THE POWER TO KNOW.

SAS® Visual Data Mining and Machine Learning 8.3
Programming Guide

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2018. *SAS® Visual Data Mining and Machine Learning 8.3: Programming Guide*. Cary, NC: SAS Institute Inc.

SAS® Visual Data Mining and Machine Learning 8.3: Programming Guide

Copyright © 2018, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

July 2018

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

SAS software may be provided with certain third-party software, including but not limited to open-source software, which is licensed under its applicable third-party software license agreement. For license information about third-party software distributed with SAS software, refer to `http://support.sas.com/thirdpartylicenses`.

Contents

Chapter 1. Analytic Store Scoring Action Set . . . 1
Chapter 2. Association Rule Mining Action Set . . . 17
Chapter 3. Audio Action Set . . . 47
Chapter 4. Autotune Action Set . . . 65
Chapter 5. Bayesian Net Classifier Action Set . . . 185
Chapter 6. BioMedImage Action Set . . . 213
Chapter 7. Boolean Rule Action Set . . . 239
Chapter 8. Factorization Machine Action Set . . . 277
Chapter 9. Fast k-Nearest Neighbors Algorithm Action Set . . . 293
Chapter 10. Generalized Linear Multitask Learning Action Set . . . 309
Chapter 11. Graph-Based Semisupervised Learning Action Set . . . 333
Chapter 12. Graphical Variable Clustering Action Set . . . 347
Chapter 13. Image Action Set . . . 369
Chapter 14. Language Model Action Set . . . 473
Chapter 15. Machine Learning Tools Action Set . . . 491
Chapter 16. The Network Action Set . . . 499
Chapter 17. Neural Network Action Set . . . 785
Chapter 18. Nonparametric Bayes Action Set . . . 831
Chapter 19. Robust PCA Action Set . . . 853
Chapter 20. Support Vector Data Description Action Set . . . 907
Chapter 21. Support Vector Machine Action Set . . . 927
Chapter 22. Text Mining Action Set . . . 943
Chapter 23. t-Distributed Stochastic Neighbor Embedding Action Set . . . 1049
Appendix A. Common Parameters . . . 1061

Subject Index **1113**

Syntax Index **1115**

Chapter 1
Analytic Store Scoring Action Set

Contents

Syntax **1**
The describe Action 2
The score Action 4
Details **12**
The describe Action 12
The score Action 13
Examples **13**
Example 1.1: Describing an Analytic Store 13
Example 1.2: Scoring the German Credit Benchmark Data 15

Analytic Store Scoring Action Set: Syntax

describe Action

CASL Syntax

aStore.describe <**result**=*results*> <**status**=*rc*> /

epcode=TRUE | FALSE

options={{

name="*string*",

value=*double*

}, {...}}

***required parameter** **rstore**={

caslib="*string*",

dataSourceOptions={*key-1*=*any-list-or-data-type-1* <, *key-2*=*any-list-or-data-type-2*, ...>},

*required parameter **name**="*table-name*"

}

;

Parameter Descriptions

epcode=TRUE | FALSE

when set to True, produces the basic DS2 code that can be edited for preprocessing of input variables and postprocessing of output variables.

Default FALSE

options={{*astore_option_list-1*} <, {*astore_option_list-2*}, ...>}

specifies a list of run-time option names and their corresponding values.

The astore_option_list value can be one or more of the following:

name="*string*"

Run-time option name.

value=*double*

Run-time option value.

* **rstore**={*castable*}

specifies a binary table that contains the analytic store.

Long form rstore={name="*table-name*"}

Shortcut form rstore="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

* **name**="*table-name*"

specifies the name of the table to use.

Analytic Store Scoring Action Set: Syntax

score Action

CASL Syntax

aStore.score <**result**=*results*> <**status**=*rc*> /

*__required parameter__ **casout**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*"

}

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

ds2code="*string*"

multiplerstores={{

caslib="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

*required parameter **name**="*table-name*"

}, {...}}

options={{

name="*string*",

value=*double*

}, {...}}

rstore={

caslib="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

```
    *required parameter name="table-name"
  }
*required parameter table={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    groupByMode="NOSORT" | "REDISTRIBUTE",
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    orderBy={{
        format="string",
        formattedLength=integer,
        label="string",
```

```
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
        }, {...}},
      singlePass=TRUE | FALSE,
      vars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
        }, {...}},
      where="where-expression"
  }
;
```

Parameter Descriptions

*** casout**={*casouttable*}

specifies an output table.

Alias out

Long form casout={name="*table-name*"}

Shortcut form casout="*table-name*"

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

ds2code="*string*"

specifies the DS2 code that preprocesses input variables and postprocesses output variables.

multiplerstores={{*castable-1*} <, {*castable-2*}, ...>}

specifies one or more binary tables that contains the analytic store.

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

* **name**="*table-name*"

specifies the name of the table to use.

options={{*astore_option_list-1*} <, {*astore_option_list-2*}, ...>}

specifies a list of run-time option names and their corresponding values.

The astore_option_list value can be one or more of the following:

name="*string*"

Run-time option name.

value=*double*

Run-time option value.

rstore={*castable*}

specifies a binary table that contains the analytic store.

Long form rstore={name="*table-name*"}

Shortcut form rstore="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

* **name**="*table-name*"

specifies the name of the table to use.

* **table**={*castable*}

specifies an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

Details

The describe Action

The **describe** action in the **aStore** action set provides information about the contents of an analytic store that is produced by an action that supports the **savestate** parameter. For more information about two such actions, see Chapter 21, "Support Vector Machine Action Set," and Chapter 8, "Factorization Machine Action Set."

An analytic store is a binary file that contains the state of an action after it completes the training phase of data analysis. The **describe** action displays some of the contents of the analytic store and helps you handle multiple analytic stores.

The **describe** action produces the following information about the analytic store:

- run information
- input variable list
- output variable list
- store key identifier, which is needed when you are using additional DS2 language programming statements. (For more information abut the DS2 language, see *SAS DS2 Language Reference*.) The key is a string that identifies each specific analytic store. The key changes every time the analytic action creates a state by using the **savestate** parameter.

The **describe** action can assist you in producing a basic form of the embedded processing (EP) code. The EP code can be edited and reused as a parameter in the scoring phase. When you edit the file, you must follow the DS2 programming language. Editing the EPCODE provides more control over the scoring process; for example, you can transform the input variables just before scoring the record and you can flag or change the decisions made by the analytic engine after scoring the record.

For more information about the capabilities of the **describe** action, see the DESCRIBE statement in Chapter 4, "The ASTORE Procedure" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*).

Results

The results that the **describe** action produces are stored in tables. You can access these tables by their names, which are shown in Table 1.1, along with a short description of the table's contents.

Table 1.1 Results Tables Produced by the **describe** Action

Table Name	Description
Key	Identifier of the store
Description	Basic information about the analytic model
InputVariables	Description of the input variables
OutputVariables	Description of the output variables

The score Action

The **score** action in the **aStore** action set scores an input data table by using an analytic store that is produced by an action that supports the **savestate** parameter. For examples of such actions, see Chapter 21, "Support Vector Machine Action Set," and Chapter 8, "Factorization Machine Action Set."

For more information about the capabilities of the **score** action, see Chapter 4, "The ASTORE Procedure" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*).

Examples

Example 1.1: Describing an Analytic Store

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example uses the dmagecr data set from the SAS Sample library to illustrate how to describe an analytic store that is created by another SAS Viya action and saved in a data table.

The dmagecr contains 1,000 observations and contains the variables age, amount, and so on.

This example assumes that the analytic store has been created and is visible by the CAS session and that all the necessary input tables are visible so that they can be accessed.

The following DATA step loads the dmagecr data set into a CAS data table named mycas.dmagecr and adds an id variable for each input record of the input table. The id variable is also copied to the output table. It acts as a record identifier and is needed to join the records in the input table to their corresponding records in the output table. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately named CAS engine libref.

```
data mycas.dmagecr;
   set sampsio.dmagecr;
   id=_n_;
run;
```

The following code uses the **svmtrain** subaction of the **svm** action set to produce an analytic store. The input table is dmagecr, and the input interval variables checking, coapp, and depend are specified in the **var** parameter of the action. The target variable telephon is specified in the **emtarget** parameter. The state of the training is saved in a table named save, as specified in the **savestate** parameter.

```
proc cas;
   action svm.svmtrain /
      table={name='dmagecr'},
      var={'checking', 'coapp', 'depends'},
      emtarget={name='telephon', options={order='formatted', leveltype='nominal'}},
      kernel=2,
      degree=2,
      id= {'id'},
      savestate={name='save', replace=true};
run;
```

The following statements use PROC CAS to load the **aStore** action set and run the **describe** action. The ODS TRACE ON statement tracks the ODS tables that the **describe** action produces.

```
ods trace on;
proc cas;
   aStore.describe rstore={name='save'}, epcode=TRUE;
   run;
quit;
```

Output 1.1.1 shows the following tables, which are produced by the preceding code:

- The “Run Information” table contains the analytic component and the time the state was saved.
- The “Input Variables” table contains the description of the input variables.
- The “Output Variables” table contains the description the output variables.

Output 1.1.1 Output Tables from the Describe Action

Run Information

Analytic Engine	svmachine
Time Created	18May2018:12:22:50

Output 1.1.1 *continued*

Input Variables

Name	Length	Role	Measurement Level	Data Type	Format
checking	8	Input	Interval	Num	
coapp	8	Input	Interval	Num	
depends	8	Input	Interval	Num	
telephon	8	Target	Classification	Num	
id	8	Id		Num	

Output Variables

Name	Length	Measurement Level	Label
id	8	Num	
_P_	8	Num	Decision Function
P_telephon2	8	Num	Predicted: telephon=2
P_telephon1	8	Num	Predicted: telephon=1
I_telephon	32	Char	Into: telephon
WARN	4	Char	Warnings

Every time you re-create the store for the same analytic problem, the Time Created row in the "Run Information" table changes.

Example 1.2: Scoring the German Credit Benchmark Data

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example uses the dmagecr data set from the SAS sample library to illustrate how to score an input data table in SAS Viya. The dmagegr data set contains 1,000 observations and contains the variables age, amount, and so on. For an exhaustive listing of the contents of this data set, you can run the following statements:

```
proc contents data=sampsio.dmagecr;
run;
```

This example assumes that the analytic store has been created and is visible by the session and that all the necessary input tables are visible so that they can be accessed.

The following DATA step loads the dmagecr data set into a CAS data table named dmycas.magecr and adds an id variable that will act as a record identifier for joining the input table to the output table. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately named CAS engine libref.

```
data mycas.dmagecr;
   set sampsio.dmagecr;
   id=_n_;
run;
```

The following statements use the **svmtrain** subaction of the **svm** action set to produce an analytic store. The input table is dmagecr, and the input interval variables are checking, coapp, and depend, as seen in the **var** parameter. The target variable telephon is specified in the **emtarget** parameter. The state of the training is saved in a table named save, as seen in the **savestate** parameter.

```
proc cas;
   action svm.svmtrain /
      table={name='dmagecr'},
      var={'checking', 'coapp', 'depends'},
      emtarget={name='telephon', options={order='formatted', leveltype='nominal'}},
      kernel=2,
      degree=2,
      id= {'id'},
      savestate={name='save', replace=true};
run;
```

The following statements use the input table mycas.dmagecr and the analytic store in the table mycas.save to produce the output table mycas.dmagecr:

```
proc cas;
   loadactionset "aStore";
   action aStore.score /
      table={name='dmagecr'},
      out={name='dmagecrout'},
      rstore={name='save'};
run;
quit;
```

The following statements print the contents of five observations from the output table dmagecrout:

```
proc print data=mycas.dmagecrout(where=(id <= 5));
run;
```

Output 1.2.1 shows the results, where you see that the id variable is not ordered. Because the scoring is done in CAS—a distributed scoring environment—it is expected that the output table might not be in the same order as the input table. You must do additional programming if you need to sort the output by the id variable.

Output 1.2.1 Scored Observations

Obs	id	_P_	P_telephon2	P_telephon1	I_telephon	_WARN_
1	1	1.00000	.000000056	1.00000	1	
2	4	1.00000	.000000014	1.00000	1	
3	2	1.00000	.000000118	1.00000	1	
4	5	1.00000	.000000060	1.00000	1	
5	3	1.00000	.000000070	1.00000	1	

Chapter 2
Association Rule Mining Action Set

Contents

Syntax **17**
The mbAnalysis Action 18
The fism Action 32
Details **39**
The mbAnalysis Action 39
The fism Action 41
Examples **43**
Example 2.1: Training a Model by Using the mbAnalysis Action 43
Example 2.2: Identifying Frequent Item Sets Using the fism Action 45
References **46**

Association Rule Mining Action Set: Syntax

mbanalysis Action

CASL Syntax

ruleMining.mbanalysis <**result**=*results*> <**status**=*rc*> /

antecedentList={*"string-1"* <, *"string-2"*, ...>}

conf=*double*

consequentList={*"string-1"* <, *"string-2"*, ...>}

hierarchy={{

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-*

```
        parameters},
    *required parameter name="table-name",
     onDemand=TRUE | FALSE,
     orderBy={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
     }, {...}},
     singlePass=TRUE | FALSE,
     vars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
     }, {...}},
     where="where-expression"
}, {...}}
*required parameter idVariable="variable-name"
 items=integer
 lift=double
 maxItems=integer
 minItems=integer
 nLHS_range={
    *required parameter lower=integer,
    *required parameter upper=integer
}
 nRHS_range={
    *required parameter lower=integer,
```

*required parameter **upper**=*integer*
}
norm=TRUE | FALSE
out={
caslib="*string*",
compress=TRUE | FALSE,
indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},
label="*string*",
maxMemSize=*64-bit-integer*,
name="*table-name*",
onDemand=TRUE | FALSE,
promote=TRUE | FALSE,
replace=TRUE | FALSE,
replication=*integer*,
timeStamp="*string*",
where={*"string-1"* <, *"string-2"*, ...>}
}
outfreq={
caslib="*string*",
compress=TRUE | FALSE,
indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},
label="*string*",
maxMemSize=*64-bit-integer*,
name="*table-name*",
onDemand=TRUE | FALSE,
promote=TRUE | FALSE,
replace=TRUE | FALSE,
replication=*integer*,
timeStamp="*string*",
where={*"string-1"* <, *"string-2"*, ...>}
}
outrule={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

saveState={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

separator="*string*"

sup_lift=*double*

supmin=*double*

suppct=*double*

***required parameter** **table**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

```
        }, {...}},
        singlePass=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
    }
   *required parameter tgtVariable="variable-name"
;
```

Parameter Descriptions

antecedentList={*"string-1"* <, *"string-2"*, ...>}

specifies the regular expression strings to match in the antecedent (left-hand side) of a rule.

conf=*double*

specifies the minimum confidence for the rules.

Default 50

Range 0–100

consequentList={*"string-1"* <, *"string-2"*, ...>}

specifies the regular expression strings to match in the consequent (right-hand side) of a rule.

hierarchy={{*castable-1*} <, {*castable-2*}, ...>}

specifies one or more hierarchy tables. If you omit this parameter, then the mbanalysis action performs simple association analysis without a hierarchy. You can specify up to five tables; each table specifies one level of the hierarchy.

Maximum items 5

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in

the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

* **idVariable**="*variable-name*"

specifies variable that is used to group the target variable into baskets.

items=*integer*

specifies the number of items in a rule. The default value is 2 when either the out parameter or outrule parameter is specified and 1 otherwise.

Default 2

Range 1–1000

lift=*double*

specifies the minimum lift value necessary to generate a rule.

Default 1

Range 0–100

maxItems=*integer*

specifies a maximum basket size, such that baskets larger than the specified value are rejected.

Default 1000

Range 0–1000

minItems=*integer*

specifies a minimum basket size, such that baskets smaller than the specified value are

rejected.

Default 1

Range 0–20

nLHS_range={*nLHSRHSOpts*}

* **lower**=*integer*

  specifies the minimum number of items in the left-hand side or right-hand side (LHS/RHS) of a rule.

  Default 1

  Range 1–1000

* **upper**=*integer*

  specifies the maximum number of items in the left-hand side or right-hand side (LHS/RHS) of a rule.

  Default 1000

  Range 1–1000

nRHS_range={*nLHSRHSOpts*}

* **lower**=*integer*

  specifies the minimum number of items in the left-hand side or right-hand side (LHS/RHS) of a rule.

  Default 1

  Range 1–1000

* **upper**=*integer*

  specifies the maximum number of items in the left-hand side or right-hand side (LHS/RHS) of a rule.

  Default 1000

  Range 1–1000

norm=TRUE | FALSE

when set to True, normalizes the values of the target variable and the items in the out tables.

Default FALSE

out={*casouttable*}

specifies the table to contain frequent item sets to generate rules. This output table also includes transaction counts and support.

For more information about specifying the out parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outfreq={*casouttable*}

specifies the table to contain the unique frequent items along with their transaction counts and support.

For more information about specifying the outfreq parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outrule={*casouttable*}

specifies the table to contain the rules. This output table includes variables that identify the left-hand and right-hand sides of the rule, the support, and the lift.

For more information about specifying the outrule parameter, see the common casouttable parameter (Appendix A: Common Parameters).

saveState={*casouttable*}

specifies the table in which to save the model for future scoring.

For more information about specifying the saveState parameter, see the common casouttable parameter (Appendix A: Common Parameters).

separator="*string*"

specifies the separator character in the antecedent (left-hand side) or the consequent (right-hand side) of a rule. The default value is "&".

sup_lift=*double*

specifies the minimum support lift necessary to generate a rule.

Default 0

supmin=*double*

specifies the minimum level of support for a rule. This represents the minimum allowable frequency of an item. This parameter overrides the suppct parameter.

Minimum value 0

suppct=*double*

specifies the minimum level of support for a rule as a percentage of the number of baskets in the input data table.

Range 0–100

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix

A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

* **tgtVariable**="*variable-name*"

specifies a single nominal variable to use as the target variable. If this variable is not present in the data table that is specified in the table parameter, then the action exits with an error.

Association Rule Mining Action Set: Syntax

fism Action

CASL Syntax

ruleMining.fism <**result**=*results*> <**status**=*rc*> /

*__required parameter__ **idVariable**="*variable-name*"

items=*integer*

nFis_Range={

*required parameter **lower**=*integer*,

*required parameter **upper**=*integer*

}

out={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outFreq={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

```
      onDemand=TRUE | FALSE,
      promote=TRUE | FALSE,
      replace=TRUE | FALSE,
      replication=integer,
      timeStamp="string",
      where={"string-1" <, "string-2", ...>}
   }
   supmin=double
   suppct=double
*required parameter table={
      caslib="string",
      computedOnDemand=TRUE | FALSE,
      computedVars={{
         format="string",
         formattedLength=integer,
         label="string",
        *required parameter name="variable-name",
         nfd=integer,
         nfl=integer
      }, {...}},
      computedVarsProgram="string",
      dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
      groupBy={{
         format="string",
         formattedLength=integer,
         label="string",
        *required parameter name="variable-name",
         nfd=integer,
         nfl=integer
      }, {...}},
      groupByMode="NOSORT" | "REDISTRIBUTE",
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
```

```
        parameters},
      *required parameter name="table-name",
       onDemand=TRUE | FALSE,
       orderBy={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
       }, {...}},
       singlePass=TRUE | FALSE,
       vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
       }, {...}},
       where="where-expression"
  }
  *required parameter tgtVariable="variable-name"
;
```

Parameter Descriptions

*** idVariable**="*variable-name*"

specifies variable that is used to group the target variable into baskets.

items=*integer*

specifies the number of items in a rule. The default value is 2 when either the out parameter or outrule parameter is specified and 1 otherwise.

Default 2

Range 1–1000

nFis_Range={*nFisRangeOpts*}

* **lower**=*integer*

  specifies the minimum number of items in the frequent item set.

  Default 1

  Range 1–1000

* **upper**=*integer*

  specifies the maximum number of items in the frequent item set.

  Default 1000

  Range 1–1000

out={*casouttable*}

specifies the table to contain frequent item sets to generate rules. This output table also includes transaction counts and support.

For more information about specifying the out parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outFreq={*casouttable*}

specifies the table to contain the unique frequent items along with their transaction counts and support.

For more information about specifying the outFreq parameter, see the common casouttable parameter (Appendix A: Common Parameters).

supmin=*double*

specifies the minimum level of support for a rule. This represents the minimum allowable frequency of an item. This parameter overrides the suppct parameter.

Minimum value 0

suppct=*double*

specifies the minimum level of support for a rule as a percentage of the number of baskets in the input data table.

Range 0–100

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

* **tgtVariable**="*variable-name*"

specifies a single nominal variable to use as the target variable. If this variable is not present in the data table that is specified in the table parameter, then the action exits with an error.

Details

The mbAnalysis Action

The **mbAnalysis** action performs association rule mining (ARM) on transaction data in conjunction with an item that is specified in one or more hierarchies (also called groups or categories). Association rule mining is also known as market basket analysis. Market basket analysis is useful in retail marketing scenarios that involve tens of thousands of distinct items that are grouped in a hierarchy. This action uses the transaction and hierarchical data and generates rules at multiple levels. These rules are called generalized association rules; they solve some of the problems, which are described in this section, that arise in simple market basket analysis.

Simple market basket analysis that is performed on transaction or point-of-sale data might miss potentially useful and significant associations. For example, consider a supermarket that sells different types of breads and a wide selection of wines, and assume that a large number of customers buy some type of bread with some type of wine. Simple market basket analysis might fail to spot a rule that links bread and wine at the transaction level. It computes the "support" for specific types of breads to specific types of wines, none of which might be large enough to generate a rule. To overcome this problem, the **mbAnalysis** action combines hierarchies with the transaction data and computes the support for any type of bread to any type of wine.

In addition, simple association mining also tends to create a large number of obvious and uninteresting rules along with the useful rules. A solution to the preceding problem that reduces the support would only exacerbate the problem. In practice, this tendency is one of the major drawbacks of association rule mining. If the support is set high, fewer rules are generated, but more of them are obvious and thus useless. If the support is set low, too many rules are generated, and the domain experts must evaluate the rules to determine which ones are useful. One solution to this drawback is generalized association rule mining.

The **mbAnalysis** action enables you to generate an interesting measure, which is called support lift. This measure is based on the deviation of a rule's support from its estimated support, which is computed on the basis of the support of the ancestors of the rule's items. Objectively, greater deviation is more likely to be interesting. If no hierarchy of ancestors is specified, then this action generates simple association rules.

For example, the following rule for a supermarket means that if customers buy butter and bread, they also buy milk:

$$butter, bread => Milk$$

Support (coverage) is an indication of how frequently the items appear in the database. Support of a rule for A and B is expressed as

$$A => B : P(A, B) = X/Y$$

where X is the number of transactions that contain both A and B, and Y is the total number of transactions.

Confidence (accuracy) indicates the number of times the conditional statement of a rule has been found to be true. Support of a rule for A and B is expressed as

$$A => B : P(B|A) = X/Z$$

where X is the number of transactions that contain both A and B, and Z is the number of transactions that contain A.

This action works in a single thread as well as in multiple threads, both on a single node and on multiple nodes.

Displayed Output

The **mbAnalysis** action does not produce any displayed output tables.

Results

The results that the **mbAnalysis** action produces are stored in tables. You can access these tables by their names, which are shown in Table 2.1.

Table 2.1 Results Tables Produced by the **mbAnalysis** Action

Table Name	Description	Action	Parameter
out	Contains required information to generate rules	**mbAnalysis**	**out**
outfreq	Contains information about the items' frequencies	**mbAnalysis**	**outFreq**
outrule	Contains detailed information about rules	**mbAnalysis**	**outRule**
savestate	Contains the output binary scoring file	**mbAnalysis**	**saveState**

Scoring with the `score` Action

The **saveState** parameter stores the analytical result in a binary file format that can be used as an input file for scoring with the **score** action. Because the transactional table is in long format and the **score** action accepts input in wide format, you must convert the input data format from long to wide before providing it as input to the **score** action.

The normalization option can be specified in the **options** parameter:

- A normalization option can be set by specifying the name **NORM** and the value **1** in the **options** parameter.

The scoring process is supported by one of the following algorithms, which you can specify in the **options** parameter:

- Rule association: A rule is selected if its antecedent and consequent item sets are included in the input item set. You can select this algorithm by specifying the name **ARM_SCORING_ALGO** and the value **1** in the **options** parameter.

- Exclusive recommendation: A rule is selected if its antecedent item set is a subset of the input item set and if its consequent item set is not a subset of the input item set. You can select this algorithm by specifying the name **`ARM_SCORING_ALGO`** and the value **`2`** in the **`options`** parameter.
- Recommendation: A rule is selected if its antecedent item set is a subset of the input item set. You can select this algorithm by specifying the name **`ARM_SCORING_ALGO`** and the value **`3`** in the **`options`** parameter.

In order to maintain consistency, the scoring output of the **`score`** action must be converted from wide to long format again.

Limitations

When the number of observations is greater than 10 million and the number of items in a rule (as specified in the **`items`** parameter) is considerably large (more than 10), the amount of memory that is required to process the analysis might be so great that the procedure stops responding.

When one item has multiple parents, only the last parent is considered for generating rules. The last parent item might vary if the data are distributed.

The maximum number of rules that are generated per thread on each node is one million.

The fism Action

The **`fism`** action uses support input to identify the frequent item sets. By default, a maximum of two item sets are generated. You can specify the item set size in the **`items`** parameter.

The **`fism`** action uses the FP-growth algorithm to find the frequent item sets. The action finds the frequent item set by using the support measure. Support is calculated as

Support (coverage) of

$$A => B : P(A, B) =$$

(Number of transactions containing both A and B) / (Total number of transactions)

FP-Growth Algorithm

Association rule mining (ARM) is an important data mining task that tries to find interesting rules from a transactional data set. Association rule mining is formally described in Agrawal, Imieliński, and Swami (1993).

Many association rule mining algorithms are based on a “support-confidence” framework, which has two major steps: First, minimum support is applied to find all frequent item sets in a data set. Second, these frequent item sets and the minimum confidence constraint are used to generate rules. Discovering all frequent item sets in a data set is difficult because the task is combinatorial.

The Apriori algorithm of Agrawal and Srikant (1994) uses a breadth-first search (BFS) strategy to count the support of item sets and uses a candidate generation function that exploits the downward-closure property of support in Agrawal, Imieliński, and Swami (1993). The basic idea of the downward-closure property is that the support of an item set is less than a particular threshold if the support of any subset of this item set is less than that threshold. In this way, many small item sets are excluded during the generation-and-test process. The Apriori algorithm can reduce the search space through a restricted generation-and-test process. The Apriori algorithm is the one of the most efficient association rule mining algorithms.

The Apriori algorithm and many similar algorithms need to scan the data set n–2 times when generating an item set that contains n items. The frequent pattern growth (FP-growth) algorithm builds a frequent-pattern tree (FP-tree) by scanning the data set twice (Han, Pei, and Yin 2000; Han et al. 2004). Then it discovers the frequent item sets by traversing the FP-tree in a depth-first search (DFS). DFS can find all the frequent item sets without enumerating all the candidates in 2^n search space. Therefore, FP-growth algorithms are faster than BFS-based algorithms. The nature of the FP-growth algorithm is restricted-test-only without generation of the n-item set.

The FP-tree construction algorithm first scans the data set and then collects the set of frequent items, F, and their supports. Next, it sorts F in support-descending order as L, the list of frequent items. Frequent items are sorted in frequency-descending order so that there is a better chance that more prefix strings can be shared.

An FP-tree has two components: an item prefix tree and a header table of frequent items. Each node in the item prefix tree consists of three fields: item name, count, and node link. Here item name registers which item this node represents; count registers the number of transactions represented by the portion of the path that reaches this node; and node link connects to the next node in the FP-tree that carries the same item name, or null if there is none. Each entry in the frequent-item header table consists of two (or three) fields: item name, count (frequency count of this one-item set), and head of node link.

More frequent items are arranged closer to the top of the FP-tree and thus are more likely to be shared when items are sorted in the support-descending order. This arrangement suggests that an FP-tree is usually a highly compact projection of the entire data table. However, if the data table is large, the FP-tree might not fit in memory. There are two ways to distribute the data:

- Parallel projection is done in parallel. Each observation is projected to every node. The problem is that the total size of the projected data tables could be multiple times the size of the original data table.
- Partition projection is done in sequence. An observation T is projected to the X-projected data table only if X is a frequent item in T and there is no other item after X in the list of frequent items that appears in the observation. The advantage of partition projection is that the total size of the projected data tables at each level is smaller than the original data table.

The frequent item sets are mined by traversing a node's conditional pattern base (under the condition of the node's existence) and thus are called a conditional FP-tree (CFPT). The mining function is defined as mine(CFPT | node), which is also a recursive function that uses a divide-and-conquer process.

Displayed Output

The **fism** action does not produce any displayed output tables.

Results

The results that the **fism** action produces are stored in tables. You can access these tables by their names, which are shown in Table 2.1.

Table 2.2 Results Tables Produced by the **fism** Action

Table Name	Description	Action	Parameter
out	Contains details about frequent item sets	**fism**	**out**
outfreq	Contains information about the items' frequencies	**fism**	**outFreq**

Limitations

When the number of observations is greater than 10 million and the number of items in a rule (as specified in the **items** parameter) is considerably large (more than 10), the amount of memory that is required to process the analysis might be so great that the procedure stops responding.

The maximum number of item sets that are generated per thread on each node is 10 million.

Examples

Example 2.1: Training a Model by Using the mbAnalysis Action

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example trains the model by using data that are available in the sampsio.assocs data set. This data set contains 1,000 transactions of 1,000 customers. Each observation contains information about the customer in the Customer variable, information about the product in the Product variable (which is the target variable), and information about time in the Time variable (which is ignored). To perform association discovery, the input data table must have a separate observation for each product that was purchased by each customer. You must assign the Customer (ID) role to one variable and the target model role to another variable when you create the data table. Only one variable is supported as the Customer variable, and only one variable is supported as the target variable.

You can load the sampsio.assocs data set into your CAS session by specifying your CAS engine libref in the first statement in the following DATA step. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.assocs;
   set sampsio.assocs;
run;
```

The following DATA steps create two level hierarchies: firstLevel and secondLevel.

```
data mycas.firstLevel;
  length Item $20 Category $20;
    Item='apples'; Category='Fruits'; output;
    Item='artichok'; Category='Vegetables'; output;
    Item='avocado'; Category='Fruits'; output;
    Item='baguette'; Category='Bread'; output;
    Item='bordeaux'; Category='Wine'; output;
    Item='bourbon'; Category='Whisky'; output;
    Item='chicken'; Category='Non-Veg'; output;
    Item='coke'; Category='Cold-Drink'; output;
    Item='corned_b'; Category='Non-Veg'; output;
    Item='cracker'; Category='Vegetables'; output;
    Item='ham'; Category='Non-Veg'; output;
    Item='heineken'; Category='Beer'; output;
    Item='hering'; Category='Non-Veg'; output;
    Item='ice_crea'; Category='Cold-Drink'; output;
    Item='olives'; Category='Vegetables'; output;
    Item='peppers'; Category='Fruits'; output;
    Item='sardines'; Category='Non-Veg'; output;
    Item='soda'; Category='Cold-Drink'; output;
    Item='steak'; Category='Non-Veg'; output;
    Item='turkey'; Category='Non-Veg'; output;
 run;

data mycas.secondLevel;
  length Category $20 Dept $20;
    Category='Fruits'; Dept='Breakfast'; output;
    Category='Vegetables'; Dept='Breakfast'; output;
    Category='Bread'; Dept='Breakfast'; output;
    Category='Wine'; Dept='Drinks'; output;
    Category='Whisky'; Dept='Drinks'; output;
    Category='Non-Veg'; Dept='Breakfast'; output;
    Category='Cold-Drink'; Dept='Drinks'; output;
    Category='Beer'; Dept='Drinks'; output;
 run;
```

The following statements run an association analysis on the mycas.assocs data table by using the **mbAnalysis** action:

```
proc cas;
    action ruleMining.mbanalysis /
        hierarchy={{name="firstLevel"}, {name="secondLevel"}}
```

```
            supmin=1 items=8 minitems=2 maxitems=10
            nLHS_range={Lower=2 UPPER=3} nRHS_range={Lower=4 UPPER=5}
            separator = "*" antecedentList={"Fruits"} consequentList={"Non-Veg" }
            idVariable="Customer"
            out={name="out", replace=true}
            outrule={name="outrule", replace=true}
            outfreq={name="outfreq", replace=true}
            table={name="assocs"}
            tgtVariable="Product"
            savestate= {name="mystateArm", replace=true} ;
    run;
    quit;
```

The **table** parameter names the input data table to be analyzed. The **supmin** parameter specifies the minimum level of support for a rule. The **items** parameter specifies the number of items in a rule. The **minitems** parameter specifies the minimum basket size to be analyzed, and the **maxitems** parameter specifies the maximum basket size to be analyzed. The **nLHS_range** parameter specifies the range of items in the left-hand side (LHS) of a rule, and the **nRHS_range** parameter specifies the range of items in the right-hand side (RHS) of a rule. The **separator** parameter specifies the separator character in the antecedent (left-hand side) or consequent (right-hand side) of a rule. The default value is "&". The **antecedentList** parameter specifies the list of regular expression strings to match in the antecedent of a rule. The **consequentList** parameter specifies the list of regular expression strings to match in the consequent of a rule. The **hierarchy** parameter specifies items' hierarchies (also called groups or categories). The **idVariable** parameter specifies the variable that is used to group the target variable into baskets. The **tgtVariable** parameter specifies a single categorical variable to use as the target variable. The **out** parameter specifies the table to contain frequent item sets to generate rules. The **outfreq** parameter specifies the table to contain unique frequent items along with their transaction counts and support. The **outrule** parameter specifies the table to contain rules along with their support, confidence, and lift.

Example 2.2: Identifying Frequent Item Sets Using the fism Action

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example identifies frequent item sets by using data that are available in the sampsio.assocs data set. This data set contains 1,000 transactions of 1,000 customers; each observation contains information about the customers, time, and item or product.

You can load the sampsio.assocs data set into your CAS session by specifying your CAS engine libref in the first statement in the following DATA step. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.assocs;
   set sampsio.assocs;
run;
```

The following statements run the FP-growth algorithm on the mycas.assocs data table by using the **fism** action:

```
proc cas;
    action ruleMining.fism /
        supmin=200 items=5 nFis_Range={Lower=2 UPPER=5}
        idVariable="Customer"
        out={name="out", replace=true}
        outfreq={name="outfreq", replace=true}
        table={name="assocs"}
        tgtVariable="Product" ;
run;
```

The **table** parameter names the input data table to be analyzed. The **supmin** parameter specifies the minimum level of support for a rule. The **nFis_Range** parameter specifies the range of number of items in the frequent item sets. The **items** parameter specifies the number of items in a rule. The **idVariable** parameter specifies the variable to be used to group the target variable into baskets. The **tgtVariable** parameter specifies the categorical variable to use as the target variable. The **out** parameter specifies the table to contain frequent item sets to generate rules. The **outfreq** parameter specifies the table to contain unique frequent items along with their transaction counts and support.

References

Agrawal, R., Imieliński, T., and Swami, A. (1993). "Mining Association Rules between Sets of Items in Large Databases." In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*. New York: ACM Press. http://crono.dei.unipd.it/~lds/MATERIALE/AgrawalIS93.pdf.

Agrawal, R., and Srikant, R. (1994). "Fast Algorithms for Mining Association Rules in Large Databases." In *Proceedings of the 20th International Conference on Very Large Data Bases*, 487–499. San Francisco: Morgan Kaufmann. http://www.vldb.org/conf/1994/P487.PDF.

Han, J., Pei, J., and Yin, Y. (2000). "Mining Frequent Patterns without Candidate Generation." In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. New York: ACM Press. https://www.cs.sfu.ca/~jpei/publications/sigmod00.pdf.

Han, J., Pei, J., Yin, Y., and Mao, R. (2004). "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach." *Data Mining and Knowledge Discovery* 8:53–87.

Chapter 3
Audio Action Set

Contents

Syntax **47**
The computeFeatures Action 48
The loadAudio Action 59
Details **61**
About the Audio Action Set 61
The computeFeatures Action 61
The loadAudio Action 62
Examples **63**
Example 3.1: Computing MFCC Features by Using Default Options 63
Example 3.2: Specifying the Input Table's Audio Column Name When Computing Features 63
Example 3.3: Specifying Input Variables and Copy Variables When Computing Features 63
Example 3.4: Loading Audio Files by Using a File Listing 63

Audio Action Set: Syntax

computeFeatures Action

CASL Syntax

```
audio.computeFeatures <result=results> <status=rc> /
    audioColumn="string"
  *required parameter casOut={
        caslib="string",
        compress=TRUE | FALSE,
        indexVars={"variable-name-1" <, "variable-name-2", ...>},
        label="string",
        maxMemSize=64-bit-integer,
        name="table-name",
        onDemand=TRUE | FALSE,
        promote=TRUE | FALSE,
        replace=TRUE | FALSE,
        replication=integer,
        timeStamp="string",
        where={"string-1" <, "string-2", ...>}
  }
    copyVars={"variable-name-1" <, "variable-name-2", ...>}
    fbankOptions={
        energyFloor=double,
        rawEnergy=TRUE | FALSE,
        useEnergy=TRUE | FALSE,
        useLogFbank=TRUE | FALSE,
        usePower=TRUE | FALSE
    }
    featureScalingMethod="NONE" | "STANDARDIZATION"
    frameExtractionOptions={
        blackmanCoeff=double,
        dither=double,
```

frameLength=*double*,

frameShift=*double*,

preemphCoeff=*double*,

removeDcOffset=TRUE | FALSE,

roundToPowerOfTwo=TRUE | FALSE,

snipEdges=TRUE | FALSE,

windowType="BLACKMAN" | "HAMMING" | "HANNING" | "RECTANGULAR"

}

melBanksOptions={

highFreq=*double*,

lowFreq=*double*,

nBins=*integer*

}

mfccOptions={

cepstralLifter=*double*,

energyFloor=*double*,

nCeps=*integer*,

rawEnergy=TRUE | FALSE,

useEnergy=TRUE | FALSE

}

nContextFrames=*integer*

nOutputFrames=*integer*

plpOptions={

cepstralLifter=*integer*,

cepstralScale=*double*,

compressFactor=*double*,

energyFloor=*double*,

lpcOrder=*integer*,

nCeps=*integer*,

rawEnergy=TRUE | FALSE,

useEnergy=TRUE | FALSE

}

***required parameter** **table**={

```
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
  }
;
```

Parameter Descriptions

audioColumn="*string*"

specifies the name of the column in the input table containing the audio data

* **casOut**={*casouttable*}

specifies the settings for an output table.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to transfer from the input table to the output table

Requirement The specified values must be unique.

fbankOptions={*fbankOptions*}

specifies the settings used to determine how to perform the FBank feature computations

energyFloor=*double*

specifies the linear floor on energy (absolute, not relative) for the FBank feature computations

Default 0

Minimum value 0

rawEnergy=TRUE | FALSE

specifies that energy should be computed before preemphasis and windowing

Default TRUE

useEnergy=TRUE | FALSE

specifies that an extra dimension containing the computed energy should be appended to each FBank feature frame

Default FALSE

useLogFbank=TRUE | FALSE

specifies that the output should contain log-filterbank values (otherwise, the output values are linear)

Default TRUE

usePower=TRUE | FALSE

specifies that power should be used in the FBank feature computations (otherwise, the magnitude is used)

Default TRUE

featureScalingMethod="NONE" | "STANDARDIZATION"

specifies the feature scaling method to apply to the computed feature vectors

Default NONE

NONE

specifies that no feature scaling method should be applied to the computed feature vectors

STANDARDIZATION

specifies that the set containing the same coefficient across all frames in an audio file should be scaled so that the resulting set has a mean of zero and unit variance

frameExtractionOptions={*frameExtractionOptions*}

specifies the settings used to determine how to break the audio file into frames

blackmanCoeff=*double*

specifies the constant coefficient used for the generalized Blackman window (this value is ignored for other window types)

Default 0.42

dither=*double*

specifies the dithering constant (0.0 means no dithering)

Default 1

frameLength=*double*

specifies the length of a frame (in milliseconds)

Default 25

Minimum value (exclusive) 0

frameShift=*double*

specifies the time difference (in milliseconds) between the beginnings of consecutive frames

Default 10

Minimum value (exclusive) 0

preemphCoeff=*double*

specifies the coefficient used in performing signal preemphasis

Default 0.97

removeDcOffset=TRUE | FALSE

specifies that the mean of the values in each frame should be subtracted from all values in that frame

Default TRUE

roundToPowerOfTwo=TRUE | FALSE

specifies that the window size should be rounded to the next power of two

Default TRUE

snipEdges=TRUE | FALSE

specifies that end effects should be handled by outputting only frames that completely fit the data

Default TRUE

windowType="BLACKMAN" | "HAMMING" | "HANNING" | "RECTANGULAR"

specifies the type of window to apply to each frame upon extraction

Default RECTANGULAR

BLACKMAN

specifies that a Blackman window should be applied when extracting frames

HAMMING

specifies that a Hamming window should be applied when extracting frames

HANNING

specifies that a Hanning window should be applied when extracting frames

RECTANGULAR

specifies that a rectangular window should be applied when extracting frames

melBanksOptions={*melBanksOptions*}

specifies the settings used to determine the mel-frequency banks

highFreq=*double*

specifies the high cutoff frequency for the mel-frequency bins (if negative, the value is added to the Nyquist frequency, which is determined from the sampling rate of the audio)

Default 0

lowFreq=*double*

specifies the low cutoff frequency for the mel-frequency bins

Default 20

nBins=*integer*

specifies the number of triangular mel-frequency bins

Default 23

Minimum value 1

mfccOptions={*mfccOptions*}

specifies the settings used to determine how to perform the MFCC feature computations

cepstralLifter=*double*

specifies the constant which controls the scaling of the MFCC feature values

Default 22

energyFloor=*double*

specifies the linear floor on energy (absolute, not relative) for the MFCC feature computations

Default 0

Minimum value 0

nCeps=*integer*

specifies the number of cepstral coefficients in each MFCC feature frame (including C0)

Default 13

Minimum value 1

rawEnergy=TRUE | FALSE

specifies that energy should be computed before preemphasis and windowing

Default TRUE

useEnergy=TRUE | FALSE

specifies that energy (not C0) should be used in the MFCC feature computations

Default TRUE

nContextFrames=*integer*

specifies the number of context frames to append before and after the current audio frame

Default 0

Minimum value 0

nOutputFrames=*integer*

specifies the exact number of frames to include in the output table (extra frames are dropped and missing frames are padded with zeros)

Minimum value 1

plpOptions={*plpOptions*}

specifies the settings used to determine how to perform the PLP feature computations

cepstralLifter=*integer*

specifies the constant which controls the scaling of the PLP feature values

Default 22

cepstralScale=*double*

specifies the cepstral scaling constant used in the PLP computation

Default 1

Minimum value (exclusive) 0

compressFactor=*double*

specifies the compression factor used in the PLP feature computations

Default 0.33333

Minimum value (exclusive) 0

energyFloor=*double*

specifies the linear floor on energy (absolute, not relative) for the PLP feature computations

Default 0

Minimum value 0

lpcOrder=*integer*

specifies the order of the linear predictive coding (LPC) analysis used in the PLP feature computations

Default 12

Minimum value 1

nCeps=*integer*

specifies the number of cepstral coefficients in each PLP feature frame (including C0)

Default 13

Minimum value 1

rawEnergy=TRUE | FALSE

specifies that energy should be computed before preemphasis and windowing

Default TRUE

useEnergy=TRUE | FALSE

specifies that energy (not C0) should be used for the zeroth feature value in the PLP feature computations

Default TRUE

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

Audio Action Set: Syntax

loadAudio Action

CASL Syntax

audio.loadAudio <**result**=*results*> <**status**=*rc*> /

*__required parameter__ **casOut**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

caslib="*string*"

*__required parameter__ **path**="*string*"

;

Parameter Descriptions

* **casOut**={*casouttable*}

specifies the settings for an output table.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

caslib="*string*"

specifies the caslib from which to read all input files

* **path**="*string*"

specifies the path to the text file containing the listing of audio file paths to load

Details

About the Audio Action Set

The **audio** action set enables you to work with audio files in SAS Viya. Through the action set, you can load audio files from disk into memory as Cloud Analytic Services (CAS) tables. You can also compute several common features that are used in speech processing for audio data loaded into these tables.

The computeFeatures Action

The **computeFeatures** action breaks audio data into small chunks known as frames and then computes the specified speech-processing features for each extracted frame. The output table of features can then be used downstream in a feature-based machine-learning application, such as that of transliterating speech to text. The primary input to this action is a table that contains audio data, such as the output table that is created by the **loadAudio** action.

There are many parameters that you can use to configure the feature computations to meet your needs. The **frameExtractionOptions** parameter and its subparameters configure how the audio data files are divided into frames. The frequency bins to use for the audio processing are configured using the **melBanksOptions** parameter and its subparameters. The action supports three types of features: filter bank energies (FBank), mel-frequency cepstral coefficients (MFCC), and perceptual linear predictions (PLP), the specifics of which can be configured using the **fbankOptions**, **mfccOptions**, and **plpOptions** parameter groups, respectively. Typically, not all feature types are used at the same time, so in order to save both time and memory, the **computeFeatures** action does not compute those features whose corresponding options parameters are not specified in the action call. For example, if you do not want PLP features, simply do not include the **plpOptions** parameter in the action call, and PLP features will not be computed.

NOTE: When more than one feature type is computed, each feature frame in the output table contains the computed features in the following order: FBank features, MFCC features, and PLP features.

In addition to using these feature configuration parameters, you can customize the output of the **computeFeatures** action by using the following parameters:

featureScalingMethod Modifies the output by scaling the features according to some feature scaling method, described in more detail after this list.

nContextFrames Modifies the output so that each frame is couched within a specified number of preceding and succeeding context frames.

nOutputFrames Modifies the output by setting the number of frames in the output table to the specified number. If this parameter is not specified, the number of frames in the output table is set to the maximum number of output frames that are produced among all audio data files in the input table. Audio data files that produce fewer output frames than the value of this parameter will have zero-padded frames appended at the end, whereas audio data files that produce more frames than the value of this parameter simply have their extra frames dropped.

In the current release, the **featureScalingMethod** parameter has two values. When the value is 'NONE', the feature values are untouched in the output table. When the value is 'STANDARDIZATION', the values in the set that contains the same coefficient across all frames in an audio file are scaled so that the resulting set has a mean of zero and unit variance. Formally, consider an audio file from which a total of N frames are extracted, and denote the jth value in the ith frame as $f_i v_j$. The sample mean of the jth coefficients across all N frames is then

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} f_i v_j$$

whereas the unbiased sample variance of the jth coefficients across all N frames is then

$$\sigma_j^2 = \frac{1}{N-1} \sum_{i=1}^{N} \left(f_i v_j - \mu_j \right)^2$$

In order to ensure that all the sets that contain the same coefficient across all frames have a mean of zero and unit variance, $f_i v_j$ in the output table is replaced by $f_i v'_j$ such that

$$f_i v'_j = \frac{f_i v_j - \mu_j}{\sigma_j}$$

for $\forall i$ and $\forall j$.

The loadAudio Action

The **loadAudio** action is the primary method of loading audio files into a CAS table. The audio files to be loaded are specified in an audio file listing, the path of which is provided to the action via the required **path** parameter. An audio file listing is simply a text file in which paths to audio files are specified on a line-by-line basis. The paths in the file listing must be relative paths within the CAS library that is specified using the **caslib** parameter. If the CAS library is not specified, then by default the **caslib** parameter assumes the name of the active CAS library in the session.

The output table that is specified using the **casOut** parameter consists of the following two columns:

path	The full source path of the audio file in the current row
audio	A (typically large) binary object that contains the entire audio file

Examples

Example 3.1: Computing MFCC Features by Using Default Options

This example is not available for the CAS programming language.

Example 3.2: Specifying the Input Table's Audio Column Name When Computing Features

This example is not available for the CAS programming language.

Example 3.3: Specifying Input Variables and Copy Variables When Computing Features

This example is not available for the CAS programming language.

Example 3.4: Loading Audio Files by Using a File Listing

This example is not available for the CAS programming language.

Chapter 4
Autotune Action Set

Contents

Syntax **65**
The tuneBnet Action 66
The tuneDecisionTree Action 73
The tuneFactMac Action 80
The tuneForest Action 87
The tuneGradientBoostTree Action 94
The tuneNeuralNet Action 101
The tuneSvm Action 108
Details **115**
Hyperparameter Tuning 115
The tuneBnet Action 118
The tuneDecisionTree Action 120
The tuneFactMac Action 120
The tuneForest Action 121
The tuneGradientBoostTree Action 122
The tuneNeuralNet Action 123
The tuneSvm Action 125
Examples **126**
Example 4.1: Tuning the Hyperparameters of a Bayesian Network Classifier 126
Example 4.2: Tuning the Hyperparameters of a Decision Tree 136
Example 4.3: Tuning the Hyperparameters of a Factorization Machine 144
Example 4.4: Tuning the Hyperparameters of a Forest 153
Example 4.5: Tuning the Hyperparameters of a Gradient Boosting Tree Model 161
Example 4.6: Tuning the Hyperparameters of a Neural Network 169
Example 4.7: Tuning the Hyperparameters of a Support Vector Machine 176

Autotune Action Set: Syntax

tuneBnet Action

CASL Syntax

autotune.tuneBnet <**result**=*results*> <**status**=*rc*> /

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

tunerOptions={

evaluationHistory=TRUE | FALSE,

logLevel=*integer*,

maxBayesianModelSize=*integer*,

maxEvals=*integer*,

maxIters=*integer*,

maxTime=*double*,

maxTrainTime=*64-bit-integer*,

nCrossValFolds=*integer*,

nParallel=*integer*,

nSubsessionWorkers=*integer*,

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU",

popSize=*integer*,

sampleSize=*integer*,

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM",

seed=*integer*,

shuffleGridPoints=TRUE | FALSE,

targetEvent="*string*",

trainPartitionFraction=*double*,

userDefinedPartition=TRUE | FALSE,

validationPartitionFraction=*double*

}

tuningParameters={{

exclude=TRUE | FALSE,

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*",

lowerBound=*double*,

namePath="*string*",

upperBound=*double*,

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

}, {...}}

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

;

Parameter Descriptions

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the score action in the aStore action set; for more information, see the parameters for that action.

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the bnet action in the bayesianNetClassifier action set; for more information, see the parameters for that action.

tunerOptions={*tunerOptions*}

specifies a list of parameters for adjusting how the tuner behaves.

Alias optMinerOpts

evaluationHistory=TRUE | FALSE

when set to True, creates a table of every evaluation (every model configuration) and returns those tables to the client.

Default TRUE

logLevel=*integer*

specifies the level of log messages to be written: no logs (0), initialization and completion logs (1), setup and iteration results (2), or detailed evaluation results (3).

Default 2

Range 0–3

maxBayesianModelSize=*integer*

specifies the maximum number of points in the kriging model. This parameter is ignored unless the value "BAYESIAN" is specified in the searchMethod parameter.

Alias maxBayesianSize

Default 100

Minimum value 10

maxEvals=*integer*

specifies the maximum number of objective evaluations (model configurations) to be trained and scored.

Alias maxEvaluations

Default 50

Minimum value 1

maxIters=*integer*

specifies the maximum number of iterations of local search optimization for model tuning. At each iteration, the number of evaluated models is less than or equal to the value of the popSize parameter.

Alias maxIterations

Default 5

Minimum value 1

maxTime=*double*

specifies the maximum time allowed for tuning. The actual time might exceed this value by the amount of time required to train and score a model configuration.

Default 36000

Minimum value 1

maxTrainTime=*64-bit-integer*

specifies the maximum time allowed for a single model training. The model training is terminated if it exceeds this time, and the objective value is set to missing.

Minimum value 0

nCrossValFolds=*integer*

specifies the number of folds to use for cross validation to assess model fit error as the tuning objective.

Alias nFolds

Default 5

Minimum value 2

nParallel=*integer*

specifies the number of threads to be used by the optimizer for parallel evaluation of model configurations.

Default 0

nSubsessionWorkers=*integer*

specifies the number of workers to be used by each subsession for parallel evaluation of model configurations.

Alias nSubWorkers

Default 0

Minimum value 0

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU"

specifies the objective metric to use during the tuning.

ASE

specifies the average square error as the objective function.

AUC

specifies area under the curve as the objective function (nominal targets only).

F05

specifies the F0.5 coefficient as the objective function (nominal targets only).

F1

specifies the F1 coefficient as the objective function (nominal targets only).

GAMMA

specifies the gamma coefficient as the objective function (nominal targets only).

GINI

specifies the Gini coefficient as the objective function (nominal targets only).

KS

specifies the Kolmogorov-Smirnov coefficient as the objective function (nominal targets only).

MAE

specifies the mean absolute error as the objective function (interval targets only).

MCE

specifies the misclassification rate as the objective function (nominal targets only).

MCLL

specifies the multiclass log loss as the objective function (nominal targets only).

MISC

specifies the misclassification error percentage as the objective function (nominal targets only).

MSE

specifies the mean square error as the objective function (interval targets only).

MSLE

specifies the mean square logarithmic error as the objective function (interval

targets only).

RASE

specifies the root average square error as the objective function.

RMAE

specifies the root mean absolute error as the objective function (interval targets only).

RMSLE

specifies the root mean square logarithmic error as the objective function (interval targets only).

TAU

specifies the tau coefficient as the objective function (nominal targets only).

popSize=*integer*

specifies the maximum number of model configurations to evaluate at each iteration.

Alias populationSize

Default 10

Minimum value 2

sampleSize=*integer*

specifies the number of model configurations to evaluate when the value of the searchMethod parameter is "LHS" or "RANDOM".

Default 50

Minimum value 2

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM"

specifies the search method to use during tuning.

Default GA

BAYESIAN

uses the Bayesian search method.

GA

uses the genetic algorithm search method.

GRID

uses the grid search method.

LHS

uses the Latin hypercube sample search method.

RANDOM

uses the random search method.

seed=*integer*

specifies the seed to use for selecting training data and validation data for a single validation partition or fold number for each observation for cross validation.

Default 0

shuffleGridPoints=TRUE | FALSE

when set to True, shuffles the points that are generated by the grid search method before execution.

Default TRUE

targetEvent="*string*"

specifies the name of the nominal target event to use for tuning.

trainPartitionFraction=*double*

specifies the size of a single data partition to use for model training during tuning.

Alias trainFraction

Range 0.01–0.99

userDefinedPartition=TRUE | FALSE

when set to True, includes a user-defined partition for training and scoring.

Default FALSE

validationPartitionFraction=*double*

specifies the size of a single validation partition to use in order to assess model fit error as the tuning objective.

Alias validateFraction

Default 0.3

Range 0.01–0.99

tuningParameters={{*autotuneTuningParmDefinition-1*} <, {*autotuneTuningParmDefinition-2*}, ...>}

specifies a list of custom tuning parameters and their definitions.

The autotuneTuningParmDefinition value can be one or more of the following:

exclude=TRUE | FALSE

when set to True, excludes a tuning parameter from the tuning process.

Default FALSE

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*"

specifies the initial (default) value of a tuning parameter.

lowerBound=*double*

specifies the lower bound of the range for a tuning parameter. This parameter can be

used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias LB

namePath="*string*"

specifies the name path of a tuning parameter. For a nested action parameter, this parameter specifies a dot-separated path that includes all its parent parameter names. For a top-level action parameter, this parameter is simply the name of the parameter.

Alias name

upperBound=*double*

specifies the upper bound of the range for a tuning parameter. This parameter can be used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias UB

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

specifies the list of values to try for a tuning parameter. If this parameter is specified, the lower bound and upper bound values of the tuning parameter are ignored.

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

specifies whether to use only standard tuning parameters, only custom tuning parameters, or a combination of both.

Default COMBINED

Autotune Action Set: Syntax

tuneDecisionTree Action

CASL Syntax

autotune.tuneDecisionTree <**result**=*results*> <**status**=*rc*> /

pruneOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

tunerOptions={

evaluationHistory=TRUE | FALSE,

logLevel=*integer*,

maxBayesianModelSize=*integer*,

maxEvals=*integer*,

maxIters=*integer*,

maxTime=*double*,

maxTrainTime=*64-bit-integer*,

nCrossValFolds=*integer*,

nParallel=*integer*,

nSubsessionWorkers=*integer*,

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU",

popSize=*integer*,

sampleSize=*integer*,

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM",

seed=*integer*,

shuffleGridPoints=TRUE | FALSE,

targetEvent="*string*",

trainPartitionFraction=*double*,

userDefinedPartition=TRUE | FALSE,

validationPartitionFraction=*double*

}

tuningParameters={{

exclude=TRUE | FALSE,

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*",

lowerBound=*double*,

namePath="*string*",

upperBound=*double*,

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

}, {...}}

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

;

Parameter Descriptions

pruneOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the dtreePrune action in the decisionTree action set; for more information, see the parameters for that action.

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the dtreeScore action in the decisionTree action set; for more information, see the parameters for that action.

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the dtreeTrain action in the decisionTree action set; for more information, see the parameters for that action.

Alias dtopts

tunerOptions={*tunerOptions*}

specifies a list of parameters for adjusting how the tuner behaves.

Alias optMinerOpts

evaluationHistory=TRUE | FALSE

when set to True, creates a table of every evaluation (every model configuration) and returns those tables to the client.

Default TRUE

logLevel=*integer*

specifies the level of log messages to be written: no logs (0), initialization and completion logs (1), setup and iteration results (2), or detailed evaluation results (3).

Default 2

Range 0–3

maxBayesianModelSize=*integer*

specifies the maximum number of points in the kriging model. This parameter is ignored unless the value "BAYESIAN" is specified in the searchMethod parameter.

Alias maxBayesianSize

Default 100

Minimum value 10

maxEvals=*integer*

specifies the maximum number of objective evaluations (model configurations) to be trained and scored.

Alias maxEvaluations

Default 50

Minimum value 1

maxIters=*integer*

specifies the maximum number of iterations of local search optimization for model tuning. At each iteration, the number of evaluated models is less than or equal to the value of the popSize parameter.

Alias maxIterations

Default 5

Minimum value 1

maxTime=*double*

specifies the maximum time allowed for tuning. The actual time might exceed this value by the amount of time required to train and score a model configuration.

Default 36000

Minimum value 1

maxTrainTime=*64-bit-integer*

specifies the maximum time allowed for a single model training. The model training is terminated if it exceeds this time, and the objective value is set to missing.

Minimum value 0

nCrossValFolds=*integer*

specifies the number of folds to use for cross validation to assess model fit error as the tuning objective.

Alias nFolds

Default 5

Minimum value 2

nParallel=*integer*

specifies the number of threads to be used by the optimizer for parallel evaluation of model configurations.

Default 0

nSubsessionWorkers=*integer*

specifies the number of workers to be used by each subsession for parallel evaluation of model configurations.

Alias nSubWorkers

Default 0

Minimum value 0

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU"

specifies the objective metric to use during the tuning.

ASE

specifies the average square error as the objective function.

AUC

specifies area under the curve as the objective function (nominal targets only).

F05

specifies the F0.5 coefficient as the objective function (nominal targets only).

F1

specifies the F1 coefficient as the objective function (nominal targets only).

GAMMA

specifies the gamma coefficient as the objective function (nominal targets only).

GINI

specifies the Gini coefficient as the objective function (nominal targets only).

KS

specifies the Kolmogorov-Smirnov coefficient as the objective function (nominal targets only).

MAE

specifies the mean absolute error as the objective function (interval targets only).

MCE

specifies the misclassification rate as the objective function (nominal targets only).

MCLL

specifies the multiclass log loss as the objective function (nominal targets only).

MISC

specifies the misclassification error percentage as the objective function (nominal targets only).

MSE

specifies the mean square error as the objective function (interval targets only).

MSLE

specifies the mean square logarithmic error as the objective function (interval targets only).

RASE

specifies the root average square error as the objective function.

RMAE

specifies the root mean absolute error as the objective function (interval targets only).

RMSLE

specifies the root mean square logarithmic error as the objective function (interval targets only).

TAU

specifies the tau coefficient as the objective function (nominal targets only).

popSize=*integer*

specifies the maximum number of model configurations to evaluate at each iteration.

Alias populationSize

Default 10

Minimum value 2

sampleSize=*integer*

specifies the number of model configurations to evaluate when the value of the searchMethod parameter is "LHS" or "RANDOM".

Default 50

Minimum value 2

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM"

specifies the search method to use during tuning.

Default GA

BAYESIAN

uses the Bayesian search method.

GA

uses the genetic algorithm search method.

GRID

uses the grid search method.

LHS

uses the Latin hypercube sample search method.

RANDOM

uses the random search method.

seed=*integer*

specifies the seed to use for selecting training data and validation data for a single validation partition or fold number for each observation for cross validation.

Default 0

shuffleGridPoints=TRUE | FALSE

when set to True, shuffles the points that are generated by the grid search method before execution.

Default TRUE

targetEvent="*string*"

specifies the name of the nominal target event to use for tuning.

trainPartitionFraction=*double*

specifies the size of a single data partition to use for model training during tuning.

Alias trainFraction

Range 0.01–0.99

userDefinedPartition=TRUE | FALSE

when set to True, includes a user-defined partition for training and scoring.

Default FALSE

validationPartitionFraction=*double*

specifies the size of a single validation partition to use in order to assess model fit error as the tuning objective.

Alias validateFraction

Default 0.3

Range 0.01–0.99

tuningParameters={{*autotuneTuningParmDefinition-1*} <, {*autotuneTuningParmDefinition-2*}, ...>}

specifies a list of custom tuning parameters and their definitions.

The autotuneTuningParmDefinition value can be one or more of the following:

exclude=TRUE | FALSE

when set to True, excludes a tuning parameter from the tuning process.

Default FALSE

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*"

specifies the initial (default) value of a tuning parameter.

lowerBound=*double*

specifies the lower bound of the range for a tuning parameter. This parameter can be used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias LB

namePath="*string*"

specifies the name path of a tuning parameter. For a nested action parameter, this parameter specifies a dot-separated path that includes all its parent parameter names. For a top-level action parameter, this parameter is simply the name of the parameter.

Alias name

upperBound=*double*

specifies the upper bound of the range for a tuning parameter. This parameter can be used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias UB

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

specifies the list of values to try for a tuning parameter. If this parameter is specified, the lower bound and upper bound values of the tuning parameter are ignored.

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

specifies whether to use only standard tuning parameters, only custom tuning parameters, or a combination of both.

Default COMBINED

Autotune Action Set: Syntax

tuneFactMac Action

CASL Syntax

autotune.tuneFactMac <**result**=*results*> <**status**=*rc*> /

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

tunerOptions={

evaluationHistory=TRUE | FALSE,

logLevel=*integer*,

maxBayesianModelSize=*integer*,

maxEvals=*integer*,

maxIters=*integer*,

maxTime=*double*,

maxTrainTime=*64-bit-integer*,

nCrossValFolds=*integer*,

nParallel=*integer*,

nSubsessionWorkers=*integer*,

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU",

popSize=*integer*,

sampleSize=*integer*,

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM",

seed=*integer*,

shuffleGridPoints=TRUE | FALSE,

targetEvent="*string*",

trainPartitionFraction=*double*,

userDefinedPartition=TRUE | FALSE,

validationPartitionFraction=*double*

}

tuningParameters={{

exclude=TRUE | FALSE,

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*",

lowerBound=*double*,

namePath="*string*",

upperBound=*double*,

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

}, {...}}

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

;

Parameter Descriptions

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the score action in the aStore action set; for more information, see the parameters for that action.

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the factmac action in the factmac action set; for more information, see the parameters for that action.

tunerOptions={*tunerOptions*}

specifies a list of parameters for adjusting how the tuner behaves.

Alias optMinerOpts

evaluationHistory=TRUE | FALSE

when set to True, creates a table of every evaluation (every model configuration) and returns those tables to the client.

Default TRUE

logLevel=*integer*

specifies the level of log messages to be written: no logs (0), initialization and completion logs (1), setup and iteration results (2), or detailed evaluation results (3).

Default 2

Range 0–3

maxBayesianModelSize=*integer*

specifies the maximum number of points in the kriging model. This parameter is ignored unless the value "BAYESIAN" is specified in the searchMethod parameter.

Alias maxBayesianSize

Default 100

Minimum value 10

maxEvals=*integer*

specifies the maximum number of objective evaluations (model configurations) to be trained and scored.

Alias maxEvaluations

Default 50

Minimum value 1

maxIters=*integer*

specifies the maximum number of iterations of local search optimization for model tuning. At each iteration, the number of evaluated models is less than or equal to the value of the popSize parameter.

Alias maxIterations

Default 5

Minimum value 1

maxTime=*double*

specifies the maximum time allowed for tuning. The actual time might exceed this value by the amount of time required to train and score a model configuration.

Default 36000

Minimum value 1

maxTrainTime=*64-bit-integer*

specifies the maximum time allowed for a single model training. The model training is terminated if it exceeds this time, and the objective value is set to missing.

Minimum value 0

nCrossValFolds=*integer*

specifies the number of folds to use for cross validation to assess model fit error as the tuning objective.

Alias nFolds

Default 5

Minimum value 2

nParallel=*integer*

specifies the number of threads to be used by the optimizer for parallel evaluation of model configurations.

Default 0

nSubsessionWorkers=*integer*

specifies the number of workers to be used by each subsession for parallel evaluation of model configurations.

Alias nSubWorkers

Default 0

Minimum value 0

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU"

specifies the objective metric to use during the tuning.

ASE

specifies the average square error as the objective function.

AUC

specifies area under the curve as the objective function (nominal targets only).

F05

specifies the F0.5 coefficient as the objective function (nominal targets only).

F1

specifies the F1 coefficient as the objective function (nominal targets only).

GAMMA

specifies the gamma coefficient as the objective function (nominal targets only).

GINI

specifies the Gini coefficient as the objective function (nominal targets only).

KS

specifies the Kolmogorov-Smirnov coefficient as the objective function (nominal targets only).

MAE

specifies the mean absolute error as the objective function (interval targets only).

MCE

specifies the misclassification rate as the objective function (nominal targets only).

MCLL

specifies the multiclass log loss as the objective function (nominal targets only).

MISC

specifies the misclassification error percentage as the objective function (nominal targets only).

MSE

specifies the mean square error as the objective function (interval targets only).

MSLE

specifies the mean square logarithmic error as the objective function (interval

targets only).

RASE

specifies the root average square error as the objective function.

RMAE

specifies the root mean absolute error as the objective function (interval targets only).

RMSLE

specifies the root mean square logarithmic error as the objective function (interval targets only).

TAU

specifies the tau coefficient as the objective function (nominal targets only).

popSize=*integer*

specifies the maximum number of model configurations to evaluate at each iteration.

Alias populationSize

Default 10

Minimum value 2

sampleSize=*integer*

specifies the number of model configurations to evaluate when the value of the searchMethod parameter is "LHS" or "RANDOM".

Default 50

Minimum value 2

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM"

specifies the search method to use during tuning.

Default GA

BAYESIAN

uses the Bayesian search method.

GA

uses the genetic algorithm search method.

GRID

uses the grid search method.

LHS

uses the Latin hypercube sample search method.

RANDOM

uses the random search method.

seed=*integer*

specifies the seed to use for selecting training data and validation data for a single validation partition or fold number for each observation for cross validation.

Default 0

shuffleGridPoints=TRUE | FALSE

when set to True, shuffles the points that are generated by the grid search method before execution.

Default TRUE

targetEvent="*string*"

specifies the name of the nominal target event to use for tuning.

trainPartitionFraction=*double*

specifies the size of a single data partition to use for model training during tuning.

Alias trainFraction

Range 0.01–0.99

userDefinedPartition=TRUE | FALSE

when set to True, includes a user-defined partition for training and scoring.

Default FALSE

validationPartitionFraction=*double*

specifies the size of a single validation partition to use in order to assess model fit error as the tuning objective.

Alias validateFraction

Default 0.3

Range 0.01–0.99

tuningParameters={{*autotuneTuningParmDefinition-1*} <, {*autotuneTuningParmDefinition-2*}, ...>}

specifies a list of custom tuning parameters and their definitions.

The autotuneTuningParmDefinition value can be one or more of the following:

exclude=TRUE | FALSE

when set to True, excludes a tuning parameter from the tuning process.

Default FALSE

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*"

specifies the initial (default) value of a tuning parameter.

lowerBound=*double*

specifies the lower bound of the range for a tuning parameter. This parameter can be

used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias LB

namePath="*string*"

specifies the name path of a tuning parameter. For a nested action parameter, this parameter specifies a dot-separated path that includes all its parent parameter names. For a top-level action parameter, this parameter is simply the name of the parameter.

Alias name

upperBound=*double*

specifies the upper bound of the range for a tuning parameter. This parameter can be used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias UB

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

specifies the list of values to try for a tuning parameter. If this parameter is specified, the lower bound and upper bound values of the tuning parameter are ignored.

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

specifies whether to use only standard tuning parameters, only custom tuning parameters, or a combination of both.

Default COMBINED

Autotune Action Set: Syntax

tuneForest Action

CASL Syntax

autotune.tuneForest <**result**=*results*> <**status**=*rc*> /

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

tunerOptions={

evaluationHistory=TRUE | FALSE,

logLevel=*integer*,

maxBayesianModelSize=*integer*,

maxEvals=*integer*,

maxIters=*integer*,

maxTime=*double*,

maxTrainTime=*64-bit-integer*,

nCrossValFolds=*integer*,

nParallel=*integer*,

nSubsessionWorkers=*integer*,

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU",

popSize=*integer*,

sampleSize=*integer*,

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM",

seed=*integer*,

shuffleGridPoints=TRUE | FALSE,

targetEvent="*string*",

trainPartitionFraction=*double*,

userDefinedPartition=TRUE | FALSE,

validationPartitionFraction=*double*

}

tuningParameters={{

exclude=TRUE | FALSE,

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*",

lowerBound=*double*,

namePath="*string*",

upperBound=*double*,

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

}, {...}}

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

;

Parameter Descriptions

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the forestScore action in the decisionTree action set; for more information, see the parameters for that action.

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the forestTrain action in the decisionTree action set; for more information, see the parameters for that action.

tunerOptions={*tunerOptions*}

specifies a list of parameters for adjusting how the tuner behaves.

Alias optMinerOpts

evaluationHistory=TRUE | FALSE

when set to True, creates a table of every evaluation (every model configuration) and returns those tables to the client.

Default TRUE

logLevel=*integer*

specifies the level of log messages to be written: no logs (0), initialization and completion logs (1), setup and iteration results (2), or detailed evaluation results (3).

Default 2

Range 0–3

maxBayesianModelSize=*integer*

specifies the maximum number of points in the kriging model. This parameter is ignored unless the value "BAYESIAN" is specified in the searchMethod parameter.

Alias maxBayesianSize

Default 100

Minimum value 10

maxEvals=*integer*

specifies the maximum number of objective evaluations (model configurations) to be trained and scored.

Alias maxEvaluations

Default 50

Minimum value 1

maxIters=*integer*

specifies the maximum number of iterations of local search optimization for model tuning. At each iteration, the number of evaluated models is less than or equal to the value of the popSize parameter.

Alias maxIterations

Default 5

Minimum value 1

maxTime=*double*

specifies the maximum time allowed for tuning. The actual time might exceed this value by the amount of time required to train and score a model configuration.

Default 36000

Minimum value 1

maxTrainTime=*64-bit-integer*

specifies the maximum time allowed for a single model training. The model training is terminated if it exceeds this time, and the objective value is set to missing.

Minimum value 0

nCrossValFolds=*integer*

specifies the number of folds to use for cross validation to assess model fit error as the tuning objective.

Alias nFolds

Default 5

Minimum value 2

nParallel=*integer*

specifies the number of threads to be used by the optimizer for parallel evaluation of model configurations.

Default 0

nSubsessionWorkers=*integer*

specifies the number of workers to be used by each subsession for parallel evaluation of model configurations.

Alias nSubWorkers

Default 0

Minimum value 0

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU"

specifies the objective metric to use during the tuning.

ASE

specifies the average square error as the objective function.

AUC

specifies area under the curve as the objective function (nominal targets only).

F05

specifies the F0.5 coefficient as the objective function (nominal targets only).

F1

specifies the F1 coefficient as the objective function (nominal targets only).

GAMMA

specifies the gamma coefficient as the objective function (nominal targets only).

GINI

specifies the Gini coefficient as the objective function (nominal targets only).

KS

specifies the Kolmogorov-Smirnov coefficient as the objective function (nominal targets only).

MAE

specifies the mean absolute error as the objective function (interval targets only).

MCE

specifies the misclassification rate as the objective function (nominal targets only).

MCLL

specifies the multiclass log loss as the objective function (nominal targets only).

MISC

specifies the misclassification error percentage as the objective function (nominal targets only).

MSE

specifies the mean square error as the objective function (interval targets only).

MSLE

specifies the mean square logarithmic error as the objective function (interval

targets only).

RASE

specifies the root average square error as the objective function.

RMAE

specifies the root mean absolute error as the objective function (interval targets only).

RMSLE

specifies the root mean square logarithmic error as the objective function (interval targets only).

TAU

specifies the tau coefficient as the objective function (nominal targets only).

popSize=*integer*

specifies the maximum number of model configurations to evaluate at each iteration.

Alias populationSize

Default 10

Minimum value 2

sampleSize=*integer*

specifies the number of model configurations to evaluate when the value of the searchMethod parameter is "LHS" or "RANDOM".

Default 50

Minimum value 2

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM"

specifies the search method to use during tuning.

Default GA

BAYESIAN

uses the Bayesian search method.

GA

uses the genetic algorithm search method.

GRID

uses the grid search method.

LHS

uses the Latin hypercube sample search method.

RANDOM

uses the random search method.

seed=*integer*

specifies the seed to use for selecting training data and validation data for a single validation partition or fold number for each observation for cross validation.

Default 0

shuffleGridPoints=TRUE | FALSE

when set to True, shuffles the points that are generated by the grid search method before execution.

Default TRUE

targetEvent="*string*"

specifies the name of the nominal target event to use for tuning.

trainPartitionFraction=*double*

specifies the size of a single data partition to use for model training during tuning.

Alias trainFraction

Range 0.01–0.99

userDefinedPartition=TRUE | FALSE

when set to True, includes a user-defined partition for training and scoring.

Default FALSE

validationPartitionFraction=*double*

specifies the size of a single validation partition to use in order to assess model fit error as the tuning objective.

Alias validateFraction

Default 0.3

Range 0.01–0.99

tuningParameters={{*autotuneTuningParmDefinition-1*} <, {*autotuneTuningParmDefinition-2*}, ...>}

specifies a list of custom tuning parameters and their definitions.

The autotuneTuningParmDefinition value can be one or more of the following:

exclude=TRUE | FALSE

when set to True, excludes a tuning parameter from the tuning process.

Default FALSE

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*"

specifies the initial (default) value of a tuning parameter.

lowerBound=*double*

specifies the lower bound of the range for a tuning parameter. This parameter can be

used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias LB

namePath="*string*"

specifies the name path of a tuning parameter. For a nested action parameter, this parameter specifies a dot-separated path that includes all its parent parameter names. For a top-level action parameter, this parameter is simply the name of the parameter.

Alias name

upperBound=*double*

specifies the upper bound of the range for a tuning parameter. This parameter can be used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias UB

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

specifies the list of values to try for a tuning parameter. If this parameter is specified, the lower bound and upper bound values of the tuning parameter are ignored.

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

specifies whether to use only standard tuning parameters, only custom tuning parameters, or a combination of both.

Default COMBINED

Autotune Action Set: Syntax

tuneGradientBoostTree Action

CASL Syntax

autotune.tuneGradientBoostTree <**result**=*results*> <**status**=*rc*> /

earlyStop=TRUE | FALSE

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

tunerOptions={

evaluationHistory=TRUE | FALSE,

logLevel=*integer*,

maxBayesianModelSize=*integer*,

maxEvals=*integer*,

maxIters=*integer*,

maxTime=*double*,

maxTrainTime=*64-bit-integer*,

nCrossValFolds=*integer*,

nParallel=*integer*,

nSubsessionWorkers=*integer*,

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU",

popSize=*integer*,

sampleSize=*integer*,

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM",

seed=*integer*,

shuffleGridPoints=TRUE | FALSE,

targetEvent="*string*",

trainPartitionFraction=*double*,

userDefinedPartition=TRUE | FALSE,

validationPartitionFraction=*double*

}

tuningParameters={{

exclude=TRUE | FALSE,

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*",

lowerBound=*double*,

namePath="*string*",

upperBound=*double*,

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

}, {...}}

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

;

Parameter Descriptions

earlyStop=TRUE | FALSE

when set to True, validation is performed during model training and training is terminated when stagnated.

Default TRUE

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the gbtreeScore action in the decisionTree action set; for more information, see the parameters for that action.

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the gbtreeTrain action in the decisionTree action set; for more information, see the parameters for that action.

tunerOptions={*tunerOptions*}

specifies a list of parameters for adjusting how the tuner behaves.

Alias optMinerOpts

evaluationHistory=TRUE | FALSE

when set to True, creates a table of every evaluation (every model configuration) and returns those tables to the client.

Default TRUE

logLevel=*integer*

specifies the level of log messages to be written: no logs (0), initialization and completion logs (1), setup and iteration results (2), or detailed evaluation results (3).

Default 2

Range 0–3

maxBayesianModelSize=*integer*

specifies the maximum number of points in the kriging model. This parameter is ignored unless the value "BAYESIAN" is specified in the searchMethod parameter.

Alias maxBayesianSize

Default 100

Minimum value 10

maxEvals=*integer*

specifies the maximum number of objective evaluations (model configurations) to be trained and scored.

Alias maxEvaluations

Default 50

Minimum value 1

maxIters=*integer*

specifies the maximum number of iterations of local search optimization for model tuning. At each iteration, the number of evaluated models is less than or equal to the value of the popSize parameter.

Alias maxIterations

Default 5

Minimum value 1

maxTime=*double*

specifies the maximum time allowed for tuning. The actual time might exceed this value by the amount of time required to train and score a model configuration.

Default 36000

Minimum value 1

maxTrainTime=*64-bit-integer*

specifies the maximum time allowed for a single model training. The model training is terminated if it exceeds this time, and the objective value is set to missing.

Minimum value 0

nCrossValFolds=*integer*

specifies the number of folds to use for cross validation to assess model fit error as the tuning objective.

Alias nFolds

Default 5

Minimum value 2

nParallel=*integer*

specifies the number of threads to be used by the optimizer for parallel evaluation of model configurations.

Default 0

nSubsessionWorkers=*integer*

specifies the number of workers to be used by each subsession for parallel evaluation of model configurations.

Alias nSubWorkers

Default 0

Minimum value 0

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU"

specifies the objective metric to use during the tuning.

ASE

specifies the average square error as the objective function.

AUC

specifies area under the curve as the objective function (nominal targets only).

F05

specifies the F0.5 coefficient as the objective function (nominal targets only).

F1

specifies the F1 coefficient as the objective function (nominal targets only).

GAMMA

specifies the gamma coefficient as the objective function (nominal targets only).

GINI

specifies the Gini coefficient as the objective function (nominal targets only).

KS

specifies the Kolmogorov-Smirnov coefficient as the objective function (nominal targets only).

MAE

specifies the mean absolute error as the objective function (interval targets only).

MCE

specifies the misclassification rate as the objective function (nominal targets only).

MCLL

specifies the multiclass log loss as the objective function (nominal targets only).

MISC

specifies the misclassification error percentage as the objective function (nominal targets only).

MSE

specifies the mean square error as the objective function (interval targets only).

MSLE

specifies the mean square logarithmic error as the objective function (interval targets only).

RASE

specifies the root average square error as the objective function.

RMAE

specifies the root mean absolute error as the objective function (interval targets only).

RMSLE

specifies the root mean square logarithmic error as the objective function (interval targets only).

TAU

specifies the tau coefficient as the objective function (nominal targets only).

popSize=*integer*

specifies the maximum number of model configurations to evaluate at each iteration.

Alias populationSize

Default 10

Minimum value 2

sampleSize=*integer*

specifies the number of model configurations to evaluate when the value of the searchMethod parameter is "LHS" or "RANDOM".

Default 50

Minimum value 2

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM"

specifies the search method to use during tuning.

Default GA

BAYESIAN

uses the Bayesian search method.

GA

uses the genetic algorithm search method.

GRID

uses the grid search method.

LHS

uses the Latin hypercube sample search method.

RANDOM

uses the random search method.

seed=*integer*

specifies the seed to use for selecting training data and validation data for a single validation partition or fold number for each observation for cross validation.

Default 0

shuffleGridPoints=TRUE | FALSE

when set to True, shuffles the points that are generated by the grid search method before execution.

Default TRUE

targetEvent="*string*"

specifies the name of the nominal target event to use for tuning.

trainPartitionFraction=*double*

specifies the size of a single data partition to use for model training during tuning.

Alias trainFraction

Range 0.01–0.99

userDefinedPartition=TRUE | FALSE

when set to True, includes a user-defined partition for training and scoring.

Default FALSE

validationPartitionFraction=*double*

specifies the size of a single validation partition to use in order to assess model fit error as the tuning objective.

Alias validateFraction

Default 0.3

Range 0.01–0.99

tuningParameters={{*autotuneTuningParmDefinition-1*} <, {*autotuneTuningParmDefinition-2*}, ...>}

specifies a list of custom tuning parameters and their definitions.

The autotuneTuningParmDefinition value can be one or more of the following:

exclude=TRUE | FALSE

when set to True, excludes a tuning parameter from the tuning process.

Default FALSE

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*"

specifies the initial (default) value of a tuning parameter.

lowerBound=*double*

specifies the lower bound of the range for a tuning parameter. This parameter can be used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias LB

namePath="*string*"

specifies the name path of a tuning parameter. For a nested action parameter, this parameter specifies a dot-separated path that includes all its parent parameter names. For a top-level action parameter, this parameter is simply the name of the parameter.

Alias name

upperBound=*double*

specifies the upper bound of the range for a tuning parameter. This parameter can be used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias UB

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

specifies the list of values to try for a tuning parameter. If this parameter is specified, the lower bound and upper bound values of the tuning parameter are ignored.

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

specifies whether to use only standard tuning parameters, only custom tuning parameters, or a combination of both.

Default COMBINED

Autotune Action Set: Syntax

tuneNeuralNet Action

CASL Syntax

autotune.tuneNeuralNet <**result**=*results*> <**status**=*rc*> /

earlyStop=TRUE | FALSE

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

tunerOptions={

evaluationHistory=TRUE | FALSE,

logLevel=*integer*,

maxBayesianModelSize=*integer*,

maxEvals=*integer*,

maxIters=*integer*,

maxTime=*double*,

maxTrainTime=*64-bit-integer*,

nCrossValFolds=*integer*,

nParallel=*integer*,

nSubsessionWorkers=*integer*,

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU",

popSize=*integer*,

sampleSize=*integer*,

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM",

seed=*integer*,

shuffleGridPoints=TRUE | FALSE,

targetEvent="*string*",

trainPartitionFraction=*double*,

userDefinedPartition=TRUE | FALSE,

validationPartitionFraction=*double*

}

tuningParameters={{

exclude=TRUE | FALSE,

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*",

lowerBound=*double*,

namePath="*string*",

upperBound=*double*,

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

}, {...}}

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

;

Parameter Descriptions

earlyStop=TRUE | FALSE

when set to True, validation is performed during model training and training is terminated when stagnated.

Default TRUE

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the annScore action in the neuralNet action set; for more information, see the parameters for that action.

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the annTrain action in the neuralNet action set; for more information, see the parameters for that action.

Alias nnopts

tunerOptions={*tunerOptions*}

specifies a list of parameters for adjusting how the tuner behaves.

Alias optMinerOpts

evaluationHistory=TRUE | FALSE

when set to True, creates a table of every evaluation (every model configuration) and returns those tables to the client.

Default TRUE

logLevel=*integer*

specifies the level of log messages to be written: no logs (0), initialization and completion logs (1), setup and iteration results (2), or detailed evaluation results (3).

Default 2

Range 0–3

maxBayesianModelSize=*integer*

specifies the maximum number of points in the kriging model. This parameter is ignored unless the value "BAYESIAN" is specified in the searchMethod parameter.

Alias maxBayesianSize

Default 100

Minimum value 10

maxEvals=*integer*

specifies the maximum number of objective evaluations (model configurations) to be trained and scored.

Alias maxEvaluations

Default 50

Minimum value 1

maxIters=*integer*

specifies the maximum number of iterations of local search optimization for model tuning. At each iteration, the number of evaluated models is less than or equal to the value of the popSize parameter.

Alias maxIterations

Default 5

Minimum value 1

maxTime=*double*

specifies the maximum time allowed for tuning. The actual time might exceed this value by the amount of time required to train and score a model configuration.

Default 36000

Minimum value 1

maxTrainTime=*64-bit-integer*

specifies the maximum time allowed for a single model training. The model training is terminated if it exceeds this time, and the objective value is set to missing.

Minimum value 0

nCrossValFolds=*integer*

specifies the number of folds to use for cross validation to assess model fit error as the tuning objective.

Alias nFolds

Default 5

Minimum value 2

nParallel=*integer*

specifies the number of threads to be used by the optimizer for parallel evaluation of model configurations.

Default 0

nSubsessionWorkers=*integer*

specifies the number of workers to be used by each subsession for parallel evaluation of model configurations.

Alias nSubWorkers

Default 0

Minimum value 0

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU"

specifies the objective metric to use during the tuning.

ASE

specifies the average square error as the objective function.

AUC

specifies area under the curve as the objective function (nominal targets only).

F05

specifies the F0.5 coefficient as the objective function (nominal targets only).

F1

specifies the F1 coefficient as the objective function (nominal targets only).

GAMMA

specifies the gamma coefficient as the objective function (nominal targets only).

GINI

specifies the Gini coefficient as the objective function (nominal targets only).

KS

specifies the Kolmogorov-Smirnov coefficient as the objective function (nominal targets only).

MAE

specifies the mean absolute error as the objective function (interval targets only).

MCE

specifies the misclassification rate as the objective function (nominal targets only).

MCLL

specifies the multiclass log loss as the objective function (nominal targets only).

MISC

specifies the misclassification error percentage as the objective function (nominal targets only).

MSE

specifies the mean square error as the objective function (interval targets only).

MSLE

specifies the mean square logarithmic error as the objective function (interval targets only).

RASE

specifies the root average square error as the objective function.

RMAE

specifies the root mean absolute error as the objective function (interval targets only).

RMSLE

specifies the root mean square logarithmic error as the objective function (interval targets only).

TAU

specifies the tau coefficient as the objective function (nominal targets only).

popSize=*integer*

specifies the maximum number of model configurations to evaluate at each iteration.

Alias populationSize

Default 10

Minimum value 2

sampleSize=*integer*

specifies the number of model configurations to evaluate when the value of the searchMethod parameter is "LHS" or "RANDOM".

Default 50

Minimum value 2

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM"

specifies the search method to use during tuning.

Default GA

BAYESIAN

uses the Bayesian search method.

GA

uses the genetic algorithm search method.

GRID

uses the grid search method.

LHS

uses the Latin hypercube sample search method.

RANDOM

uses the random search method.

seed=*integer*

specifies the seed to use for selecting training data and validation data for a single validation partition or fold number for each observation for cross validation.

Default 0

shuffleGridPoints=TRUE | FALSE

when set to True, shuffles the points that are generated by the grid search method before execution.

Default TRUE

targetEvent="*string*"

specifies the name of the nominal target event to use for tuning.

trainPartitionFraction=*double*

specifies the size of a single data partition to use for model training during tuning.

Alias trainFraction

Range 0.01–0.99

userDefinedPartition=TRUE | FALSE

when set to True, includes a user-defined partition for training and scoring.

Default FALSE

validationPartitionFraction=*double*

specifies the size of a single validation partition to use in order to assess model fit error as the tuning objective.

Alias validateFraction

Default 0.3

Range 0.01–0.99

tuningParameters={{*autotuneTuningParmDefinition-1*} <, {*autotuneTuningParmDefinition-2*}, ...>}

specifies a list of custom tuning parameters and their definitions.

The autotuneTuningParmDefinition value can be one or more of the following:

exclude=TRUE | FALSE

when set to True, excludes a tuning parameter from the tuning process.

Default FALSE

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*"

specifies the initial (default) value of a tuning parameter.

lowerBound=*double*

specifies the lower bound of the range for a tuning parameter. This parameter can be used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias LB

namePath="*string*"

specifies the name path of a tuning parameter. For a nested action parameter, this parameter specifies a dot-separated path that includes all its parent parameter names. For a top-level action parameter, this parameter is simply the name of the parameter.

Alias name

upperBound=*double*

specifies the upper bound of the range for a tuning parameter. This parameter can be used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias UB

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

specifies the list of values to try for a tuning parameter. If this parameter is specified, the lower bound and upper bound values of the tuning parameter are ignored.

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

specifies whether to use only standard tuning parameters, only custom tuning parameters, or a combination of both.

Default COMBINED

Autotune Action Set: Syntax

tuneSvm Action

CASL Syntax

autotune.tuneSvm <**result**=*results*> <**status**=*rc*> /

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

tunerOptions={

evaluationHistory=TRUE | FALSE,

logLevel=*integer*,

maxBayesianModelSize=*integer*,

maxEvals=*integer*,

maxIters=*integer*,

maxTime=*double*,

maxTrainTime=*64-bit-integer*,

nCrossValFolds=*integer*,

nParallel=*integer*,

nSubsessionWorkers=*integer*,

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU",

popSize=*integer*,

sampleSize=*integer*,

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM",

seed=*integer*,

shuffleGridPoints=TRUE | FALSE,

targetEvent="*string*",

trainPartitionFraction=*double*,

userDefinedPartition=TRUE | FALSE,

validationPartitionFraction=*double*

}

tuningParameters={{

exclude=TRUE | FALSE,

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*",

lowerBound=*double*,

namePath="*string*",

upperBound=*double*,

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

}, {...}}

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

;

Parameter Descriptions

scoreOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used for support vector machine scoring.

trainOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters to be used by the svmTrain action in the svm action set; for more information, see the parameters for that action.

Alias svmopts

tunerOptions={*tunerOptions*}

specifies a list of parameters for adjusting how the tuner behaves.

Alias optMinerOpts

evaluationHistory=TRUE | FALSE

when set to True, creates a table of every evaluation (every model configuration) and returns those tables to the client.

Default TRUE

logLevel=*integer*

specifies the level of log messages to be written: no logs (0), initialization and completion logs (1), setup and iteration results (2), or detailed evaluation results (3).

Default 2

Range 0–3

maxBayesianModelSize=*integer*

specifies the maximum number of points in the kriging model. This parameter is ignored unless the value "BAYESIAN" is specified in the searchMethod parameter.

Alias maxBayesianSize

Default 100

Minimum value 10

maxEvals=*integer*

specifies the maximum number of objective evaluations (model configurations) to be trained and scored.

Alias maxEvaluations

Default 50

Minimum value 1

maxIters=*integer*

specifies the maximum number of iterations of local search optimization for model tuning. At each iteration, the number of evaluated models is less than or equal to the value of the popSize parameter.

Alias maxIterations

Default 5

Minimum value 1

maxTime=*double*

specifies the maximum time allowed for tuning. The actual time might exceed this value by the amount of time required to train and score a model configuration.

Default 36000

Minimum value 1

maxTrainTime=*64-bit-integer*

specifies the maximum time allowed for a single model training. The model training is terminated if it exceeds this time, and the objective value is set to missing.

Minimum value 0

nCrossValFolds=*integer*

specifies the number of folds to use for cross validation to assess model fit error as the tuning objective.

Alias nFolds

Default 5

Minimum value 2

nParallel=*integer*

specifies the number of threads to be used by the optimizer for parallel evaluation of model configurations.

Default 0

nSubsessionWorkers=*integer*

specifies the number of workers to be used by each subsession for parallel evaluation of model configurations.

Alias nSubWorkers

Default 0

Minimum value 0

objective="ASE" | "AUC" | "F05" | "F1" | "GAMMA" | "GINI" | "KS" | "MAE" | "MCE" | "MCLL" | "MISC" | "MSE" | "MSLE" | "RASE" | "RMAE" | "RMSLE" | "TAU"

specifies the objective metric to use during the tuning.

ASE

specifies the average square error as the objective function.

AUC

specifies area under the curve as the objective function (nominal targets only).

F05

specifies the F0.5 coefficient as the objective function (nominal targets only).

F1

specifies the F1 coefficient as the objective function (nominal targets only).

GAMMA

specifies the gamma coefficient as the objective function (nominal targets only).

GINI

specifies the Gini coefficient as the objective function (nominal targets only).

KS

specifies the Kolmogorov-Smirnov coefficient as the objective function (nominal targets only).

MAE

specifies the mean absolute error as the objective function (interval targets only).

MCE

specifies the misclassification rate as the objective function (nominal targets only).

MCLL

specifies the multiclass log loss as the objective function (nominal targets only).

MISC

specifies the misclassification error percentage as the objective function (nominal targets only).

MSE

specifies the mean square error as the objective function (interval targets only).

MSLE

specifies the mean square logarithmic error as the objective function (interval targets only).

RASE

specifies the root average square error as the objective function.

RMAE

specifies the root mean absolute error as the objective function (interval targets only).

RMSLE

specifies the root mean square logarithmic error as the objective function (interval targets only).

TAU

specifies the tau coefficient as the objective function (nominal targets only).

popSize=*integer*

specifies the maximum number of model configurations to evaluate at each iteration.

Alias populationSize

Default 10

Minimum value 2

sampleSize=*integer*

specifies the number of model configurations to evaluate when the value of the searchMethod parameter is "LHS" or "RANDOM".

Default 50

Minimum value 2

searchMethod="BAYESIAN" | "GA" | "GRID" | "LHS" | "RANDOM"

specifies the search method to use during tuning.

Default GA

BAYESIAN

uses the Bayesian search method.

GA

uses the genetic algorithm search method.

GRID

uses the grid search method.

LHS

uses the Latin hypercube sample search method.

RANDOM

uses the random search method.

seed=*integer*

specifies the seed to use for selecting training data and validation data for a single validation partition or fold number for each observation for cross validation.

Default 0

shuffleGridPoints=TRUE | FALSE

when set to True, shuffles the points that are generated by the grid search method before execution.

Default TRUE

targetEvent="*string*"

specifies the name of the nominal target event to use for tuning.

trainPartitionFraction=*double*

specifies the size of a single data partition to use for model training during tuning.

Alias trainFraction

Range 0.01–0.99

userDefinedPartition=TRUE | FALSE

when set to True, includes a user-defined partition for training and scoring.

Default FALSE

validationPartitionFraction=*double*

specifies the size of a single validation partition to use in order to assess model fit error as the tuning objective.

Alias validateFraction

Default 0.3

Range 0.01–0.99

tuningParameters={{*autotuneTuningParmDefinition-1*} <, {*autotuneTuningParmDefinition-2*}, ...>}

specifies a list of custom tuning parameters and their definitions.

The autotuneTuningParmDefinition value can be one or more of the following:

exclude=TRUE | FALSE

when set to True, excludes a tuning parameter from the tuning process.

Default FALSE

initValue=*integer* | *64-bit-integer* | *double* | TRUE | FALSE | "*string*"

specifies the initial (default) value of a tuning parameter.

lowerBound=*double*

specifies the lower bound of the range for a tuning parameter. This parameter can be used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias LB

namePath="*string*"

specifies the name path of a tuning parameter. For a nested action parameter, this parameter specifies a dot-separated path that includes all its parent parameter names. For a top-level action parameter, this parameter is simply the name of the parameter.

Alias name

upperBound=*double*

specifies the upper bound of the range for a tuning parameter. This parameter can be used only for numeric tuning parameters, and is ignored if a value list is specified for the tuning parameter.

Alias UB

valueList={*any-list-or-data-type-1* <, *any-list-or-data-type-2*, ...>}

specifies the list of values to try for a tuning parameter. If this parameter is specified, the lower bound and upper bound values of the tuning parameter are ignored.

useParameters="COMBINED" | "CUSTOM" | "STANDARD"

specifies whether to use only standard tuning parameters, only custom tuning parameters, or a combination of both.

Default COMBINED

Details

Hyperparameter Tuning

The **autotune** action set includes a number of actions for tuning predictive models that are produced by machine learning algorithms. The quality of the predictive model that a machine learning algorithm creates depends on the values for various options that govern the training process; these options are called *hyperparameters*. The default values of these hyperparameters might not be suitable for all applications. In order to reduce the manual effort in adjusting these hyperparameters, you can use an action in the **autotune** action set to identify the best settings for them.

To tune hyperparameters, all actions in the **autotune** action set engage a search method (tuner) that searches for the best possible combination of values of these hyperparameters while trying to minimize or maximize an objective function. The objective function is a validation error or accuracy estimate—for example, MISC (misclassification error) for nominal targets or ASE (average square error) for interval targets. The tuning process involves multiple evaluations of the objective function (one for each model hyperparameter configuration it evaluates) and might include multiple iterations, depending on the specified search method. The tuning process also requires data partitioning in order to avoid overfitting to training data; models are trained on a training partition and validated on a validation partition to obtain the objective function value for each chosen hyperparameter configuration. The options for data partitioning are described in the next section, followed by descriptions of the default hyperparameter tuning process and determination of sizing for parallel tuning.

Data Partitioning

Each evaluation of the objective function can consist of one or several training and scoring (validation) executions, depending on whether single-partition validation or *k*-fold cross validation is used. In both cases, the tuner partitions the data unless the **userDefinedPartition** parameter is set to True and the **scoreOptions** parameter is specified (in this case, the tuner does not partition the training data, but instead uses the data table specified in the **trainOptions** parameter for training and uses the data table specified in the **scoreOptions** parameter for validation).

Single-Partition Validation

By default, the tuner uses single-partition validation. In this process, the tuner partitions all the data into two subsets: one subset for model training and one subset for model validation. For each newly generated configuration of hyperparameters, a new model is trained on the training subset and then the validation subset is scored using the trained model to find the resulting objective function value.

The **sampling** action set is used to create the training and validation partitions. The **stratified** action is used for a target of nominal type (if all target levels can be included in both the training and validation partitions), and the **srs** action is used for a target of interval type and for cases in which stratified sampling is not possible. By default, a validation partition of 30% is used and the remaining 70% is used for model training. The validation partition size can be adjusted by specifying the **validationPartitionFraction** parameter. For more information about the **sampling** action set, see Chapter 16, "Sampling and Partitioning Action Set" (*SAS Visual Statistics: Programming Guide*).

For large data tables, tuning efficiency can be increased by subsampling the remaining data for training. For example, 30% of the data can be used for model training and 30% for validation, leaving 40% unused. The training partition can be specified using the `trainPartitionFraction` parameter. Stratified sampling ensures that the model training and validation partitions are representative of the full data table.

Cross Validation

If *k*-fold cross validation is specified, the tuner partitions all the data into *k* subsets (folds). For each fold, a new model is trained on all folds except the selected (holdout) fold (that is, it is trained on *k*–1 folds) and then validated using the selected (holdout) fold. The objective function value is averaged over the set of fold validation executions to obtain a single error estimate value. Cross validation is specified using the `nCrossValFolds` parameter.

For efficiency, the cross validation process might be terminated before all *k* folds are evaluated. Cross validation is terminated under the following conditions: the validation score is 2 times worse than the current best score after the first fold, the validation score is 1.75 times worse than the current best score after two folds, or the validation score is 1.5 times worse than the current best score after three or more folds.

Hyperparameter Tuning Process

By default, the tuner chooses the model hyperparameter configurations to evaluate by using a hybrid optimization strategy that is based on a genetic algorithm (GA), which applies principles of natural selection and evolution to find an improved configuration. The tuner performs the following steps:

1. A default model configuration (default values of select model hyperparameters) is evaluated first and designated as Iteration 0. The objective function value is obtained by using either single-partition validation or *k*-fold cross validation and then recorded for comparison.
2. An initial set of hyperparameter configurations, also called a "population," is generated using a technique called Latin hypercube sampling (LHS). Each configuration of hyperparameters in the Latin hypercube sample is evaluated, and the objective function value for each is again recorded for comparison. This set becomes Iteration 1. All or a portion of the sample configurations can be evaluated in parallel, as described in the section "Determining the Number of Parallel Evaluations" on page 117.
3. The best model configurations from the Iteration 1 are used to generate the next population of model configurations, Iteration 2, which are then evaluated, in parallel when possible (see the section "Determining the Number of Parallel Evaluations" on page 117). This process is repeated for the remaining iterations, as long as the maximum number of evaluations or the maximum time has not been reached. The default number of iterations is 5, and the default maximum time is 36,000 seconds (10 hours).
4. The best model configuration (the one that minimizes or maximizes the objective function) is reevaluated by executing a single model training and scoring, and information about the model training and scoring for this configuration is returned.
5. All evaluated model configurations are ranked, and the hyperparameter and objective function values of the top 10 configurations are returned in the TunerResults table.

You can specify alternate search methods: Bayesian, random, LHS, and grid. The preceding steps are performed differently, depending on the search method:

- For the Bayesian search method, the preceding steps are performed in the same sequence, with the following changes. Instead of using genetic operations for generating the new combinations of hyperparameter values, an internal surrogate model is created after the first iteration. This surrogate model is then explored for potential candidate configurations, and the new set of configurations is created and evaluated in parallel. After each iteration, the internal surrogate model is updated with the new data and explored again for new potential configurations.
- For the random, LHS, or grid search methods, step 3 in the preceding sequence is eliminated; a single sample of candidate configurations is generated and evaluated at step 2. This single sample of configurations is generated in one of the following ways:
  - randomly for the random search method
  - using a Latin hypercube sample for the LHS search method
  - by creating all possible combinations of hyperparameter levels for the grid search method. If the **`valueList`** attribute is specified for any hyperparameter, the levels are taken from that list. Otherwise, the tuner takes the lower bound, the upper bound, and the middle value between the two bounds, and uses these values as the grid levels for each hyperparameter. When evaluating the grid configurations, the tuner shuffles them randomly, unless the value of the **`shuffleGridPoints`** parameter is False. The total number of grid configurations is the product of the numbers of levels for each hyperparameter. If the number of hyperparameters and the number of levels is greater than a very small value, the total number of combinations can be many hundreds or even thousands. If the **`maxEvaluations`** parameter is specified, it is used by the tuner to terminate execution of the grid search method before all grid points are evaluated.

If the tuner had partitioned the data for training and scoring (if the **`scoreOptions`** parameter is not specified), then the final (best) model configuration is trained on all the data and scored on all the data; as a result the final model that is returned and its reported validation score might not match the value that is observed during tuning, when the model was trained to a portion of the data and validated on the remaining validation partition.

As an option for increased efficiency in tuning, you can specify the **`maxTrainTime`** parameter to limit the maximum time for model training. Complex long-running models are terminated when the specified time limit has been reached. The greatest gain in tuning efficiency, though, is observed through parallel evaluation of the hyperparameter configurations in steps 2 and 3, if sufficient resources exist for parallel tuning.

Determining the Number of Parallel Evaluations

By default, the number of possible parallel evaluations during step 2 and step 3 (if performed) in the section "Hyperparameter Tuning Process" on page 116 is determined by the population size for GA or Bayesian search methods or by the sample size for random, LHS, or grid search methods. The maximum number of parallel evaluations is capped by the total number of worker nodes that are connected to the server divided by the number of worker nodes that are used by the parallel sessions that are created. For example, if the server is configured with 100 worker nodes and four workers are used in the sessions, at most 25 parallel evaluations are performed by default. This limit can be overridden by a factor of 2, resulting in up to 50 parallel evaluations in this example scenario.

Specifically, the number of parallel evaluations is determined as follows:

1. If you do not specify the number of workers to use in parallel subsessions by using the **nSubsessionWorkers** parameter, the number of workers is determined using the size of the data table: **nSubsessionWorkers** = 1 + *nDataRows* * *nDataColumns* * 2E–8. If determined based on the data table size, the number of workers in each subsession is fairly aggressively set at one node per 50 million values. The efficiency benefits of tuning in parallel generally outweigh the time savings from distributed training if the data can be managed on fewer nodes. For example, if training a single model on four workers takes 1 minute versus 2 minutes on one worker, then training four models in parallel on one worker each requires only 2 minutes, whereas training four sequential models each on four workers would require 4 minutes.
2. The number of possible parallel evaluations is then determined as follows:
   - one less than the population size for the GA search method
   - population size for the Bayesian search method
   - sampling size for the random or LHS search methods
   - the grid size for the grid search method
3. The number of parallel evaluations is then limited by the server configuration:
   - In single-machine mode, if the number of possible parallel evaluations is greater than 4 and not specified in the **nParallel** parameter, it is limited to 4. This limit can be overridden up to a value of 32 by specifying the **nParallel** parameter (the value of the **nParallel** parameter is reduced if it is greater than 32).
   - In distributed mode, the upper limit for the number of parallel evaluations is calculated as *W/n*, where *W* is the number of workers connected to the server and *n* is the number of workers in the parallel subsessions. This limit can be overridden by a factor of 2 by specifying the **nParallel** parameter; the new limit is *2W/n* (the value of the **nParallel** parameter is reduced if it is greater than *2W/n*).

The tuneBnet Action

The **tuneBnet** action selects different hyperparameter configurations to run the **bnet** action of the **bayesianNetClassifier** action set, the **score** action of the **astore** action set, and the **assess** action of the **percentile** action set to train and validate Bayesian network classifier models as it searches for a model that has reduced validation error. For more information about the **bnet** and **score** actions, see *SAS Visual Data Mining and Machine Learning: Programming Guide*. For more information about the **assess** action, see *SAS Visual Analytics: Programming Guide*.

The **tuneBnet** action searches for the best combination of values of the following hyperparameters:

- **alpha**, the significance level for independence tests that use chi-square or G-square statistics, where the value of **alpha** is a real number whose value is equal to or greater than 0.0 and less than or equal to 1.0.
- **maxParents**, the maximum number of parents allowed for each node in the network, where the value of **maxParents** is an integer whose value is equal to or greater than 1 and less than or equal to 16.

- **`parenting`**, the structure learning method, where the value of **`parenting`** can be one of two strings: BESTONE or BESTSET.
- **`preScreening`**, the initial screening for the input variables, where the value of **`preScreening`** can be one of two strings: ZERO or ONE.
- **`structure`**, the type of network structure, where the value of **`structure`** can be one of four strings: NAIVE, TAN, PC, or MB.
- **`varSelect`**, the selection for the input variables, where the value of **`varSelect`** can be one of four strings: ZERO, ONE, TWO, or THREE.
- **`numBin`**, the binning number for interval variables, where the value of **`numBin`** is an integer whose value is equal to or greater than 2 and less than or equal to 1,024.
- **`miAlpha`**, the significance level for independence tests that use mutual information, where the value of **`miAlpha`** is a real number whose value is equal to or greater than 0.0 and less than or equal to 1.0.
- **`indepTest`**, the method for independence tests, where the value of **`indepTest`** can be one of four strings: CHISQUARE, GSQUARE, MI, or CHIGSQUARE.
- **`missingInt`**, missing values policy for interval variables, where the value of **`missingInt`** can be one of two strings: IGNORE or IMPUTE. By default, the tuner does not vary the **`missingInt`** hyperparameter but rather keeps its value fixed as IGNORE. You can enable this hyperparameter by specifying it in the **`tuningParameters`** parameter.
- **`missingNom`**, missing values policy for nominal variables, where the value of **`missingNom`** can be one of three strings: IGNORE, IMPUTE, or LEVEL. By default, the tuner does not vary the **`missingNom`** hyperparameter but rather keeps its value fixed as IGNORE. You can enable this hyperparameter by specifying it in the **`tuningParameters`** parameter.

Table 4.1 shows the default initial value and the default tuning range or values list for each of these tuning parameters. You can change the initial value and the tuning range or values list for a hyperparameter to any value or values allowed by the parameter by specifying the **`initValue`**, **`lowerBound`** and **`upperBound`**, or **`valueList`** subparameters, respectively, in the **`tuningParameters`** parameter. For numeric hyperparameters listed in Table 4.1, as an alternative to specifying lower and upper bound values, you can specify a value list by using the **`valueList`** subparameter.

Table 4.1 Default Tuning Parameters of the **`tuneBnet`** Action

Hyperparameter	Initial Value	Lower Bound	Upper Bound	Value List
`alpha`	0.05	0.01	0.99	
`maxParents`	5	1	16	
`parenting`	BESTSET			BESTONE, BESTSET
`preScreening`	ONE			ZERO, ONE
`structure`	PC			NAIVE, TAN, PC, MB
`varSelect`	ONE			ZERO, ONE, TWO, THREE
`numBin`	5	2	20	
`miAlpha`	0.05	0.0	1.0	
`indepTest`	CHIGSQUARE			CHISQUARE, GSQUARE, MI, CHIGSQUARE

The tuneDecisionTree Action

The **tuneDecisionTree** action selects different hyperparameter configurations to run the **dtreeTrain**, **dtreePrune** (optionally), and **dtreeScore** actions multiple times to train and validate decision tree models as it searches for a model that has reduced validation error. For more information about the **dtreeTrain**, **dtreePrune**, and **dtreeScore** actions in the **decisionTree** action set, see *SAS Visual Analytics: Programming Guide*.

The **tuneDecisionTree** action searches for the best combination of values of the following hyperparameters:

- **maxLevel**, the maximum number of decision tree levels, where **maxLevel** is a positive nonzero integer.
- **nBins**, the number of bins to use for numeric variables during calculation of the decision tree, where **nBins** in a positive nonzero integer. If all inputs are nominal, binning is not necessary and so **nBins** is not tuned.
- **crit**, the split criterion for each tree node. See Table 4.2 for the list of all valid **crit** values.

Table 4.2 shows the default initial value and the default tuning range or values list for each of these tuning parameters. You can change the initial, lower bound, and upper bound values, or the value list, for any hyperparameter to any value or values allowed by the parameter by specifying the **initValue**, **lowerBound**, **upperBound**, and **valueList** subparameters, respectively, in the **tuningParameters** parameter.

Table 4.2 Default Tuning Parameters of the **tuneDecisionTree** Action

Hyperparameter	Initial Value	Lower Bound	Upper Bound
maxLevel	11	2	20
nBins	20	20	200
crit	GAINRATIO (nominal target) VARIANCE (interval target)	**Value List** GAIN, GAINRATIO, GINI, CHISQUARE, CHAID (nominal target) VARIANCE, FTEST, CHAID (interval target)	

The tuneFactMac Action

The **tuneFactMac** action selects different hyperparameter configurations to run the **factmac** action of the **factmac** action set, the **score** action of the **astore** action set, and the **assess** action of the **percentile** action set to train and validate factorization machine models as it searches for a model that has reduced validation error. For more information about the **factmac** and **score** actions, see *SAS Visual Data Mining and Machine Learning: Programming Guide*. For more information about the **assess** action, see *SAS Visual Analytics: Programming Guide*.

The **tuneFactMac** action searches for the best combination of values of the following hyperparameters:

- **nFactors**, the number of factors to be estimated, where **nFactors** is a positive nonzero integer.
- **maxIter**, the maximum number of iterations for the internal training optimization, where **maxIter** is a positive nonzero integer.
- **learnStep**, the learning step size for the internal training optimization, where **learnStep** is a positive nonzero real number.

Table 4.3 shows the default initial value and the default tuning range (expressed as a list of values) for each of these tuning parameters. You can change the initial value or value list for any hyperparameter to any value or values allowed by the parameter by specifying the **initValue** and **valueList** subparameters, respectively, in the **tuningParameters** parameter. As an alternative to specifying a value list, you can specify a tuning range by using the **lowerBound** and **upperBound** subparameters.

Table 4.3 Default Tuning Parameters of the **tuneFactMac** Action

Hyperparameter	Initial Value	Value List
nFactors	5	5, 10, 15, 20, 25, 30
maxIter	30	10, 20, 30, . . . , 200
learnStep	1E–3	1E–6, 1E–5, 1E–4, 1E–3, 1E–2, 1E–1, 1.0

The tuneForest Action

The **tuneForest** action selects different hyperparameter configurations to run the **forestTrain** and **forestScore** actions multiple times to train and validate forest models as it searches for a model that has reduced validation error. For more information about the **forestTrain** and **forestScore** actions in the **decisionTree** action set, see *SAS Visual Analytics: Programming Guide*.

The **tuneForest** action searches for the best combination of values of the following hyperparameters:

- **nTree**, the number of trees to create, where **nTree** is a positive nonzero integer.
- **m**, the number of input variables to consider for splitting on a node, where **m** is a positive nonzero integer.
- **bootstrap**, the fraction of the data for the bootstrap sample, where **bootstrap** is a real number whose value is greater than 0.0 and less than or equal to 1.0.
- **maxLevel**, the maximum number of the tree level, where **maxLevel** is a positive nonzero integer.

Table 4.4 shows the default initial value and the default tuning range for each of these tuning parameters. You can change the initial, lower bound, and upper bound values for any hyperparameter to any value or values allowed by the parameter by specifying the **initValue**, **lowerBound**, and **upperBound** subparameters, respectively, in the **tuningParameters** parameter. As an alternative to specifying lower bound and upper bound values, you can specify a value list by using the **valueList** subparameter.

Table 4.4 Default Tuning Parameters of the `tuneForest` Action

Hyperparameter	Initial Value	Lower Bound	Upper Bound
`nTree`	100	20	150
`m`	sqrt(*nInputs*)	1	min(*nInputs*, 100)
`bootstrap`	0.6	0.1	0.9
`maxLevel`	21	2	30

nInputs is the number of inputs to the model.

The tuneGradientBoostTree Action

The `tuneGradientBoostTree` action selects different hyperparameter configurations to run the `gbtreeTrain` and `gbtreeScore` actions multiple times to train and validate gradient boosting tree models as it searches for a model that has reduced validation error. For more information about the `gbtreeTrain` and `gbtreeScore` actions in the `decisionTree` action set, see *SAS Visual Analytics: Programming Guide*.

The `tuneGradientBoostTree` action searches for the best combination of values of the following hyperparameters:

- `nTree`, the number of trees to create, where `nTree` is a positive nonzero integer.
- `m`, the number of input variables to consider splitting on in a node, where `m` is a positive nonzero integer.
- `learningRate`, the learning rate of each tree, where `learningRate` is a real number whose value is greater than 0.0 and less than or equal to 1.0.
- `subSampleRate`, the fraction of the data to use for building each tree, where `subSampleRate` is a real number whose value is greater than 0.0 and less than or equal to 1.0.
- `lasso`, the L1 norm regularization on prediction, where `lasso` is a positive real number including 0.0.
- `ridge`, the L2 norm regularization on prediction, where `ridge` is a positive real number including 0.0.

Table 4.5 shows the default initial value and the default tuning range for each of these tuning parameters. You can change the initial, lower bound, and upper bound values for any hyperparameter to any value or values allowed by the parameter by specifying the `initValue`, `lowerBound`, and `upperBound` subparameters, respectively, in the `tuningParameters` parameter. As an alternative to specifying lower and upper bound values, you can specify a value list by using the `valueList` subparameter.

Table 4.5 Default Tuning Parameters of the `tuneGradientBoostTree` Action

Hyperparameter	Initial Value	Lower Bound	Upper Bound
`nTree`	100	20	150
`m`	Number of inputs	1	Number of inputs
`learningRate`	0.1	0.01	1.0
`subSampleRate`	0.5	0.1	1.0
`lasso`	0.0	0.0	10.0
`ridge`	0.0	0.0	10.0

By default, the `tuneGradientBoostTree` action activates early stopping for more efficient tuning. Early stopping stops the training action at the current iteration if no improvement (compared to the current error) is achieved by the next *n* iterations (*n* is specified in the `stagnation` parameter, which is set to 4 for autotuning). As a result, the actual final number of trees used in the reported top model can be less than the "best" value that the autotuning action selects. To disable early stopping and allow all models to train to completion, specify a value of False for the `earlyStop` parameter.

The tuneNeuralNet Action

The `tuneNeuralNet` action selects different hyperparameter configurations to run the `annTrain` and `annScore` actions multiple times to train and validate neural network models as it searches for a model that has reduced validation error. For more information about the `annTrain` and `annScore` actions in the `neuralNet` action set, see *SAS Visual Data Mining and Machine Learning: Programming Guide*.

The `tuneNeuralNet` action searches for the best combination of values of the following hyperparameters:

- `hiddens`, specifies both the number of hidden layers and the number of hidden neurons for each hidden layer in the feed-forward model. For example, a value of {5,3} for the `hiddens` parameter specifies two hidden layers: one with 5 hidden neurons and the other with 3 hidden neurons. This parameter in the `annTrain` action specifies multiple levels of information about the model to be constructed. To capture these multiple levels in the tuner, this single parameter results in up to six tuning parameters. The first tuning parameter represents the number of layers (`nHidden`), and you can specify up to five other tuning parameters (`nUnits1-5`) to specify the number of hidden neurons in each layer. It is not always necessary for the tuner to use all the additional five tuning parameters. For example, if the tuner determines that the best configuration uses three hidden layers, then it does not use the values of `nUnits4` and `nUnits5` tuning parameters.

  `nHidden` is a positive integer including 0.

  `nUnits1-5` are positive nonzero integers.

- `nloOpts.optmlOpt.regL1`, the L1 regularization parameter. `nloOpts.optmlOpt.regL1` is a positive real number including 0.0.

- `nloOpts.optmlOpt.regL2`, the L2 regularization parameter. `nloOpts.optmlOpt.regL2` is a positive real number including 0.0.

- **`nloOpts.sgdOpt.learningRate`**, the learning rate parameter for the internal optimization for neural net training. This parameter applies only when the stochastic gradient descent (SGD) is used for training. **`nloOpts.sgdOpt.learningRate`** is a positive real number including 0.0.

- **`nloOpts.sgdOpt.annealingRate`**, the annealing rate parameter for the internal optimization for neural net training. This parameter applies only when stochastic gradient descent (SGD) is used for training. **`nloOpts.sgdOpt.annealingRate`** is a positive real number including 0.0.

Table 4.6 shows the default initial value and the default tuning range for each of these tuning parameters. You can change the initial, lower bound, and upper bound values for any hyperparameter to any value or values allowed by the parameter by specifying the **`initValue`**, **`lowerBound`**, and **`upperBound`** subparameters, respectively, in the **`tuningParameters`** parameter. As an alternative to specifying lower and upper bound values, you can specify a value list by using the **`valueList`** subparameter.

Table 4.6 Default Tuning Parameters of the **`tuneNeuralNet`** Action

Hyperparameter	Initial Value	Lower Bound	Upper Bound
`nHidden`	0	0	2
`nUnits1`	1	1	min(3**nInputs*, 100)
`nUnits2`	1	1	min(3**nInputs*, 100)
`nUnits3`[1]	1	1	min(3**nInputs*, 100)
`nUnits4`[1]	1	1	min(3**nInputs*, 100)
`nUnits5`[1]	1	1	min(3**nInputs*, 100)
`regL1`	0	0	10.0
`regL2`	0	0	10.0
`learningRate`[2]	1E–3	1E–6	1E–1
`annealingRate`[2]	1E–6	1E–13	1E–2

For the default upper bound of the **`nUnits1-nUnits5`** tuning parameters, *nInputs* is the number of inputs to the model.

If the number of internal neural network training iterations (specified in the **`trainOptions`** parameter) is 25 or greater (the default is 10 iterations), the **`tuneNeuralNet`** action activates early stopping for more efficient tuning, with the **`stagnation`** parameter set to 4 and the **`frequency`** parameter set to 1 for autotuning. The **`stagnation`** parameter specifies the number of consecutive validations with increasing error rates that are allowed before early termination of the model training optimization process, and the **`frequency`** parameter specifies how frequently (in epochs) validation occurs during model training. To disable early stopping and allow all models to train to completion, specify a value of False for the **`earlyStop`** parameter.

[1] **`nUnits3-nUnits5`** are tuned only if the upper bound for **`nHidden`** is increased.
[2] **`learningRate`** and **`annealingRate`** are tuned only if stochastic gradient descent (SGD) is used for training.

The tuneSvm Action

The **tuneSvm** action selects different hyperparameter configurations to run the **svmTrain** action of the **svm** action set, the **score** action of the **astore** action set, and the **assess** action of the **percentile** action set to train and validate support vector machine models as it searches for a model that has reduced validation error. For more information about the **svmTrain** and **score** actions, see *SAS Visual Data Mining and Machine Learning: Programming Guide*. For more information about the **assess** action, see *SAS Visual Analytics: Programming Guide*.

The **tuneSvm** action searches for the best combination of values of the following hyperparameters:

- **C**, the penalty parameter for SVM training, where **C** is a positive nonzero real number.
- **degree**, the degree of the polynomial kernel, where **degree** is an integer whose value is equal to or greater than 1 and less than or equal to 3.

Table 4.7 shows the default initial value and the default tuning range for each of these tuning parameters. You can change the initial, lower bound, and upper bound values for any hyperparameter to any value or values allowed by the parameter by specifying the **initValue**, **lowerBound**, and **upperBound** subparameters, respectively, in the **tuningParameters** parameter. As an alternative to specifying lower and upper bound values, you can specify a value list by using the **valueList** subparameter.

Table 4.7 Default Tuning Parameters of the **tuneSvm** Action

Hyperparameter	Initial Value	Lower Bound	Upper Bound
C	1.0	1E–10	100.0
degree	1	1	3

Examples

Example 4.1: Tuning the Hyperparameters of a Bayesian Network Classifier

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the **tuneBnet** action to automatically tune the hyperparameters of a Bayesian network classifier model from observations in the German credit benchmark data, which are available in the sampsio.dmagecr data set. This data set contains 1,000 observations, each of which contains an applicant's information, including the applicant's credit rating ("GOOD" or "BAD"). The binary target is named GOOD_BAD. Other input variables are Checking, Duration, History, and so on.

The contents of the sampsio.dmagecr data set are described at http://support.sas.com/documentation/cdl/en/emgs/59885/HTML/default/a001026918.htm.

You can load the sampsio.dmagecr data set into your CAS session by specifying your CAS engine libref in the following DATA step. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.dmagecr;
   set sampsio.dmagecr;
   seed=12345;
   if ranuni(seed) < 0.7 then partind=1;
   else                       partind=0;
run;
```

The following PROC CAS code uses the **tuneBnet** action to automatically tune the hyperparameters of a Bayesian network classifier model that is trained on the dmagecr data table (note that the syntax of the **trainOptions** parameter here is the same as the syntax of the **bnet** action):

```
proc cas noqueue;
    autotune.tuneBnet /
        trainOptions     = {
            table        = "dmagecr"
            structure    = {"NAIVE" "TAN" "PC" "MB"}
            outnetwork   = {name="network" replace=1}
            nominals     = {
                "checking" "history"  "purpose" "savings"  "employed"
                "installp" "marital"  "coapp"   "resident" "property"
                "other"    "housing"  "existcr" "job"
                "depends"  "telephon" "foreign" "good_bad"
            }
            inputs       = {
                "duration" "amount" "age"
```

```
                "depends"  "checking" "history"  "purpose" "savings"
                "employed" "installp" "marital"  "coapp"   "resident"
                "property" "other"    "job"
                "existcr"  "housing"  "telephon" "foreign"
            }
            target       = "good_bad"
            partbyvar    = {name="partind", train="1", valid="0"}
        }
        tunerOptions     = {logLevel=3}
    ;
    ods output TunerResults      = TuneResults(keep=MisclassErr);
    ods output EvaluationHistory = EvalHistory;
    ods output IterationHistory  = IterHistory;
    run;
quit;
```

The following optional code includes two DATA steps and two SGPLOT procedure steps that generate the following two plots from the **tuneBnet** results:

- history of all evaluations (the objective value versus the evaluation number, grouped by iteration), as shown in Output 4.1.9
- iteration history (the best objective value and total tuning time versus the iteration number), as shown in Output 4.1.10

```
data _null_;
   set TuneResults;
   if _n_ = 1 then call symput('default', MisclassErr);
   else if _n_ = 2 then call symput('best', MisclassErr);
run;

data EvalHistory;
   set EvalHistory;
   default = &default.;
   best = &best.;
run;

proc sgplot data=EvalHistory;
   title "Tune Bnet with GA on DMAGECR data";
   scatter x=Evaluation y=MisclassErr / group=Iteration;
   series x=Evaluation y=default /
      curvelabel="default" lineattrs=(pattern=shortdash);
   series x=Evaluation y=best /
      curvelabel="best" lineattrs=(color=green pattern=shortdash);
run;

proc sgplot data=IterHistory;
   title "Tune Bnet with GA on DMAGECR data";
   series x=Iteration y=Best_Obj;
   series x=Iteration y=Time_Sec / y2axis;
run;
```

Output 4.1.1 shows the "Tuner Information" output table, which displays the values of parameters that the tuner used.

Output 4.1.1 Tuner Information Output

Results from bayesianNetClassifier.bnet

Tuner Information	
Model Type	Bayesian Network Classifier
Tuner Objective Function	Misclassification Error Percentage
Search Method	GA
Maximum Evaluations	50
Population Size	10
Maximum Iterations	5
Maximum Tuning Time in Seconds	36000
Validation Type	Validation Type
Log Level	3
Seed	458893390
Number of Parallel Evaluations	9
Number of Workers per Subsession	1

Output 4.1.2 shows the "Tuner Results" output table, which displays the default configuration (Iteration 0) and up to 10 of the best configurations that the tuner found. For each configuration, the hyperparameter values and objective value are provided.

Output 4.1.2 Tuner Results Output

Tuner Results
Default and Best Configurations

Evaluation	Alpha	Maximum Number of Parents	Parenting	Prescreening	Variable Selection	Number of Bins	Mutual Information Alpha	Independence Test	Misclassification Error Percentage
0	0.050000	5	BESTSET	ONE	ONE	5	0.050000	CHIGSQUARE	23.93
33	0.912000	7	BESTSET	ZERO	ZERO	13	0.582894	CHIGSQUARE	22.62
42	0.871882	7	BESTSET	ZERO	ZERO	13	0.540431	CHIGSQUARE	22.62
43	0.959758	7	BESTSET	ZERO	ZERO	13	0.612419	CHIGSQUARE	22.62
10	0.050000	5	BESTSET	ONE	ONE	5	0.050000	CHIGSQUARE	23.93
18	0.010000	9	BESTONE	ZERO	ZERO	15	0.187116	GSQUARE	23.93
28	0.067586	5	BESTSET	ONE	ONE	5	0.064496	CHIGSQUARE	23.93
31	0.052938	5	BESTSET	ONE	ONE	5	0.052422	CHIGSQUARE	23.93
39	0.052781	5	BESTSET	ONE	ONE	5	0.052231	CHIGSQUARE	23.93
44	0.097758	5	BESTSET	ONE	ONE	5	0.079525	CHIGSQUARE	23.93
1	0.118889	8	BESTONE	ZERO	ZERO	14	0.222222	GSQUARE	24.26

Tuner Results Default and Best Configurations Time In Seconds
1.01
0.62
0.60
0.65
0.00
0.47
0.44
0.46
0.41
0.36
0.33

Output 4.1.3 shows the "Tuner Iteration History" output table, which displays the results of each iteration. For each iteration, the number of evaluations, the best objective value so far, and the tuning time in seconds are provided.

Output 4.1.3 Iteration History Output

Tuner Iteration History

Iteration	Evaluations	Best Objective	Time In Seconds
0	1	23.93	1.01
1	10	23.93	3.46
2	19	23.93	4.47
3	28	23.93	5.16
4	37	22.62	5.78
5	46	22.62	6.43

Output 4.1.4 shows the "Tuner Evaluation History" output table, which displays all model configurations that the tuner evaluated. For each configuration, the hyperparameter values and objective value are provided.

Output 4.1.4 Tuner Evaluation History Output

Tuner History
All Evaluated Configurations

Evaluation	Iteration	Alpha	Maximum Number of Parents	Parenting	Prescreening	Variable Selection	Number of Bins	Mutual Information Alpha	Independence Test
0	0	0.050000	5	BESTSET	ONE	ONE	5	0.050000	CHIGSQUARE
1	1	0.118889	8	BESTONE	ZERO	ZERO	14	0.222222	GSQUARE
2	1	0.554444	16	BESTSET	ONE	TWO	4	0.111111	CHISQUARE
3	1	0.010000	13	BESTONE	ONE	ONE	2	0.666667	CHIGSQUARE
4	1	0.772222	11	BESTSET	ZERO	ZERO	20	0.777778	MI
5	1	0.227778	1	BESTONE	ZERO	THREE	6	0.888889	CHISQUARE
6	1	0.445556	14	BESTONE	ZERO	ZERO	16	1.000000	CHIGSQUARE
7	1	0.663333	9	BESTSET	ONE	THREE	8	0.555556	GSQUARE
8	1	0.336667	6	BESTSET	ONE	THREE	12	0.333333	CHIGSQUARE
9	1	0.990000	4	BESTONE	ZERO	TWO	10	0.444444	MI
10	1	0.050000	5	BESTSET	ONE	ONE	5	0.050000	CHIGSQUARE
11	2	0.010000	5	BESTSET	ONE	ONE	5	0.012989	MI
12	2	0.618432	9	BESTSET	ONE	THREE	8	0.518544	GSQUARE
13	2	0.515858	15	BESTSET	ONE	TWO	4	0.106437	CHISQUARE
14	2	0.010000	1	BESTSET	ONE	ZERO	6	0.009036	GSQUARE
15	2	0.216308	9	BESTSET	ONE	ONE	5	0.070147	MI
16	2	0.010000	5	BESTONE	ZERO	ZERO	13	0	CHISQUARE
17	2	0.288964	11	BESTONE	ZERO	ZERO	15	0.627164	MI
18	2	0.010000	9	BESTONE	ZERO	ZERO	15	0.187116	GSQUARE
19	2	0.852384	5	BESTONE	ZERO	ZERO	11	0.409338	MI
20	3	0.620777	9	BESTSET	ONE	TWO	8	0.527968	GSQUARE
21	3	0.518204	15	BESTSET	ONE	TWO	4	0.115860	CHISQUARE
22	3	0.767852	5	BESTONE	ZERO	ZERO	10	0.371482	MI
23	3	0.027212	11	BESTSET	ZERO	ONE	19	0.244087	CHISQUARE
24	3	0.033380	7	BESTSET	ONE	ONE	9	0.106970	MI
25	3	0.990000	8	BESTSET	ZERO	ZERO	15	0.697293	CHIGSQUARE
26	3	0.615963	4	BESTONE	ZERO	ZERO	10	0.296991	MI
27	3	0.010000	1	BESTSET	ONE	ZERO	2	0	MI
28	3	0.067586	5	BESTSET	ONE	ONE	5	0.064496	CHIGSQUARE
29	4	0.816168	5	BESTONE	ZERO	ZERO	10	0.395177	MI
30	4	0.664278	4	BESTONE	ZERO	ZERO	10	0.320686	MI
31	4	0.052938	5	BESTSET	ONE	ONE	5	0.052422	CHIGSQUARE
32	4	0.907326	10	BESTSET	ZERO	ZERO	18	0.908706	CHIGSQUARE
33	4	0.912000	7	BESTSET	ZERO	ZERO	13	0.582894	CHIGSQUARE
34	4	0.010000	5	BESTSET	ONE	ZERO	10	0	CHIGSQUARE
35	4	0.167278	7	BESTSET	ONE	ONE	9	0.202937	MI
36	4	0.010000	8	BESTONE	ONE	ONE	9	0.071819	MI
37	4	0.508191	5	BESTONE	ZERO	ZERO	10	0.261839	MI
38	5	0.095720	5	BESTSET	ONE	TWO	2	0.104652	CHIGSQUARE
39	5	0.052781	5	BESTSET	ONE	ONE	5	0.052231	CHIGSQUARE
40	5	0.723281	6	BESTSET	ZERO	ZERO	12	0.432850	CHIGSQUARE
41	5	0.990000	10	BESTONE	ZERO	ZERO	16	0.802639	MI
42	5	0.871882	7	BESTSET	ZERO	ZERO	13	0.540431	CHIGSQUARE
43	5	0.959758	7	BESTSET	ZERO	ZERO	13	0.612419	CHIGSQUARE
44	5	0.097758	5	BESTSET	ONE	ONE	5	0.079525	CHIGSQUARE

Output 4.1.4 *continued*

Tuner History All Evaluated Configurations	
Misclassification Error Percentage	Time In Seconds
23.93	1.01
24.26	0.33
31.80	2.22
38.69	2.14
27.54	2.13
37.05	2.45
24.59	2.34
24.59	2.06
37.05	2.31
25.90	2.17
23.93	0.00
24.26	1.01
24.59	0.51
37.05	0.40
37.05	0.42
27.21	0.40
37.05	0.40
26.23	0.48
23.93	0.47
25.25	0.47
24.59	0.55
37.05	0.47
25.90	0.48
37.05	0.43
24.26	0.55
24.92	0.69
26.89	0.50
26.89	0.41
23.93	0.44
25.90	0.46
26.89	0.50
23.93	0.46
25.90	0.46
22.62	0.62
24.59	0.44
26.23	0.37
24.26	0.61
26.89	0.51
28.20	0.42
23.93	0.41
25.57	0.40
26.89	0.44
22.62	0.60
22.62	0.65
23.93	0.36

Output 4.1.4 *continued*

Tuner History
All Evaluated Configurations

Evaluation	Iteration	Alpha	Maximum Number of Parents	Parenting	Prescreening	Variable Selection	Number of Bins	Mutual Information Alpha	Independence Test	Misclassification Error Percentage	Time In Seconds
45	5	0.010000	11	BESTONE	ONE	TWO	8	0.142845	GSQUARE	37.05	0.32
46	5	0.010000	8	BESTONE	ONE	ONE	9	0.071026	MI	24.26	0.56

Output 4.1.5 shows the "Best Configuration" output table, which displays the values of hyperparameters and the objective value for the best configuration that the tuner found.

Output 4.1.5 Best Configuration Output

Best Configuration	
Evaluation	33
Alpha	0.91199969
Maximum Number of Parents	7
Parenting Algorithm	BESTSET
Prescreening	ZERO
Network Structure	NAIVE
Variable Selection	ZERO
Number of Bins	13
Mutual Information Alpha	0.58289433
Independence Test Method	CHIGSQUARE
Misclassification Error Percentage	22.62295082

Output 4.1.6 shows the "Tuner Summary" output table, which displays statistics about the tuning process.

Output 4.1.6 Tuner Summary Output

Tuner Summary	
Initial Configuration Objective Value	23.9344
Best Configuration Objective Value	22.6230
Worst Configuration Objective Value	38.6885
Initial Configuration Evaluation Time in Seconds	1.0053
Best Configuration Evaluation Time in Seconds	0.6184
Number of Improved Configurations	1
Number of Evaluated Configurations	46
Total Tuning Time in Seconds	8.2710
Parallel Tuning Speedup	3.5091

Output 4.1.7 shows the "Tuner Task Timing" output table, which displays timing information about the different tasks that were performed during the tuning process.

Output 4.1.7 Tuner Task Timing Output

Tuner Task Timing

Task	Seconds	Percent
Model Training	20.44	70.44
Model Scoring	5.50	18.94
Total Objective Evaluations	25.94	89.39
Tuner	3.08	10.61
Total CPU Time	29.02	100.00

Output 4.1.8 shows the "Output CAS Tables" output table, which displays a list of CAS tables that were created during the tuning process.

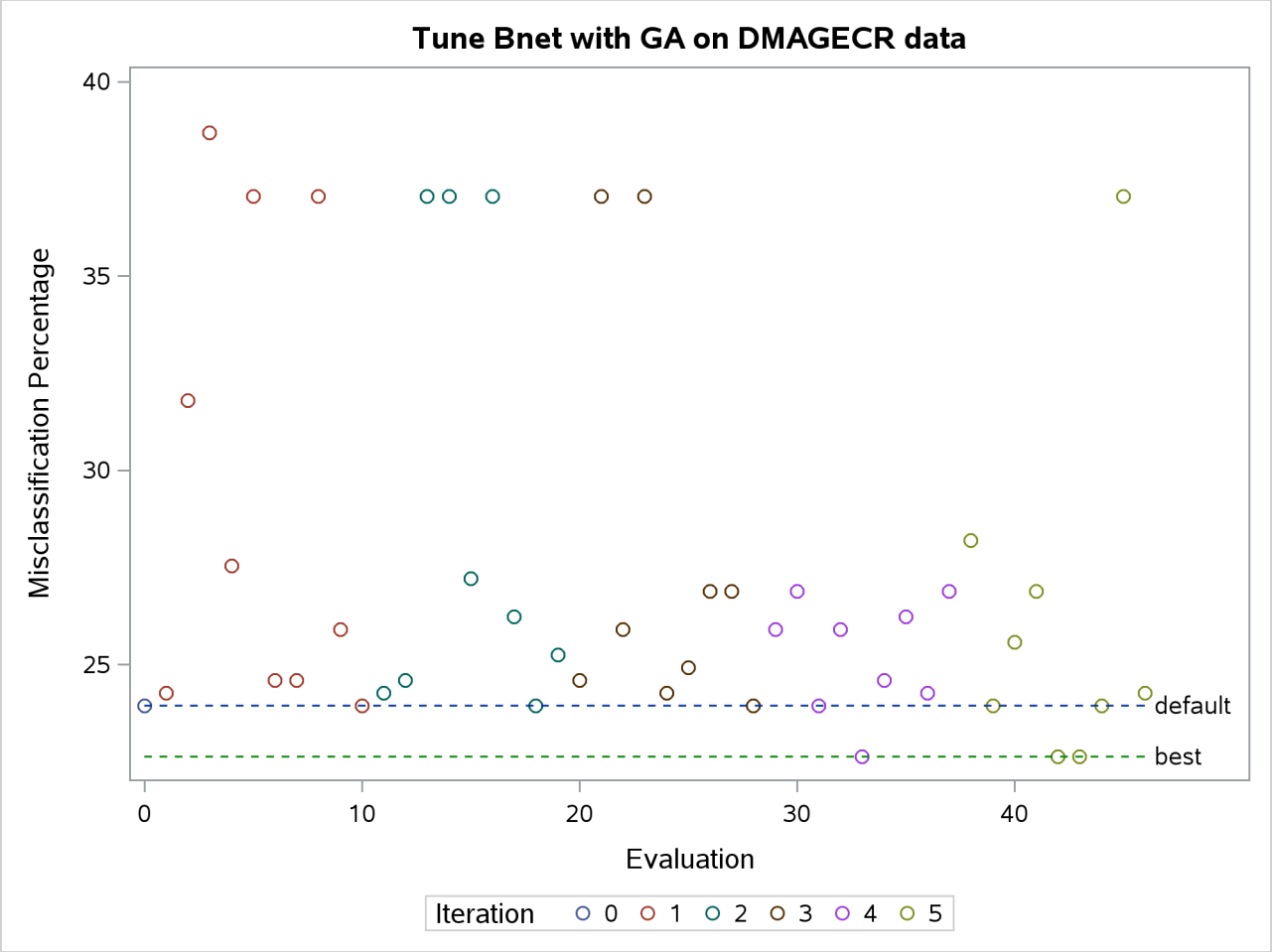
Output 4.1.8 Output CAS Tables

Output CAS Tables

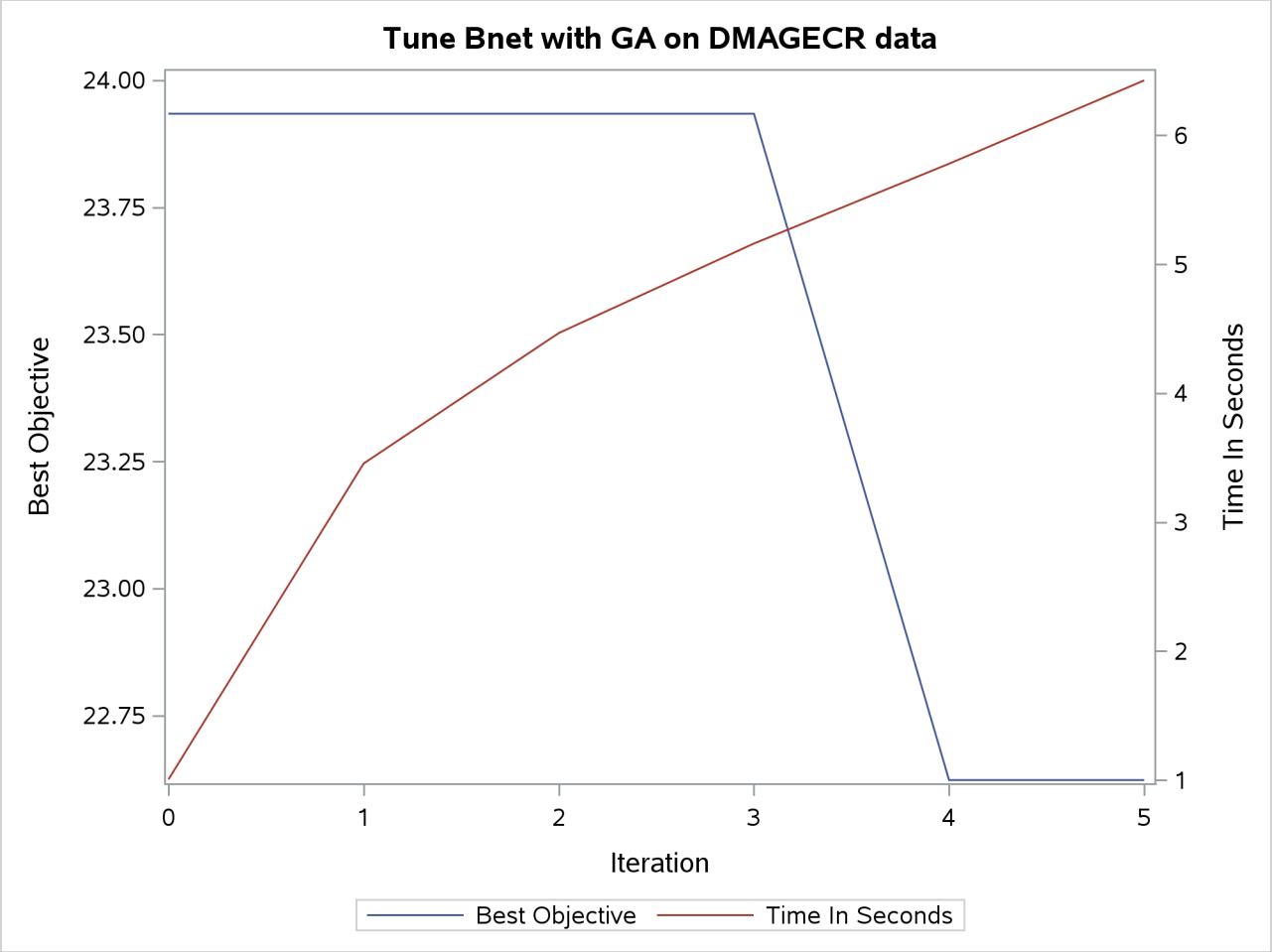
CAS Library	Name	Number of Rows	Number of Columns
YOUR_CAS_LIB	network	327	10

The first PROC SGPLOT step creates a scatter plot of the objective value (root average square error in this case) evaluation history by iteration, which is shown in Output 4.1.9. The second PROC SGPLOT step creates a line plot of the best objective value by iteration and a line plot of the total tuning time by iteration, which is shown in Output 4.1.10.

Output 4.1.9 Evaluation History Plot



Output 4.1.10 Iteration History Plot



Example 4.2: Tuning the Hyperparameters of a Decision Tree

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the **tuneDecisionTree** action to automatically tune the hyperparameters of a decision tree. The following DATA step loads the hmeq data set from the Sampsio library into a data table named mycas.hmeq. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
   data mycas.hmeq;
      length Bad Loan MortDue Value 8 Reason Job $7
             YoJ Derog Delinq CLAge nInq CLNo DebtInc 8;
      set sampsio.hmeq;
   run;
```

The following PROC CAS code uses the **tuneDecisionTree** action to automatically tune the hyperparameters of a decision tree model that is trained on the hmeq data table (note that the syntax of the **trainOptions** parameter here is the same as the syntax of the **dtreeTrain** action):

```
proc cas noqueue;
   autotune.tuneDecisionTree /
      trainOptions={
         table={
            name='hmeq',
            where='Bad NE .'
         },
         inputs={
            "CLAge",   "CLNo",  "DebtInc", "Loan",
            "MortDue", "Value", "YoJ",     "Delinq",
            "Derog",   "Job",   "nInq",    "Reason"
         },
         target='Bad',
         nominals={
            "Delinq", "Derog", "Job", "nInq",
            "Reason", "Bad"
         },
         casOut={
            name='decisiontree_hmeq_model',
            replace=true
         }
      }
      tunerOptions={seed=12345}
   ;
   ods output TunerResults      = TuneResults(keep=MisclassErr);
   ods output EvaluationHistory = EvalHistory;
   ods output IterationHistory  = IterHistory;
   run;
quit;
```

The following optional code includes two DATA steps and two SGPLOT procedure steps that generate the following two plots from the **tuneDecisionTree** results:

- history of all evaluations (the objective value versus the evaluation number, grouped by iteration)
- iteration history (the best objective value and total tuning time versus the iteration number)

```
data _null_;
   set TuneResults;
   if _n_ = 1 then call symput('default', MisclassErr);
   else if _n_ = 2 then call symput('best', MisclassErr);
run;
```

```
data EvalHistory;
   set EvalHistory;
   default = &default.;
   best = &best.;
run;

proc sgplot data=EvalHistory;
   title "Tune Decision Tree with GA on HMEQ data";
   scatter x=Evaluation y=MisclassErr / group=Iteration;
   series x=Evaluation y=default /
      curvelabel="default" lineattrs=(pattern=shortdash);
   series x=Evaluation y=best /
      curvelabel="best" lineattrs=(color=green pattern=shortdash);
run;

proc sgplot data=IterHistory;
   title "Tune Decision Tree with GA on HMEQ data";
   series x=Iteration y=Best_Obj;
   series x=Iteration y=Time_Sec / y2axis;
run;
```

The "Tuner Information" output displays the values of the options that the tuner used.

Output 4.2.1 Tuner Information Output

Results from decisionTree.dtreeTrain

Tuner Information	
Model Type	Decision Tree
Tuner Objective Function	Misclassification Error Percentage
Search Method	GA
Maximum Evaluations	50
Population Size	10
Maximum Iterations	5
Maximum Tuning Time in Seconds	36000
Validation Type	Single Partition
Validation Partition Fraction	0.30
Log Level	2
Seed	12345
Number of Parallel Evaluations	9
Number of Workers per Subsession	1

The "Tuner Results" output displays the default configuration (Iteration 0) and up to 10 of the best configurations that the tuner found. For each configuration, the hyperparameter values and objective value are provided.

Output 4.2.2 Tuner Results Output

Tuner Results
Default and Best Configurations

Evaluation	Maximum Tree Levels	Number of Bins	Criterion	Misclassification Error Percentage	Time In Seconds
0	11	20	GAINRATIO	12.92	0.96
44	6	43	CHAID	10.68	0.46
45	6	46	CHISQUARE	10.68	0.53
9	6	40	CHAID	11.07	2.09
39	6	37	CHAID	11.13	0.46
40	6	41	CHAID	11.13	0.37
3	20	160	GAINRATIO	11.19	2.01
16	15	144	GAIN	11.19	0.62
17	19	164	GAINRATIO	11.24	0.57
18	9	66	GINI	11.35	0.48
23	4	20	CHAID	11.35	0.37

The "Tuner Iteration History" output displays the results of each iteration. For each iteration, the number of evaluations, the best objective value so far, and the tuning time in seconds are provided.

Output 4.2.3 Iteration History Output

Tuner Iteration History

Iteration	Evaluations	Best Objective	Time In Seconds
0	1	12.92	0.96
1	10	11.07	4.46
2	19	11.07	5.11
3	28	11.07	5.66
4	37	11.07	6.17
5	46	10.68	6.70

The "Tuner Evaluation History" output displays all model configurations that the tuner evaluated. For each configuration, the hyperparameter values and objective value are provided.

Output 4.2.4 Evaluation History Output

Tuner History
All Evaluated Configurations

Evaluation	Iteration	Maximum Tree Levels	Number of Bins	Criterion	Misclassification Error Percentage	Time In Seconds
0	0	11	20	GAINRATIO	12.92	0.96
1	1	8	20	GINI	12.08	0.27
2	1	14	180	CHISQUARE	12.64	2.40
3	1	20	160	GAINRATIO	11.19	2.01
4	1	16	120	CHAID	13.70	2.54
5	1	2	60	GAIN	15.38	2.05
6	1	10	80	GINI	11.80	2.18
7	1	4	140	GAIN	11.91	2.35
8	1	12	100	CHISQUARE	12.53	2.44
9	1	6	40	CHAID	11.07	2.09
10	1	11	20	GAINRATIO	12.92	0.00
11	2	12	200	GAINRATIO	12.92	0.55
12	2	12	65	GINI	11.58	0.43
13	2	10	166	GINI	12.86	0.53
14	2	17	176	GINI	12.02	0.63
15	2	17	156	GAINRATIO	11.58	0.62
16	2	15	144	GAIN	11.19	0.62
17	2	19	164	GAINRATIO	11.24	0.57
18	2	9	66	GINI	11.35	0.48
19	2	13	166	CHISQUARE	12.30	0.65
20	3	8	60	GINI	11.97	0.41
21	3	2	150	GAINRATIO	18.62	0.33
22	3	14	132	GAIN	12.02	0.54
23	3	4	20	CHAID	11.35	0.37
24	3	8	116	CHISQUARE	11.69	0.53
25	3	4	20	GAINRATIO	14.26	0.40
26	3	13	119	GAIN	12.42	0.49
27	3	4	20	CHISQUARE	11.35	0.43
28	3	12	98	GINI	12.36	0.47
29	4	5	37	GINI	11.52	0.35
30	4	3	77	CHISQUARE	13.87	0.45
31	4	4	21	CHISQUARE	11.80	0.47
32	4	2	56	CHISQUARE	15.10	0.40
33	4	9	78	GINI	12.02	0.51
34	4	2	96	CHAID	15.44	0.41
35	4	9	77	GAINRATIO	13.31	0.47
36	4	2	55	GINI	15.10	0.37
37	4	6	44	CHISQUARE	11.41	0.51
38	5	8	56	CHAID	11.63	0.52
39	5	6	37	CHAID	11.13	0.46
40	5	6	41	CHAID	11.13	0.37
41	5	5	32	CHAID	12.08	0.44
42	5	8	65	GAIN	11.80	0.50
43	5	3	20	GINI	14.26	0.35
44	5	6	43	CHAID	10.68	0.46

Output 4.2.4 *continued*

Tuner History
All Evaluated Configurations

Evaluation	Iteration	Maximum Tree Levels	Number of Bins	Criterion	Misclassification Error Percentage	Time In Seconds
45	5	6	46	CHISQUARE	10.68	0.53
46	5	4	23	CHISQUARE	11.97	0.46

The "Best Configuration" output displays the values of hyperparameters and the objective value for the best configuration that the tuner found.

Output 4.2.5 Best Configuration Output

Best Configuration	
Evaluation	44
Maximum Tree Levels	6
Maximum Bins	43
Criterion	CHAID
Misclassification Error Percentage	10.68

The "Tuner Summary" output displays statistics about the tuning process.

Output 4.2.6 Tuner Summary Output

Tuner Summary	
Initial Configuration Objective Value	12.9195
Best Configuration Objective Value	10.6823
Worst Configuration Objective Value	18.6242
Initial Configuration Evaluation Time in Seconds	0.9577
Best Configuration Evaluation Time in Seconds	0.4623
Number of Improved Configurations	4
Number of Evaluated Configurations	46
Total Tuning Time in Seconds	6.8219
Parallel Tuning Speedup	4.0993

The "Tuner Task Timing" output displays timing information about the different tasks that were performed during the tuning process.

Output 4.2.7 Tuner Timing Output

Tuner Task Timing		
Task	**Seconds**	**Percent**
Model Training	19.73	70.56
Model Scoring	5.60	20.02
Total Objective Evaluations	25.33	90.59
Tuner	2.63	9.41
Total CPU Time	27.96	100.00

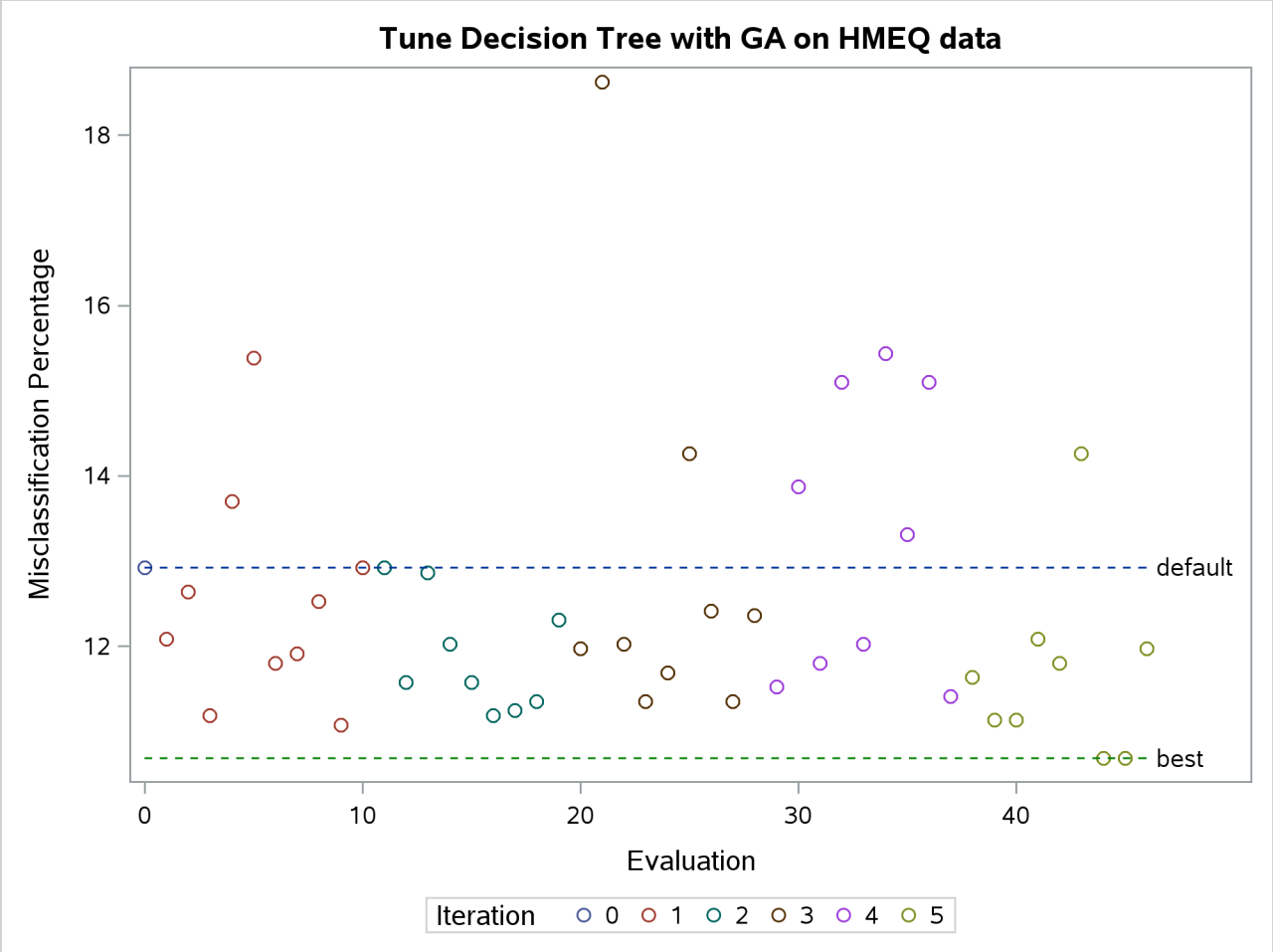
The "Output CAS Tables" output displays a list of CAS tables that were created during the tuning process.

Output 4.2.8 Output CAS Tables

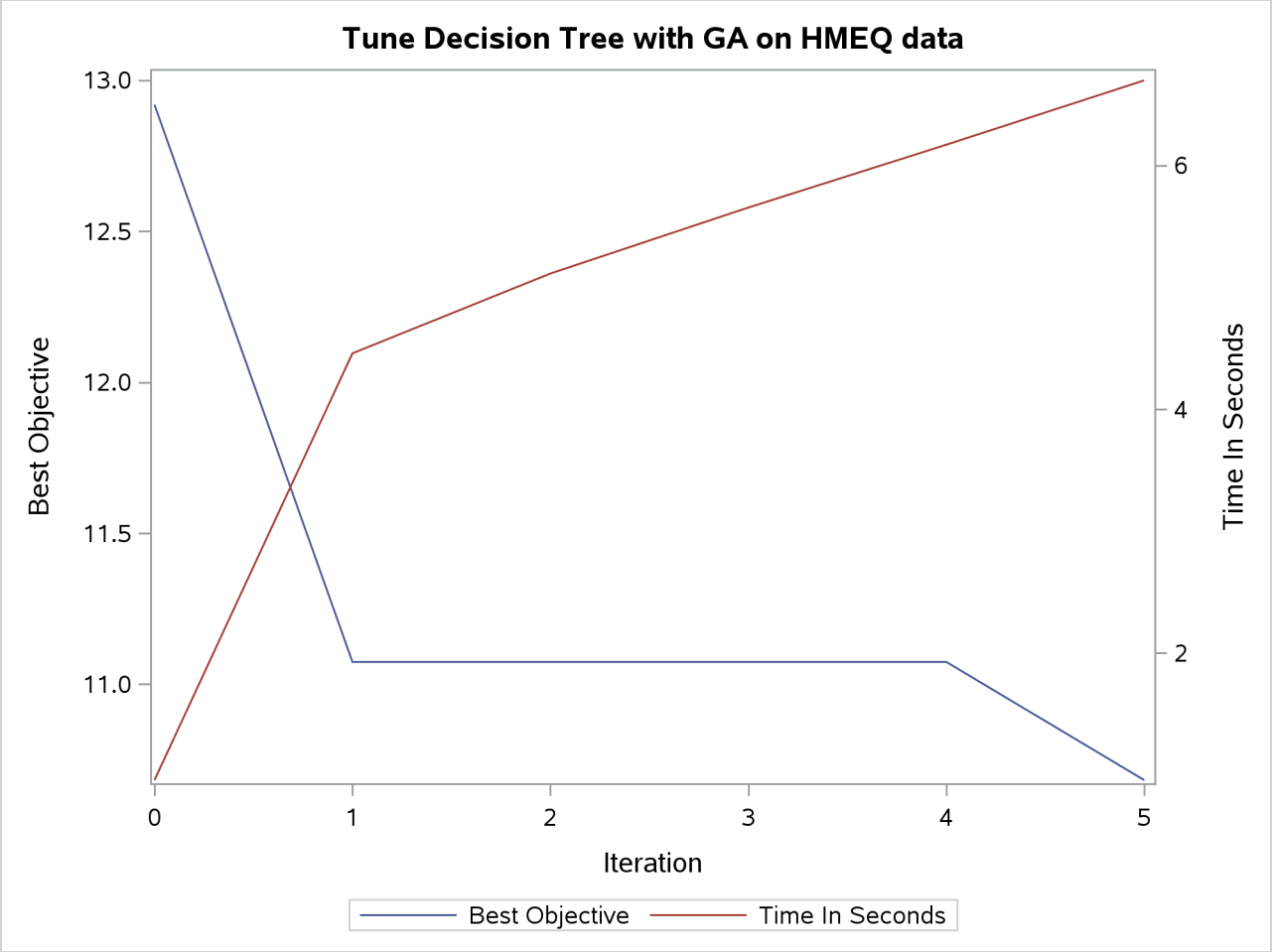
Output CAS Tables			
CAS Library	**Name**	**Number of Rows**	**Number of Columns**
YOUR_CAS_LIB	**decisiontree_hmeq_model**	55	53

The first PROC SGPLOT step creates a scatter plot of the objective value (misclassification error in this case) evaluation history by iteration. The second PROC SGPLOT step creates a line plot of the best objective value by iteration and a line plot of the total tuning time by iteration.

Output 4.2.9 Evaluation History Plot



Output 4.2.10 Iteration History Plot



Example 4.3: Tuning the Hyperparameters of a Factorization Machine

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the **tuneFactMac** action to automatically tune the hyperparameters of a factorization machine model from observations in the MovieLens data table. In this case, the data are derived from companies that provide movies for online viewing. A company wants to offer its customers recommendations of movies that they might like. These recommendations are based on ratings that are provided by users. The MovieLens data set was developed by the GroupLens project at the University of Minnesota and is available at http://grouplens.org/datasets/movielens. This example uses the MovieLens 100K version.

There are four columns in the MovieLens 100K data set: user ID, item ID (each item is a movie), rating, and timestamp. This example predicts the rating for a specified user ID and an item ID. The data set is very sparse because most combinations of users and movies are not rated.

You can download the compressed archive file from the website at http://files.grouplens.org/datasets/movielens/ml-100k.zip and use any third-party unzip tool to extract all the files in the archive to the destination directory of your choice.[1] The file inside the archive that contains the ratings is *u.data*. This file contains data in a tab-separated format.

The following statements load the data table from a directory accessible from the CAS client into your CAS session:

```
proc casutil;
   load
      file          = "/my/data/u.data" /* full path to the data file */
      casout        = "movlens"
      importoptions = (filetype="CSV" delimiter="TAB" getnames=false
                       vars=("user" "item" "rating" "timestamp"));
run;
```

The following PROC CAS code uses the **tuneFactMac** action to automatically tune the hyperparameters of a factorization machine model that is trained on the movlens data table (note that the syntax of the **trainOptions** parameter here is the same as the syntax of the **factmac** action):

```
proc cas noqueue;
   action autotune.tuneFactMac /
      trainOptions    = {
         table        = {name="movlens"},
         inputs       = {"user", "item"},
         nominals     = {"user", "item"},
         target       = "rating",
         maxIter      = 20,
         nFactors     = 10,
         learnStep    = 0.15,
         saveState    = {name='factmac_movlens_model', replace=true},
         output       = {
            casout    = {name="score_out", replace=true},
            copyvars  = {"user","item","rating"}
         },
         outModel     = {name="factors_out", replace=true}
      }
      tunerOptions    = {
         searchMethod = "GA",
         popSize      = 15,
         logLevel     = 3,
         objective    = "RMSLE",
```

[1] Disclaimer: SAS may reference other websites or content or resources for use at Customer's sole discretion. SAS has no control over any websites or resources that are provided by companies or persons other than SAS. Customer acknowledges and agrees that SAS is not responsible for the availability or use of any such external sites or resources, and does not endorse any advertising, products, or other materials on or available from such websites or resources. Customer acknowledges and agrees that SAS is not liable for any loss or damage that may be incurred by Customer or its end users as a result of the availability or use of those external sites or resources, or as a result of any reliance placed by Customer or its end users on the completeness, accuracy, or existence of any advertising, products, or other materials on, or available from, such websites or resources.

```
         seed         = 12345
      }
   ;
   ods output TunerResults      = TuneResults(keep=RootMeanSqLogErr);
   ods output EvaluationHistory = EvalHistory;
   ods output IterationHistory  = IterHistory;
   run;
quit;
```

The following optional code includes two DATA steps and two SGPLOT procedure steps that generate the following two plots from the **tuneFactMac** results:

- history of all evaluations (the objective value versus the evaluation number, grouped by iteration)
- iteration history (the best objective value and total tuning time versus the iteration number)

```
data _null_;
   set TuneResults;
   if _n_ = 1 then call symput('default', RootMeanSqLogErr);
   else if _n_ = 2 then call symput('best', RootMeanSqLogErr);
run;

data EvalHistory;
   set EvalHistory;
   default = &default.;
   best = &best.;
run;

proc sgplot data=EvalHistory;
   title "Tune FactMac with GA on MovieLens data";
   scatter x=Evaluation y=RootMeanSqLogErr / group=Iteration;
   series x=Evaluation y=default /
      curvelabel="default" lineattrs=(pattern=shortdash);
   series x=Evaluation y=best /
      curvelabel="best" lineattrs=(color=green pattern=shortdash);
run;

proc sgplot data=IterHistory;
   title "Tune FactMac with GA on MovieLens data";
   series x=Iteration y=Best_Obj;
   series x=Iteration y=Time_Sec / y2axis;
run;
```

The “Tuner Information” output displays the values of parameters that the tuner used.

Output 4.3.1 Tuner Information Output

Results from factmac.factmac

Tuner Information	
Model Type	Factorization Machine
Tuner Objective Function	Root Mean Square Logarithmic Error
Search Method	GA
Maximum Evaluations	75
Population Size	15
Maximum Iterations	5
Maximum Tuning Time in Seconds	36000
Validation Type	Single Partition
Validation Partition Fraction	0.30
Log Level	3
Seed	12345
Number of Parallel Evaluations	14
Number of Workers per Subsession	1

The "Tuner Results" output displays the default configuration (Iteration 0) and up to 10 of the best configurations that the tuner found. For each configuration, the hyperparameter values and objective value are provided.

Output 4.3.2 Tuner Results Output

Tuner Results
Default and Best Configurations

Evaluation	Number of Factors	Maximum Number of Iterations	Learning Step Size	Root Mean Squared Log Error	Time In Seconds
0	5	30	0.001000	0.3982	1.49
29	5	160	0.001000	0.3352	1.62
51	5	150	0.001000	0.3420	1.70
47	5	40	0.010000	0.3579	0.97
32	5	80	0.010000	0.3611	1.09
48	5	150	1.000000	0.3648	1.68
39	5	60	0.010000	0.3650	1.02
49	5	90	0.010000	0.3721	1.21
40	5	180	1.000000	0.3732	1.73
23	5	200	1.000000	0.3787	1.94
44	5	100	0.010000	0.3877	1.33

The "Tuner Iteration History" output displays the results of each iteration. For each iteration, the number of evaluations, the best objective value so far, and the tuning time in seconds are provided.

Output 4.3.3 Iteration History Output

Tuner Iteration History			
Iteration	Evaluations	Best Objective	Time In Seconds
0	1	0.40	1.49
1	15	0.40	6.65
2	26	0.38	8.75
3	33	0.34	10.49
4	42	0.34	12.77
5	54	0.34	14.85

The "Tuner Evaluation History" output displays all model configurations that the tuner evaluated. For each configuration, the hyperparameter values and objective value are provided.

Output 4.3.4 Tuner Evaluation History Output

Tuner History
All Evaluated Configurations

Evaluation	Iteration	Number of Factors	Maximum Number of Iterations	Learning Step Size	Root Mean Squared Log Error	Time In Seconds
0	0	5	30	0.001000	0.3982	1.49
1	1	20	10	1.000000	0.5619	0.69
2	1	25	20	0.001000	0.7420	3.74
3	1	30	110	0.010000	0.7012	4.64
4	1	5	130	0.000001000	0.5459	4.27
5	1	30	40	0.000100	0.8976	4.06
6	1	10	90	0.000010000	0.7049	4.14
7	1	25	120	0.100000	0.4282	4.99
8	1	20	150	0.000001000	0.9209	4.74
9	1	15	60	0.000001000	0.7881	3.75
10	1	5	80	0.000010000	0.5552	4.16
11	1	30	190	0.001000	0.7316	5.04
12	1	10	170	1.000000	0.4432	5.16
13	1	15	50	0.010000	0.5125	4.37
14	1	20	200	0.100000	0.4400	4.60
15	1	5	30	0.001000	0.3982	0.00
16	2	30	90	0.010000	0.7024	1.70
17	2	20	140	0.100000	0.4308	2.09
18	2	25	100	0.000100	0.8310	1.61
19	2	15	160	0.000010000	0.8257	1.73
20	2	10	70	0.000010000	0.7449	1.60
21	2	5	140	0.000001000	0.5455	1.67
22	2	5	90	0.000010000	0.5471	1.59
23	2	5	200	1.000000	0.3787	1.94
24	2	15	120	1.000000	0.5048	1.88
25	2	30	80	0.010000	0.6905	1.65
26	2	20	130	0.100000	0.4330	1.77
27	3	25	140	0.010000	0.6297	1.72
28	3	5	170	0.100000	0.4342	1.69
29	3	5	160	0.001000	0.3352	1.62
30	3	5	150	0.100000	0.4169	1.34
31	3	5	50	0.100000	0.3889	0.89
32	3	5	80	0.010000	0.3611	1.09
33	3	20	150	0.100000	0.4243	1.75
34	4	10	150	0.100000	0.5023	1.73
35	4	15	150	0.100000	0.4610	2.03
36	4	15	80	0.010000	0.5188	1.20
37	4	20	160	0.100000	0.4387	2.27
38	4	20	10	0.100000	0.4380	0.74
39	4	5	60	0.010000	0.3650	1.02
40	4	5	180	1.000000	0.3732	1.73
41	4	5	170	1.000000	0.3960	2.03
42	4	5	120	0.100000	0.4211	1.46
43	5	5	10	0.001000	0.4676	0.74
44	5	5	100	0.010000	0.3877	1.33

Output 4.3.4 *continued*

Tuner History All Evaluated Configurations						
Evaluation	Iteration	Number of Factors	Maximum Number of Iterations	Learning Step Size	Root Mean Squared Log Error	Time In Seconds
45	5	20	80	0.100000	0.4399	1.61
46	5	20	20	0.100000	0.4454	1.09
47	5	5	40	0.010000	0.3579	0.97
48	5	5	150	1.000000	0.3648	1.68
49	5	5	90	0.010000	0.3721	1.21
50	5	5	200	0.000100	0.4198	1.90
51	5	5	150	0.001000	0.3420	1.70
52	5	10	170	0.000100	0.5385	2.05
53	5	15	150	0.010000	0.5035	2.08
54	5	5	160	0.000100	0.4227	1.76

The "Best Configuration" output displays the values of hyperparameters and the objective value for the best configuration that the tuner found.

Output 4.3.5 Best Configuration Output

Best Configuration	
Evaluation	29
Number of Factors	5
Maximum Number of Iterations	160
Learning Step Size	0.001
Root Mean Square Logarithmic Error	0.3351544415

The "Tuner Summary" output displays statistics about the tuning process.

Output 4.3.6 Tuner Summary Output

Tuner Summary	
Initial Configuration Objective Value	0.3982
Best Configuration Objective Value	0.3352
Worst Configuration Objective Value	0.9209
Initial Configuration Evaluation Time in Seconds	1.4852
Best Configuration Evaluation Time in Seconds	1.6237
Number of Improved Configurations	3
Number of Evaluated Configurations	54
Total Tuning Time in Seconds	19.0695
Parallel Tuning Speedup	5.2790

The "Tuner Task Timing" output displays timing information about the different tasks that were performed during the tuning process.

Output 4.3.7 Tuner Task Timing Output

Tuner Task Timing

Task	Seconds	Percent
Model Training	72.31	71.83
Model Scoring	22.97	22.81
Total Objective Evaluations	95.28	94.65
Tuner	5.39	5.35
Total CPU Time	100.67	100.00

The "Output CAS Tables" output displays a list of CAS tables that were created during the tuning process.

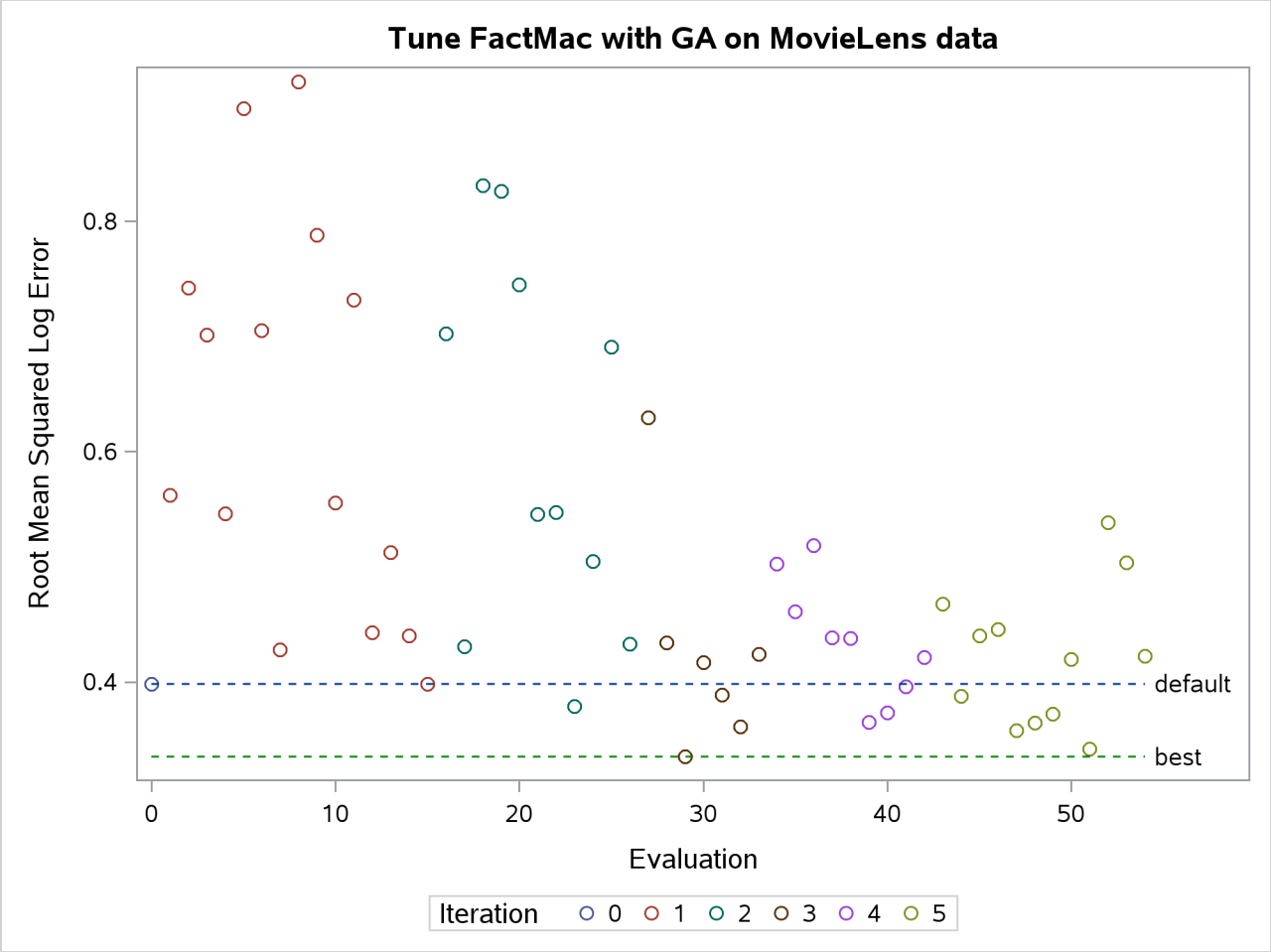
Output 4.3.8 Output CAS Tables

Output CAS Tables

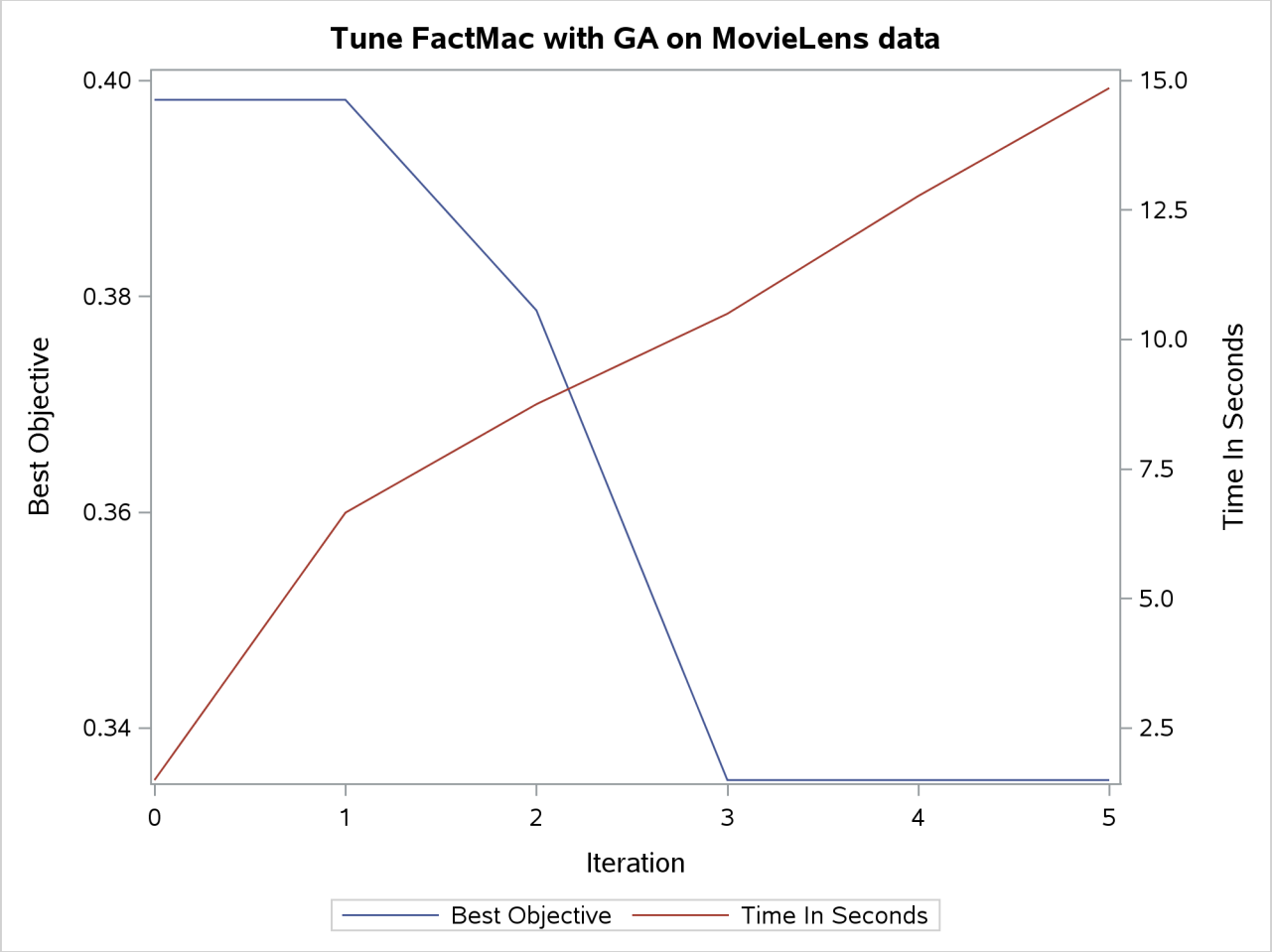
CAS Library	Name	Number of Rows	Number of Columns
YOUR_CAS_LIB	**factmac_movlens_model**	1	2
YOUR_CAS_LIB	**score_out**	10000	4
YOUR_CAS_LIB	**factors_out**	1640	8

The first PROC SGPLOT step creates a scatter plot of the objective value (root mean squared logarithmic error in this case) evaluation history by iteration. The second PROC SGPLOT step creates a line plot of the best objective value by iteration and a line plot of the total tuning time by iteration.

Output 4.3.9 Evaluation History Plot



Output 4.3.10 Iteration History Plot



Example 4.4: Tuning the Hyperparameters of a Forest

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the **tuneForest** action to automatically tune the hyperparameters of a forest. The following DATA step loads the junkmail data set from the Sashelp library into a data table named mycas.junkmail. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.junkmail;
   set sashelp.junkmail;
run;
```

The following PROC CAS code uses the **tuneForest** action to automatically tune the hyperparameters of a forest model that is trained on the junkmail data table (note that the syntax of the **trainOptions** parameter here is the same as the syntax of the **forestTrain** action):

```
proc cas noqueue;
   autotune.tuneForest /
      trainOptions={
         table={
            name='junkmail',
            where='Class NE .'
         },
         inputs={
            "Address",      "Addresses",  "All",        "Bracket",
            "Business",     "CS",         "CapAvg",     "CapLong",
            "CapTotal",     "Conference", "Credit",     "Data",
            "Direct",       "Dollar",     "Edu",        "Email",
            "Exclamation",  "Font",       "Free",       "George",
            "HP",           "HPL",        "Internet",   "Lab",
            "Labs",         "Mail",       "Make",       "Meeting",
            "Money",        "Order",      "Original",   "Our",
            "Over",         "PM",         "Paren",      "Parts",
            "People",       "Pound",      "Project",    "RE",
            "Receive",      "Remove",     "Semicolon",  "Table",
            "Technology",   "Telnet",     "Will",       "You",
            "Your",         "_000",       "_85",        "_415",
            "_650",         "_857",       "_1999",      "_3D"
         },
         target='Class',
         nominals={
            {name='Class'}
         },
         casOut={
            name='forest_junkmail_model', replace=true
         }
      }
      tunerOptions={seed=54321}
   ;
   ods output TunerResults      = TuneResults(keep=MisclassErr);
   ods output EvaluationHistory = EvalHistory;
   ods output IterationHistory  = IterHistory;
   run;
quit;
```

The following optional code includes two DATA steps and two SGPLOT procedure steps that generate the following two plots from the **tuneForest** results:

- history of all evaluations (the objective value versus the evaluation number, grouped by iteration)
- iteration history (the best objective value and total tuning time versus the iteration number)

```
data _null_;
   set TuneResults;
   if _n_ = 1 then call symput('default', MisclassErr);
   else if _n_ = 2 then call symput('best', MisclassErr);
run;

data EvalHistory;
   set EvalHistory;
   default = &default.;
   best = &best.;
run;

proc sgplot data=EvalHistory;
   title "Tune Forest with GA on Junkmail data";
   scatter x=Evaluation y=MisclassErr / group=Iteration;
   series x=Evaluation y=default /
      curvelabel="default" lineattrs=(pattern=shortdash);
   series x=Evaluation y=best /
      curvelabel="best" lineattrs=(color=green pattern=shortdash);
run;

proc sgplot data=IterHistory;
   title "Tune Forest with GA on Junkmail data";
   series x=Iteration y=Best_Obj;
   series x=Iteration y=Time_Sec / y2axis;
run;
```

The “Tuner Information” output displays the values of the options that the tuner used.

Output 4.4.1 Tuner Information Output

Results from decisionTree.forestTrain

Tuner Information	
Model Type	Forest
Tuner Objective Function	Misclassification Error Percentage
Search Method	GA
Maximum Evaluations	50
Population Size	10
Maximum Iterations	5
Maximum Tuning Time in Seconds	36000
Validation Type	Single Partition
Validation Partition Fraction	0.30
Log Level	2
Seed	54321
Number of Parallel Evaluations	9
Number of Workers per Subsession	1

The "Tuner Results" output displays the default configuration (Iteration 0) and up to 10 of the best configurations that the tuner found. For each configuration, the hyperparameter values and objective value are provided.

Output 4.4.2 Tuner Results Output

Tuner Results
Default and Best Configurations

Evaluation	Maximum Tree Levels	Number of Trees	Number of Variables to Try	Bootstrap	Misclassification Error Percentage	Time In Seconds
0	21	100	8	0.600000	9.28	1.77
43	25	124	6	0.730145	7.90	1.49
32	25	125	6	0.739258	8.04	1.65
46	26	141	4	0.844666	8.04	1.92
44	26	141	5	0.835910	8.19	1.80
25	26	137	6	0.812460	8.26	1.76
37	25	134	5	0.803448	8.26	1.76
31	26	142	5	0.845023	8.33	1.79
38	25	134	5	0.803419	8.33	1.69
40	25	130	7	0.775919	8.41	1.74
30	26	141	4	0.844598	8.55	1.78

The "Tuner Iteration History" output displays the results of each iteration. For each iteration, the number of evaluations, the best objective value so far, and the tuning time in seconds are provided.

Output 4.4.3 Iteration History Output

Tuner Iteration History

Iteration	Evaluations	Best Objective	Time In Seconds
0	1	9.28	1.77
1	10	8.70	8.31
2	19	8.70	10.39
3	28	8.26	12.26
4	37	8.04	14.05
5	46	7.90	15.97

The "Tuner Evaluation History" output displays all model configurations that the tuner evaluated. For each configuration, the hyperparameter values and objective value are provided.

Output 4.4.4 Evaluation History Output

Tuner History
All Evaluated Configurations

Evaluation	Iteration	Maximum Tree Levels	Number of Trees	Number of Variables to Try	Bootstrap	Misclassification Error Percentage	Time In Seconds
0	0	21	100	8	0.600000	9.28	1.77
1	1	14	63	56	0.544444	13.77	1.75
2	1	30	78	25	0.188889	10.72	3.23
3	1	27	136	19	0.900000	10.14	4.78
4	1	2	107	44	0.366667	22.97	1.99
5	1	18	20	32	0.277778	12.83	2.82
6	1	8	34	50	0.811111	18.04	2.83
7	1	5	150	38	0.455556	18.62	2.83
8	1	24	121	7	0.722222	8.70	4.20
9	1	21	49	1	0.100000	20.22	2.48
10	1	21	100	8	0.600000	9.28	0.00
11	2	30	150	1	0.878741	12.39	1.63
12	2	23	114	13	0.700963	10.00	2.07
13	2	19	128	15	0.654146	10.51	1.76
14	2	23	150	1	0.856353	12.32	1.20
15	2	20	84	13	0.534131	10.00	1.94
16	2	24	123	7	0.735323	8.77	1.76
17	2	21	102	8	0.613101	8.99	1.32
18	2	26	125	1	0.636816	12.68	1.31
19	2	19	88	24	0.581260	11.09	1.68
20	3	18	125	17	0.623507	10.80	1.74
21	3	22	147	3	0.825714	8.99	1.87
22	3	24	119	8	0.717141	8.62	1.64
23	3	25	125	4	0.733183	9.06	1.63
24	3	24	118	10	0.100000	11.01	0.91
25	3	26	137	6	0.812460	8.26	1.76
26	3	23	118	7	0.703339	9.06	1.50
27	3	27	150	2	0.900000	8.84	1.43
28	3	23	116	8	0.699303	8.99	1.78
29	4	23	112	9	0.666820	9.42	1.58
30	4	26	141	4	0.844598	8.55	1.78
31	4	26	142	5	0.845023	8.33	1.79
32	4	25	125	6	0.739258	8.04	1.65
33	4	24	120	7	0.716339	8.99	1.59
34	4	20	150	2	0.897260	9.42	1.33
35	4	23	142	3	0.804729	8.84	1.64
36	4	22	103	11	0.620589	9.86	1.43
37	4	25	134	5	0.803448	8.26	1.76
38	5	25	134	5	0.803419	8.33	1.69
39	5	23	142	3	0.804701	8.84	1.47
40	5	25	130	7	0.775919	8.41	1.74
41	5	24	128	5	0.773873	8.70	1.59
42	5	25	136	5	0.815448	8.62	1.85
43	5	25	124	6	0.730145	7.90	1.49
44	5	26	141	5	0.835910	8.19	1.80

Output 4.4.4 *continued*

Tuner History
All Evaluated Configurations

Evaluation	Iteration	Maximum Tree Levels	Number of Trees	Number of Variables to Try	Bootstrap	Misclassification Error Percentage	Time In Seconds
45	5	26	142	1	0.845091	12.03	1.13
46	5	26	141	4	0.844666	8.04	1.92

The "Best Configuration" output displays the values of hyperparameters and the objective value for the best configuration that the tuner found.

Output 4.4.5 Best Configuration Output

Best Configuration	
Evaluation	43
Number of Trees	124
Number of Variables to Try	6
Bootstrap	0.73014475
Maximum Tree Levels	25
Misclassification Error Percentage	7.90

The "Tuner Summary" output displays statistics about the tuning process.

Output 4.4.6 Tuner Summary Output

Tuner Summary	
Initial Configuration Objective Value	9.2754
Best Configuration Objective Value	7.8986
Worst Configuration Objective Value	22.9710
Initial Configuration Evaluation Time in Seconds	1.7669
Best Configuration Evaluation Time in Seconds	1.4949
Number of Improved Configurations	5
Number of Evaluated Configurations	46
Total Tuning Time in Seconds	17.7453
Parallel Tuning Speedup	4.5427

The "Tuner Task Timing" output displays timing information about the different tasks that were performed during the tuning process.

Output 4.4.7 Tuner Timing Output

Tuner Task Timing

Task	Seconds	Percent
Model Training	60.41	74.95
Model Scoring	16.73	20.75
Total Objective Evaluations	77.15	95.70
Tuner	3.46	4.30
Total CPU Time	80.61	100.00

The "Output CAS Tables" output displays a list of CAS tables that were created during the tuning process.

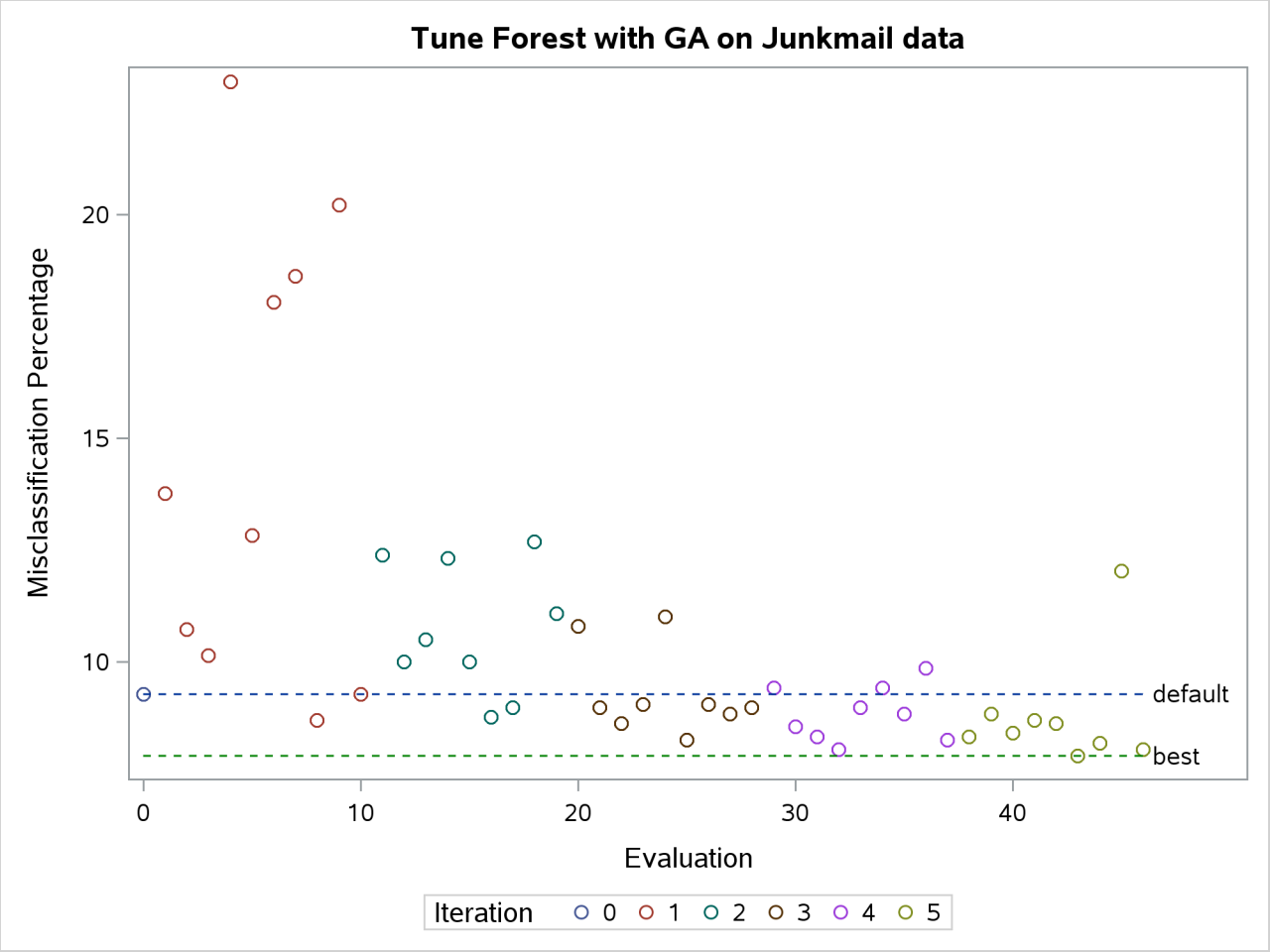
Output 4.4.8 Output CAS Tables

Output CAS Tables

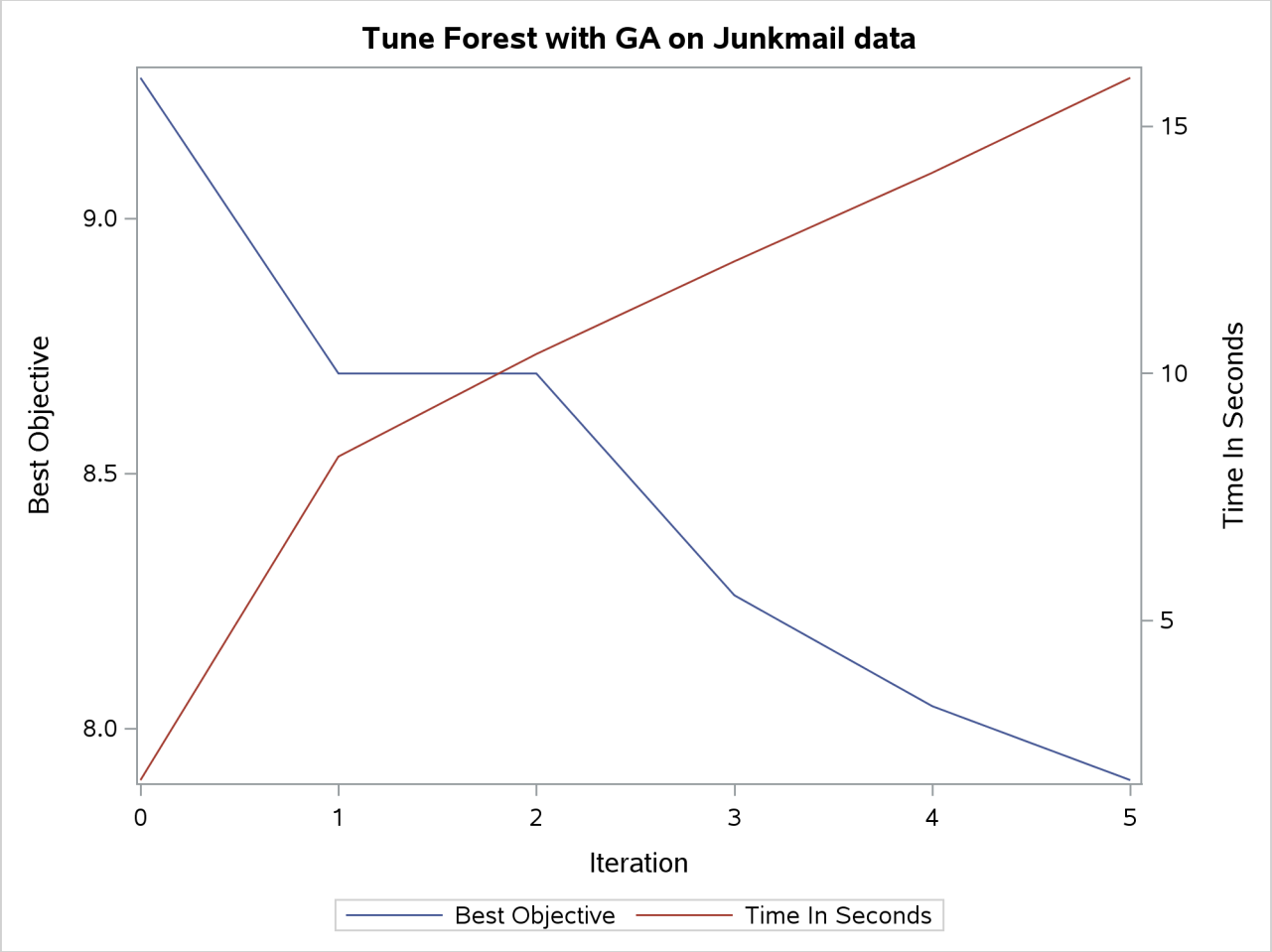
CAS Library	Name	Number of Rows	Number of Columns
YOUR_CAS_LIB	forest_junkmail_model	19358	29

The first PROC SGPLOT step creates a scatter plot of the objective value (misclassification error in this case) evaluation history by iteration. The second PROC SGPLOT step creates a line plot of the best objective value by iteration and a line plot of the total tuning time by iteration.

Output 4.4.9 Evaluation History Plot



Output 4.4.10 Iteration History Plot



Example 4.5: Tuning the Hyperparameters of a Gradient Boosting Tree Model

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the **tuneGradientBoostTree** action to automatically tune the hyperparameters of a gradient boosting tree model. The following DATA step loads the iris data set from the Sashelp library into a data table named mycas.iris. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.iris;
   set sashelp.iris;
run;
```

The following PROC CAS code uses the **tuneGradientBoostTree** action to automatically tune the hyperparameters of a gradient boosting tree model trained on the iris data table (note that the syntax of the **trainOptions** parameter here is the same as the syntax of the **gbtreeTrain** action):

```
proc cas noqueue;
   autotune.tuneGradientBoostTree /
      trainOptions = {
         table     = {name="iris"},
         inputs    = {"SepalLength",
                      "SepalWidth",
                      "PetalLength",
                      "PetalWidth"
                     },
         target     = "Species",
         nominal    = {"Species"},
         casout     = {name="gradboost_iris_model",
                       replace=true
                      }
      }
     tunerOptions={seed=12345}
   ;
   ods output TunerResults      = TuneResults(keep=MisclassErr);
   ods output EvaluationHistory = EvalHistory;
   ods output IterationHistory  = IterHistory;
   run;
quit;
```

The following optional code includes two DATA steps and two SGPLOT procedure steps that generate the following two plots from the **tuneGradientBoostTree** results:

- history of all evaluations (the objective value versus the evaluation number, grouped by iteration)
- iteration history (the best objective value and total tuning time versus the iteration number)

```
data _null_;
   set TuneResults;
   if _n_ = 1 then call symput('default', MisclassErr);
   else if _n_ = 2 then call symput('best', MisclassErr);
run;

data EvalHistory;
   set EvalHistory;
   default = &default.;
   best = &best.;
run;

proc sgplot data=EvalHistory;
   title "Tune GradBoost with GA on Iris data";
   scatter x=Evaluation y=MisclassErr / group=Iteration;
   series x=Evaluation y=default /
      curvelabel="default" lineattrs=(pattern=shortdash);
   series x=Evaluation y=best /
      curvelabel="best" lineattrs=(color=green pattern=shortdash);
run;

proc sgplot data=IterHistory;
   title "Tune GradBoost with GA on Iris data";
   series x=Iteration y=Best_Obj;
   series x=Iteration y=Time_Sec / y2axis;
run;
```

The "Tuner Information" output displays the values of options that the tuner used.

Output 4.5.1 Tuner Information Output

Results from decisionTree.gbtreeTrain

Tuner Information	
Model Type	Gradient Boosting Tree
Tuner Objective Function	Misclassification Error Percentage
Search Method	GA
Maximum Evaluations	50
Population Size	10
Maximum Iterations	5
Maximum Tuning Time in Seconds	36000
Validation Type	Single Partition
Validation Partition Fraction	0.30
Log Level	2
Seed	12345
Number of Parallel Evaluations	9
Number of Workers per Subsession	1

The "Tuner Results" output displays the default configuration (Iteration 0) and up to 10 of the best configurations that the tuner found. For each configuration, the hyperparameter values and objective value are provided.

Output 4.5.2 Tuner Results Output

Tuner Results
Default and Best Configurations

Evaluation	Number of Trees	Number of Variables to Try	Learning Rate	Sampling Rate	Lasso	Ridge	Misclassification Error Percentage	Time In Seconds
0	100	4	0.100000	0.500000	0	0	11.11	1.11
30	86	2	0.808008	0.124101	1.305009	9.650734	2.22	0.47
45	43	2	0.646442	0.164256	3.479246	9.337534	2.22	0.46
14	72	2	1.000000	0.137222	2.015434	10.000000	4.44	0.51
24	59	2	0.775284	0.265152	4.006443	7.922743	4.44	0.48
43	100	2	0.738707	0.146162	1.731733	10.000000	4.44	0.38
44	85	2	0.814418	0.122061	1.265539	9.479030	4.44	0.52
46	72	2	0.996278	0.136968	2.001660	9.993229	4.44	0.47
1	63	1	0.230000	0.900000	3.333333	3.333333	6.67	0.26
2	107	1	0.890000	0.300000	1.111111	8.888889	6.67	2.79
4	121	2	0.450000	0.400000	10.000000	10.000000	6.67	2.51

The "Tuner Iteration History" output displays the results of each iteration. For each iteration, the number of evaluations, the best objective value so far, and the tuning time in seconds are provided.

Output 4.5.3 Iteration History Output

Tuner Iteration History

Iteration	Evaluations	Best Objective	Time In Seconds
0	1	11.11	1.11
1	10	6.67	5.31
2	19	4.44	5.86
3	28	4.44	6.38
4	37	2.22	6.93
5	46	2.22	7.46

The "Tuner Evaluation History" output displays all model configurations that the tuner evaluated. For each configuration, the hyperparameter values and objective value are provided.

Output 4.5.4 Evaluation History Output

Tuner History
All Evaluated Configurations

Evaluation	Iteration	Number of Trees	Number of Variables to Try	Learning Rate	Sampling Rate	Lasso	Ridge	Misclassification Error Percentage	Time In Seconds
0	0	100	4	0.100000	0.500000	0	0	11.11	1.11
1	1	63	1	0.230000	0.900000	3.333333	3.333333	6.67	0.26
2	1	107	1	0.890000	0.300000	1.111111	8.888889	6.67	2.79
3	1	150	4	0.120000	0.500000	0	0	11.11	2.79
4	1	121	2	0.450000	0.400000	10.000000	10.000000	6.67	2.51
5	1	20	3	0.560000	0.100000	4.444444	5.555556	66.67	2.38
6	1	78	4	0.780000	1.000000	6.666667	6.666667	6.67	2.85
7	1	34	3	0.010000	0.600000	8.888889	1.111111	6.67	3.08
8	1	92	1	0.340000	0.800000	5.555556	7.777778	6.67	2.90
9	1	49	2	1.000000	0.700000	2.222222	4.444444	6.67	2.98
10	1	100	4	0.100000	0.500000	0	0	11.11	0.00
11	2	20	2	0.098818	0.650260	10.000000	1.669554	6.67	0.42
12	2	92	3	0.064714	0.550260	4.467544	0.558443	11.11	0.51
13	2	74	3	0.210891	0.508686	9.396190	5.169519	6.67	0.38
14	2	72	2	1.000000	0.137222	2.015434	10.000000	4.44	0.51
15	2	115	2	0.746695	0.337222	0.904323	7.234580	13.33	0.55
16	2	87	1	0.154881	0.968290	7.051470	7.403799	6.67	0.46
17	2	102	1	0.704881	0.468290	2.607026	8.514910	6.67	0.55
18	2	74	3	0.764280	0.481830	7.460614	6.349088	15.56	0.44
19	2	103	1	0.874280	0.400037	1.905058	8.571310	6.67	0.49
20	3	105	1	0.881852	0.394407	1.522611	8.724289	6.67	0.43
21	3	76	3	0.771852	0.476200	7.078166	6.502067	15.56	0.36
22	3	88	1	0.140584	0.902009	0	6.318428	6.67	0.50
23	3	111	2	0.455329	0.100000	0	9.009152	11.11	0.47
24	3	59	2	0.775284	0.265152	4.006443	7.922743	4.44	0.48
25	3	77	2	1.000000	0.100000	1.974250	7.048484	42.22	0.41
26	3	54	2	1.000000	0.587919	2.181039	5.550875	6.67	0.36
27	3	20	2	0.933277	0.429084	2.367812	9.189433	11.11	0.45
28	3	109	2	0.805692	0.366269	1.367077	7.852703	13.33	0.52
29	4	47	2	0.910187	0.161323	3.320442	9.877213	6.67	0.35
30	4	86	2	0.808008	0.124101	1.305009	9.650734	2.22	0.47
31	4	101	2	0.600682	0.109933	0.537848	9.273575	8.89	0.50
32	4	20	2	0.646232	0.946828	3.785397	3.006987	11.11	0.42
33	4	69	2	0.855986	0.458910	1.604359	6.465265	6.67	0.53
34	4	71	2	0.883719	0.100000	0.838814	7.622657	6.67	0.55
35	4	75	3	0.824876	0.397419	5.901547	7.315014	11.11	0.51
36	4	107	1	0.969528	0.334369	1.702702	9.008849	6.67	0.50
37	4	90	1	0.228260	0.841971	0.180091	6.602988	6.67	0.45
38	5	59	2	0.934047	0.565613	1.930456	5.606593	6.67	0.43
39	5	91	2	0.678744	0.216637	0.863946	0	8.89	0.53
40	5	75	2	0.960939	0.134553	1.870899	9.928942	6.67	0.48
41	5	68	2	0.988477	0.112414	2.054786	9.365979	8.89	0.53
42	5	103	1	0.973362	0.309560	1.742055	9.133571	6.67	0.37
43	5	100	2	0.738707	0.146162	1.731733	10.000000	4.44	0.38
44	5	85	2	0.814418	0.122061	1.265539	9.479030	4.44	0.52

Output 4.5.4 *continued*

Tuner History
All Evaluated Configurations

Evaluation	Iteration	Number of Trees	Number of Variables to Try	Learning Rate	Sampling Rate	Lasso	Ridge	Misclassification Error Percentage	Time In Seconds
45	5	43	2	0.646442	0.164256	3.479246	9.337534	2.22	0.46
46	5	72	2	0.996278	0.136968	2.001660	9.993229	4.44	0.47

The "Best Configuration" output displays the values of hyperparameters and the objective value for the best configuration that the tuner found.

Output 4.5.5 Best Configuration Output

Best Configuration	
Evaluation	30
Number of Trees	86
Number of Variables to Try	2
Learning Rate	0.80800762
Sampling Rate	0.1241015
Lasso	1.30500872
Ridge	9.65073374
Misclassification Error Percentage	2.22

The "Tuner Summary" output displays statistics about the tuning process.

Output 4.5.6 Tuner Summary Output

Tuner Summary	
Initial Configuration Objective Value	11.1111
Best Configuration Objective Value	2.2222
Worst Configuration Objective Value	66.6667
Initial Configuration Evaluation Time in Seconds	1.1146
Best Configuration Evaluation Time in Seconds	0.4654
Number of Improved Configurations	3
Number of Evaluated Configurations	46
Total Tuning Time in Seconds	8.1517
Parallel Tuning Speedup	3.5271

The "Tuner Task Timing" output displays timing information about the different tasks that were performed during the tuning process.

Output 4.5.7 Tuner Timing Output

Tuner Task Timing

Task	Seconds	Percent
Model Training	17.86	62.10
Model Scoring	6.95	24.19
Total Objective Evaluations	24.81	86.30
Tuner	3.94	13.70
Total CPU Time	28.75	100.00

The "Output CAS Tables" output displays a list of CAS tables that were created during the tuning process.

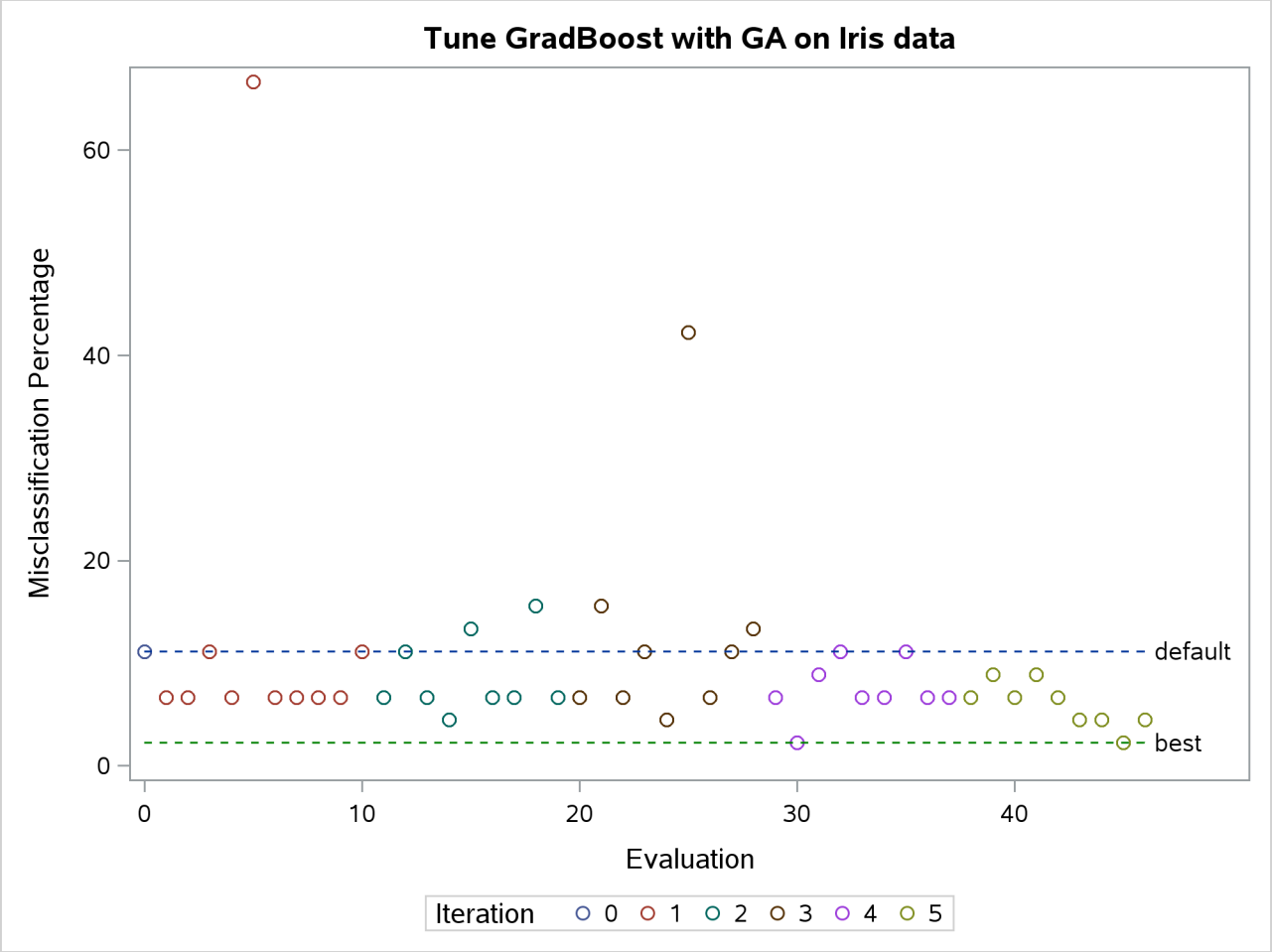
Output 4.5.8 Output CAS Tables

Output CAS Tables

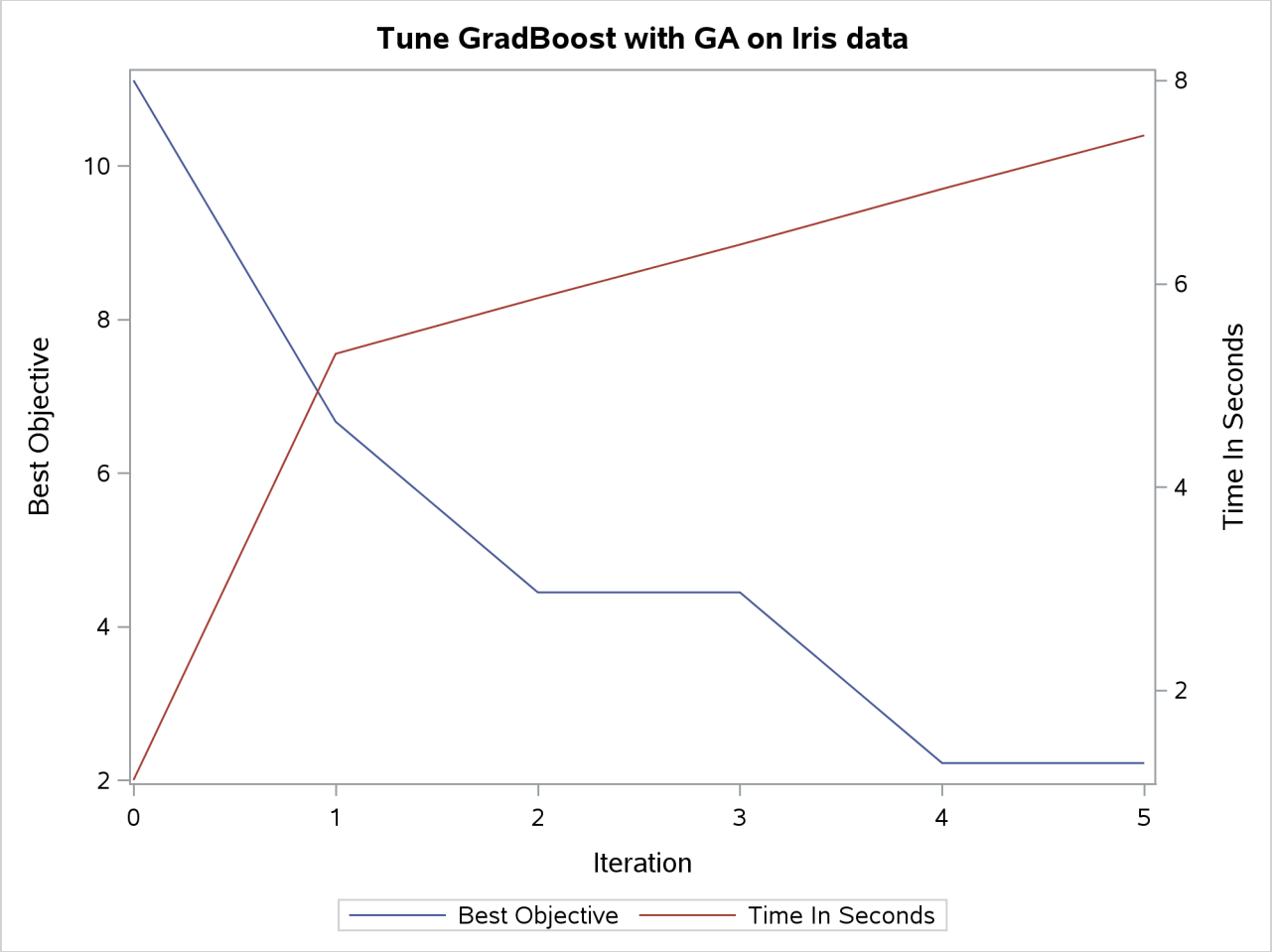
CAS Library	Name	Number of Rows	Number of Columns
YOUR_CAS_LIB	**gradboost_iris_model**	160	19

The first PROC SGPLOT step creates a scatter plot of the objective value (misclassification error in this case) evaluation history by iteration. The second PROC SGPLOT step creates a line plot of the best objective value by iteration and a line plot of the total tuning time by iteration.

Output 4.5.9 Evaluation History Plot



Output 4.5.10 Iteration History Plot



Example 4.6: Tuning the Hyperparameters of a Neural Network

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the **`tuneNeuralNet`** action to automatically tune the hyperparameters of a neural network. The following DATA step loads the iris data set from the Sashelp library into a data table named mycas.iris. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.iris;
   set sashelp.iris;
run;
```

The following PROC CAS code uses the **tuneNeuralNet** action to automatically tune the hyperparameters of a neural network model trained on the iris data table (note that the syntax of the **trainOptions** parameter here is the same as the syntax of the **annTrain** action):

```
proc cas noqueue;
   autotune.tuneNeuralNet /
      trainOptions = {
         table     = {name="iris"},
         inputs    = {"SepalLength",
                      "SepalWidth",
                      "PetalLength",
                      "PetalWidth"
                     },
         target     = "Species",
         nominal    = {"Species"},
         casout     = {name="neuralnet_iris_model",
                       replace=true
                      }
      }
     tunerOptions={seed=54321}
   ;
   ods output TunerResults      = TuneResults(keep=MisclassErr);
   ods output EvaluationHistory = EvalHistory;
   ods output IterationHistory  = IterHistory;
   run;
quit;
```

The following optional code includes two DATA steps and two SGPLOT procedure steps that generate the following two plots from the **tuneNeuralNet** results:

- history of all evaluations (the objective value versus the evaluation number, grouped by iteration)
- iteration history (the best objective value and total tuning time versus the iteration number)

```
data _null_;
   set TuneResults;
   if _n_ = 1 then call symput('default', MisclassErr);
   else if _n_ = 2 then call symput('best', MisclassErr);
run;

data EvalHistory;
   set EvalHistory;
   default = &default.;
   best = &best.;
run;

proc sgplot data=EvalHistory;
   title "Tune Neural Net with GA on Iris data";
   scatter x=Evaluation y=MisclassErr / group=Iteration;
   series x=Evaluation y=default /
      curvelabel="default" lineattrs=(pattern=shortdash);
   series x=Evaluation y=best /
      curvelabel="best" lineattrs=(color=green pattern=shortdash);
run;

proc sgplot data=IterHistory;
   title "Tune Neural Net with GA on Iris data";
   series x=Iteration y=Best_Obj;
   series x=Iteration y=Time_Sec / y2axis;
run;
```

The "Tuner Information" output displays the values of options that the tuner used.

Output 4.6.1 Tuner Information Output

Results from neuralNet.annTrain

Tuner Information	
Model Type	Neural Net
Tuner Objective Function	Misclassification Error Percentage
Search Method	GA
Maximum Evaluations	50
Population Size	10
Maximum Iterations	5
Maximum Tuning Time in Seconds	36000
Validation Type	Single Partition
Validation Partition Fraction	0.30
Log Level	2
Seed	54321
Number of Parallel Evaluations	9
Number of Workers per Subsession	1

The "Tuner Results" output displays the default configuration (Iteration 0) and up to 10 of the best configurations that the tuner found. For each configuration, the hyperparameter values and objective value are provided.

Output 4.6.2 Tuner Results Output

Tuner Results
Default and Best Configurations

Evaluation	Hidden Layers	Neurons in Hidden Layer 1	Neurons in Hidden Layer 2	L1 Regularization	L2 Regularization	Misclassification Error Percentage	Time In Seconds
0	0	0	0	0	0	2.22	0.98
1	0	0	0	0.000359	0.166810	0.00	0.26
5	0	0	0	0.002782	0	0.00	2.22
11	0	0	0	3.5941462E-8	0.000099904	0.00	0.55
12	0	0	0	3.3533588E-8	0.000000911	0.00	0.47
13	0	0	0	0.004089	7.8125345E-8	0.00	0.56
15	1	2	0	0.000708	0.000004367	0.00	0.55
17	0	0	0	0.000020401	0	0.00	0.47
19	0	0	0	0.000022610	0.001328	0.00	0.39
20	0	0	0	0.000301	0.000000142	0.00	0.45
21	0	0	0	0.042055	3.5639784E-8	0.00	0.50

The "Tuner Iteration History" output displays the results of each iteration. For each iteration, the number of evaluations, the best objective value so far, and the tuning time in seconds are provided.

Output 4.6.3 Iteration History Output

Tuner Iteration History

Iteration	Evaluations	Best Objective	Time In Seconds
0	1	2.22	0.98
1	10	0.00	4.41
2	19	0.00	5.00
3	28	0.00	5.53
4	37	0.00	6.06
5	46	0.00	6.61

The "Tuner Evaluation History" output displays all model configurations that the tuner evaluated. For each configuration, the hyperparameter values and objective value are provided.

Output 4.6.4 Evaluation History Output

Tuner History
All Evaluated Configurations

Evaluation	Iteration	Hidden Layers	Neurons in Hidden Layer 1	Neurons in Hidden Layer 2	L1 Regularization	L2 Regularization	Misclassification Error Percentage	Time In Seconds
0	0	0	0	0	0	0	2.22	0.98
1	1	0	0	0	0.000359	0.166810	0.00	0.26
2	1	1	6	0	10.000000	0.021544	66.67	2.05
3	1	2	5	12	1.291550	0.000005895	66.67	2.08
4	1	0	0	0	0	0.000359	2.22	1.88
5	1	0	0	0	0.002782	0	0.00	2.22
6	1	1	11	0	0.000005895	10.000000	66.67	2.44
7	1	0	0	0	0.000000674	1.291550	13.33	2.16
8	1	1	2	0	0.166810	0.000046316	13.33	2.12
9	1	2	1	1	0.021544	0.000000674	33.33	2.34
10	1	0	0	0	0	0	2.22	0.00
11	2	0	0	0	3.5941462E-8	0.000099904	0.00	0.55
12	2	0	0	0	3.3533588E-8	0.000000911	0.00	0.47
13	2	0	0	0	0.004089	7.8125345E-8	0.00	0.56
14	2	0	0	0	0.054370	0.000001161	11.11	0.47
15	2	1	2	0	0.000708	0.000004367	0.00	0.55
16	2	0	0	0	0.570439	0	2.22	0.59
17	2	0	0	0	0.000020401	0	0.00	0.47
18	2	0	0	0	0.081617	10.000000	66.67	0.56
19	2	0	0	0	0.000022610	0.001328	0.00	0.39
20	3	0	0	0	0.000301	0.000000142	0.00	0.45
21	3	0	0	0	0.042055	3.5639784E-8	0.00	0.50
22	3	1	5	0	0.010495	0.000000866	2.22	0.40
23	3	0	0	0	10.000000	0.000000444	66.67	0.35
24	3	0	0	0	0.001929	8.8951198E-9	0.00	0.50
25	3	0	0	0	0	0.010146	2.22	0.53
26	3	0	0	0	0.000002106	0.000017973	0.00	0.52
27	3	0	0	0	0.009146	0.000001436	0.00	0.49
28	3	0	0	0	0.002327	6.4310774E-8	0.00	0.40
29	4	0	0	0	0.002250	6.4676081E-8	0.00	0.42
30	4	0	0	0	0.040669	3.5941347E-8	2.22	0.51
31	4	0	0	0	0.002534	2.9649691E-8	0.00	0.50
32	4	0	0	0	2.047772	0	66.67	0.32
33	4	0	0	0	0.000004769	0.000350	0.00	0.42
34	4	0	0	0	0.001519	0	0.00	0.42
35	4	0	0	0	0.005970	0.000000666	0.00	0.51
36	4	0	0	0	0.033844	0.000014122	0.00	0.52
37	4	0	0	0	0.007138	0.000000908	0.00	0.50
38	5	0	0	0	0.016520	0.000003377	0.00	0.51
39	5	0	0	0	0.004202	0.000000354	0.00	0.51
40	5	0	0	0	0.005079	0.000000230	0.00	0.54
41	5	0	0	0	0.013809	0.000002739	0.00	0.38
42	5	0	0	0	0.004352	0.000000364	0.00	0.51
43	5	0	0	0	0.007629	0.000001017	0.00	0.55
44	5	0	0	0	0.006380	0.000000748	0.00	0.51

Output 4.6.4 *continued*

Tuner History
All Evaluated Configurations

Evaluation	Iteration	Hidden Layers	Neurons in Hidden Layer 1	Neurons in Hidden Layer 2	L1 Regularization	L2 Regularization	Misclassification Error Percentage	Time In Seconds
45	5	0	0	0	0.002648	1.8670238E-8	0.00	0.44
46	5	0	0	0	0.002351	5.07304E-8	0.00	0.50

The "Best Configuration" output displays the values of hyperparameters and the objective value for the best configuration that the tuner found.

Output 4.6.5 Best Configuration Output

Best Configuration	
Evaluation	1
Hidden Layers	0
Neurons in Hidden Layer 1	0
Neurons in Hidden Layer 2	0
L1 Regularization	0.00035928
L2 Regularization	0.16680996
Misclassification Error Percentage	0.00

The "Tuner Summary" output displays statistics about the tuning process.

Output 4.6.6 Tuner Summary Output

Tuner Summary	
Initial Configuration Objective Value	2.2222
Best Configuration Objective Value	0
Worst Configuration Objective Value	66.6667
Initial Configuration Evaluation Time in Seconds	0.9845
Best Configuration Evaluation Time in Seconds	0.2622
Number of Improved Configurations	1
Number of Evaluated Configurations	46
Total Tuning Time in Seconds	7.2274
Parallel Tuning Speedup	3.8869

The "Tuner Task Timing" output displays timing information about the different tasks that were performed during the tuning process.

Output 4.6.7 Tuner Timing Output

Tuner Task Timing

Task	Seconds	Percent
Model Training	18.08	64.35
Model Scoring	6.89	24.53
Total Objective Evaluations	24.97	88.89
Tuner	3.12	11.11
Total CPU Time	28.09	100.00

The "Output CAS Tables" output displays a list of CAS tables that were created during the tuning process.

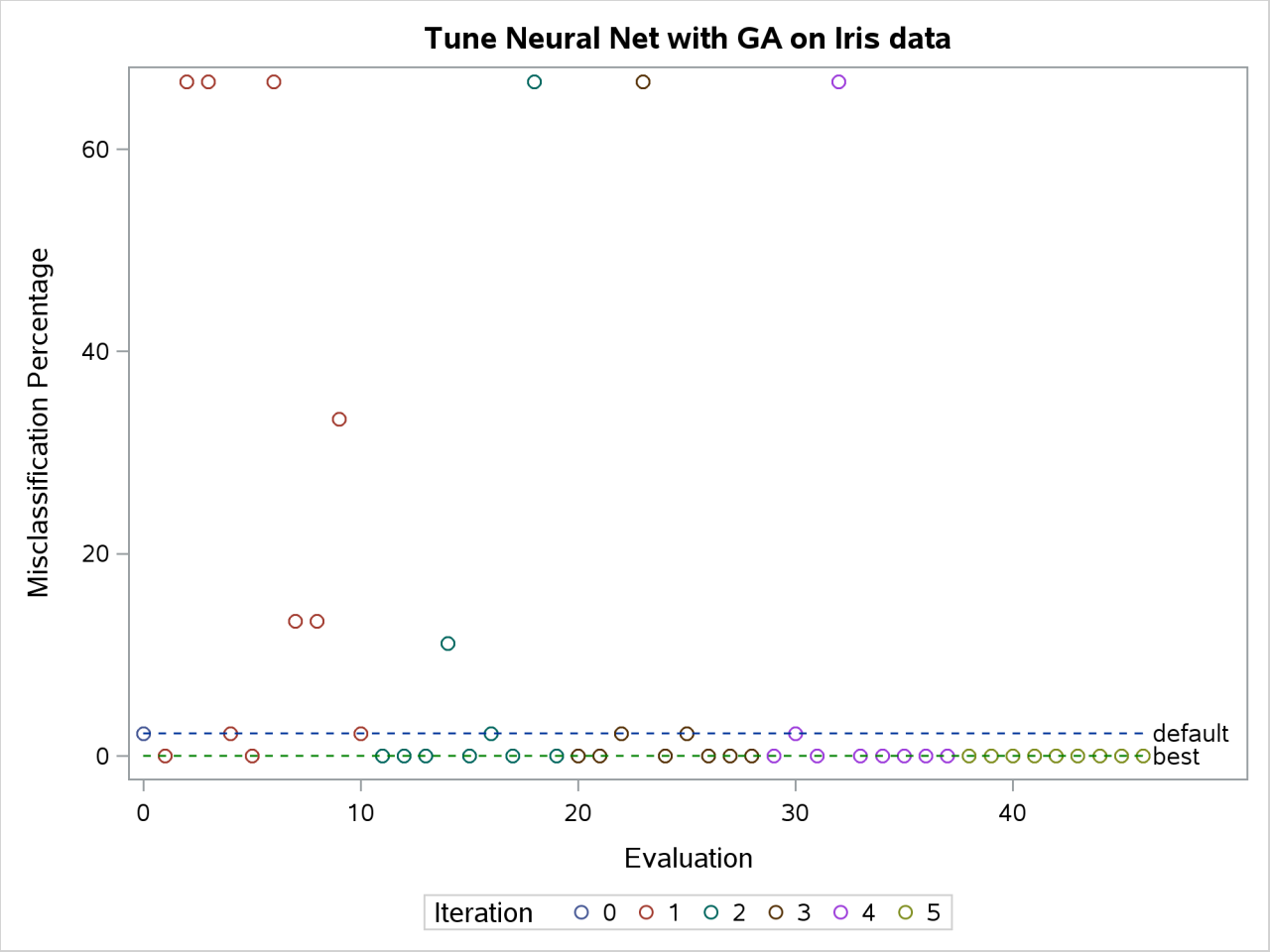
Output 4.6.8 Output CAS Tables

Output CAS Tables

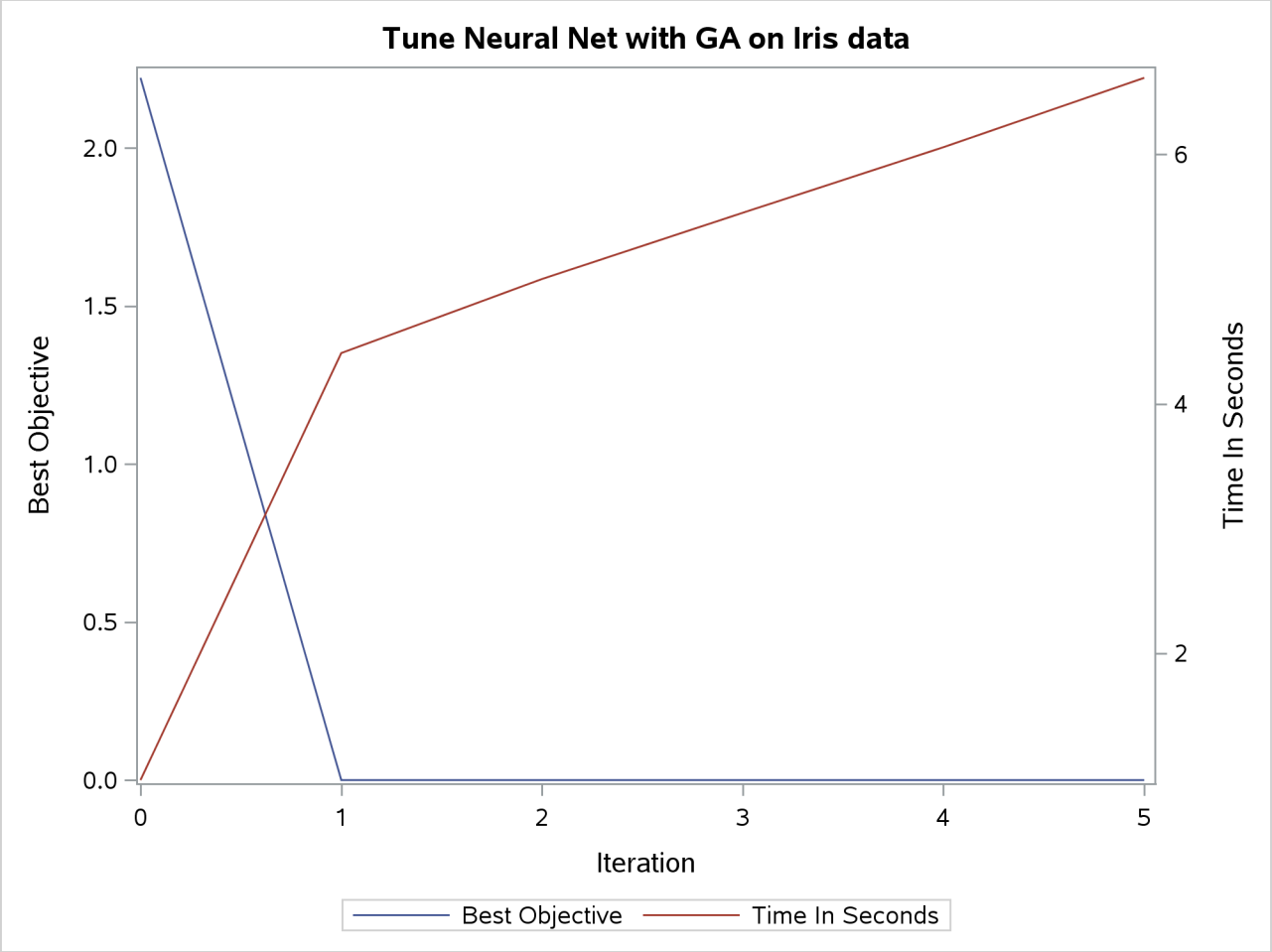
CAS Library	Name	Number of Rows	Number of Columns
YOUR_CAS_LIB	neuralnet_iris_model	15	15

The first PROC SGPLOT step creates a scatter plot of the objective value (misclassification error in this case) evaluation history by iteration. The second PROC SGPLOT step creates a line plot of the best objective value by iteration and a line plot of the total tuning time by iteration.

Output 4.6.9 Evaluation History Plot



Output 4.6.10 Iteration History Plot



Example 4.7: Tuning the Hyperparameters of a Support Vector Machine

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the **tuneSvm** action to automatically tune the hyperparameters of a support vector machine model from observations in German credit benchmark data, which are available in the sampsio.dmagecr data set. This data set contains 1,000 observations, each of which contains an applicant's information, including the applicant's credit rating ("GOOD" or "BAD"). The binary target is named GOOD_BAD. Other input variables are Checking, Duration, History, and so on.

The contents of the sampsio.dmagecr data set are described at `http://support.sas.com/documentation/cdl/en/emgs/59885/HTML/default/a001026918.htm`.

You can load the sampsio.dmagecr data set into your CAS session by specifying your CAS engine libref in the second statement in the following DATA step. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.dmagecr;
   set sampsio.dmagecr;
run;
```

The following PROC CAS code uses the **tuneSvm** action to automatically tune the hyperparameters of a support vector machine model that is trained on the dmagecr data table (note that the syntax of the **trainOptions** parameter here is the same as the syntax of the **svmTrain** action):

```
proc cas noqueue;
   action autotune.tuneSvm /
      trainOptions  = {
         table      = {name="dmagecr"},
         nominals   = {
            "checking",   "history",   "purpose",   "savings",
            "employed",   "marital",   "coapp",     "property",
            "other",      "job",       "housing",   "telephon",
            "foreign",    "good_bad"
         }
         inputs     = {
            "duration",   "amount",    "installp",  "resident",
            "existcr",    "depends",   "age",       "checking",
            "history",    "purpose",   "savings",   "employed",
            "marital",    "coapp",     "property",  "other",
            "job",        "housing",   "telephon",  "foreign"
         }
         target     = "good_bad"
         saveState  = {name='svm_dmagecr_model', replace=true}
      },
      tunerOptions  = {
         searchMethod = "GA",
         popSize      = 10,
         logLevel     = 3,
         objective    = "RASE",
         seed         = 12345
      }
   ;
   ods output TunerResults      = TuneResults(keep=RootASE);
   ods output EvaluationHistory = EvalHistory;
   ods output IterationHistory  = IterHistory;
   run;
quit;
```

The following optional code includes two DATA steps and two SGPLOT procedure steps that generate the following two plots from the **tuneSvm** results:

- history of all evaluations (the objective value versus the evaluation number, grouped by iteration)
- iteration history (the best objective value and total tuning time versus the iteration number)

```
data _null_;
   set TuneResults;
   if _n_ = 1 then call symput('default', RootASE);
   else if _n_ = 2 then call symput('best', RootASE);
run;

data EvalHistory;
   set EvalHistory;
   default = &default.;
   best = &best.;
run;

proc sgplot data=EvalHistory;
   title "Tune SVM with GA on DMAGECR data";
   scatter x=Evaluation y=RootASE / group=Iteration;
   series x=Evaluation y=default /
      curvelabel="default" lineattrs=(pattern=shortdash);
   series x=Evaluation y=best /
      curvelabel="best" lineattrs=(color=green pattern=shortdash);
run;

proc sgplot data=IterHistory;
   title "Tune SVM with GA on DMAGECR data";
   series x=Iteration y=Best_Obj;
   series x=Iteration y=Time_Sec / y2axis;
run;
```

The “Tuner Information” output displays the values of parameters that the tuner used.

Output 4.7.1 Tuner Information Output

Results from svm.svmTrain

Tuner Information	
Model Type	Support Vector Machine
Tuner Objective Function	Root Average Square Error
Search Method	GA
Maximum Evaluations	50
Population Size	10
Maximum Iterations	5
Maximum Tuning Time in Seconds	36000
Validation Type	Single Partition
Validation Partition Fraction	0.30
Log Level	3
Seed	12345
Number of Parallel Evaluations	9
Number of Workers per Subsession	1

The "Tuner Results" output displays the default configuration (Iteration 0) and up to 10 of the best configurations that the tuner found. For each configuration, the hyperparameter values and objective value are provided.

Output 4.7.2 Tuner Results Output

Tuner Results
Default and Best Configurations

Evaluation	Penalty (C)	Polynomial Degree	Root Average Squared Error	Time In Seconds
0	1.000000	1	0.4236	2.64
36	0.150412	1	0.4182	0.75
31	0.161299	1	0.4191	0.71
34	0.212150	1	0.4192	0.78
29	0.893490	1	0.4235	0.81
35	1.001229	1	0.4236	0.67
10	1.000000	1	0.4236	0.00
37	1.698753	1	0.4236	0.78
32	1.131814	1	0.4237	0.86
28	1.876446	1	0.4237	0.86
30	1.864005	1	0.4237	0.81

The "Tuner Iteration History" output displays the results of each iteration. For each iteration, the number of evaluations, the best objective value so far, and the tuning time in seconds are provided.

Output 4.7.3 Iteration History Output

Tuner Iteration History

Iteration	Evaluations	Best Objective	Time In Seconds
0	1	0.42	2.64
1	10	0.42	5.95
2	19	0.42	6.96
3	26	0.42	7.85
4	33	0.42	8.71
5	38	0.42	9.49

The "Tuner Evaluation History" output displays all model configurations that the tuner evaluated. For each configuration, the hyperparameter values and objective value are provided.

Output 4.7.4 Tuner Evaluation History Output

Tuner History
All Evaluated Configurations

Evaluation	Iteration	Penalty (C)	Polynomial Degree	Root Average Squared Error	Time In Seconds
0	0	1.000000	1	0.4236	2.64
1	1	33.333333	1	0.4252	0.75
2	1	66.666667	3	0.4583	3.05
3	1	100.000000	2	0.4580	2.51
4	1	77.777778	3	0.4583	3.11
5	1	1E-10	3	0.5477	3.31
6	1	44.444444	1	0.4252	2.63
7	1	11.111111	1	0.4251	2.77
8	1	55.555556	2	0.4580	3.11
9	1	22.222222	2	0.4580	2.85
10	1	1.000000	1	0.4236	0.00
11	2	4.892068	1	0.4246	0.85
12	2	27.114290	1	0.4252	0.90
13	2	56.567796	2	0.4580	0.99
14	2	1E-10	1	0.5477	0.73
15	2	15.799170	1	0.4251	0.86
16	2	45.317138	2	0.4580	0.98
17	2	89.761582	2	0.4580	0.95
18	2	9.523217	1	0.4249	0.85
19	2	20.634328	2	0.4580	1.01
20	3	11.465910	1	0.4251	0.85
21	3	1.970515	1	0.4239	0.70
22	3	5.528154	1	0.4248	0.78
23	3	4.216458	1	0.4247	0.84
24	3	15.327569	2	0.4580	0.88
25	3	6.311667	1	0.4248	0.85
26	3	12.587619	1	0.4251	0.77
27	4	8.559231	1	0.4249	0.81
28	4	1.876446	1	0.4237	0.86
29	4	0.893490	1	0.4235	0.81
30	4	1.864005	1	0.4237	0.81
31	4	0.161299	1	0.4191	0.71
32	4	1.131814	1	0.4237	0.86
33	4	2.548341	1	0.4242	0.71
34	5	0.212150	1	0.4192	0.78
35	5	1.001229	1	0.4236	0.67
36	5	0.150412	1	0.4182	0.75
37	5	1.698753	1	0.4236	0.78
38	5	4.261586	1	0.4247	0.70

The "Best Configuration" output displays the values of hyperparameters and the objective value for the best configuration that the tuner found.

Output 4.7.5 Best Configuration Output

Best Configuration	
Evaluation	36
Penalty (C)	0.15041246
Degree	1
Root Average Square Error	0.4182194135

The "Tuner Summary" output displays statistics about the tuning process.

Output 4.7.6 Tuner Summary Output

Tuner Summary	
Initial Configuration Objective Value	0.4236
Best Configuration Objective Value	0.4182
Worst Configuration Objective Value	0.5477
Initial Configuration Evaluation Time in Seconds	2.6383
Best Configuration Evaluation Time in Seconds	0.7491
Number of Improved Configurations	2
Number of Evaluated Configurations	38
Total Tuning Time in Seconds	11.9005
Parallel Tuning Speedup	3.4606

The "Tuner Task Timing" output displays timing information about the different tasks that were performed during the tuning process.

Output 4.7.7 Tuner Task Timing Output

Tuner Task Timing		
Task	**Seconds**	**Percent**
Model Training	20.73	50.33
Model Scoring	16.77	40.72
Total Objective Evaluations	37.50	91.05
Tuner	3.68	8.95
Total CPU Time	41.18	100.00

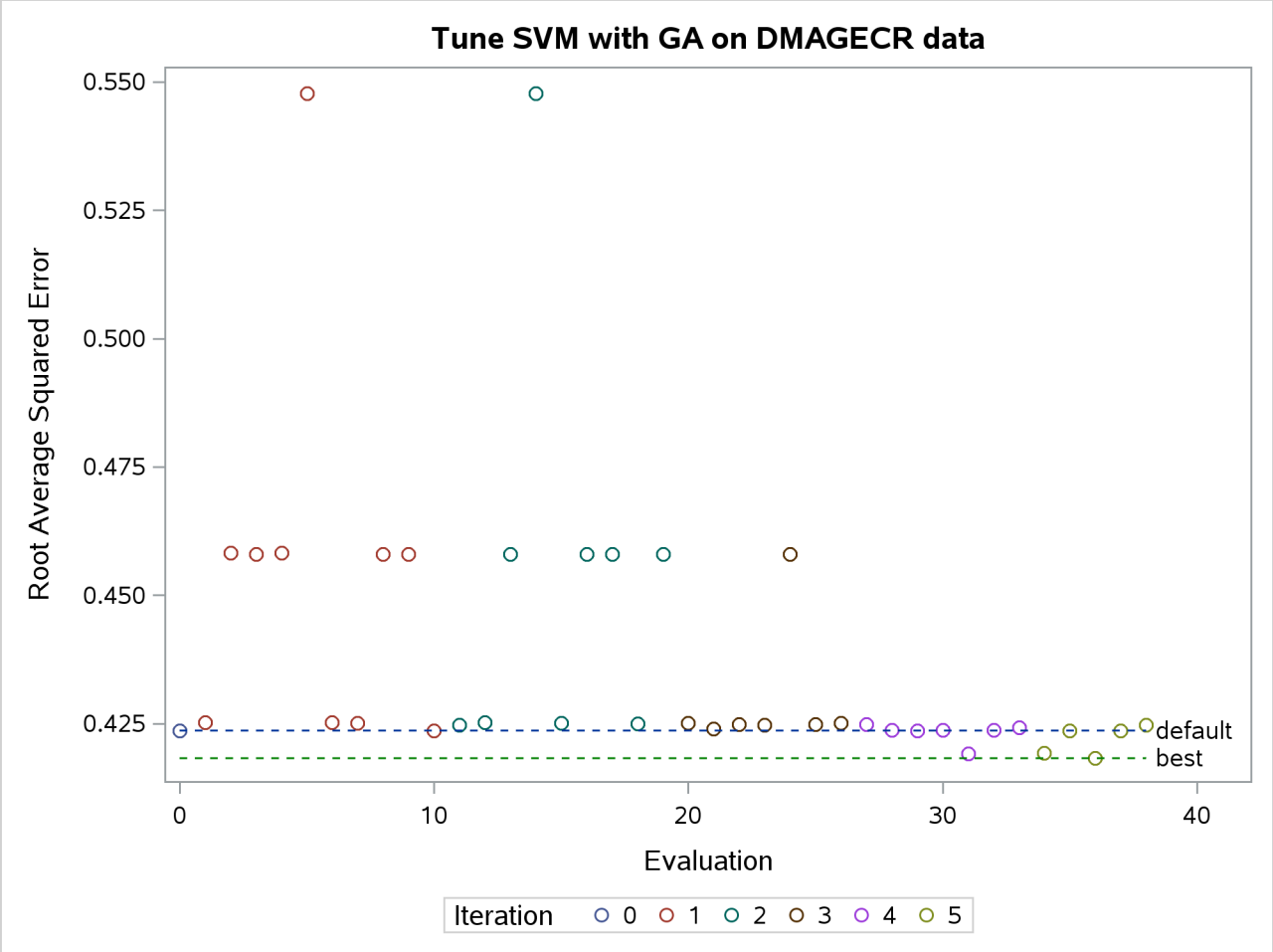
The "Output CAS Tables" output displays a list of CAS tables that were created during the tuning process.

Output 4.7.8 Output CAS Tables

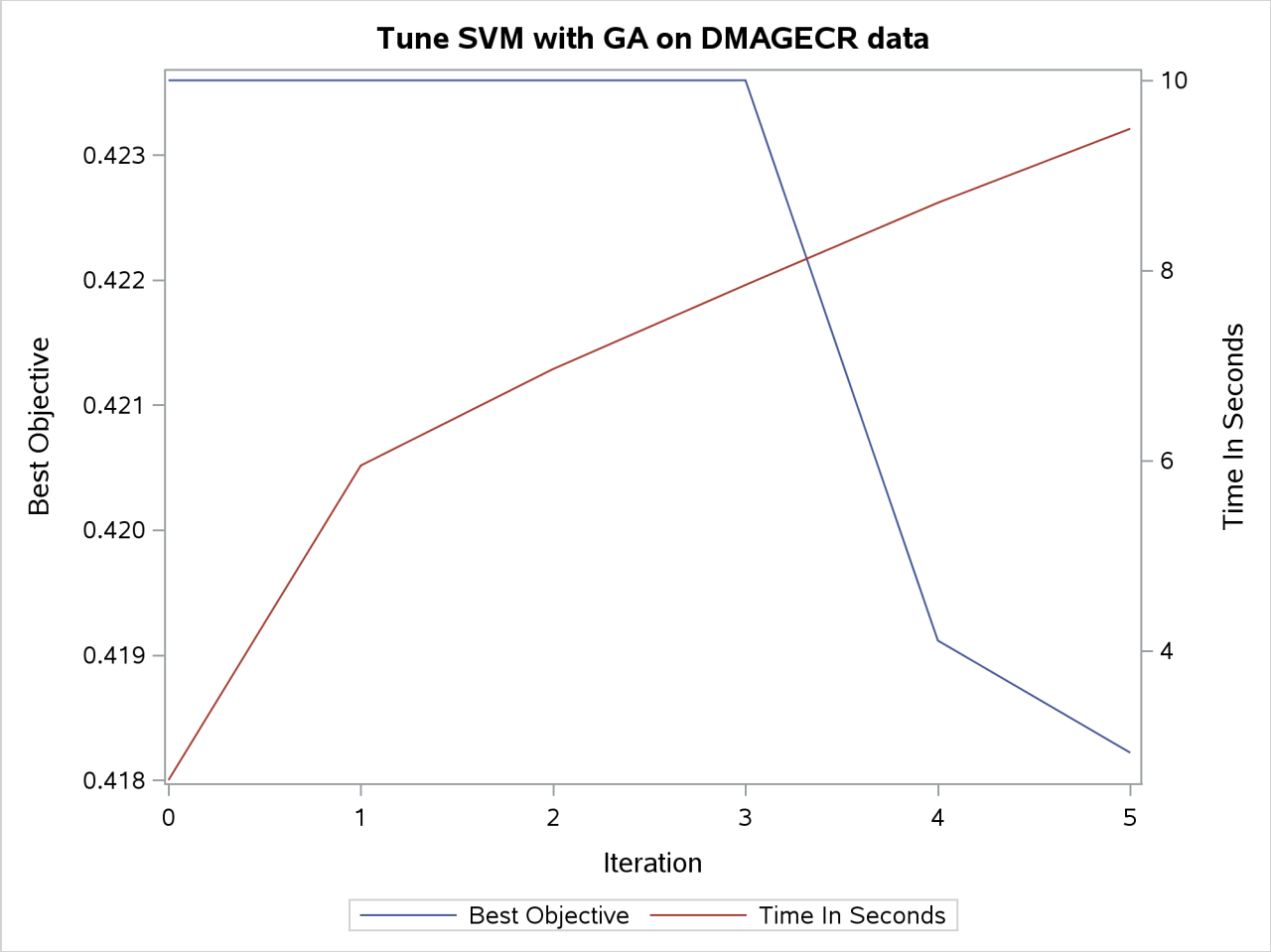
Output CAS Tables			
CAS Library	**Name**	**Number of Rows**	**Number of Columns**
YOUR_CAS_LIB	**svm_dmagecr_model**	1	2

The first PROC SGPLOT step creates a scatter plot of the objective value (root average squared error in this case) evaluation history by iteration. The second PROC SGPLOT step creates a line plot of the best objective value by iteration and a line plot of the total tuning time by iteration.

Output 4.7.9 Evaluation History Plot



Output 4.7.10 Iteration History Plot



Chapter 5
Bayesian Net Classifier Action Set

Contents

Syntax **185**
The bnet Action 186
Details **204**
Independence Tests 204
Variable Selection 205
Structure Learning 205
Parameter Learning 207
Displayed Output 207
Results 208
Example **209**
Example 5.1: German Credit Benchmark Data 209

Bayesian Network Classifier Action Set: Syntax

bnet Action

CASL Syntax

bayesianNetClassifier.bnet <**result**=*results*> <**status**=*rc*> /

alpha={*double-1* <, *double-2*, ...>}

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

bestModel=TRUE | FALSE

code={

casOut={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

comment=TRUE | FALSE,

fmtWdth=*integer*,

iProb=TRUE | FALSE,

indentSize=*integer*,

intoCutPt=*double*,

labelId=*integer*,

lineSize=*integer*,

noTrim=TRUE | FALSE,

pCatAll=TRUE | FALSE,

tabForm=TRUE | FALSE

}

codeGroup="*string*"

diagnostics={

eyecatcher="*string*"

}

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

freq="*string*"

id={*"variable-name-1"* <, *"variable-name-2"*, ...>}

indepTest="ALL" | "CHIGSQUARE" | "CHISQUARE" | "GSQUARE" | "MI"

inputs={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

maxParents=*integer*

miAlpha=*double*

missingInt="IGNORE" | "IMPUTE"

missingNom="IGNORE" | "IMPUTE" | "LEVEL"

nominals={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

numBin=*integer*

outNetwork={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

output={

*required parameter **casOut**={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

copyVars="ALL" | "ALL_MODEL" | "ALL_NUMERIC" | {*"variable-name-1"* <, *"variable-name-2"*, ...>},

role="*string*"

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

parenting={"BESTONE", "BESTSET"}

partByFrac={

seed=*integer*,

test=*double*,

validate=*double*

}

partByVar={

*required parameter **name**="*variable-name*",

test="*string*",

train="*string*",

validate="*string*"

}

```
  preScreening={"ONE", "ZERO"}
  printtarget=TRUE | FALSE
  resident=TRUE | FALSE
  saveState={
      caslib="string",
      name="table-name",
      promote=TRUE | FALSE,
      replace=TRUE | FALSE
  }
  structures={"MB", "NAIVE", "PC", "TAN"}
*required parameter table={
      caslib="string",
      computedOnDemand=TRUE | FALSE,
      computedVars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      computedVarsProgram="string",
      dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
      groupBy={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      groupByMode="NOSORT" | "REDISTRIBUTE",
```

```
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
        parameters},
     *required parameter name="table-name",
      onDemand=TRUE | FALSE,
      orderBy={{
          format="string",
          formattedLength=integer,
          label="string",
         *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      singlePass=TRUE | FALSE,
      vars={{
          format="string",
          formattedLength=integer,
          label="string",
         *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      where="where-expression"
  }
   target="string"
   varSelect={"ONE", "THREE", "TWO", "ZERO"}
;
```

Parameter Descriptions

alpha={*double-1* <, *double-2*, ...>}

specifies the significance level for independence tests by using chi-square or G-square statistics. If you want to choose the best model among several, you can specify up to five numbers, separated by spaces. If you specify multiple numbers but you do not specify the value True for the bestModel parameter, the action uses the first number and ignores the remaining numbers.

Requirement The specified values must be unique.

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

changes the attributes of variables that are used in this action.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

bestModel=TRUE | FALSE

when set to True, selects the best model.

Default FALSE

code={*aircodegen*}

For more information about specifying the code parameter, see the common aircodegen parameter (Appendix A: Common Parameters).

codeGroup="*string*"

Code Group

diagnostics={*_diagnostics*}

eyecatcher="*string*"

specifies a quoted string that will be prefixed to any messages that are associated with this action invocation.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

freq="*string*"

specifies the frequency variable.

id={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the variables to copy to the generated table.

indepTest="ALL" | "CHIGSQUARE" | "CHISQUARE" | "GSQUARE" | "MI"

specifies the method for independence tests.

Default CHIGSQUARE

ALL

uses the chi-square statistic, the G-square statistic, and the normalized mutual information for independence tests. A variable is independent of the target if the p-

values of both the chi-square and the G-square statistics are greater than the value of the alpha parameter and the normalized mutual information is less than the value of the miAlpha parameter.

CHIGSQUARE

uses both the chi-square and G-square statistics for independence tests. A variable is independent of the target if the p-values of both the chi-square and G-square statistics are greater than the value of the alpha parameter.

CHISQUARE

uses the chi-square statistic for independence tests. A variable is independent of the target if the p-value of the statistic is greater than the value of the alpha parameter.

GSQUARE

uses the G-square statistic for independence tests. A variable is independent of the target if the p-value of the statistic is greater than the value of the alpha parameter.

MI

uses the normalized mutual information for independence tests. A variable is independent of the target if the normalized mutual information is less than the value of the miAlpha parameter.

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies variables to use for analysis.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

maxParents=*integer*

specifies the maximum number of parents allowed for each node in the network.

Default 5

Range 1–16

miAlpha=*double*

specifies the significance level for independence tests that use mutual information.

Default 0.05

Range 0–1

missingInt="IGNORE" | "IMPUTE"

specifies how to handle missing values for interval variables.

Default IGNORE

IGNORE

ignores the observations that have missing values in any of the interval variables.

IMPUTE

replaces the missing values in any interval variable by the mean of the variable.

missingNom="IGNORE" | "IMPUTE" | "LEVEL"

specifies how to handle missing values for nominal variables.

Default IGNORE

IGNORE

ignores the observations that have missing values in any of the nominal variables.

IMPUTE

replaces the missing values in any nominal variable by the mode of the variable.

LEVEL

treats the missing values in any nominal variable as a separate level of the variable.

nominals={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies nominal variables to use for analysis.

For more information about specifying the nominals parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

numBin=*integer*

specifies the binning number for interval variables.

Default 5

Range 2–1024

outNetwork={*casouttable*}

specifies the name of the output table for the network structure and the probability distributions.

For more information about specifying the outNetwork parameter, see the common casouttable parameter (Appendix A: Common Parameters).

output={*BnetOutputStatement*}

creates an output table to contain the predicted target values of the input table.

* **casOut**={*casouttable*}

specifies the settings for an output table.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

copyVars="ALL" | "ALL_MODEL" | "ALL_NUMERIC" | {*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies a list of one or more variables to be copied from the input table to the output table. You can alternatively specify the value ALL, ALL_MODEL, or ALL_NUMERIC, which respectively copies all variables, all variables used in the modeling, or all numeric variables from the input table to the output table.

role="*string*"

renames the generated column _ROLE_ in the output data table to the specified role name.

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

parenting={"BESTONE", "BESTSET"}

specifies the structure learning methods. If you want the action to choose between the two methods, you can specify both BESTONE and BESTSET and also specify the value True for the bestModel parameter. If you specify both methods but you do not specify the value True for the bestModel parameter, the action uses the first specified method and ignores the other.

Default BESTSET

BESTONE

uses a greedy approach to determine the parents of each node; that is, for each node, the best candidate is added as a parent of the node in each iteration.

BESTSET

determines the best set of variables among possible candidate sets as the parents of each node; that is, instead of adding one variable in an iteration, the action tests multiple sets of variables together and chooses the best set as the parents of the node.

partByFrac={*partByFracStatement*}

The partByFracStatement value can be one or more of the following:

seed=*integer*

specifies the seed to use in the random number generator that is used for partitioning the data.

Default 0

test=*double*

randomly assigns the specified proportion of observations in the input table to the testing role. The sum of the fractions that are specified in the test and validate parameters must be less than 1.

Range 0–1

validate=*double*

randomly assigns the specified proportion of observations in the input table to the validation role. The sum of the fractions that are specified in the test and validate parameters must be less than 1.

Alias valid

Range 0–1

partByVar={*partByVarStatement*}

Long form partByVar={name="*variable-name*"}

Shortcut form partByVar="*variable-name*"

The partByVarStatement value can be one or more of the following:

* **name**="*variable-name*"

names the variable in the input table whose values are used to assign rows to each observation.

test="*string*"

specifies the formatted value of the variable that is used to assign observations to the testing role.

train="*string*"

specifies the formatted value of the variable that is used to assign observations to the training role. If you do not specify the train parameter, then all observations whose roles are not determined by the test and validate parameters are assigned to training.

validate="*string*"

specifies the formatted value of the variable that is used to assign observations to the validation role.

Alias valid

preScreening={"ONE", "ZERO"}

specifies the initial screening for the input variables. If you want the action to choose the best model with or without prescreening, you can specify {"ZERO","ONE"} or {"ONE","ZERO"} for the parameter and also specify the value True for the bestModel parameter. If you specify both ONE and ZERO but you do not specify the value True for the bestModel parameter, the action uses the first specified value and ignores the other.

Default ONE

Requirement The specified values must be unique.

ONE

uses only the input variables that are dependent on the target.

ZERO

uses all the input variables.

printtarget=TRUE | FALSE

when set to True, generates names for the predicted target variable and the predicted probability variables.

Default FALSE

resident=TRUE | FALSE

Default TRUE

saveState={*casouttable*}

specifies the table in which to save the model for future scoring.

Long form saveState={name="*table-name*"}

Shortcut form saveState="*table-name*"

caslib="*string*"

specifies the name of the caslib to use.

name="*table-name*"

specifies the name to associate with the table.

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

structures={"MB", "NAIVE", "PC", "TAN"}

specifies the network structure types. Together with the maxParents parameter, this parameter determines which network structure the action learns from the training data. If you want the action to choose the best structure among several structures, you can

specify multiple values in any combination, separated by spaces, and also specify the value True for the bestModel parameter. If you specify multiple structures but you do not specify the value True for the bestModel parameter, the first value that you specify is used and the rest are ignored.

Alias structure

Default PC

Requirement The specified values must be unique.

MB

learns the Markov blanket of the target variable. The Markov blanket includes the parents, the children, and the other parents of the children. After learning the Markov blanket, the action further determines the parents of the target, the links from the parents to the children, and the links among the children. When you specify the value MB for the structure parameter, the action learns the Markov blanket regardless of the values of the preScreening and the varSelect parameters.

NAIVE

learns a naive Bayesian network structure (that is, the target has a direct link to each input variable). If you specify the value 1 for maxParents, the structure being trained is a naive Bayesian network. If you specify a value greater than 1 for maxParents, the structure is a Bayesian network-augmented naive Bayesian network.

PC

learns the parent-child Bayesian network structure. PC structure differs from NAIVE structure in that some input variables could be learned as the parents of the target variable. In addition, links from the parents to the children and among the children are also possible in PC.

TAN

learns the tree-augmented naive Bayesian network structure. The TAN structure includes a direct link from the target to each input variable plus a tree structure among the input variables.

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

target="*string*"

specifies the target variable to use for analysis.

varSelect={"ONE", "THREE", "TWO", "ZERO"}

specifies how input variables are selected beyond prescreening. If you specify the value "ONE", "TWO", or "THREE", the action automatically tests each input variable for unconditional independence of the target regardless of the value of the preScreening parameter. If no variables are left at a particular variable selection level, the action rolls back to the previous level. For example, if you specify "THREE" and there are no variables in the Markov blanket of the target, the action uses the variables from the previous level "TWO". If you want to choose the best model among different levels of variable selection, you can specify any combination of values for this parameter and also specify the value True for the bestModel parameter. If you specify multiple values for the varSelect parameter but you do not specify the value True for the bestModel parameter, the action uses the first specified value and ignores the remaining values.

Default ONE

Requirement The specified values must be unique.

ONE

tests each input variable for conditional independence of the target variable given any

other input variable. This type of selection uses only the variables that are conditionally dependent on the target given any other input variable.

THREE

determines the Markov blanket of the target variable and uses only the variables in the Markov blanket.

TWO

tests each input variable further for conditional independence of the target variable given any subset of other input variables. This type of selection uses only the variables that are conditionally dependent on the target given any subset of other input variables.

ZERO

uses all input variables that remain after the initial screening is performed as specified in the PRESCREENING= parameter.

Details

Independence Tests

Both variable selection and structure learning require either independence tests between two variables or conditional independence tests given some other variables. The **bnet** action supports independence tests by using the chi-square statistic, G-square statistic, normalized mutual information, or some combination of them.

Given two nominal variables X and Y (or interval variables after binning) that have levels r and c, respectively, the chi-square statistic is computed as

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

where O_{ij} is an observed frequency in a cell of the contingency table of the two variables and E_{ij} is the expected frequency of the cell. The degrees of freedom for the test is $(r-1)\times(c-1)$. If the p-value of the test statistic is greater than the specified significance level, the two variables are considered to be independent.

Similarly, the G-square statistic is calculated as

$$G^2 = 2\sum_{i=1}^{r}\sum_{j=1}^{c} O_{ij}\ln(\frac{O_{ij}}{E_{ij}})$$

where O_{ij} is an observed frequency, E_ij is the expected frequency, and the degrees of freedom for the test is $(r-1)\times(c-1)$. If the p-value of the test statistic is greater than the specified significance level, the two variables are independent.

The mutual information between X and Y is defined as

$$I(X,Y) = \sum_{x=1}^{r}\sum_{y=1}^{c} p(x,y)\ln(\frac{p(x,y)}{p(x)p(y)})$$

where $p(x,y) = \frac{O_{xy}}{N}$ is the joint distribution function of X and Y; $p(x) = \frac{\sum_{y=1}^{c} O_{xy}}{N}$ and $p(y) = \frac{\sum_{x=1}^{r} O_{xy}}{N}$ are the marginal probability distributions of X and Y, respectively; and N is the total number of observations in the training data.

The mutual information is then normalized to be between 0 and 1 as

$$NI(X,Y) = \sqrt{1 - e^{-2I(X,Y)}}$$

where $I(X, Y)$ is the mutual information between X and Y.

If the value of the normalized mutual information is less than the specified significance level, the two variables are considered to be independent.

To test the conditional independence between two variables X and Y given a subset S of other variables ($X \notin S$ and $Y \notin S$), the corresponding statistics are summed for each value combination of S, and the corresponding degrees of freedom for the chi-square and G-square statistics is $(r-1) \times (c-1) \times q$, where q is the number of value combinations for S.

The following **bnet** action parameters are related to independence tests:

- **indepTest** specifies which test statistic or combination of them to use.
- **alpha** specifies the significance level for the chi-square and the G-square statistics.
- **miAlpha** specifies the significance level for the normalized mutual information.

Variable Selection

A Bayesian network is a graphical model that consists of two parts, $< G, P >$, where G is a directed acyclic graph (DAG) whose nodes correspond to the random variables in U (U is the set of input variables plus the target variable in the **bnet** action and P is a set of local probability distributions, one for each node conditional on each value combination of the parents). The joint probability distribution of U can be factorized into the product of the local probability distributions; that is $p(U) = \Pi_{X \in U} p(X|\pi(X))$, where $\pi(X)$ are the parents of X. It is assumed that the network structure G and the probability distribution P are faithful to each other; that is, every conditional independence in the structure G is also present in P, and vice versa.

Given a target variable T, a Markov blanket of T is defined as a subset of input variables $MB \subseteq U - \{T\}$ such that T is conditionally independent of each of the remaining input variables $X \in U - MB - \{T\}$ given MB. Under the faithful assumption, the Markov blanket of T is unique. According to the definition of the Markov blanket, the probability distribution of T is completely determined by its Markov blanket; therefore, the Markov blanket can be used for variable selection.

The **bnet** action supports two types of variable selection: one by independence tests between each input variable and the target (when PRESCREENING=1) and the other by conditional independence tests between each input variable and the target given any subset of other input variables (when VARSELECT=1, 2, or 3).

The **bnet** action uses specialized data structures to efficiently compute the contingency tables for any variable combination, and it uses dynamic candidate generation to reduce the variable combinations that are false candidates.

Structure Learning

In general, there are two approaches to learning the network structure: one is score-based, and the other is constraint-based. The score-based approach uses a score function to measure how well a structure fits the training data and tries to find the structure that has the best score. The constraint-based approach uses independence tests to determine the edges and the directions.

The **bnet** action uses both score-based and constraint-based approaches to learn the network structure. It uses the BIC (Bayesian information criterion) score, which is defined as

$$\mathrm{BIC}(G, D) = N \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} p(\pi_{ij}) p(X_i = v_{ik} | \pi_{ij}) \ln p(X_i = v_{ik} | \pi_{ij}) - \frac{M}{2} \ln N$$

where G is a network, D is the training data set, N is the number of observations in D, n is the number of variables, X_i is a random variable, r_i is the number of levels for X_i, v_{ik} is the kth value of X_i, q_i is the number of value combinations of X_i's parents, π_{ij} is the jth value combination of X_i's parents, and $M = \sum_{i=1}^{n} (r_i - 1) \times q_i$ is the number of parameters for the probability distributions.

The **bnet** action uses independence tests to determine the edges and the directions as follows. Assume that you have three variables, X, Y and Z, and that it has been determined (using independent tests) that there are edges between X and Z and Y and Z, but no edge between X and Y. If X is conditionally dependent of Y given any subset of variables $S = \{Z\} \bigcup S', S' \subseteq U - \{X, Y, Z\}$, then the direction between X and Z is $X \rightarrow Z$ and the direction between Y and Z is $Y \rightarrow Z$. Notice that using independence tests alone might not be able to orient all edges because some structures are equivalent with respect to conditional independence tests. For example, $X \leftarrow Y \leftarrow Z$, $X \rightarrow Y \rightarrow Z$, and $X \leftarrow Y \rightarrow Z$ belong to the same equivalence class. In these cases, the **bnet** action uses the BIC score to determine the directions of the edges.

The **bnet** action learns different types of network structures: naive Bayesian (NB), tree-augmented naive (TAN), Bayesian network-augmented naive (BAN), parent-child Bayesian network (PC), and Markov blanket (MB). Based on the specified network structure type, it uses different algorithms. For example, if you specify STRUCTURE=TAN, the action uses the maximum spanning tree to learn the tree structure, where the weight for an edge is the mutual information between the two nodes. The **bnet** action uses either the BESTONE or BESTSET value of the PARENTING= option to learn the other network structures (BAN, PC, and MB).

bnet action orders the input variables based on the BIC score with the target. The BIC score of an input variable X with the target is defined as

$$\mathrm{BIC}(X, T) = \max(\mathrm{BIC}(X \rightarrow T), \mathrm{BIC}(T \rightarrow X))$$

where $\mathrm{BIC}(X \rightarrow T)$ is the BIC score when X is the parent of T (ignoring all the remaining variables) and $\mathrm{BIC}(T \rightarrow X)$ is the BIC score when X is the child of T (ignoring all the remaining variables).

The **bnet** action learns the parents of the target first for structures PC and MB. Then it learns the parents of the input variable that has the highest BIC score with the target. It continues learning the parents of the input variable that has the next highest BIC score, and so on. When learning the parents of a node, it first determines the edges by using independence tests. Then it orients the edges by using both independence tests and the BIC score. The **bnet** action uses the BIC score not only for orienting the edges but also for controlling the network complexity, because a complex network that has more parents is penalized in the BIC score.

Both the BESTONE and BESTSET values of the PARENTING= option try to find the local optimum structure for each node. BESTONE adds the best candidate variable to the parents at each iteration, whereas BESTSET tries to pick the best set of variables among the candidate sets.

If you have many input variables, structure learning can be time-consuming, because the number of variable combinations is exponential. Therefore, variable selection is strongly recommended.

Parameter Learning

Parameter learning determines the probability distribution for each node in a network structure. In the **bnet** action, the probability distribution is discrete because the interval variables are binned.

You can use the resulting probability distribution table to score an observation $(x_1, x_2, \ldots, x_{n-1})$ as

$$\begin{aligned} \text{argmax}_c\, p(T = c|x_1, x_2, \ldots, x_{n-1}) &= p(x_1, x_2, \ldots, x_{n-1}|T = c) \times K \\ &= \Pi_i\, p(x_i|\pi(X_i)) \times K \end{aligned}$$

where c is a level of the target variable, $\pi(X_i)$ are the parents of X_i, K is a constant, and $X_n = T$ (target) for convenience.

To estimate the parameters $p(x_i|\pi(X_i))$, the **bnet** action uses an additive smoothing technique,

$$\hat{p}(x_i|\pi(X_i)) = \frac{\text{counts\_of}(xi, pi(X_i)) + \alpha}{\text{counts\_of}(pi(X_i)) + \alpha \times n_i}$$

where α is the smoothing parameter ($\alpha = 0$ corresponds to no smoothing, and the **bnet** action uses $\alpha = 1$) and n_i is the number of possible values for X_i.

The general reason for smoothing is to avoid overfitting the data. The case where the count of some class is 0 is just a particular case of overfitting. The **bnet** action still smooths the probabilities when every class is observed.

Displayed Output

The following sections describe the output that the **bnet** action produces by default. The output is organized into various tables, which are discussed in the order of their appearance.

Model Information

The "Model Information" table contains the initial training settings, such as significance level, structure, and number of bins.

Fit Statistics

The "Fit Statistics" table contains the fit statistics of the Bayesian network.

Number of Observations

The "Number of Observations" table contains the number of observations and the number of observations used.

Variable Level

The "Variable Level" table contains the details of each level of the variables. The columns include the observed target, predicted event, predicted nonevent, and total numbers of events or nonevents for the training data.

Variable Order

The "Variable Order" table contains the order of the input variables based on the BIC score with the target.

Variable Information

The "Variable Information" table contains the variable information such as number of levels, number of missing values, and so on.

Iteration Report

The "Iteration Report" table contains the variable selection results.

Validation Information

The "Validation Information" table contains the validation results. If the PARTITION statement is specified, then the misclassification errors mean misclassification errors in the validation data; if not, then the misclassification errors mean misclassification errors in the training data.

Results

The results that the **bnet** action produces are stored in tables. You can access these tables by their names, which are shown in Table 5.1. You use these table names when you specify the **display** and **outputTables** parameters.

Table 5.1 Results Tables Produced by the **bnet** action

Table Name	Description	Action	Parameter
FitStatistics	Contains the fit statistics of the network	**bnet**	Default
ModelInfo	Contains the initial training settings, such as significance level, structure, and number of bins	**bnet**	Default

Table 5.1 *continued*

Table Name	Description	Action	Parameter
NObs	Contains the number of observations for training, validation and testing, and so on	**bnet**	Default
PredIntoName	Predicted target variable	**bnet**	**printTarget**
PredProbName	Predicted probability variables	**bnet**	**printTarget**
ValidInfo	Contains the validation results	**bnet**	**bestModel**
VarInfo	Contains the variable information such as number of levels, number of missing values, and so on	**bnet**	Default
VarLevel	Contains the details of each level of the variables	**bnet**	Default
VarOrder	Contains the order of the input variables	**bnet**	Default
VarSelect	Contains the variable selection results	**bnet**	Default

Example

Example 5.1: German Credit Benchmark Data

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example trains the model by using German credit benchmark data, which are available in the sampsio.dmagecr data set. This data set contains 1,000 observations, each of which contains an applicant's information, including the applicant's credit rating (GOOD or BAD). The binary target is named GOOD_BAD. Other input variables are Checking, Duration, History, and so on.

The contents of the sampsio.dmagecr data set are described at `http://support.sas.com/documentation/cdl/en/emgs/59885/HTML/default/a001026918.htm`.

You can load the sampsio.dmagecr data set into your CAS session by specifying your CAS engine libref in the first statement in the following DATA step:

```
data mycas.dmagecr;
   set sampsio.dmagecr;
run;
```

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

The following statements use the **bnet** action to learn a Bayesian network classifier on the mycas.dmagecr data table:

```
proc cas;
action bayesianNetClassifier.bnet/
    table="dmagecr"
    structure={"NAIVE" "TAN" "PC" "MB"}
    bestmodel=true
    outnetwork={name="network" replace=1}
    nominals={"checking" "history" "purpose" "savings" "employed"
              "installp" "marital" "coapp" "resident" "property"
              "other" "housing" "existcr" "job"
              "depends" "telephon" "foreign" "good_bad"}
    inputs={"duration" "amount" "age"
            "depends"  "checking" "history" "purpose" "savings"
            "employed" "installp" "marital" "coapp" "resident"
            "property" "other" "job"
            "existcr" "housing" "telephon" "foreign"}
    target="good_bad"
    partbyfrac = {valid = 0.3 test=0 seed=12345}
    outputTables = {names = { FITSTATISTICS = "fit"
                              VALIDINFO = "validinfo"
                              NOBS ="nobs"}}
     ;
run;
quit;
```

The **table** parameter names the input data table to be analyzed. The **structure** parameter lists the network structures to be trained. The **bestmodel=true** parameter specifies that you want the action to select the best network structure. The **outnetwork** parameter names the table that saves the output network structure and probability table. The **nominals** parameter lists the nominal input variables and the target variable to use in the training. The **inputs** parameter lists all the input variables to use in the training. The **target** parameter specifies the variable to be predicted. The **partbyfrac** parameter specifies that 30% of the input data is to be used for validation and the rest for training. The **outputTables** parameter specifies ODS tables to be saved in the CAS library.

The following statements produce Output 5.1.1, which shows the number of observations in the input data, including the fact that 711 observations are used for training and 289 are used for validation.

```
proc print data=mycas.nobs noobs label;
run;
```

Output 5.1.1 Model Selection: Number of Obs

RowId	Description	Value
NREAD	Number of Observations Read	1000
NUSED	Number of Observations Used	1000
NTRAINUSED	Number of Observations Used for Training	691
NVALUSED	Number of Observations Used for Validation	309
NTESTUSED	Number of Observations Used for Testing	0

The following statements produce Output 5.1.2, which shows the fit statistics, including the fact that there are 4 nodes, 3 links between the nodes, and the number of parameters is 23 in the resulting network.

```
proc print data=mycas.fit noobs label;
run;
```

Output 5.1.2 Model Selection: Fit Statistics

RowId	Description	Value
NUMNODES	Number of Nodes	5.00
NUMLINKS	Number of Links	5.00
AVGDEGREE	Average Degree	2.00
MAXNUMPARENTS	Maximum Number of Parents in Network	2.00
NUMPARMS	Number of Parameters	63.00
SCORE	Score	-3446.84
VALIDERRORSRATE	Validation Misclassification Rate	0.28
TESTERRORSRATE	Test Misclassification Rate	.

The following statements produce Output 5.1.3, which shows that the **bnet** action has learned a PC Bayesian network structure. In the structure, Checking is the parent of Good_Bad, and Good_Bad is the parent of all the other input variables.

```
proc print data=mycas.network noobs label;
    var _parentnode_ _childnode_;
    where _type_="STRUCTURE";
run;
```

Output 5.1.3 Model Selection: Best Structure

Parent Node	Child Node
good_bad	checking
good_bad	history
good_bad	amount
duration	amount
good_bad	duration

The following statements produce Output 5.1.3, which shows the validation results for each parameter combination. The PC Bayesian network structure has misclassified 55 observations out of 289 validation observations when MAXPARENTS is greater than or equal to 2 and VARSELECT=1. The naive Bayesian network has 115 misclassification errors. The TAN structure has 116 misclassification errors. The MB

Bayesian network structure has 153 misclassification errors when MAXPARENTS is greater than or equal to 1 and VARSELECT=3.

```
proc print data=mycas.validinfo noobs label;
run;
```

Output 5.1.4 Model Selection: Validation Information

Best Model	Misclassification Rate	Misclassification Errors	Number of Observations for Assessment	Significance Threshold	Prescreening
Yes	0.27508	85	309	0.05	1
	0.27508	85	309	0.05	1
	0.27508	85	309	0.05	1
	0.27508	85	309	0.05	1
	0.27832	86	309	0.05	1
	0.27832	86	309	0.05	1
	0.27832	86	309	0.05	1
	0.27832	86	309	0.05	1
	0.27832	86	309	0.05	1
	0.28479	88	309	0.05	1
	0.28479	88	309	0.05	1
	0.33981	105	309	0.05	1
	0.33981	105	309	0.05	1
	0.33981	105	309	0.05	1
	0.33981	105	309	0.05	1
	0.33981	105	309	0.05	1

Variable Selection	Structure	Parenting Method	Maximum Number of Parents
1	Naive	BestSet	2
1	Naive	BestSet	3
1	Naive	BestSet	4
1	Naive	BestSet	5
1	PC	BestSet	2
1	PC	BestSet	3
1	PC	BestSet	4
1	PC	BestSet	5
1	TAN	BestSet	2
1	PC	BestSet	1
1	Naive	BestSet	1
3	MB	BestSet	1
3	MB	BestSet	2
3	MB	BestSet	3
3	MB	BestSet	4
3	MB	BestSet	5

Chapter 6
BioMedImage Action Set

Contents

Syntax **213**
The buildSurface Action 214
The processBioMedImages Action 218
The segmentBioMedImages Action 229
Details **234**
The buildSurface Action 234
The processBioMedImages Action 235
The segmentBioMedImages Action 235
Examples **236**
Example 6.1: Build a Surface from a 3-D Medical Image for Visualization 236
Example 6.2: Extract a Lung Nodule Region from a 3-D Torso Image and Process It in 2-D 236
Example 6.3: Segment a Lung Nodule Region in a 3-D Image 236
References **236**

BioMedImage Action Set: Syntax

buildSurface Action

CASL Syntax

bioMedImage.buildSurface <**result**=*results*> <**status**=*rc*> /

*__required parameter__ **images**={

bioMedDimension="*variable-name*",

bioMedId="*variable-name*",

dimension="*variable-name*",

id="*variable-name*",

image="*variable-name*",

imageFormat="*variable-name*",

label="*variable-name*",

orientation="*variable-name*",

path="*variable-name*",

position="*variable-name*",

resolution="*variable-name*",

size="*variable-name*",

sliceIndex="*variable-name*",

spacing="*variable-name*",

*required parameter **table**={

caslib="*string*"

computedOnDemand=TRUE | FALSE

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

computedVarsProgram="*string*"

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

*required parameter **name**="*table-name*"

onDemand=TRUE | FALSE

singlePass=TRUE | FALSE

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

where="*where-expression*"

},

type="*variable-name*"

}

intensities={*double-1* <, *double-2*, ...>}

***required parameter outputFaces**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

***required parameter** **outputVertices**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

smoothing={

iterations=*integer*,

relaxationFactor=*double*

}

thresholds={{

high=*double*,

low=*double*

}, {...}}

;

Parameter Descriptions

* **images**={*biomedimageTableIn*}

specifies the parameter list of input image table.

For more information about specifying the images parameter, see the common biomedimageTableIn parameter (Appendix A: Common Parameters).

intensities={*double-1* <, *double-2*, ...>}

specifies the intensity values of desired regions.

* **outputFaces**={*casouttable*}

specifies the output table containing surface faces.

For more information about specifying the outputFaces parameter, see the common casouttable parameter (Appendix A: Common Parameters).

* **outputVertices**={*casouttable*}

specifies the output table containing surface vertices.

For more information about specifying the outputVertices parameter, see the common casouttable parameter (Appendix A: Common Parameters).

smoothing={*binarySmoothingParms*}

specifies the parameters for smoothing.

iterations=*integer*

specifies the maximum number of smoothing iterations.

Default 0

Minimum value 1

relaxationFactor=*double*

specifies the degree of smoothing in each iteration.

Default 1

Range 0–1

thresholds={{*buildSurfaceThresholds-1*} <, {*buildSurfaceThresholds-2*}, ...>}

specifies the threshold values of desired regions.

The buildSurfaceThresholds value can be one or more of the following:

high=*double*

specifies the high threshold.

Default MACBIG

low=*double*

specifies the low threshold.

Default -MACBIG

BioMedImage Action Set: Syntax

processBioMedImages Action

CASL Syntax

bioMedImage.processBioMedImages <**result**=*results*> <**status**=*rc*> /

addColumns={"CHANNELCOUNT", "CHANNELTYPE", "DEPTH", "HEIGHT", "ORIENTATION", "POSITION", "SPACING", "SPAN", "WIDTH"}

***required parameter** **casOut**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

decode=TRUE | FALSE

***required parameter** **images**={

bioMedDimension="*variable-name*",

bioMedId="*variable-name*",

dimension="*variable-name*",

id="*variable-name*",

image="*variable-name*",

imageFormat="*variable-name*",

label="*variable-name*",

orientation="*variable-name*",

path="*variable-name*",

position="*variable-name*",

resolution="*variable-name*",

size="*variable-name*",

sliceIndex="*variable-name*",

spacing="*variable-name*",

*required parameter **table**={

caslib="*string*"

computedOnDemand=TRUE | FALSE

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

computedVarsProgram="*string*"

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

*required parameter **name**="*table-name*"

onDemand=TRUE | FALSE

singlePass=TRUE | FALSE

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

```
            }, {...}}
            where="where-expression"
        },
        type="variable-name"
    }
    *required parameter steps={{
        stepParameters={stepType="CROP" | "EXPORT" | "GRADIENT_MAGNITUDE" |
          "IMPORT" | "RESAMPLE" | "RESCALE" | "SMOOTH" | "THRESHOLD", stepType-
          specific-parameters}
    }, {...}}
;
```

Parameter Descriptions

addColumns={"CHANNELCOUNT", "CHANNELTYPE", "DEPTH", "HEIGHT", "ORIENTATION", "POSITION", "SPACING", "SPAN", "WIDTH"}

specifies additional metadata columns to be added to the output table.

Range 1–9 values

Requirement The specified values must be unique.

* **casOut**={*casouttable*}

specifies the output table containing processed images.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of columns to copy from the input table to the output table.

Alias copyVar

decode=TRUE | FALSE

when set to True, writes decompressed images and metadata to the output table.

Alias decompress

Default FALSE

* **images**={*biomedimageTableIn*}

specifies the parameter list of input image table.

For more information about specifying the images parameter, see the common biomedimageTableIn parameter (Appendix A: Common Parameters).

* **steps**={{*processStepParms-1*} <, {*processStepParms-2*}, ...>}

specifies the sequence of processing steps and their parameters.

Minimum items 1

stepParameters={**stepType**="CROP" | "EXPORT" | "GRADIENT_MAGNITUDE" | "IMPORT" | "RESAMPLE" | "RESCALE" | "SMOOTH" | "THRESHOLD", *stepType-specific-parameters*}

Alias functionOptions

The value that you specify for stepType determines the other parameters that apply.

Parameters for cropType="BASIC"

imageSize={*64-bit-integer-1* <, *64-bit-integer-2*, ...>}

specifies the size of an image in terms of number of pixels.

pixelIndex={*64-bit-integer-1* <, *64-bit-integer-2*, ...>}

specifies the index of a pixel in an image.

Parameters for gradientType="BASIC"

useSpacing=TRUE | FALSE

when set to True, uses image spacing in calculations.

Default TRUE

Parameters for resampleType="SIZE"

depth=*integer*

specifies the image depth.

Default 0

Minimum value 0

height=*integer*

specifies the image height.

Default 0

Minimum value 0

width=*integer*

specifies the image width.

Default 0

Minimum value 0

Parameters for resampleType="SPACING"

depth=*double*

specifies the pixel depth.

Default 0

Minimum value 0

height=*double*

specifies the pixel height.

Default 0

Minimum value 0

width=*double*

specifies the pixel width.

Default 0

Minimum value 0

Parameters for rescaleType="CHANNELTYPE_32F"

max=*double*

specifies the maximum value.

Default 1e+37

Range 2–1e+37

min=*double*

specifies the minimum value.

Default 0

Range 0–1e+36

Parameters for rescaleType="CHANNELTYPE_64F"

max=*double*

specifies the maximum value.

Default 1e+307

Range 2–1e+307

min=*double*

specifies the minimum value.

Default 0

Range 0–1e+306

Parameters for rescaleType="CHANNELTYPE_16S"

max=*64-bit-integer*

specifies the maximum value.

Default 32767

Range -32766–32767

min=*64-bit-integer*

specifies the minimum value.

Default -32767

Range -32767–32766

Parameters for rescaleType="CHANNELTYPE_32S"

max=*64-bit-integer*

specifies the maximum value.

Default 2147483647

Range -2147483646–2147483647

min=*64-bit-integer*

specifies the minimum value.

Default -2147483647

Range -2147483647–2147483646

Parameters for rescaleType="CHANNELTYPE_64S"

max=*64-bit-integer*

specifies the maximum value.

Default 2.3058430092137e+18

Range 1–2.3058430092137e+18

min=*64-bit-integer*

specifies the minimum value.

Default 0

Range 0–2305843009213693951

Parameters for rescaleType="CHANNELTYPE_8S"

max=*64-bit-integer*

specifies the maximum value.

Default 127

Range -126–127

min=*64-bit-integer*

specifies the minimum value.

Default -127

Range -127–126

Parameters for rescaleType="CHANNELTYPE_16U"

max=*64-bit-integer*

specifies the maximum value.

Default 65535

Range 1–65535

min=*64-bit-integer*

specifies the minimum value.

Default 0

Range 0–65534

Parameters for rescaleType="CHANNELTYPE_32U"

max=*64-bit-integer*

specifies the maximum value.

Default 4294967295

Range 1–4294967295

min=*64-bit-integer*

specifies the minimum value.

Default 0

Range 0–4294967294

Parameters for rescaleType="CHANNELTYPE_64U"

max=*64-bit-integer*

specifies the maximum value.

Default 2.3058430092137e+18

Range 1–2.3058430092137e+18

min=*64-bit-integer*

specifies the minimum value.

Default 0

Range 0–2305843009213693951

Parameters for rescaleType="CHANNELTYPE_8U"

max=*64-bit-integer*

specifies the maximum value.

Default 255

Range 1–255

min=*64-bit-integer*

specifies the minimum value.

Default 0

Range 0–254

Parameters for smoothType="GAUSSIAN"

kernelWidth=*integer*

specifies the width of the Gaussian kernel.

Default 32

Minimum value 1

variance=*double*

specifies the variance of the Gaussian function.

Default 0

Minimum value 0

Parameters for smoothType="GRADIENT"

conductance=*double*

specifies the conductance for the algorithm to use.

Default 3

iterations=*integer*

specifies the number of iterations for the algorithm to use.

Default 1

Minimum value 1

timeStep=*double*

specifies the time step for the algorithm to use.

Default 0.0625

Minimum value 1e-06

Parameters for stepType="CROP"

cropParameters={**cropType**="BASIC", *cropType-specific-parameters*}

The value that you specify for cropType determines the other parameters that apply.

Parameters for stepType="EXPORT"

encodeType="*string*"

specifies the type of encoding to use.

Default "png"

Parameters for stepType="GRADIENT_MAGNITUDE"

gradientMagnitudeParameters={**gradientType**="BASIC", *gradientType-specific-parameters*}

The value that you specify for gradientType determines the other parameters that apply.

Parameters for stepType="IMPORT"

targetDimension=*integer*

specifies the target dimension.

Default 2

Range 2–3

Parameters for stepType="RESAMPLE"

interpolationType="LINEAR" | "NEAREST"

specifies the interpolation algorithm.

Default LINEAR

LINEAR

specifies the linear interpolation algorithm.

NEAREST

specifies the nearest neighbor interpolation algorithm.

resampleParameters={**resampleType**="**SIZE**" | "**SPACING**", *resampleType-specific-parameters*}

The value that you specify for resampleType determines the other parameters that apply.

Parameters for stepType="RESCALE"

rescaleParameters={**rescaleType**="CHANNELTYPE_32F" | "CHANNELTYPE_64F" | "CHANNELTYPE_16S" | "CHANNELTYPE_32S" | "CHANNELTYPE_64S" | "CHANNELTYPE_8S" | "CHANNELTYPE_16U" | "CHANNELTYPE_32U" | "CHANNELTYPE_64U" | "CHANNELTYPE_8U", *rescaleType-specific-parameters*}

The value that you specify for rescaleType determines the other parameters that apply.

Parameters for stepType="SMOOTH"

smoothParameters={**smoothType**="GAUSSIAN" | "GRADIENT", *smoothType-specific-parameters*}

The value that you specify for smoothType determines the other parameters that apply.

Parameters for stepType="THRESHOLD"

thresholdParameters={**thresholdType**="BASIC" | "OTSU", *thresholdType-specific-parameters*}

The value that you specify for thresholdType determines the other parameters that apply.

Parameters for thresholdType="BASIC"

high=*double*

specifies the high threshold.

Default MACBIG

low=*double*

specifies the low threshold.

Default -MACBIG

Parameters for thresholdType="OTSU"

bins=*integer*

specifies the number of histogram bins to use in the algorithm.

Default 128

Minimum value 2

regions=*integer*

specifies the number of regions in the output images.

Default 2

Minimum value 2

BioMedImage Action Set: Syntax

segmentBioMedImages Action

CASL Syntax

bioMedImage.segmentBioMedImages <**result**=*results*> <**status**=*rc*> /

addColumns={"CHANNELCOUNT", "CHANNELTYPE", "DEPTH", "HEIGHT", "ORIENTATION", "POSITION", "SPACING", "SPAN", "WIDTH"}

***required parameter** **casOut**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

decode=TRUE | FALSE

***required parameter** **images**={

bioMedDimension="*variable-name*",

bioMedId="*variable-name*",

dimension="*variable-name*",

id="*variable-name*",

image="*variable-name*",

imageFormat="*variable-name*",

label="*variable-name*",

orientation="*variable-name*",

path="*variable-name*",

position="*variable-name*",

resolution="*variable-name*",

size="*variable-name*",

sliceIndex="*variable-name*",

spacing="*variable-name*",

*required parameter **table**={

caslib="*string*"

computedOnDemand=TRUE | FALSE

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

computedVarsProgram="*string*"

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

*required parameter **name**="*table-name*"

onDemand=TRUE | FALSE

singlePass=TRUE | FALSE

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

```
            }, {...}}
            where="where-expression"
        },
        type="variable-name"
    }
    method={name="KMEANS" | "REGIONGROWING" | "WATERSHED", name-specific-
      parameters}
;
```

Parameter Descriptions

addColumns={"CHANNELCOUNT", "CHANNELTYPE", "DEPTH", "HEIGHT", "ORIENTATION", "POSITION", "SPACING", "SPAN", "WIDTH"}

specifies additional metadata columns to be added to the output table.

Range 1–9 values

Requirement The specified values must be unique.

* **casOut**={*casouttable*}

specifies the output table containing processed images.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of columns to copy from the input table to the output table.

Alias copyVar

decode=TRUE | FALSE

when set to True, writes decompressed images and metadata to the output table.

Alias decompress

Default FALSE

* **images**={*biomedimageTableIn*}

specifies the parameter list of input image table.

For more information about specifying the images parameter, see the common biomedimageTableIn parameter (Appendix A: Common Parameters).

method={**name**="KMEANS" | "REGIONGROWING" | "WATERSHED", *name-specific-parameters*}

specifies the segmentation method.

The value that you specify for name determines the other parameters that apply.

Parameters for kmeansType="SEEDED"

* **intensities**={*double-1* <, *double-2*, ...>}
  specifies the mean image intensities to use to initialize the k-means algorithm.

Parameters for kmeansType="BASIC"

nClasses=*64-bit-integer*
specifies the number of segmented regions to use.

Alias	regions
Default	2
Minimum value	2

Parameters for name="KMEANS"

kmeansParameters={**kmeansType**="SEEDED" | "BASIC", *kmeansType-specific-parameters*}
The value that you specify for kmeansType determines the other parameters that apply.

Parameters for name="REGIONGROWING"

regionGrowingParameters={**regionGrowingType**="BASIC", *regionGrowingType-specific-parameters*}
The value that you specify for regionGrowingType determines the other parameters that apply.

Parameters for name="WATERSHED"

watershedParameters={**watershedType**="BASIC", *watershedType-specific-parameters*}
The value that you specify for watershedType determines the other parameters that apply.

Parameters for regionGrowingType="BASIC"

regionIntensity=*double*
specifies the region intensity.

Default	255

seeds={{*list-1*} <,{*list-2*}, ...>}

specifies the seeds for the region-growing algorithm.

thresholds={*thresholdPair*}

specifies the thresholds for the region-growing algorithm.

high=*double*

specifies the high threshold.

Default MACBIG

low=*double*

specifies the low threshold.

Default -MACBIG

Parameters for watershedType="BASIC"

level=*double*

specifies the depth of the metaphorical flooding of the image.

Default 0.4

Range 0–1

threshold=*double*

specifies the minimum pixel intensity value for the algorithm to use.

Default 0.004

Range 0–1

Details

The **bioMedImage** action set offers basic and advanced biomedical image processing capabilities and enables biomedical images to be used as a basic data type in SAS Viya.

The buildSurface Action

The **buildSurface** action generates surfaces of desired regions in three-dimensional (3-D) images. The generated surfaces can be used for interactive visualization and exploration of anatomical structures within the images. The primary inputs to the action are a CAS table that contains the images and a list of pairs of image intensity values that specify the low and high thresholds of each desired region. An optional parameter named **smoothing** can be specified to direct the action to produce smoothed surfaces. The action generates one surface for each image and each threshold pair. The action assigns a unique identifier to each surface and represents each surface as lists of vertex coordinates and triangles in the output CAS tables that are specified in the **outputVertices** and **outputFaces** parameters, respectively. The action also produces a results table that contains the correspondence between the surface identifiers, images, and threshold pairs.

The table specified in the **outputVertices** parameter has the following columns:

surfaceId	An integer that uniquely identifies each surface
_id_	An integer that uniquely identifies each vertex of a surface
_x_	The X coordinate of a vertex
_y_	The Y coordinate of a vertex
_z_	The Z coordinate of a vertex

The table specified in the **outputFaces** parameter has the following columns:

surfaceId	An integer that uniquely identifies each surface
_v1_	The identifier of the first vertex of a triangle
_v2_	The identifier of the second vertex of a triangle
_v3_	The identifier of the third vertex of a triangle

The results table has the following columns:

Surface Identifier	An integer that uniquely identifies each surface
Image Identifier	An integer that uniquely identifies each image
Low Threshold	The low threshold in a threshold pair
High Threshold	The high threshold in a threshold pair
Number of Vertices	The number of vertices in a surface
Number of Faces	The number of triangles in a surface

The processBioMedImages Action

The **processBioMedImages** action performs basic processing of two-dimensional (2-D) and three-dimensional (3-D) input images to generate a table of processed images. The primary inputs to the action are a CAS table that contains the input images and a sequence of steps that are applied to each image in the input. Each step in the sequence is a parameter list that can specify the image processing algorithm and parameters to be used in that step. The action has two other notable parameters. The **addColumns** parameter enables you to add metadata columns to the output table (like the **loadImages** action parameter of the same name), and the **copyVars** parameter facilitates the copying of columns from the input table to the output table. Note that the copying of certain metadata columns is not supported in order to protect the integrity of the output table. See the action syntax for brief descriptions of the image processing algorithms that the **processBioMedImages** action supports. More information about these algorithms can be found elsewhere (Johnson et al. 2018).

A remarkable feature of this action is its ability to convert grayscale photographic images (such as an image loaded from a .bmp file) into biomedical image format and vice versa, in its IMPORT and EXPORT steps (which you specify in the **stepType** parameter). Specifically, when an IMPORT step is used at the beginning of the step sequence, the action performs an in-memory conversion of any grayscale photographic images in the input table to a biomedical image format. If there are any additional steps, the action then processes the converted images along with any biomedical images in the input table. Conversely, when an EXPORT step is used at the end of the step sequence, all output images will be in a photographic format that is accepted by other actions, such as the ones in the **image** or **deepLearn** action set. Thus, the **processBioMedImages** action serves as a bridge between the **bioMedImage** action set and action sets that process only photographic images. Note that the EXPORT step processes 3-D images by splitting them into 2-D slices and inserting each slice in a separate row of the output table. If you want, the exported slices can be processed by an image processing action outside the **bioMedImage** action set, and the processed slices can be reassembled into 3-D images that preserve their original order by using the IMPORT step. To facilitate this reassembly of slices, the output table has the following additional columns whenever an EXPORT step is used in the action:

bioMedId	The identifier of the biomedical image from which an output image was derived
bioMedDimension	The dimension of the biomedical image from which an output image was derived
sliceIndex	The index of the biomedical image slice from which an output image was derived

Apart from these columns, the names and contents of the various columns in the output table match those of the output table in the **loadImages** action in the **image** action set.

The segmentBioMedImages Action

The **segmentBioMedImages** action performs segmentation of two-dimensional (2-D) and three-dimensional (3-D) biomedical images. Segmentation is the process of labeling pixels that belong to tissues of interest, a task that is fundamental to most biomedical image analytics pipelines. The primary inputs to the action are a CAS table that contains the input images, and the type and parameters of the selected segmentation method. The output of the action is a table of segmented images. The names and contents of the various columns in the output table match those of the output table in the **loadImages** action in the **image** action set. Note that, although the THRESHOLD step of the **processBioMedImages** action (which you specify in the **stepType** parameter) might be sufficient for simple segmentation tasks, the **segmentBioMedImages** action

offers advanced methods that can assist with more complex scenarios. You can specify the following values for the **name** subparameter of the **method** parameter to choose the segmentation algorithm:

KMEANS	specifies an algorithm that achieves segmentation by performing *k*-means clustering on the image intensity values.
REGION_GROWING	specifies an algorithm that uses the same region value to label all pixels that are connected to a set of seed pixels, where connectivity is determined by the parameters of the algorithm.
WATERSHED	specifies an algorithm that treats an image like a topographic map, where the intensity of a pixel at a point on the map represents the elevation at that point. Segmentation is achieved by "flooding" the map and using the same region value to label all pixels that lie within a single "catchment basin." The primary parameter of this algorithm is **level**, which controls the water depth in the metaphorical flooding.

See the action syntax for brief descriptions of parameters each of these algorithms. More information about these algorithms can be found elsewhere (Johnson et al. 2018). Note that the **segmentBioMedImages** action, like the **processBioMedImages** action, supports the **addColumns** and **copyVars** parameters.

Examples

Example 6.1: Build a Surface from a 3-D Medical Image for Visualization

This example is not available for the SAS language.

Example 6.2: Extract a Lung Nodule Region from a 3-D Torso Image and Process It in 2-D

This example is not available for the SAS language.

Example 6.3: Segment a Lung Nodule Region in a 3-D Image

This example is not available for the SAS language.

References

Armato, S. G., III, Drukker, K., Li, F., Hadjiiski, L., Tourassi, G. D., Engelmann, R. M., Giger, M. L., Redmond, G., Farahani, K., Kirby, J. S., and Clarke, L. P. (2016). "LUNGx Challenge for Computerized Lung Nodule Classification." *Journal of Medical Imaging* 3:044506–1–044506–9.

Armato, S. G., III, Hadjiiski, L., Tourassi, G. D., Drukker, K., Giger, M. L., Li, F., Redmond, G., Farahani, K., Kirby, J. S., and Clarke, L. P. (2015a). "Guest Editorial: LUNGx Challenge for Computerized Lung Nodule Classification: Reflections and Lessons Learned." *Journal of Medical Imaging* 2:020103–1–020103–5.

Armato, S. G., III, Hadjiiski, L., Tourassi, G. D., Drukker, K., Giger, M. L., Li, F., Redmond, G., Farahani, K., Kirby, J. S., and Clarke, L. P. (2015b). "SPIE-AAPM-NCI Lung Nodule Classification Challenge Dataset. The Cancer Imaging Archive." http://doi.org/10.7937/K9/TCIA.2015.UZLSU3FL.

Clark, K., Vendt, B., Smith, K., Freymann, J., Kirby, J., Koppel, P., Moore, S., Phillips, S., Maffitt, D., Pringle, M., Tarbox, L., and Prior, F. (2013). "The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository." *Journal of Digital Imaging* 26:1045–1057.

Johnson, H. J., McCormick, M. M., Ibáñez, L., and Insight Software Consortium (2018). *The ITK Software Guide, Book 2: Design and Functionality*. 4th ed. Clifton Park, NY: Kitware Inc.

Ramachandran, P., and Varoquaux, G. (2011). "Mayavi: 3D Visualization of Scientific Data." *IEEE Computing in Science and Engineering* 13:40–51.

Chapter 7
Boolean Rule Action Set

Contents

Syntax **239**
The brScore Action 240
The brTrain Action 250
Details **272**
About Boolean Rules 272
How Are the Rules Extracted? 272
How Do You Apply Boolean Rules to New Data? 273
Example **273**
Example 7.1: Deriving Boolean Rules from a Term-by-Document Frequency Matrix . 273

Boolean Rule Action Set: Syntax

brScore Action

CASL Syntax

boolRule.brScore <**result**=*results*> <**status**=*rc*> /

casOut={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

docId="*variable-name*"

nThreads=*integer*

***required parameter** **ruleTerms**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

singlePass=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

```
        where="where-expression"
}
*required parameter table={
        caslib="string",
        computedOnDemand=TRUE | FALSE,
        computedVars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        computedVarsProgram="string",
        dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
        groupBy={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        groupByMode="NOSORT" | "REDISTRIBUTE",
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
      *required parameter name="table-name",
        onDemand=TRUE | FALSE,
        orderBy={{
            format="string",
            formattedLength=integer,
            label="string",
```

```
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      singlePass=TRUE | FALSE,
      vars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      where="where-expression"
  }
  termId="variable-name"
  useOldNames=TRUE | FALSE
;
```

Parameter Descriptions

casOut={*casouttable*}

specifies a data table to contain the rule matching (whether a document satisfies a rule) results.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

docId="*variable-name*"

specifies the variable in the TABLE= data table that contains the document ID.

Default "_document_"

nThreads=*integer*

specifies the number of threads to be used per node. The value must be an integer.

Default 0

Range 0–1024

* **ruleTerms**={*castable*}

specifies a data table that contains the terms in each rule that is generated by the training action.

Long form ruleTerms={name="*table-name*"}

Shortcut form ruleTerms="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the

computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

* **table**={*castable*}

specifies the input data table for rule scoring.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

termId="*variable-name*"

specifies the variable in the TABLE= data table that contains the term ID.

Default "_termnum_"

useOldNames=TRUE | FALSE

specifies whether to use the old variable names used in HPBOOLRULE

Alias legacyName, legacyNames

Default FALSE

Boolean Rule Action Set: Syntax

brTrain Action

CASL Syntax

boolRule.brTrain <**result**=*results*> <**status**=*rc*> /

casOuts={

candidateTerms={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

ruleTerms={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

```
            timeStamp="string"
            where={"string-1" <, "string-2", ...>}
        },
        rules={
            caslib="string"
            compress=TRUE | FALSE
            indexVars={"variable-name-1" <, "variable-name-2", ...>}
            label="string"
            maxMemSize=64-bit-integer
            name="table-name"
            onDemand=TRUE | FALSE
            promote=TRUE | FALSE
            replace=TRUE | FALSE
            replication=integer
            timeStamp="string"
            where={"string-1" <, "string-2", ...>}
        }
    }
*required parameter docId="variable-name"
    docInfo={
        events={"string-1" <, "string-2", ...>},
      *required parameter id="variable-name",
        table={
            caslib="string"
            computedOnDemand=TRUE | FALSE
            computedVars={{
                format="string",
                formattedLength=integer,
                label="string",
              *required parameter name="variable-name",
                nfd=integer,
                nfl=integer
            }, {...}}
```

computedVarsProgram="*string*"

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

groupByMode="NOSORT" | "REDISTRIBUTE"

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

*required parameter **name**="*table-name*"

onDemand=TRUE | FALSE

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

singlePass=TRUE | FALSE

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

```
        }, {...}}
        where="where-expression"
    },
    targetType="BINARY" | "MULTICLASS" | "MULTILABEL",
    targets={"variable-name-1" <, "variable-name-2", ...>}
}
gNegative=integer
gPositive=integer
mNegative=integer
mPositive=integer
maxCandidates=integer
maxTriesIn=integer
maxTriesOut=integer
minSupports=integer
nThreads=integer
*required parameter table={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
```

```
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      groupByMode="NOSORT" | "REDISTRIBUTE",
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
        parameters},
    *required parameter name="table-name",
      onDemand=TRUE | FALSE,
      orderBy={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      singlePass=TRUE | FALSE,
      vars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      where="where-expression"
}
*required parameter termId="variable-name"
  termInfo={
    *required parameter id="variable-name",
      label="variable-name",
```

table={

caslib="*string*"

computedOnDemand=TRUE | FALSE

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

computedVarsProgram="*string*"

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

groupByMode="NOSORT" | "REDISTRIBUTE"

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

*required parameter **name**="*table-name*"

onDemand=TRUE | FALSE

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

```
            nfd=integer,
            nfl=integer
        }, {...}}
        singlePass=TRUE | FALSE
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}}
        where="where-expression"
    }
}
useOldNames=TRUE | FALSE
;
```

Parameter Descriptions

casOuts={*output_parm*}

specifies the data tables that contain the results generated by the action.

candidateTerms={*casouttable*}

specifies a data table to contain the terms that have been selected for rule creation.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should

allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

ruleTerms={*casouttable*}

specifies a data table to contain the terms in each rule that is generated by the action.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

rules={*casouttable*}

specifies a data table to contain the rules that have been generated by the action for each category.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger

values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

* **docId**="*variable-name*"

specifies the variable in the TABLE= data table that contains the document ID.

Default "_document_"

docInfo={*docinfo_parm*}

specifies the information about the document table.

events={*"string-1"* <, *"string-2"*, ...>}

specifies the values of target variables that are considered as positive events; incompatible with MULTICLASS target type.

* **id**="*variable-name*"

specifies the variable that contains the document ID.

table={*castable*}

specifies the input data table that contains information about documents.

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

targetType="BINARY" | "MULTICLASS" | "MULTILABEL"

specifies the type of the target variables, either BINARY or MULTICLASS.

Default BINARY

targets={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the target variables.

gNegative=*integer*

specifies the minimum g-score needed for a negative term to be considered for rule

extraction.

Alias gNeg

Default 8

Range 0–32767

gPositive=*integer*

specifies the minimum g-score needed for a positive.

Alias gPos

Default 8

Range 1–32767

mNegative=*integer*

specifies the m value for computing estimated precision for negative terms.

Alias mNeg

Default 4

Range 0–32767

mPositive=*integer*

specifies the m value for computing estimated precision for positive terms.

Alias mPos

Default 2

Range 1–32767

maxCandidates=*integer*

specifies the number of term candidates to be selected for each category.

Default 500

Range 0–32767

maxTriesIn=*integer*

specifies the k-in value for k-best search in the term ensemble process for creating rules.

Default 150

Range 0–32767

maxTriesOut=*integer*

specifies the k-out value for k-best search in the rule ensemble process for creating a rule set.

Default 50

Range 0–32767

minSupports=*integer*

specifies the minimum number of documents in which a term needs to appear in order for the term to be used for creating a rule.

Default 3

Range 0–32767

nThreads=*integer*

specifies the number of threads to be used per node. The value must be an integer.

Default 0

Range 0–1024

* **table**={*castable*}

specifies the input data table for rule extraction.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

* **termId**="*variable-name*"

specifies the variable in the TABLE= data table that contains the term ID.

Default "_termnum_"

termInfo={*terminfo_parm*}

specifies the information about the terms table.

* **id**="*variable-name*"

specifies the variable that contains the term ID.

label="*variable-name*"

specifies the variable that contains the text of the terms. This variable must be a text variable.

table={*castable*}

specifies the input data table that contains information about terms.

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

useOldNames=TRUE | FALSE

specifies whether to use the old variable names used in HPBOOLRULE

Alias legacyName, legacyNames

Default FALSE

Details

About Boolean Rules

A Boolean rule is a set of terms that use the logical operators ^ (indicates AND) and ~ (indicates negation) and that are generated for text categorization. For example, the following rule is generated for documents that are related to bank interest:

```
(cut ^ rate ^ bank ^ percent ^ ~sell)
```

The interpretation of the rule is as follows: If a document contains the terms *cut*, *rate*, *bank*, and *percent* but does not contain the term *sell*, it belongs to the bank interest category. The **`boolRule`** action set contains a **`brTrain`** action, which extracts a set of Boolean rules from labeled documents, and a **`brScore`** action, which matches rules (that is, determines whether a document satisfies a rule) in new, unlabeled documents.

How Are the Rules Extracted?

The extraction (training) process is as follows:

1. Use an information gain criterion to form an ordered list of candidate terms. The term that best predicts the category is first on the list, and so on. Remove terms that do not have a significant relationship to the category from the list. Set the current term to the first term.
2. Determine the *estimated precision* of the current term. The estimated precision is the projected percentage of the term's occurrence within the category in out-of-sample data, if additive smoothing is used. Create a rule that consists of that term.
3. If the estimated precision of the current rule cannot be improved by adding more terms as qualifiers, continue with Step 6.
4. Starting with the next term on the list, determine whether the combination of the current rule with that term (through either presence or absence of the term) significantly improves the information gain and also improves the estimated precision.
5. If at least one combination meets the criterion in Step 4, choose the combination that yields the best estimated precision and go to Step 3 with that combination. Otherwise, continue to Step 6.
6. If the best rule that is obtained in Step 3 has a higher estimated precision than the current highest-precision rule, replace the current rule by the new rule.
7. Increment the current term to the next term on the ordered list of candidate terms and go to Step 2. Continue until all terms in the list have been considered.
8. Determine whether the harmonic mean of precision and recall (the F1 score) of the current rule set is improved by adding the best rule obtained by the previous steps. If there is no improvement, then exit. If there is improvement, remove all documents that match the new rule from the document set, add this rule to the rule set, and go to Step 1 to start creating the next rule in the rule set.

The **brTrain** action extracts Boolean rules from documents. To run the **brTrain** action on a set of documents, you first need to parse the documents and generate two data tables: the terms table and the term-by-document matrix (parent) table. These two tables can be generated by the **tmMine** action.

How Do You Apply Boolean Rules to New Data?

To apply Boolean rules to new, unlabeled documents, you first need to generate the term-by-document matrix (parent) table from the **tmMine** action set. Then you run the **brScore** action on the term-by-document matrix table. The **brScore** action evaluates all the rule and document pairs (rule, doc) and determines whether the document satisfies the rule. A document satisfies a rule (in other words, a rule is matched in the document) if and only if all the positive terms in the rule are present in the document and none of the negative terms is present in the document. If a document satisfies a rule, then the document and the rule will be contained in the output.

The **brScore** action also outputs a special rule for which ID=0. A document satisfies the rule for which ID=0 if and only if the document does not satisfy any rule in the input Boolean rules.

Example

Example 7.1: Deriving Boolean Rules from a Term-by-Document Frequency Matrix

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example first executes the **tmMine** action to generate a term-by-document matrix for training data and uses the **brTrain** action to extract Boolean rules. Then it executes the **tmScore** action to generate a term-by-document matrix for testing data and uses the **score** action to match the extracted rules.

The following DATA step creates the data table mycas.reviews. This DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.reviews;
  infile datalines delimiter='|' missover;
  length text $300 category $20;
  input text$ positive category$ did;
  datalines;
This is the greatest phone ever! love it!|1|electronics|1
```

```
The phone's battery life is too short and screen resolution is low.|0|electronics|2
The screen resolution is low, but I love this tv.|1|electronics|3
The movie itself is great and I like it, although the resolution is low.|1|movies|4
The movie's story is boring and the acting is poor.|0|movies|5
I watched this movie on tv, it's not good on a small screen. |0|movies|6
watched the movie first and loved it, the book is even better!|1|books |7
I like the story in this book, they should put it on screen.|1|books|8
I love the author, but this book is a waste of time, don't buy it.|0|books|9
;
```

The following three PROC CAS steps execute the **tmMine** action to generate a term-by-document matrix for the training data, execute the **brTrain** action to extract Boolean rules, and then print the rules, which are shown in Output 7.1.1.

```
proc cas;
  loadactionset "textMining";
  tmMine / documents='reviews' text='text' docid='did'
           parent={name='parent', replace=1} terms={name='terms', replace=1}
           parseConfig={name='config', replace=1} offset={name='pos', replace=1}
           entities='None' NounGroups=0 stemming=1 tagging=0 reduce=1;
run;


proc cas;
  loadactionset "boolRule";
  brTrain / table='parent' docid='_document_' termid='_termnum_' minsupports=1
            mPositive=1 gPositive=1
            docinfo={table='reviews', id='did', targettype='BINARY',
               targets={'positive'}}
            terminfo={table='terms', id='_termnum_', label='_term_'}
            casOuts={rules={name='review_rules' replace=1},
               ruleterms={name='review_ruleterms' replace=1}};
run;

proc cas;
  fetch / table='review_rules';
run;
```

Output 7.1.1 Results of the Training Data

Results from table.fetch

Selected Rows from Table REVIEW_RULES

Index	target	target ID	target variable	target value	rule ID	rule	rule set's true positive	rule set's false positive	rule set's support	rule's true positive	rule's false positive	rule's support	F-1 score
1	positive	1	positive	1	1	like	2	0	2	2	0	2	0.5714285714
2	positive	1	positive	1	2	better	3	0	3	1	0	1	0.75
3	positive	1	positive	1	3	great	4	0	4	1	0	1	0.8888888889
4	positive	1	positive	1	4	love	5	1	6	1	1	2	0.9090909091

Selected Rows from Table REVIEW_RULES

precision	recall
1	0.4
1	0.6
1	0.8
0.8333333333	1

The following DATA step generates testing data for this example:

```
data mycas.new_reviews;
  infile datalines delimiter='|' missover;
  length text $300;
  input text$ did;
  datalines;
love it! a great phone, even better than advertised|1
I like the book, GREATEST in this genre|2
;
```

The following three PROC CAS steps execute the **tmScore** action to generate a term-by-document matrix for the testing data, execute the **brScore** action to score the data, and then print the scoring results, which are shown in Output 7.1.2.

```
proc cas;
  tmscore / documents='new_reviews' text='text' docid='did' terms='terms'
            parent={name='new_parent' replace=1} parseconfig='config';
run;


proc cas;
  brScore / table='new_parent' ruleterms='review_ruleterms'
            casOut={name='score_results',replace=1};
run;


proc cas;
  fetch / table='score_results';
run;
```

Output 7.1.2 Scoring Results

Results from table.fetch

Selected Rows from Table SCORE_RESULTS			
Index	document ID	target ID	rule ID
1	1	1	4
2	1	1	3
3	1	1	2
4	2	1	3
5	2	1	1

Chapter 8
Factorization Machine Action Set

Contents

Syntax **277**
The factmac Action 278
Details **288**
Displayed Output 288
Results 289
Output Data Tables 290
Example **290**
Example 8.1: MovieLens Data 290
References **291**

Factorization Machine Action Set: Syntax

factmac Action

CASL Syntax

factmac.factmac <**result**=*results*> <**status**=*rc*> /

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

code={

casOut={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

comment=TRUE | FALSE,

fmtWdth=*integer*,

iProb=TRUE | FALSE,

indentSize=*integer*,

intoCutPt=*double*,

labelId=*integer*,

lineSize=*integer*,

noTrim=TRUE | FALSE,

pCatAll=TRUE | FALSE,

tabForm=TRUE | FALSE

}

codeGroup="*string*"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

freq="*variable-name*"

id={*"variable-name-1"* <, *"variable-name-2"*, ...>}

inputs={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

learnStep=*double*

maxIter=*integer*

modelId="*string*"

nFactors=*integer*

nominals={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

nonNegative=TRUE | FALSE

outModel={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

output={

*required parameter **casOut**={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

copyVars="ALL" | "ALL_MODEL" | "ALL_NUMERIC" | {*"variable-name-1"* <, *"variable-name-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

printTarget=TRUE | FALSE

saveState={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

seed=*double*

***required parameter** **table**={

```
      caslib="string",
      computedOnDemand=TRUE | FALSE,
      computedVars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      computedVarsProgram="string",
      dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
      groupByMode="NOSORT" | "REDISTRIBUTE",
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
        parameters},
     *required parameter name="table-name",
      onDemand=TRUE | FALSE,
      singlePass=TRUE | FALSE,
      vars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      where="where-expression"
  }
   target="variable-name"
;
```

Parameter Descriptions

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variable attributes.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

code={*aircodegen*}

produces SAS score code.

For more information about specifying the code parameter, see the common aircodegen parameter (Appendix A: Common Parameters).

codeGroup="*string*"

specifies the code group.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

freq="*variable-name*"

specifies the frequency variable.

id={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the variables to use as record identifiers and to transfer to the state output table.

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to be used in the training.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

learnStep=*double*

specifies the learning step size for the optimization.

Default 0.001

maxIter=*integer*

specifies the maximum number of iterations.

Default 30

modelId="*string*"

specifies the model ID variable name.

nFactors=*integer*

specifies the number of factors to be estimated.

Default 5

nominals={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the nominal variables to be used in the training.

For more information about specifying the nominals parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

nonNegative=TRUE | FALSE

when set to True, performs nonnegative factorization.

Default FALSE

outModel={*casouttable*}

specifies the output data table in which to save the estimated factorization machine parameters.

For more information about specifying the outModel parameter, see the common casouttable parameter (Appendix A: Common Parameters).

output={*outputStatement*}

specifies the output data table in which to save the scored observations.

For more information about specifying the output parameter, see the common outputStatement parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

printTarget=TRUE | FALSE

when set to True, produces an output table containing the name of the predicted target variable.

Default FALSE

saveState={*casouttable*}

specifies the output data table in which to save the state of the factorization machine for future scoring.

For more information about specifying the saveState parameter, see the common casouttable parameter (Appendix A: Common Parameters).

seed=*double*

specifies the seed value for random number generation.

Default 0

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering

upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

target="*variable-name*"

specifies the target variable.

Details

Following Rendle (2012), the factorization machine model is defined as

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j x_{j'} \sum_{f=1}^{k} v_{jf} v_{j'f}$$

where $\mathbf{x} = (x_1, \ldots, x_p)$ is an observed p-dimensional input feature vector, $\hat{y}$ is the predicted target, w_0 is a global bias, w_j are per-feature biases, and v_{jf} denotes coordinate f of the vector $\mathbf{v}_j \in \mathbb{R}^k$. The overall factor matrix $\mathbf{V} \in \mathbb{R}^{p \times k}$ is the concatenation of the row vectors $\mathbf{v}_j$ for $j = 1, \ldots, p$. The number of factors is k. The **factmac** action estimates the model parameters $w_0, w_1, \ldots, w_p$ and $\mathbf{V}$. The estimation is done by minimizing the root mean square error (RMSE), which is defined by

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2}$$

over the training set, subject to the max-norm regularization constraint

$$\|\mathbf{v}_j\|_\infty < B, j = 1, \ldots, p$$

The optimization uses a projected-gradient version of the stochastic gradient descent (SGD) algorithm. The constant B is automatically set, based on the range of the input and target variables.

The results of running the **factmac** action are reproducible only if you specify a value greater than 0 for the **seed** parameter and set the **nThreads** parameter to 1, because the **factmac** action uses a threaded SGD solver that purposefully uses shared memory without locks in each computation node. The variability between runs is nevertheless expected to be small. The **factmac** action can still use multiple machines for the analysis even when **nThreads** is set to 1.

Displayed Output

The **factmac** action displays various tables for analysis. The following sections describe the output in the order of its appearance when the related options are specified.

Number of Observations

The "Number of Observations" table displays the number of observations that are read from the input data table and used.

Model Information

The "Model Information" table displays basic information about the parameters that are used in the analysis. This information includes the maximum number of iterations, number of factors, seed value for random number generation, and learning step size.

Iteration History

The "Iteration History" table displays the iteration number and the approximate loss. The "displayed loss" is an approximation to increase computational efficiency. The final exact loss is shown in the "Final Exact Loss" table.

Final Exact Loss

The "Final Exact Loss" table displays the exact mean square error (MSE) and root mean square error (RMSE) of the learned factorization machines model solution, which are computed on the training data.

Interval Variables

The "Interval Variables" table shows the mean and the standard deviation of the interval variables.

Nominal Variables

The "Nominal Variables" table shows the number of levels for the nominal variables.

OutputCasTables Table

The OutputCasTables table is a special table that has information about each CAS table that is created during a CAS action execution. The information for each CAS table consists of the CAS table name, the caslib in which the table resides, and the number of columns and rows in the CAS table. This table is created when you specify the **`output`**, **`outModel`**, or **`saveState`** parameter.

Results

The results that the **`factmac`** action produces are stored in tables. You can access these tables by their names, which are shown in Table 8.1. For more information about the contents of these tables, see the section "Displayed Output" (Chapter 7, *SAS Visual Data Mining and Machine Learning: Procedures Guide*).

Table 8.1 Results Tables Produced by the **`factmac`** Action

Table Name	Description	Action	Parameter
DescStatsInt	Descriptive statistics for interval variables	**`factmac`**	Default
DescStatsNom	Descriptive statistics for nominal variables	**`factmac`**	Default
FinalLoss	Final exact loss	**`factmac`**	Default
ModelInfo	Model information	**`factmac`**	Default
NObs	Number of observations	**`factmac`**	Default
OptIterHistory	Iteration history	**`factmac`**	Default
OutputCasTables	A special table that has information about all the CAS tables that are created during a CAS action execution	**`factmac`**	Default

Output Data Tables

The **factmac** action creates a data table to which it writes the biases and the factors. You specify the name of this data table in the **outModel** parameter. Details about the data table are listed in Table 8.2.

Table 8.2 Output Data Table Produced by the **factmac** Action

Data Table	Description
FACTORS	Lists the global bias, the name of each input variable, each level, and the values of the estimated bias and factors

Example

Example 8.1: MovieLens Data

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example shows how to use the **factmac** action to train a factorization machine model from observations in a data table. In this case, the data are derived from companies that provide movies for online viewing. A company wants to offer its customers recommendations of movies that they might like. These recommendations are based on ratings that are provided by users. The MovieLens data set was developed by the GroupLens project at the University of Minnesota and is available at `http://grouplens.org/datasets/movielens` (Harper and Konstan 2015). This example uses the MovieLens 100K version.

There are four columns in the MovieLens 100K data set: user ID, item ID (each item is a movie), timestamp, and rating. This example predicts the rating for a specified user ID and an item ID. The data set is very sparse because most combinations of users and movies are not rated.

You can download the compressed archive file from the website at http://files.grouplens.org/datasets/movielens/ml-100k.zip and use any third-party unzip tool to extract all the files in the archive to the destination directory of your choice.[1] The file that contains the ratings is *u.data*. The following DATA step loads the data table from the directory into your CAS session:

```
proc casutil;
    load file="u.data"    /*or other user-defined location*/
    casout="movlens"
    importoptions=(filetype="CSV" delimiter="TAB" getnames="FALSE"
                   vars=("userid" "itemid" "rating" "timestamp"));
run;
```

The following statements train a factorization machine model on the MovieLens data by using the **factmac** action.

The **table** parameter names the input data table to be analyzed. The **outModel** parameter outputs the fitted parameter estimates to the factors_out data table. The **inputs** parameter specifies the input variables to be used. The **nominals** parameter specifies that the variables userid and itemid are nominal. The **target** parameter specifies the target variable to be used. The **maxIter** parameter specifies the number of iterations for the optimization. The **nFactors** parameter specifies the number of factors to be estimated for the factorization machine model. The **learnStep** parameter specifies the learning step size for the optimization.

```
proc cas;
   loadactionset "tkfactmac";
   action factmac result=R / table={name="movlens"},
      outModel={name="factors_out", replace=true},
      inputs={"userid", "itemid"},
      nominals={"userid", "itemid"},
      target="rating",
      maxIter=20, nFactors=10, learnStep=0.15,
      output={casout={name="score_out", replace="TRUE"},
      copyvars={"userid","itemid","rating"}};
run;
```

References

Harper, F. M., and Konstan, J. A. (2015). "The MovieLens Datasets: History and Context." *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5:19 pages. http://dx.doi.org/10.1145/2827872.

Rendle, S. (2012). "Factorization Machines with libFM." *ACM Transactions on Intelligent Systems and Technology* 3:1–22.

[1] Disclaimer: SAS may reference other websites or content or resources for use at Customer's sole discretion. SAS has no control over any websites or resources that are provided by companies or persons other than SAS. Customer acknowledges and agrees that SAS is not responsible for the availability or use of any such external sites or resources, and does not endorse any advertising, products, or other materials on or available from such websites or resources. Customer acknowledges and agrees that SAS is not liable for any loss or damage that may be incurred by Customer or its end users as a result of the availability or use of those external sites or resources, or as a result of any reliance placed by Customer or its end users on the completeness, accuracy, or existence of any advertising, products, or other materials on, or available from, such websites or resources.

Chapter 9
Fast k-Nearest Neighbors Algorithm Action Set

Contents

Syntax **293**
The fastKnn Action 294
Details **305**
Displayed Output 305
Example **306**
Example 9.1: Home Mortgage Data 306

Fast k-Nearest Neighbor Action Set: Syntax

fastknn Action

CASL Syntax

fastKnn.fastknn <**result**=*results*> <**status**=*rc*> /

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

id={*"variable-name-1"* <, *"variable-name-2"*, ...>}

inputs={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

k=*64-bit-integer*

outDist={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

output={

*required parameter **casOut**={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

copyVars="ALL" | "ALL_MODEL" | "ALL_NUMERIC" | {*"variable-name-1"* <, *"variable-name-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

parallelization="INPUT" | "QUERY" | *64-bit-integer*

***required parameter query**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

singlePass=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

where="*where-expression*"

}

***required parameter table**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

***required parameter name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>*},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

***required parameter name**="*table-name*",

onDemand=TRUE | FALSE,

singlePass=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

***required parameter name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

```
        where="where-expression"
    }
    threshDist=double
;
```

Parameter Descriptions

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variable attributes.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

id={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the variable to use as the record identifier.

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the analysis.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

k=*64-bit-integer*

specifies the number of neighbors to be returned.

Default 2

outDist={*casouttable*}

specifies the output data table in which to save the computed distances.

For more information about specifying the outDist parameter, see the common casouttable parameter (Appendix A: Common Parameters).

output={*outputStatement*}

specifies the output data table in which to save the computed neighbors.

For more information about specifying the output parameter, see the common outputStatement parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

parallelization="INPUT" | "QUERY" | *64-bit-integer*

specifies the type of parallelization to use.

Alias par

* **query**={*castable*}

specifies the query input data table, which contains the query observations.

Long form query={name="*table-name*"}

Shortcut form query="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again

on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

threshDist=*double*

specifies the threshold value to use for distance computation.

Default 100

Details

The **fastknn** action implements the k-nearest neighbor (k-NN) search algorithm. The k-NN algorithm has numerous applications, including recommendation systems, image search, fingerprint recognition, and clustering.

The **fastknn** action uses the Euclidean distance metric to compute distances between the input and query observations. The **fastknn** action stores the search results in the output data table that is specified by the **output** parameter. For each row in the **query** data table, the **output** table returns the identifiers of the k rows from the input data table that are nearest. You specify k by using the **k** parameter. In addition, the **fastknn** action stores the distances between the query and input observations in an output data table that is specified in the **outDist** parameter. You can exclude the distances that are greater than a threshold value by specifying by the **threshDist** parameter. You can choose to distribute either the input data table or the query data table by specifying the **parallelization** parameter.

The distance metric in the **fastknn** action is defined as

$$D_{p,q} = \sqrt{\sum_{m=1}^{d} (p_m - q_m)^2}$$

where D is the Euclidean distance metric, d is the dimension, and p and q represent the input and query observations, respectively.

The distances are sorted in ascending order, and the k neighbors that correspond to the smallest distances are returned.

Displayed Output

The **fastknn** action displays various tables for analysis.

The following sections describe the output in the order of their appearance when the related options are specified.

Number of Observations

The "Number of Observations" table displays the number of observations that are read from the input data table and the number that are used.

Model Information

The "Model Information" table displays basic information about the parameters that are used in the analysis. This information includes the number of neighbors and the distance threshold.

Output CAS Tables

The **fastknn** action always produces a table that lists the CAS library, the data table name, and the number of rows and columns in each output data table. When you specify the **output** or the **outDist**, the **fastknn** action also produces the output data tables that display, respectively, the identifiers of the nearest k rows from the input data table and the distances between the query and the input observations.

Example

Example 9.1: Home Mortgage Data

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example shows how to use the **fastknn** action to retrieve the nearest neighbors between observations in a query data table and an input data table. In this case, the data are from the hmeq data set, which contains observations for mortgage applicants. This example selects 4,000 applicants for the input data table, and 100 applicants for the query data table. The **fastknn** action returns the input applicants that are most similar to each of the query applicants. The analysis uses eleven variables: bad, loan, mortdue, value, yoj, derog, delinq, clage, ninq, clno, and debitinc. The remaining variables in the data set are not used.

You can load the hmeq and query data sets into your CAS session by using the following DATA steps. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.hmeq;
set sampsio.hmeq(obs=4000);
id1=_n_;
run;

data mycas.query;
set sampsio.hmeq(firstobs=4001 obs=4100);
id1=_n_;
run;
```

The following statements retrieve the *k*-nearest neighbors from the hmeq data by using the **fastknn** action. In this example, the value of the **k** parameter is 4.

```
proc cas;
   loadactionset "fastknn";
   action fastknn result=R / table={name="hmeq"},
      query="query",
      inputs={"bad", "loan", "mortdue", "value", "yoj", "derog",
              "delinq", "clage", "ninq", "clno", "debtinc"},
      id="id1",
      k=4,
      output={casout={name="knn_out", replace="TRUE"}},
      outDist={name="dist_out", replace=true},
      parallelization="input",
      threshDist=10000;
run;
```

The **table** parameter names the input data table. The **query** parameter names the query data table. The **inputs** parameter specifies the input variables to be used. The **id** parameter specifies that the variable id1 contains the numeric identifier for each observation. The **k** parameter specifies 4 as the number of neighbors to be returned for each row of the query table. The **output** parameter outputs the list of neighbors to the knn_out data table. The **outDist** parameter outputs the computed distances to the dist_out data table. The **parallelization** parameter requests that the input, rather than the query data table, be distributed among workers. The **threshDist** parameter requests that distances greater than 10,000 not be computed.

Chapter 10
Generalized Linear Multitask Learning Action Set

Contents

Syntax **309**
The graphMultiReg Action 310
Details **324**
Examples of Using the graphType Parameter 324
Missing Values 329
Displayed Output 329
Results 330
Output Data Tables 330
Example **330**
Example 10.1: Synthetic Data 330

Generalized Linear Multitask Learning Action Set: Syntax

graphMultiReg Action

CASL Syntax

gleam.graphMultiReg <**result**=*results*> <**status**=*rc*> /

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

graphOut={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

graphTable={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*
}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*
}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*
}, {...}},

singlePass=TRUE | FALSE,

vars={{

format="*string*",

```
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
graphType="CLUSTER" | "CUSTOM" | "FUSE" | "INDEP"
inputs={{
    format="string",
    formattedLength=integer,
    label="string",
   *required parameter name="variable-name",
    nfd=integer,
    nfl=integer
}, {...}}
maxIters=64-bit-integer
modelId="string"
modelOut={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
    replace=TRUE | FALSE,
    replication=integer,
    timeStamp="string",
    where={"string-1" <, "string-2", ...>}
}
```

output={

*required parameter **casOut**={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

copyVars="ALL" | "ALL_MODEL" | "ALL_NUMERIC" | {*"variable-name-1"* <, *"variable-name-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

regL1=*double*

regL2=*double*

saveState={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

seed=*double*

***required parameter table**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

***required parameter name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

***required parameter name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

```
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    orderBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
  }
  *required parameter targets={"variable-name-1" <, "variable-name-2", ...>}
   tolerance=double
;
```

Parameter Descriptions

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variable attributes.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

graphOut={*casouttable*}

specifies the output data table in which to save the graph table.

For more information about specifying the graphOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

graphTable={*castable*}

specifies the graph table that the user defines. The first column of the graph table must contain the row ID variable. The graph table must also include all target variables that you specify in the target list.

Alias R

Long form graphTable={name="*table-name*"}

Shortcut form graphTable="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

*** name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

graphType="CLUSTER" | "CUSTOM" | "FUSE" | "INDEP"

specifies the type of graph table.

Default CLUSTER

CLUSTER

uses a cluster graph table, in which all tasks are connected.

CUSTOM

uses a custom graph table that the user specifies.

FUSE

uses a fuse graph table, in which each task is connected to the next task in the target list.

INDEP

uses an independent graph table, in which all tasks are independent of one another.

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to be used in the training.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

maxIters=*64-bit-integer*

specifies the maximum number of iterations.

modelId="*string*"

specifies the model ID variable name.

modelOut={*casouttable*}

specifies the output data table in which to save the estimated multitask regression weights.

For more information about specifying the modelOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

output={*outputStatement*}

specifies the output data table in which to save the scored observations.

For more information about specifying the output parameter, see the common outputStatement parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

regL1=*double*

specifies the L1 (LASSO) penalization weight.

regL2=*double*

specifies the L2 graph penalization weight.

saveState={*casouttable*}

specifies the output data table in which to save the state of the multitask learning model for

future scoring.

For more information about specifying the saveState parameter, see the common casouttable parameter (Appendix A: Common Parameters).

seed=*double*

specifies the seed to use for pseudorandom number generation.

Default 0

* **table**={*castable*}

specifies the settings for an input table. The table must include one variable named id as the row ID variable.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

* **targets**={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the target variables.

Alias mltargets, target

tolerance=*double*

specifies the optimization tolerance (absolute L2 difference of solution) as a stopping criterion.

Default 1e-06

Details

The **graphMultiReg** action implements the multitask learning technique for least squares loss with ℓ_1 and graph structure penalizations. It solves multiple related sparse linear regression problems simultaneously. A graph structure encodes the relationships between the problems. The **graphMultiReg** action shares the data and model parameters among different regression problems and solves the problems simultaneously in order to produce a more robust and accurate predictive model. When used with independent graph tables, the **graphMultiReg** action also serves as an elastic net regularization solver, which is widely used in feature selection.

Consider a total number of K different tasks. For each task $i = 1, \ldots, K$, there are n_i observations in the observation matrix $X_i \in \mathbb{R}^{n_i \times p}$. The targets are denoted by $y_i \in \mathbb{R}^{n_i}$. Let $W = [W_1, \ldots, W_K] \in \mathbb{R}^{p \times K}$ be the model that you want to learn, such that

$$y_i = X_i W_i + \epsilon_i$$

where ϵ_i is random noise, and you want to learn W_i from the observations X_i and the targets y_i.

The model W is estimated by solving the following optimization problem,

$$\widehat{W} = \arg\min_{W} \frac{1}{2} \sum_{i=1}^{K} \| X_i W_i - y_i \|_2^2 + \frac{1}{2} \rho_{\ell_2} \| R^\top W \|_F^2 + \rho_{\ell_1} \| W \|_1$$

where R is the graph table (matrix) that represents the relationships between K tasks. Note that the specific form of R can be generated in various ways. One simple way is to model the relationships among tasks by using an undirected graph, where each task is a node, and the two nodes are connected if the two tasks are related. To encode the connections, let

$$\mathcal{E} = \{e_k | k = 1, \ldots, n_e\}$$

be the set of edges in the graph, where $n_e = |\mathcal{E}|$ is the total number of edges in the graph. Define a matrix

$$R \in \mathbb{R}^{|\mathcal{E}| \times K}$$

where

$$R_{k,i} = \begin{cases} 1, & \text{if } e_k \text{ connects targets } i \text{ and } j, i < j \\ -1, & \text{if } e_k \text{ connects targets } i \text{ and } j, i > j \\ 0, & \text{otherwise} \end{cases}$$

Examples of Using the graphType Parameter

The **graphType** parameter values that you can specify are CLUSTER, INDEP, FUSE, and CUSTOM. When the **graphTable** parameter is not specified, the **graphMultiReg** action can automatically generate a graph table if the **graphType** parameter value is CLUSTER, INDEP, or FUSE. If the **graphType** parameter value is CUSTOM, the **graphMultiReg** action does not generate a graph table and instead expects a graph table to be specified in the **graphTable** parameter. If a graph table is specified in the **graphTable** parameter, the **graphMultiReg** action uses the user-specified graph table, and the value of the **graphType** parameter is overwritten by CUSTOM.

- When the value of the **`graphType`** parameter is CLUSTER and no **`graphTable`** parameter is specified, the **`graphMultiReg`** action generates a cluster graph table, where all tasks are connected (to a virtual mean task). The graph table $R \in \mathbb{R}^{K \times K}$ is defined as

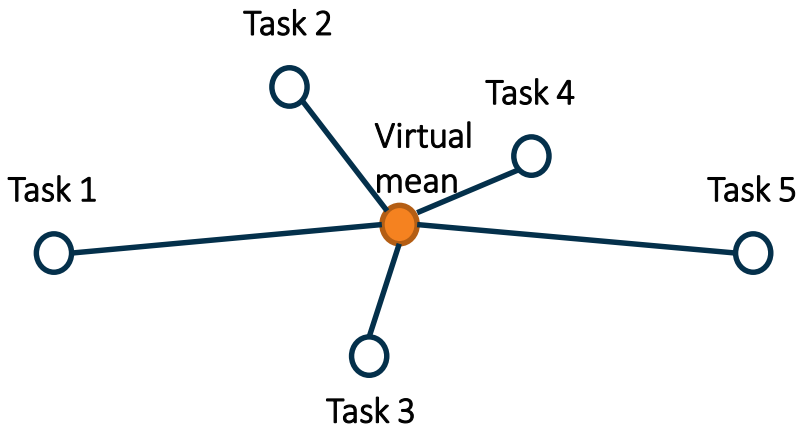
$$R_{k,i} = \begin{cases} 1, & \text{if } k = i \\ -\frac{1}{K-1}, & \text{otherwise} \end{cases}$$

When $K = 1$ (only one target variable), the value of the graph table is set to 1. Figure 10.1 shows an example of a cluster graph structure of five tasks, whose corresponding graph table matrix is

$$R = \begin{bmatrix} 1 & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{4} & 1 & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & 1 & -\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & 1 & -\frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & 1 \end{bmatrix}$$

The cluster graph type is best used when all tasks are similarly related to one another.

Figure 10.1 Cluster Graph Structure with Five Targets (Tasks 1 to 5)



- When the value of the **`graphType`** parameter is INDEP and no **`graphTable`** parameter is specified, the **`graphMultiReg`** action generates an independent graph table, where all tasks are independent of one another. The graph table matrix $R \in \mathbb{R}^{K \times K}$ is an identity matrix,

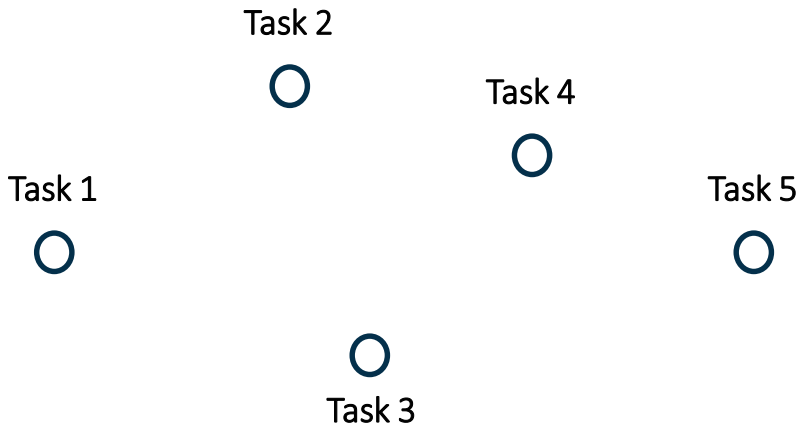
$$R_{k,i} = \begin{cases} 1, & \text{if } k = i \\ 0, & \text{otherwise} \end{cases}$$

When $K = 1$ (only one target variable), the value of the graph table is set to 1. Figure 10.2 shows an

example of an independent graph structure of five tasks, whose corresponding graph table matrix is

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 10.2 Independent Graph Structure with Five Targets (Tasks 1 to 5)



The independent graph type is best used when all tasks are independent of one another. It can also be used for elastic net regression problems.

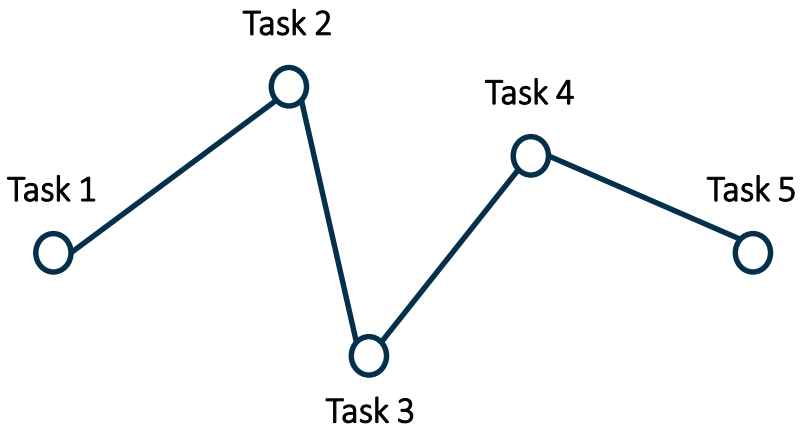
- When the value of the **`graphType`** parameter is FUSE and no **`graphTable`** parameter is specified, the **`graphMultiReg`** action generates a fuse graph table, where each task is connected to the next task in the target list. The graph table $R \in \mathbb{R}^{(K-1)\times K}$ is defined as

$$R_{k,i} = \begin{cases} 1, \text{ if } k = i \\ -1, \text{ if } k = i + 1 \\ 0, \text{ otherwise} \end{cases}$$

When $K = 1$ (only one target variable), the value of the graph table is set to 0. Figure 10.3 shows an example of a fuse graph structure of five tasks, whose corresponding graph table matrix is

$$R = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

Figure 10.3 Fuse Graph Structure with Five Targets (Tasks 1 to 5)



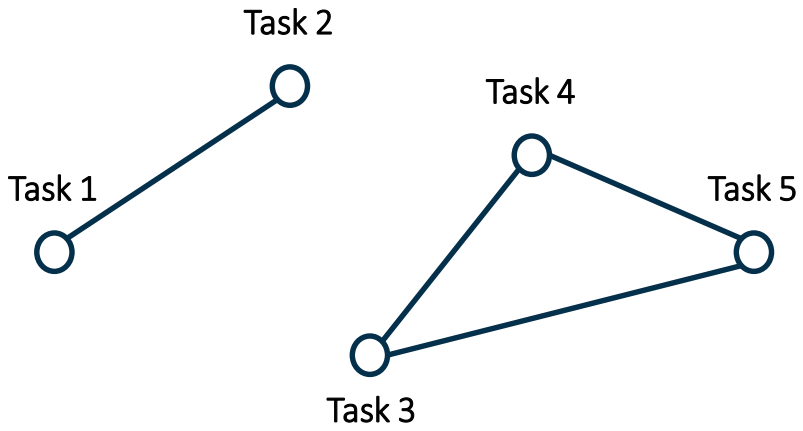
The fuse graph type is best used when tasks have known orders, such as temporal relationships.

- When the value of the **`graphType`** parameter is CUSTOM, the **`graphMultiReg`** action uses the graph table that you specify in the **`graphTable`** parameter. The user-specified custom graph table must have one variable named id as the row ID variable, and also must have variables that match all the target variables.

  Figure 10.4 shows an example of a custom graph. Using the definition of R near the beginning of the section "Details" on page 324, the corresponding graph table matrix is

$$R = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

Figure 10.4 Example of a Custom Graph Structure with Five Targets (Tasks 1 to 5)



The following DATA step creates the custom graph table shown in Figure 10.1. This data table contains six variables: id is the row ID variable, and Task1, Task2, Task3, Task4, and Task5 are the target variables.

```
data customR;
  input id Task1 Task2 Task3 Task4 Task5;
  cards;
  0   1  -1   0   0   0
  1   0   0   1  -1   0
  2   0   0   1   0  -1
  3   0   0   0   1  -1
  ;
run;
```

The following statements display the generated graph table, which is shown in Output 10.5.

```
proc print noobs data=customR;
run;
```

Figure 10.5 Example of a Custom Graph Table

id	Task1	Task2	Task3	Task4	Task5
0	1	-1	0	0	0
1	0	0	1	-1	0
2	0	0	1	0	-1
3	0	0	0	1	-1

Missing Values

In the **graphMultiReg** action, missing values are not allowed in the input variables, but they are allowed in the target variables. For any observation in the input table x_k, let the corresponding target variables be $y_i, i = 1, \ldots, K$. Let $\mathcal{S} = \{1, \ldots, K\}$ be the set of all task indices, $\mathcal{S}_M$ be the set of indices whose target variables are missing, and $\mathcal{S}_N = \mathcal{S} \backslash \mathcal{S}_M$ be the set of indices whose target variables are present. The **graphMultiReg** action considers that the observation $x_k \in X_i$, where $i \in \mathcal{S}_N$, and $x_k \notin X_j$, where $j \in \mathcal{S}_M$.

If all target variables are missing for an observation, the observation is skipped and not used in the optimization. However, the observation is still scored in the output data table.

Displayed Output

The **graphMultiReg** action displays various tables that are related to the multitask regression. The following sections describe the output tables in the order of their appearance when the related options are specified.

Model Information

The "Model Information" table displays basic information about the parameters that are used in the multitask regression analysis. This information includes the seed value, ℓ_1 penalization weight, ℓ_2 penalization weight, tolerance value, and maximum number of iterations.

Number of Observations

The "Number of Observations" table displays the number of observations that are read from the input data table and used.

Interval Variables

The "Interval Variables" table shows the mean and the standard deviation of the interval variables.

Iteration History

The "Iteration History" table displays the iteration history of the objective function.

OutputCasTables Table

The OutputCasTables table is a special table that has information about each CAS table that is created during a CAS action execution. The information for each CAS table consists of the CAS table name, the caslib in which the table resides, and the number of columns and rows in the CAS table. This table is created when you specify the **output**, **outModel**, or **saveState** parameter.

Results

The results that the **graphMultiReg** action produces are stored in tables. You can access these tables by their names, which are shown in Table 10.1. You use these table names when you specify the **outputTables** parameter.

Table 10.1 ODS Tables Produced by the **graphMultiReg** Action

Table Name	Description	Parameter
DescStats	Descriptive statistics for interval variables	Inputs and targets
ModelInfo	Model information	Default
NObs	Number of observations	Default
OptIterHistory	Iteration history	Default
OutputCasTables	A special table that has information about all the CAS tables that are created during a CAS action execution	Default

Output Data Tables

The **graphMultiReg** action creates two data tables:

- The output graph table lists the value of the graph table. You specify the name of this data table in the **graphOut** parameter.
- The output model table lists the value of the estimated multitask regression weights. You specify the name of this data table in the **modelOut** parameter.

Example

Example 10.1: Synthetic Data

This section contains PROC CAS code.

For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example shows how to use the **graphMultiReg** action to estimate regression models from observations in a data table. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

The following DATA step creates the input data table, mycas.ToyData, in your CAS session. This data table contains seven variables: id is the row ID variable; X1, X2, and X3 are the input variables; and Y1, Y2, and Y3 are the target variables.

```
data mycas.toydata;
  input id X1 X2 X3 Y1 Y2 Y3;
  cards;
  0  0.912621   1.288759   1.568040   2.813182   2.792080    2.792080
  1 -0.696839  -1.487805   1.383056   0.187085   0.559484    0.559484
  2  0.970062   2.405601  -1.638553  -1.903736  -2.301208   -2.301208
  3  0.291515  -0.176170  -0.175393   3.842113   3.799427    3.799427
  4  0.721477   0.472668  -0.522688   5.171737   4.906125    .
  5  1.563781  -1.04760   -0.298725  21.160617  20.762561    .
  6  0.426927   1.319816  -0.022467  -2.231491  -2.361400    .
  7 -1.046538  -0.190017  -1.75929   -9.438538  -9.388092    .
  8  0.155960   0.299715  -0.366969   0.109865  -0.00258535  .
  9  1.568768  -0.900524   0.334290  20.422254  19.799980   19.799980
  ;
run;
```

The following DATA step creates the input graph table, mycas.ToyR, in your CAS session. This data table encodes the relationships between the targets. In this example, the targets Y2 and Y3 are connected, meaning that the regression weights for Y2 and Y3 are expected to be similar. This data table contains four variables: id is the row ID variable, and Y1, Y2, and Y3 are the target variables.

```
data mycas.toyr;
  input id Y1 Y2 Y3;
  cards;
  0   0   1  -1
  ;
run;
```

The following statements run the **graphMultiReg** action and output the results to ODS tables:

```
proc cas;
   loadactionset "gleam";
   action gleam.graphMultiReg result=r / table={name='toydata'},
      inputs={{name='X1'}, {name='X2'}, {name='X3'}},
      targets={'Y1', 'Y2', 'Y3'},
      graphTable={name='toyr'},
      graphType='CUSTOM',
      regL1=0.5,
      regL2=1.0,
      maxIters=200,
      tolerance=1e-2,
      seed=123,
      modelOut={name='gmr_outW', replace=true},
      graphOut={name='gmr_outR', replace=true},
      saveState={name='gmr_ast', replace=true},
      output={casOut={name='gmr_out', replace=true},
              copyVars={'id', 'X1', 'X2', 'X3'}};
   run;
```

The **table** parameter names the input data table. The **inputs** parameter specifies that the X1, X2, and X3 variables be used as inputs. The **targets** parameter specifies that the Y1, Y2, and Y3 variables be used

as targets. The **graphTable** parameter specifies the user-defined graph table. The **graphType** parameter indicates the use of a custom graph table that is specified in the **graphTable** parameter. The **regL1** parameter specifies the value of the ℓ_1 penalization weight. The **regL2** parameter specifies the value of the ℓ_2 penalization weight. The **maxiters** parameter specifies the maximum number of iterations. The **tolerance** parameter specifies the optimization tolerance as a stopping criterion. The **seed** parameter specifies the seed to use for pseudorandom number generation. The **modelOut** parameter writes the estimated multitask regression weights to the data table mycas.gmr_outW. The **graphOut** parameter writes the graph table to the data table mycas.gmr_outR. The **savestate** parameter stores the estimated model in the data table mycas.mtl_ast for future scoring. The **output** parameter writes the scored results to the data table mycas.gmr_out, and the **copyVars** parameter copies the id, X1, X2, and X3 variables to the output.

Chapter 11
Graph-Based Semisupervised Learning Action Set

Contents

Syntax **333**
The labelSpread Action 334
Details **343**
Displayed Output 343
Example **344**
Example 11.1: Home Mortgage Data 344
References **346**

Graph-Based Semisupervised Learning Action Set: Syntax

labelSpread Action

CASL Syntax

graphSemiSupLearn.labelSpread <**result**=*results*> <**status**=*rc*> /

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

gamma=*double*

inputs={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

***required parameter labeledTable**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

```
        }, {...}},
        computedVarsProgram="string",
        dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
      *required parameter name="table-name",
        onDemand=TRUE | FALSE,
        singlePass=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
}
  maxIters=integer
  nominals={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
  }, {...}}
  output={
      *required parameter casOut={
            caslib="string"
            compress=TRUE | FALSE
            indexVars={"variable-name-1" <, "variable-name-2", ...>}
```

```
        label="string"
        maxMemSize=64-bit-integer
        name="table-name"
        onDemand=TRUE | FALSE
        promote=TRUE | FALSE
        replace=TRUE | FALSE
        replication=integer
        timeStamp="string"
        where={"string-1" <, "string-2", ...>}
    },
    copyVars="ALL" | "ALL_MODEL" | "ALL_NUMERIC" | {"variable-name-1" <, "variable-
      name-2", ...>}
}
*required parameter table={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    singlePass=TRUE | FALSE,
    vars={{
```

```
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
  }
  target="variable-name"
;
```

Parameter Descriptions

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

alters attributes on variables used in this action.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

gamma=*double*

specified the inverse of variance for Gaussian kernels, which computes the similarity between pairs of data samples.

Default	20
Minimum value (exclusive)	0

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies variables to use for analysis.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

* **labeledTable**={*castable*}

specifies the labeled input data table, which contains the labeled data observations.

Alias	labels
Long form	labeledTable={name="*table-name*"}
Shortcut form	labeledTable="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" |

"ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

maxIters=*integer*

specifies the maximum number iterations.

nominals={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies nominal variables to use for analysis.

For more information about specifying the nominals parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

output={*outputStatement*}

For more information about specifying the output parameter, see the common outputStatement parameter (Appendix A: Common Parameters).

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

target="*variable-name*"

specifies target variable to use for analysis.

Details

The **labelSpread** action implements the graph-based semisupervised learning algorithm that relies on label propagation (Zhou et al. 2003). The semisupervised learning algorithm has numerous applications, including web page classification, image recognition, medical imaging, natural language processing, and action recognition.

The **labelSpread** action uses the well-known radial basis function (RBF) kernel metric to compute distances. Because the value of the RBF kernel decreases with distance and ranges between zero and one, it has good interpretation as a similarity measure between the pairs of unlabeled and labeled data observations. The **labelSpread** action stores the predicted output for the target variables for both the unlabeled data and the labeled data in the output data table that is specified by the **output** parameter. For each row in the output data table, the WARN column indicates whether the row of data is coming from labeled data or unlabeled data. The WARN value is 1 for labeled data and 0 for unlabeled data.

Following Chang et al. (2010), the distance metric in the **labelSpread** action is defined as

$$D_{p,q} = \exp(-\gamma * \sqrt{\sum_{k=1}^{K}(p_k - q_k)^2})$$

where D is the RBF kernel distance metric, K is the total number of variables in the data, k is the feature index for the input observation, γ is the inverse of the variance for the Gaussian kernels, and p and q represent the index of the data in the combined input from both the unlabeled data and the labeled data.

Because in most cases the labeled data are relatively small, the **labelSpread** action distributes the unlabeled data and repeats the labeled data in different threads in order to maximize the prediction performance. For each thread, the distance matrix computes the pairwise distance in the combined list of unlabeled data and labeled data. If there are M observations for the unlabeled data and N observations for the labeled data in one thread, the distance matrix is of the size M+N in that thread.

Displayed Output

The **labelSpread** action displays various tables for analysis. The following sections describe the output tables in the order of their appearance when the related parameters are specified.

Model Information

The "Model Information" table displays basic information about the parameters that are used in the analysis. This information includes the number of unlabeled observations, number of labeled observations, number of levels for target variable, **gamma** value, and maximum number of iterations. Here the number of unlabeled observations and labeled observations represents the used observations, which are the observations that exclude missing data.

Output CAS Tables

The **labelSpread** action always produces a table that lists the CAS library, the data table name, and the number of rows and columns in each output data table. For the variables in the unlabeled data, when you specify the **copyvars** subparameter in the **output** parameter, the **labelSpread** action also produces the output data tables that display the variables.

For the variables in the labeled data, when you specify the **copyvars** subparameter in the **output** parameter, if the variables match the input variables list, the **labelSpread** action transfers the values of the variables to the output table. If the variables do not match the input variables list, the **labelSpread** action shows the values of the variables as missing for the labeled data.

Example

Example 11.1: Home Mortgage Data

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example shows how to use the **labelSpread** action to predict the target variables for the unlabeled data between observations in a labeled data table and an unlabeled data table. In this case, the data are from the hmeq data set, which contains observations for mortgage applicants. This example selects 3,000 applicants for the unlabeled data table and 200 applicants for the labeled data table. The **labelSpread** action returns the predicted labels for applicants in the unlabeled data table and the labeled data table by combining the information from the unlabeled data and labeled data. The algorithm propagates the labels from labeled data to unlabeled data according to the similarity measure in an iterative algorithm. The analysis uses eight variables: bad, loan, mortdue, value, yoj, derog, clage, and debitinc. The remaining variables in the data table are not used. Because the **labelSpread** action relies on the pairwise distances as a similarity measure between two observations, any observations in the unlabeled and labeled data tables that contain missing data are ignored.

You can load the Unlabel and Label data sets into your CAS session by using the following DATA steps. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
cas mysess;
libname mycas sasioca sessref=mysess;

data hmeq;
set sampsio.hmeq;
if cmiss(of _all_) then delete;
```

```
run;
data mycas.unlabel(drop=bad);
    set hmeq(obs=3000);
run;

data mycas.label;
    set hmeq(obs=200);
run;
```

The following statements use the **labelSpread** action to predict the labels for the Unlabel data table from the hmeq data. In this example, the value of the **gamma** parameter is 1,000.

```
proc cas;
 loadactionset "tkssl";
 action tkssl.labelSpread /
table={name="unlabel"},
labeledTable="label",
 inputs={"loan", "mortdue", "value", "yoj", "derog",
              "clage", "debtinc"},
 output=  {casout={name="out",   replace="TRUE"}},
         target= "bad",
   gamma=1000,

          nominals={"bad"}            ;
 run;
quit;
```

The **table** parameter names the unlabeled data table, which contains the data without target variables. The **labeledTable** parameter names the labeled data table, which contains the data with target variables. The **inputs** parameter specifies the input variables to be used. The **gamma** parameter specifies 1,000 as the inverse of the variance for Gaussian kernels when you are computing the pairwise distance between data observations. The **output** parameter outputs the list of predicted labels for both the unlabeled data and the labeled data to the Out data table. The **target** parameter specifies the target variable name for the labeled data.

Figure 11.1.1 shows the number of unlabeled observations, number of labeled observations, number of levels for target variable, gamma value, and maximum number of iterations.

Output 11.1.1 Model Information

Results from graphSemiSupLearn.labelSpread

Model Information	
Labeled Observations Used	200
Unlabeled Observations Used	3000
Maximum Iterations	3
Target Number of Levels	2
Gamma	1000

References

Chang, Y.-W., Hsieh, C.-J., Chang, K.-W., Ringgaard, M., and Lin, C.-J. (2010). "Training and Testing Low-Degree Polynomial Data Mapping via Linear SVM." *Journal of Machine Learning Research* 11:1471–1490.

Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Schölkopf, B. (2003). "Learning with Local and Global Consistency." In *Advances in Neural Information Processing Systems 16 (NIPS 2003)*, 321–328. Cambridge, MA: MIT Press.

Chapter 12
Graphical Variable Clustering Action Set

Contents

Syntax **347**
The gVarCluster Action 348
Details **363**
Missing Values 363
Graphical Lasso 363
Graphical Representation of Precision Matrix 364
Preprocessing 364
Results 365
Examples **365**
Example 12.1: Analyzing United States Senate Voting Data 365
References **368**

Graphical Variable Clustering Action Set: Syntax

gvarcluster Action

CASL Syntax

gVarCluster.gvarcluster <**result**=*results*> <**status**=*rc*> /

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

collection={{

details=TRUE | FALSE,

*required parameter **name**="*string*",

*required parameter **vars**={*"variable-name-1"* <, *"variable-name-2"*, ...>}

}, {...}}

diagnostics={

eyecatcher="*string*"

}

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

exact=TRUE | FALSE

freq="*variable-name*"

```
inputs={{
    format="string",
    formattedLength=integer,
    label="string",
  *required parameter name="variable-name",
    nfd=integer,
    nfl=integer
}, {...}}
maxIter=64-bit-integer
maxMember=64-bit-integer
maxSteps=64-bit-integer
minCluster=64-bit-integer
multimember={{
    details=TRUE | FALSE,
  *required parameter name="string",
    noEffect=TRUE | FALSE,
    stdize=TRUE | FALSE,
  *required parameter vars={"variable-name-1" <, "variable-name-2", ...>},
    weight={"variable-name-1" <, "variable-name-2", ...>}
}, {...}}
nominals={{
    format="string",
    formattedLength=integer,
    label="string",
  *required parameter name="variable-name",
    nfd=integer,
    nfl=integer
}, {...}}
outCP={
  *required parameter casOut={
      caslib="string"
      compress=TRUE | FALSE
```

```
        indexVars={"variable-name-1" <, "variable-name-2", ...>}
        label="string"
        maxMemSize=64-bit-integer
        name="table-name"
        onDemand=TRUE | FALSE
        promote=TRUE | FALSE
        replace=TRUE | FALSE
        replication=integer
        timeStamp="string"
        where={"string-1" <, "string-2", ...>}
    },
    eps=double,
    list=TRUE | FALSE
}
outEdge={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
    replace=TRUE | FALSE,
    replication=integer,
    timeStamp="string",
    where={"string-1" <, "string-2", ...>}
}
outTree={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
```

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outVert={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

polynomial={{

```
      degree=integer,
      details=TRUE | FALSE,
      labelStyle={
          expand=TRUE | FALSE
          exponent="string"
          includeName=TRUE | FALSE
          productSymbol="NONE" | "string"
      },
      mDegree=integer,
    *required parameter name="string",
      noSeparate=TRUE | FALSE,
      standardize={
          method="MOMENTS" | "MRANGE" | "WMOMENTS"
          options="CENTER" | "CENTERSCALE" | "NONE" | "SCALE"
          prefix="NONE" | "string"
      },
    *required parameter vars={"variable-name-1" <, "variable-name-2", ...>}
  }, {...}}
  rho=double
  select="ADJBIC" | "CV" | "NONE" | "PENALIZED"
  stop=64-bit-integer
*required parameter table={
      caslib="string",
      computedOnDemand=TRUE | FALSE,
      computedVars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      computedVarsProgram="string",
```

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

singlePass=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

where="*where-expression*"

}

target="*string*"

xTol=*double*

;

Parameter Descriptions

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

changes the attributes of variables that are used in this action.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

collection={{*collection-1*} <, {*collection-2*}, ...>}

defines a set of variables that are treated as a single effect that has multiple degrees of freedom.

details=TRUE | FALSE

when set to True, requests a table that shows additional details that are related to this effect.

Default FALSE

* **name**="*string*"

specifies the name of the effect.

* **vars**={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies a set of variables that are treated as a single effect that has multiple degrees of freedom. The columns in the design matrix that are contributed by a collection effect are the design columns of its constituent variables in the order in which they appear in the definition of the collection effect.

diagnostics={*_diagnostics*}

eyecatcher="*string*"

specifies a quoted string that will be prefixed to any messages that are associated with this action invocation.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

exact=TRUE | FALSE

when set to True, performs graphical variable clustering without preprocessing by

thresholding the sample covariance into connected components. By default, the preprocessing step is performed.

Alias noblock

Default FALSE

freq="*variable-name*"

names the numeric variable that contains the frequency of occurrence for each observation.

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies variables to use for analysis.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

maxIter=*64-bit-integer*

specifies the maximum number of iterations for estimating the sparse precision covariance matrix by using coordinate descent.

Default 50

Range 1–100000

maxMember=*64-bit-integer*

stops the action when the number of members within any cluster is greater than or equal to the specified value.

Range 1–100000

maxSteps=*64-bit-integer*

specifies the maximum number of clustering steps.

Default 3

Range 1–50

minCluster=*64-bit-integer*

stops the action when the number of clusters is less than or equal to the specified value.

Default 3

Range 1–100000

multimember={{*multimember-1*} <, {*multimember-2*}, ...>}

uses one or more classification variables specified in the vars parameter in such a way that each observation can be associated with one or more levels of the union of the levels of the classification variables.

For more information about specifying the multimember parameter, see the common multimember parameter (Appendix A: Common Parameters).

nominals={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies nominal variables to use for analysis.

For more information about specifying the nominals parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

outCP={*OutputCPStatement*}

creates a data set that contains a symmetric matrix that depicts the covariances among variables and also creates a set of statistics about the input data set and variables.

* **casOut**={*casouttable*}

specifies the output table.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other

sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

eps=*double*

specifies an epsilon value such that matrix entries that have an absolute value smaller than epsilon are ignored in the output. You must specify the list parameter when you specify the eps parameter.

Default 0

Minimum value 0

list=TRUE | FALSE

when set to True, outputs the symmetric matrix in the list-of-lists (LIL) format.

Default FALSE

outEdge={*casouttable*}

creates a data set for use with the Hypergroup action in the tkhypgrp action library. This table contains the information that defines the edges in the network: _FROM_, _TO_ and _WEIGHT_.

For more information about specifying the outEdge parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outTree={*casouttable*}

creates a data set that depicts a tree diagram to display the hierarchical clustering results. The tree diagram can be plotted using the DENDROGRAM statement in the Graph

Template Language.

For more information about specifying the outTree parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outVert={*casouttable*}

creates a data set for use with the Hypergroup action in the tkhypgrp action library. This table contains the vertices in the network and their size.

For more information about specifying the outVert parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

polynomial={{*polynomial-1*} <, {*polynomial-2*}, ...>}

specifies a polynomial effect. All specified variables must be numeric. A design matrix column is generated for each term of the specified polynomial. By default, each of these terms is treated as a separate effect for the purpose of model building.

For more information about specifying the polynomial parameter, see the common polynomial parameter (Appendix A: Common Parameters).

rho=*double*

specifies the value of rho that determines the sequence of regulation parameters [the first power of rho, the second power of rho, and so on], that are used on sequential clustering steps.

Default 0.8

select="ADJBIC" | "CV" | "NONE" | "PENALIZED"

when set to True, performs graphical variable clustering without preprocessing by thresholding the sample covariance into connected components. By default, the preprocessing step is performed.

Default NONE

stop=*64-bit-integer*

requests that the action stop if the clustering results do not change in the previous number of consecutive step that is specified in this parameter.

Default 3

Range 2–100

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

*** name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

target="*string*"

specifies the target variable to use for analysis.

xTol=*double*

specifies the minimal absolute tolerance at which an iteration stops.

Default	0.001
Minimum value	1e-12

Details

The **gVarCluster** action analyzes relationships among variables in a multivariate analysis. It estimates the inverse covariance matrix, which encodes partial correlations between pairs of variables given the other variables. At each step, a regularization parameter is used to control the sparsity of the estimated inverse covariance matrix. Elements whose value is 0 imply conditional independence between variables, and the variables are divided into disjoint clusters. Tuning the regularization parameter from low to high increases the number of disconnected components and splits larger clusters into smaller ones. The divided clusters from sequential steps naturally form a hierarchical structure during the analysis.

Missing Values

Any observation that has missing values for the frequency or effects is excluded from the analysis. Observations that have a frequency less than 1 are also excluded.

Graphical Lasso

The **gVarCluster** action uses the graphical lasso of Friedman, Hastie, and Tibshirani (2008) to estimate the inverse covariance matrix. Assume that the observations have a multivariate Gaussian distribution with mean μ and covariance Σ. Zeros in the inverse covariance matrix (also called the precision matrix) correspond to conditional independencies among the variables, given other variables. The precision matrix is estimated by solving a maximum likelihood problem with an added l_1-norm penalty to increase the sparsity of the precision matrix.

Suppose you have n multivariate normal observations of dimension p, $\mathbf{X} \sim \mathbf{N}(\mu, \Sigma)$. Let $\Theta = \Sigma^{-1}$, and let S be the empirical covariance matrix. The problem is to maximize the following penalized log likelihood over nonnegative definite matrices Θ:

$$\log\det\Theta - \mathrm{Trace}(S\Theta) - \lambda||\Theta||_1$$

Here det denotes the determinant and $||\Theta||_1$ is the L_1 norm, the sum of the absolute values of the elements of Σ^{-1}. The scalar parameter λ controls the size of the penalty. The penalty term is a proxy for the number of nonzero elements in Θ.

Let W be the estimate of Σ. Banerjee, El Ghaoui, and d'Aspremont (2008) show that the preceding problem is convex and can be solved by optimizing over each row and the corresponding column of W in a block coordinate descending fashion. They further show that the problem is equivalent to the following dual problem:

$$\min\{\frac{1}{2}||W_{11}^{1/2}\beta - b||^2 + \lambda||\beta||_1\}$$

where $b = W_{11}^{-1/2}$ and W is partitioned as follows:

$$W = \begin{bmatrix} W_{11} & w_{12} \\ w_{12}^T & w_{22} \end{bmatrix}$$

Permute the rows and columns so that the one to be updated is always the last. Solve the dual problem, and then update the estimate of W after each stage.

The graphical lasso algorithm solves the dual problem as follows:

1. Start with $W = S + \lambda I$, where S is partitioned as follows:

$$S = \begin{bmatrix} S_{11} & s_{12} \\ s_{12}^T & s_{22} \end{bmatrix}$$

   The diagonal of W remains unchanged in the following steps.

2. For each $j = 1, 2, \ldots p, \ldots$, solve the lasso problem by coordinate descent (Friedman, Hastie, and Tibshirani 2008), which takes as input the inner products W_{11} and s_{12}. This produces a $p - 1$ vector solution $\hat{\beta}$. Fill in the corresponding row and column of W using $w_{12} = W_{11}\hat{\beta}$.

3. Continue until convergence.

Graphical Representation of Precision Matrix

Elements that have the value 0 in a precision matrix imply conditional independence between variables given the other variables in the network. So the estimate of a precision matrix can recover the conditional independence structure, which can be displayed in an undirected graph. If node X_i is connected to node X_j by an arc, then X_i is called a neighbor of X_j. If X_i is connected to X_k through some chain of arcs, then X_i is called a connected component of X_k.

The regularization parameter in penalized log likelihood can be seen as a quasi-measure for the strength of connection between X_i and X_j, because as λ increases from small to large, some connections drop out, and the connections that drop out should be weaker than those that stay in.

Preprocessing

Prior to estimating the precision matrix by using the graphical lasso at each step, the **`gVarCluster`** action creates a threshold for the entries of the sample covariance matrix at a regularization parameter λ and decomposes the covariance matrix into connected components. In other words, if an entry of sample covariance is less than λ in its absolute value, there is no graph edge between the variables. This graph is called the thresholded sample covariance graph. The thresholded covariance matrix can be reordered and decomposed into connected components.

If you already have the solution to maximize the penalized log likelihood, you can also present the edge set via a symmetric 0-1 matrix. Let $\hat{\Theta}^{\lambda}$ denote the solution at λ. The element (i, j) in the matrix is 1 when $\hat{\Theta}^{\lambda}_{ij} \neq 0$ and $i \neq j$ and is 0 otherwise. This graph is called the estimated concentration graph.

Mazumder and Hastie (2012) prove that the vertex partition of connected components that is obtained through thresholding the covariance matrix is exactly equal to the one that is obtained from the estimated concentration graph.

Results

The results that the **gVarCluster** action produces are stored in tables. You can access these tables by their names, which are shown in Table 12.1. You use these table names when you specify the **display** parameter.

Table 12.1 Results Tables Produced by the **gVarCluster** Action

Table Name	Description	Parameter
ClassInfo	Level information for classification variables	Default
ClusterSummary	Summary of clustering results	Default
NObs	Number of observations read and used; number of events and trials, if applicable	Default

Examples

Example 12.1: Analyzing United States Senate Voting Data

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example analyzes United States Senate voting records data from the 106th Congress, 1999–2000. The data set contains 100 variables, which correspond to 100 senators. Each of the 542 samples is a bill that was put to a vote. The votes are recorded as 0 for a No vote and 1 for a Yes vote.

Download the Senate data set to the Work library from the Github repository: https://github.com/sassoftware/sas-viya-machine-learning/tree/master/data/senate. You can then load the Work.Senate data set into your CAS session by naming your CAS engine libref in the first statement of the following DATA step. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.senate;
    set work.senate;
run;
```

The following statements run the **gVarCluster** action:

```
proc cas;
    loadactionset "gvarcluster";
    action gvarcluster /
    table={name="SENATE"}
    inputs={ "AbrahamRMI" "AkakaDHI" "AllardRCO" "AshcroftRMO" "BaucusDMT"
     "BayhDIN" "BennettRUT" "BidenDDE" "BingamanDNM" "BondRMO" "BoxerDCA"
     "BreauxDLA" "BrownbackRKS" "BryanDNV" "BunningRKY" "BurnsRMT" "ByrdDWV"
     "CampbellRCO" "ChafeeRRI" "ClelandDGA" "CochranRMS" "CollinsRME"
     "ConradDND" "CoverdellRGA" "CraigRID" "CrapoRID" "DaschleDSD"
     "DeWineROH" "DoddDCT" "DomeniciRNM" "DorganDND" "DurbinDIL" "EdwardsDNC"
     "EnziRWY" "FeingoldDWI" "FeinsteinDCA" "FitzgeraldRIL" "FristRTN"
     "GortonRWA" "GrahamDFL" "GrammRTX" "GramsRMN" "GrassleyRIA" "GreggRNH"
     "HagelRNE" "HarkinDIA" "HatchRUT" "HelmsRNC" "HollingsDSC" "HutchinsonRAR"
     "HutchisonRTX" "InhofeROK" "InouyeDHI" "JeffordsRVT" "JohnsonDSD"
     "KennedyDMA" "KerreyDNE" "KerryDMA" "KohlDWI" "KylRAZ" "LandrieuDLA"
     "LautenbergDNJ" "LeahyDVT" "LevinDMI" "LiebermanDCT" "LincolnDAR"
     "LottRMS" "LugarRIN" "MackRFL" "McCainRAZ" "McConnellRKY" "MikulskiDMD"
     "MoynihanDNY" "MurkowskiRAK" "MurrayDWA" "NicklesROK" "ReedDRI" "ReidDNV"
     "RobbDVA" "RobertsRKS" "RockefellerDWV" "RothRDE" "SantorumRPA"
     "SarbanesDMD" "SchumerDNY" "SessionsRAL" "ShelbyRAL" "SmithRNH" "SmithROR"
     "SnoweRME" "SpecterRPA" "StevensRAK" "ThomasRWY" "ThompsonRTN" "ThurmondRSC"
     "TorricelliDNJ" "VoinovichROH" "WarnerRVA" "WellstoneDMN" "WydenDOR"}
    nominals={ "AbrahamRMI" "AkakaDHI" "AllardRCO" "AshcroftRMO" "BaucusDMT"
     "BayhDIN" "BennettRUT" "BidenDDE" "BingamanDNM" "BondRMO" "BoxerDCA"
     "BreauxDLA" "BrownbackRKS" "BryanDNV" "BunningRKY" "BurnsRMT" "ByrdDWV"
     "CampbellRCO" "ChafeeRRI" "ClelandDGA" "CochranRMS" "CollinsRME"
     "ConradDND" "CoverdellRGA" "CraigRID" "CrapoRID" "DaschleDSD"
     "DeWineROH" "DoddDCT" "DomeniciRNM" "DorganDND" "DurbinDIL" "EdwardsDNC"
     "EnziRWY" "FeingoldDWI" "FeinsteinDCA" "FitzgeraldRIL" "FristRTN"
     "GortonRWA" "GrahamDFL" "GrammRTX" "GramsRMN" "GrassleyRIA" "GreggRNH"
     "HagelRNE" "HarkinDIA" "HatchRUT" "HelmsRNC" "HollingsDSC" "HutchinsonRAR"
     "HutchisonRTX" "InhofeROK" "InouyeDHI" "JeffordsRVT" "JohnsonDSD"
     "KennedyDMA" "KerreyDNE" "KerryDMA" "KohlDWI" "KylRAZ" "LandrieuDLA"
     "LautenbergDNJ" "LeahyDVT" "LevinDMI" "LiebermanDCT" "LincolnDAR"
     "LottRMS" "LugarRIN" "MackRFL" "McCainRAZ" "McConnellRKY" "MikulskiDMD"
     "MoynihanDNY" "MurkowskiRAK" "MurrayDWA" "NicklesROK" "ReedDRI" "ReidDNV"
     "RobbDVA" "RobertsRKS" "RockefellerDWV" "RothRDE" "SantorumRPA"
     "SarbanesDMD" "SchumerDNY" "SessionsRAL" "ShelbyRAL" "SmithRNH" "SmithROR"
     "SnoweRME" "SpecterRPA" "StevensRAK" "ThomasRWY" "ThompsonRTN" "ThurmondRSC"
     "TorricelliDNJ" "VoinovichROH" "WarnerRVA" "WellstoneDMN" "WydenDOR"}
    maxsteps=5
    rho=0.800000
    outtree={name="TREE", replace=1 }
    outedge={name="EDGE", replace=1 }
    outvert={name="VERT", replace=1 }
    ;
run;
```

The **table** parameter names the input data table to be analyzed. The **inputs** parameter lists all the input variables to be used in the training. The **nominals** parameter lists the nominal input variables to be used in the training.

By specifying the value of the **maxsteps** parameter to be more than 1, you can let the **gVarCluster** action perform variable clustering at multiple ρ values, which decrease at each step. A **rho** parameter value of 0.8 specifies the initial ρ value for step 1, and then ρ is 0.8^2 for step 2, 0.8^3 for step 3, and so on. The regularization parameter ρ controls the sparsity of the estimation of the inverse covariance matrix at each step, and a sequence of steps yields a hierarchical structure of clustering. The hierarchies that the **gVarCluster** action provides are interpretable with tree output. The **outtree** parameter saves the tree to a CAS table, and you can plot the dendrogram by using the SAS Graph Template Language (GTL). The **gVarCluster** action also outputs the edge and vertex information for representing the partial correlations in an undirected graph. The **outedge** parameter saves the edges in the network to a CAS table named mycas.edge, and the **outvert** parameter saves the vertices and their sizes in the network to CAS table named mycas.vert.

The following statements download the tree table to the Work library:

```
data work.tree (drop = last);
    set mycas.tree;
    last = scan(_CHILD_, -1);
    if last=0   then delete;
run;
```

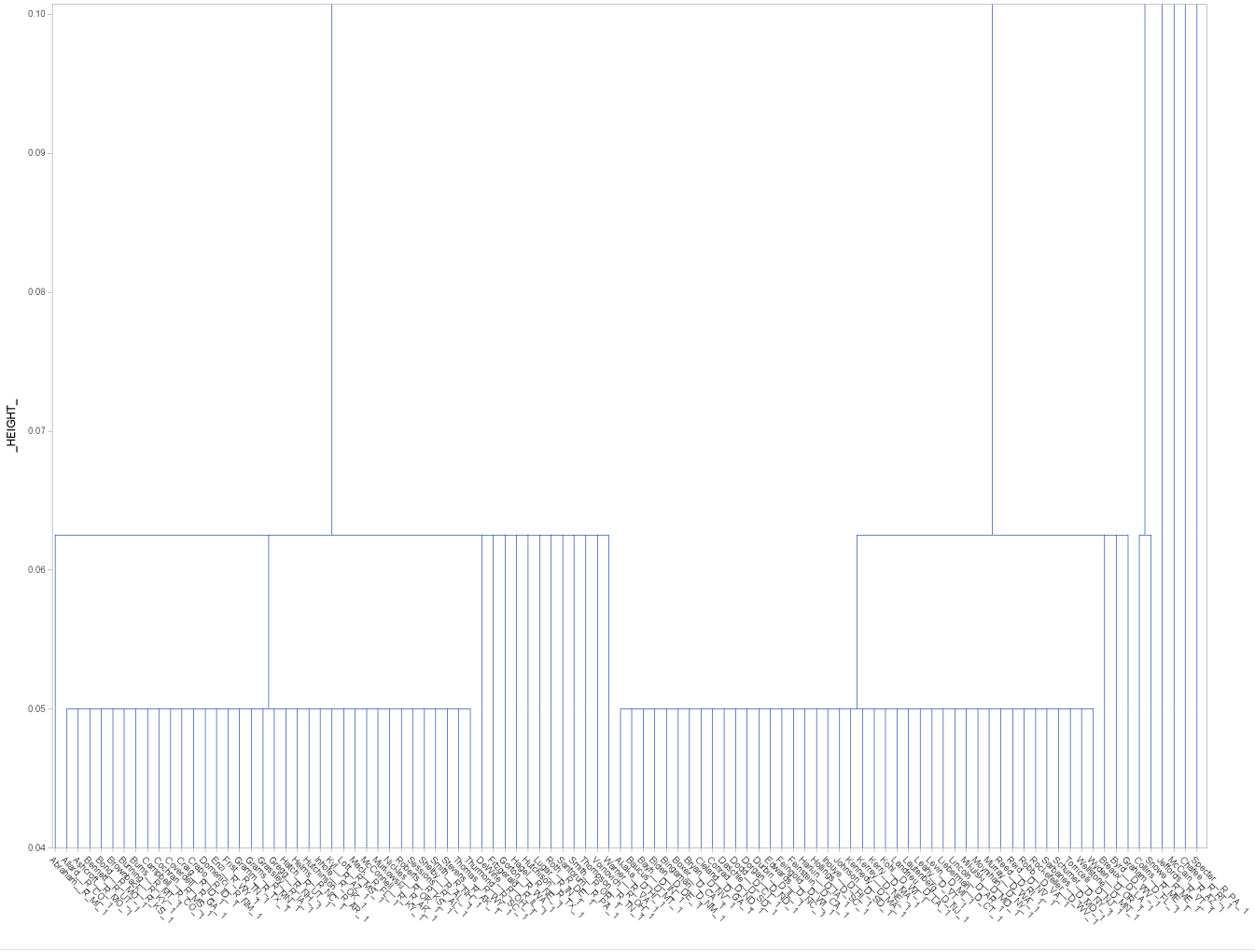
The following code uses the DENDROGRAM statement in GTL to define the parent node, the child node, and the height in the dendrogram. The dendrogram is then plotted using the SGRENDER procedure.

```
proc template;
    define statgraph dendrogram;
    begingraph;
    layout overlay/ yaxisopts=(linearopts=(viewmax=0.1));
    dendrogram nodeID=_CHILD_ parentID=_PARENT_ clusterheight=_HEIGHT_;
    endlayout;
    endgraph;
end;run;

ods graphics /width =2300px;
proc sgrender data=work.tree1 template=dendrogram;
run;
```

Output 12.1.1 shows that there are two large clusters in the dendrogram. All the Democratic senators' votes are united and in one cluster. Most of the Republican senators' votes are in the other cluster, but there are some small cliques.

Output 12.1.1 Senate Voting Records from the 106th Congress



References

Banerjee, O., El Ghaoui, L., and d'Aspremont, A. (2008). "Model Selection through Sparse Maximum Likelihood Estimation for Multivariate Gaussian or Binary Data." *Journal of Machine Learning Research* 9:485–516.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2008). "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics* 9:432–441.

Mazumder, R., and Hastie, T. (2012). "Exact Covariance Thresholding into Connected Components for Large-Scale Graphical Lasso." *Journal of Machine Learning Research* 13:781–794.

Chapter 13
Image Action Set

Contents

Syntax **371**
The augmentImages Action 372
The compareImages Action 382
The condenseImages Action 397
The extractDetectedObjects Action 404
The fetchImages Action 410
The flattenImages Action 416
The loadImages Action 422
The matchImages Action 426
The processImages Action 433
The saveImages Action 445
The summarizeImages Action 449
Details **454**
About the Image Action Set 454
The augmentImages Action 454
The compareImages Action 454
The condenseImages Action 455
The extractDetectedObjects Action 455
The fetchImages Action 456
The flattenImageTable action 456
The loadImages Action 457
The matchImages Action 459
The processImages Action 460
The saveImages Action 463
The summarizeImages Action 463
Examples **464**
Example 13.1: Use Sliding Window without Mutating Using the augmentImages Action 464
Example 13.2: Use Sliding Window and Mutate Using the augmentImages Action . . 464
Example 13.3: Create a Patch Using the augmentImages Action 464
Example 13.4: Create a Patch and Mutate Using the augmentImages Action 464
Example 13.5: Mutate without Creating a Patch Using the augmentImages Action . . 464
Example 13.6: Introduce Randomness Using the augmentImages Action 464
Example 13.7: Comparing Images 464
Example 13.8: Condense Images 465
Example 13.9: Condense Images Using the groupedChannels Parameter 466

Example 13.10: Condense Images Using the depth Parameter 466
Example 13.11: Condense Images Using the numberOfChannels Parameter 466
Example 13.12: Condense Images Using the inputs Parameter 466
Example 13.13: Visualize Detections 466
Example 13.14: Visualize Only One Detection 466
Example 13.15: Visualize Detections in the COCO Format 466
Example 13.16: Visualize Detections in the YOLO Format 466
Example 13.17: Cropping Detections and Saving Them in Multiple Columns Using the extractDetectedObjects Action 466
Example 13.18: Cropping Detections and Saving Them in Multiple Rows Using the extractDetectedObjects Action 467
Example 13.19: Fetch Images 467
Example 13.20: Fetch Only Two Images 467
Example 13.21: Copying Variables While Fetching Images 467
Example 13.22: Process Images and Then Fetch Them 467
Example 13.23: Fetch Images Using SortBy 467
Example 13.24: Simple Flattening 467
Example 13.25: Transpose While Flattening an Image Table 467
Example 13.26: Group Channels While Flattening an Image Table 467
Example 13.27: Loading an Image 468
Example 13.28: Loading a Directory Tree and Metadata 468
Example 13.29: Recursively Loading All DICOM Series under a Directory 469
Example 13.30: Loading and Displaying a 3-D Image 469
Example 13.31: Loading DICOM Attributes Along with DICOM Images 469
Example 13.32: Template Matching 469
Example 13.33: Descriptor Matching 469
Example 13.34: Resizing Images Using the processImages Action 469
Example 13.35: Creating a Patch from an Input Image Using the processImages Action 470
Example 13.36: Calculating Image Derivatives Using the processImages Action . . . 470
Example 13.37: Detecting Edges with Canny Using the processImages Action 470
Example 13.38: Applying the Laplace Operator Using the processImages Action . . . 470
Example 13.39: Normalizing Images Using the processImages Action 470
Example 13.40: Applying a Threshold Using the processImages Action 470
Example 13.41: Converting Color Spaces Using the processImages Action 470
Example 13.42: Performing a Morphological Transformation Using the processImages Action 470
Example 13.43: Blurring Images Using the processImages Action 471
Example 13.44: Reducing Noise and Detecting Edges Using the processImages Action 471
Example 13.45: Creating a Custom Filter and Applying It Using the processImages Action 471
Example 13.46: Performing Histogram Equalization Using the processImages Action 471
Example 13.47: Linear Transformation Using the processImages Action 471
Example 13.48: Saving Images 471
Example 13.49: Saving Image Labels 472

Example 13.50: Converting DICOM Series Into Sequences of JPEG Images 472
Example 13.51: Summarize an Image Table 472
Example 13.52: Summarize an Image Table with Custom Image Column 472
References **472**

Image Action Set: Syntax

augmentImages Action

CASL Syntax

image.augmentImages <**result**=*results*> <**status**=*rc*> /

*__required parameter__ **casOut**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

cropList={{

height=*64-bit-integer*,

mask=TRUE | FALSE,

mutations={

colorJittering=TRUE | FALSE

colorShifting=TRUE | FALSE

darken=TRUE | FALSE

horizontalFlip=TRUE | FALSE

invertPixels=TRUE | FALSE

lighten=TRUE | FALSE

pyramidDown=TRUE | FALSE

```
            pyramidUp=TRUE | FALSE
            rotateLeft=TRUE | FALSE
            rotateRight=TRUE | FALSE
            sharpen=TRUE | FALSE
            verticalFlip=TRUE | FALSE
        },
        outputHeight=64-bit-integer,
        outputWidth=64-bit-integer,
        stepSize=64-bit-integer,
        sweepImage=TRUE | FALSE,
        useWholeImage=TRUE | FALSE,
        width=64-bit-integer,
        x=64-bit-integer,
        y=64-bit-integer
    }, {...}}
    decode=TRUE | FALSE
    image="variable-name"
    seed=64-bit-integer
*required parameter table={
        caslib="string",
        computedOnDemand=TRUE | FALSE,
        computedVars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        computedVarsProgram="string",
        dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
        groupBy={{
```

```
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        groupByMode="NOSORT" | "REDISTRIBUTE",
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
      *required parameter name="table-name",
        onDemand=TRUE | FALSE,
        orderBy={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        singlePass=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
    }
    writeRandomly=TRUE | FALSE
;
```

Parameter Descriptions

* **casOut**={*casouttable*}

specifies the settings for an output table.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the variables to copy from the input table to the output table.

Alias copyVar

cropList={{*cropOptions-1*} <, {*cropOptions-2*}, ...>}

specifies the list of cropping options.

The cropOptions value can be one or more of the following:

height=*64-bit-integer*

specifies the height (number of rows) of a subimage.

Alias h

mask=TRUE | FALSE

when set to True, masks the region of interest.

Default FALSE

mutations={*mutations*}

specifies the list of mutations that can be invoked.

colorJittering=TRUE | FALSE

when set to True, applies color jittering to an input image.

Default FALSE

colorShifting=TRUE | FALSE

when set to True, randomly changes pixel intensity values of an input image.

Default FALSE

darken=TRUE | FALSE

when set to True, darkens the input image.

Alias darker

Default FALSE

horizontalFlip=TRUE | FALSE

when set to True, flips the input image horizontally.

Alias hFlip

Default FALSE

invertPixels=TRUE | FALSE

when set to True, inverts all pixels in the input image.

Alias bitwiseNot

Default FALSE

lighten=TRUE | FALSE

when set to True, lightens the input image.

Alias lighter

Default FALSE

pyramidDown=TRUE | FALSE

when set to True, downsamples and then blurs the input image.

Alias pyrDown

Default FALSE

pyramidUp=TRUE | FALSE

when set to True, upsamples and then blurs the input image.

Alias pyrUp

Default FALSE

rotateLeft=TRUE | FALSE

when set to True, rotates the input image to the left.

Alias rotLeft

Default FALSE

rotateRight=TRUE | FALSE

when set to True, rotates the input image to the right.

Alias rotRight

Default FALSE

sharpen=TRUE | FALSE

when set to True, sharpens the input image.

Default FALSE

verticalFlip=TRUE | FALSE

when set to True, flips the input image vertically.

Alias vFlip

Default FALSE

outputHeight=*64-bit-integer*

specifies the height (number of rows) of the output subimage.

Alias outH

outputWidth=*64-bit-integer*

specifies the width (number of columns) of the output subimage.

Alias outW

stepSize=*64-bit-integer*

specifies the step size in the sliding window.

Alias step

Default 1

sweepImage=TRUE | FALSE

when set to True, sweeps (sliding window) the whole image.

Default FALSE

useWholeImage=TRUE | FALSE

when set to True, uses the whole image (the image is not cropped).

Default FALSE

width=*64-bit-integer*

specifies the width (number of columns) of a subimage.

Alias w

x=*64-bit-integer*

specifies the X location.

Default 0

y=*64-bit-integer*

specifies the Y location.

Default 0

decode=TRUE | FALSE

when set to True, decompresses images before writing them to the output table.

Alias decompress

Default FALSE

image="*variable-name*"

specifies the name of the column that contains image binaries.

seed=*64-bit-integer*

specifies a seed for writing images randomly.

* **table**={*castable*}

specifies the table name, caslib, and other common parameters.

Alias imageTable

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

writeRandomly=TRUE | FALSE

when set to True, randomly writes the resulting image to the output table.

Alias random

Default FALSE

Image Action Set: Syntax

compareImages Action

CASL Syntax

image.compareImages <**result**=*results*> <**status**=*rc*> /

*__required parameter__ **casOut**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

maximum=*double*

method="PSNR" | "SSIM"

minimum=*double*

pairAll=TRUE | FALSE

pairOnPath=TRUE | FALSE

pairReferenceOn="*variable-name*"

pairSourceOn="*variable-name*"

*__required parameter__ **referenceImages**={

dimension="*variable-name*",

id="*variable-name*",

image="*variable-name*",

imageFormat="*variable-name*",

label="*variable-name*",

path="*variable-name*",

resolution="*variable-name*",

size="*variable-name*",

*required parameter **table**={

caslib="*string*"

computedOnDemand=TRUE | FALSE

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

computedVarsProgram="*string*"

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

groupByMode="NOSORT" | "REDISTRIBUTE"

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

*required parameter **name**="*table-name*"

onDemand=TRUE | FALSE

orderBy={{

```
                format="string",
                formattedLength=integer,
                label="string",
              *required parameter name="variable-name",
                nfd=integer,
                nfl=integer
            }, {...}}
            singlePass=TRUE | FALSE
            vars={{
                format="string",
                formattedLength=integer,
                label="string",
              *required parameter name="variable-name",
                nfd=integer,
                nfl=integer
            }, {...}}
            where="where-expression"
        },
        type="variable-name"
}
  separateChannels=TRUE | FALSE
*required parameter sourceImages={
        dimension="variable-name",
        id="variable-name",
        image="variable-name",
        imageFormat="variable-name",
        label="variable-name",
        path="variable-name",
        resolution="variable-name",
        size="variable-name",
      *required parameter table={
            caslib="string"
            computedOnDemand=TRUE | FALSE
```

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

computedVarsProgram="*string*"

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

groupByMode="NOSORT" | "REDISTRIBUTE"

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

*required parameter **name**="*table-name*"

onDemand=TRUE | FALSE

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

```
        singlePass=TRUE | FALSE
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}}
        where="where-expression"
    },
     type="variable-name"
}
;
```

Parameter Descriptions

*** casOut**={*casouttable*}

specifies the settings for an output table.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

maximum=*double*

specifies the maximum comparison value to report.

Default MACBIG

method="PSNR" | "SSIM"

specifies the name of the image comparison method.

Default SSIM

PSNR

uses the peak signal-to-noise ratio method

SSIM

uses the structural similarity index method

minimum=*double*

specifies the minimum comparison value to report.

Default -MACBIG

pairAll=TRUE | FALSE

when set to True, compares all images from the source table with all images from the reference table.

Default FALSE

pairOnPath=TRUE | FALSE

when set to True, specifies that the pairing variables are file paths.

Alias isPairVariablePath

Default TRUE

pairReferenceOn="*variable-name*"

specifies the name of the column in the reference images table that contains the pairing variable.

pairSourceOn="*variable-name*"

specifies the name of the column in the source images table that contains the pairing variable.

* **referenceImages**={*imageTableIn*}

specifies the parameter list of the input image table that contains reference images.

Alias referenceImage

dimension="*variable-name*"

specifies the name of the column that contains dimensions of images.

Alias dimensionVar

Default "_dimension_"

id="*variable-name*"

specifies the name of the variable that identifies each image.

Alias idVar

Default "_id_"

image="*variable-name*"

specifies the name of the column that contains image binaries.

Alias imageVar

Default "_image_"

imageFormat="*variable-name*"

specifies the name of the column that contains formats of image binaries.

Alias imageFormatVar

Default "_imageFormat_"

label="*variable-name*"

specifies the name of the column that contains labels of images.

Alias labelVar

Default "_label_"

path="*variable-name*"

specifies the name of the column that contains file paths.

Alias pathVar

Default "_path_"

resolution="*variable-name*"

specifies the name of the column that contains resolutions of images.

Alias resolutionVar

Default "_resolution_"

size="*variable-name*"

specifies the name of the column that contains byte lengths of image binaries.

Alias sizeVar

Default "_size_"

* **table**={*castable*}

specifies the input table that contains image data.

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with

variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

type="*variable-name*"

specifies the name of the column that contains the image type, which is shown as jpg, bmp, and so on.

Alias typeVar

Default "_type_"

separateChannels=TRUE | FALSE

when set to True, compares each channel separately.

Default TRUE

* **sourceImages**={*imageTableIn*}

specifies the list of parameters for the input image table that contains source images.

Alias sourceImage

dimension="*variable-name*"

specifies the name of the column that contains dimensions of images.

Alias dimensionVar

Default "_dimension_"

id="*variable-name*"

specifies the name of the variable that identifies each image.

Alias idVar

Default "_id_"

image="*variable-name*"

specifies the name of the column that contains image binaries.

Alias imageVar

Default "_image_"

imageFormat="*variable-name*"

specifies the name of the column that contains formats of image binaries.

Alias imageFormatVar

Default "_imageFormat_"

label="*variable-name*"

specifies the name of the column that contains labels of images.

Alias labelVar

Default "_label_"

path="*variable-name*"

specifies the name of the column that contains file paths.

Alias pathVar

Default "_path_"

resolution="*variable-name*"

specifies the name of the column that contains resolutions of images.

Alias resolutionVar

Default "_resolution_"

size="*variable-name*"

specifies the name of the column that contains byte lengths of image binaries.

Alias sizeVar

Default "_size_"

* **table**={*castable*}

specifies the input table that contains image data.

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

type="*variable-name*"

specifies the name of the column that contains the image type, which is shown as jpg, bmp, and so on.

Alias typeVar

Default "_type_"

Image Action Set: Syntax

condenseImages Action

CASL Syntax

image.condenseImages <**result**=*results*> <**status**=*rc*> /

*__required parameter__ **casOut**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

depth="BIT32" | "BIT64" | "BIT8" | *64-bit-integer*

groupedChannels=TRUE | FALSE

height=*64-bit-integer*

inputs={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

numberOfChannels="COLOR_IMAGE" | "GRAY_SCALE_IMAGE" | *64-bit-integer*

*required parameter **table**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

```
            nfl=integer
        }, {...}},
        singlePass=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
    }
    width=64-bit-integer
;
```

Parameter Descriptions

*** casOut**={*casouttable*}

specifies the settings for an output table.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the variables to be copy from the input table to the output table.

Alias copyVar

depth="BIT32" | "BIT64" | "BIT8" | *64-bit-integer*

Specifies the depth of a pixel of the output images.

groupedChannels=TRUE | FALSE

When set to True, assumes the pixels of images in the input table is grouped (for example, BB...GG...RR...).

Alias grouped

Default FALSE

height=*64-bit-integer*

Specifies the height of the output images.

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the input variables to use while constructing the images. If this parameter is not set, then all numeric columns are used.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

numberOfChannels="COLOR_IMAGE" | "GRAY_SCALE_IMAGE" | *64-bit-integer*

Specifies the number of channels of the output images.

Alias c

* **table**={*castable*}

specifies the table name, caslib, and other common parameters.

Alias imageTable

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common

casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

width=*64-bit-integer*

Specifies the width of the output images.

Image Action Set: Syntax

extractDetectedObjects Action

CASL Syntax

```
image.extractDetectedObjects <result=results> <status=rc> /
   *required parameter casOut={
         caslib="string",
         compress=TRUE | FALSE,
         indexVars={"variable-name-1" <, "variable-name-2", ...>},
         label="string",
         maxMemSize=64-bit-integer,
         name="table-name",
         onDemand=TRUE | FALSE,
         promote=TRUE | FALSE,
         replace=TRUE | FALSE,
         replication=integer,
         timeStamp="string",
         where={"string-1" <, "string-2", ...>}
   }
    coordType="COCO" | "RECT" | "YOLO"
    extractType="CROP" | "HIGHLIGHT"
    image="variable-name"
    maxObjects=64-bit-integer
    outputFormat="MULTIPLE_COLUMNS" | "MULTIPLE_ROWS"
   *required parameter table={
         caslib="string",
         computedOnDemand=TRUE | FALSE,
         computedVars={{
              format="string",
              formattedLength=integer,
              label="string",
```

```
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
        }, {...}},
      computedVarsProgram="string",
      dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
        parameters},
     *required parameter name="table-name",
      onDemand=TRUE | FALSE,
      singlePass=TRUE | FALSE,
      vars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
        }, {...}},
      where="where-expression"
  }
;
```

Parameter Descriptions

*** casOut**={*casouttable*}

specifies the settings for an output table.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

coordType="COCO" | "RECT" | "YOLO"

specifies the type of input format to use for the bounding boxes around the detected objects.

Default RECT

COCO

uses the Common Objects in Context format for the coordinates of a bounding box around an object detection.

RECT

uses the rectangular format for the coordinates of a bounding box around an object detection.

YOLO

uses the "you only look once" format for the coordinates of a bounding box around an object detection.

extractType="CROP" | "HIGHLIGHT"

specifies the type of extract action to perform.

Default HIGHLIGHT

CROP

performs the crop action while extracting object detections.

HIGHLIGHT

performs the highlight action while extracting object detections.

image="*variable-name*"

specifies the name of the column that contains image binaries.

maxObjects=*64-bit-integer*

specifies the maximum number of detected objects to extract from the image data in the input table.

Default 10

Minimum value 0

outputFormat="MULTIPLE_COLUMNS" | "MULTIPLE_ROWS"

specifies whether to save the cropped detections from a single image in multiple columns and one row or in one column and multiple rows in the output table.

Default MULTIPLE_ROWS

MULTIPLE_COLUMNS

saves the cropped object detections in multiple columns and one row in the output table.

MULTIPLE_ROWS

saves the cropped object detections in one column and multiple rows in the output table.

* **table**={*castable*}

specifies the table name, caslib, and other common parameters.

Alias imageTable

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

Image Action Set: Syntax

fetchImages Action

CASL Syntax

```
image.fetchImages <result=results> <status=rc> /
    fetchImagesVars={"variable-name-1" <, "variable-name-2", ...>}
    from=64-bit-integer
    image="variable-name"
    seed=64-bit-integer
    sortBy={{
        formatted="FORMATTED" | "RAW",
       *required parameter name="variable-name",
        order="ASCENDING" | "DESCENDING"
    }, {...}}
  *required parameter table={
        caslib="string",
        computedOnDemand=TRUE | FALSE,
        computedVars={{
            format="string",
            formattedLength=integer,
            label="string",
           *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        computedVarsProgram="string",
        dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
        groupBy={{
            format="string",
            formattedLength=integer,
            label="string",
           *required parameter name="variable-name",
```

```
            nfd=integer,
            nfl=integer
        }, {...}},
    groupByMode="NOSORT" | "REDISTRIBUTE",
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    orderBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
  to=64-bit-integer
;
```

Parameter Descriptions

fetchImagesVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the variables to copy from the input table to the output table.

Alias fetchVars

from=*64-bit-integer*

specifies the ordinal position of the first row to return.

Default 0

image="*variable-name*"

specifies the name of the column that contains image binaries.

seed=*64-bit-integer*

specifies a seed for writing images randomly.

Default 0

sortBy={{*fetchorderbyelement-1*} <, {*fetchorderbyelement-2*}, ...>}

specifies the variables and variable settings for sorting results.

The fetchorderbyelement value can be one or more of the following:

Alias orderBy

formatted="FORMATTED" | "RAW"

specifies whether the variables are formatted or use raw values.

Default RAW

* **name**="*variable-name*"

specifies the name of the variable to use for sorting.

order="ASCENDING" | "DESCENDING"

specifies whether the variables are ordered in ascending or descending order.

Default ASCENDING

* **table**={*castable*}

specifies the table name, caslib, and other common parameters.

Alias imageTable

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

to=*64-bit-integer*

specifies the ordinal position of the last image row to return.

Default 20

Minimum value 0

Image Action Set: Syntax

flattenImageTable Action

CASL Syntax

image.flattenImageTable <**result**=*results*> <**status**=*rc*> /

*__required parameter__ **casOut**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

groupChannels=TRUE | FALSE

height=*64-bit-integer*

image="*variable-name*"

*__required parameter__ **table**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*
}, {...}},
computedVarsProgram="*string*",
dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},
groupBy={{
format="*string*",
formattedLength=*integer*,
label="*string*",
*required parameter **name**="*variable-name*",
nfd=*integer*,
nfl=*integer*
}, {...}},
groupByMode="NOSORT" | "REDISTRIBUTE",
importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},
*required parameter **name**="*table-name*",
onDemand=TRUE | FALSE,
orderBy={{
format="*string*",
formattedLength=*integer*,
label="*string*",
*required parameter **name**="*variable-name*",
nfd=*integer*,
nfl=*integer*
}, {...}},
singlePass=TRUE | FALSE,
vars={{
format="*string*",
formattedLength=*integer*,
label="*string*",
*required parameter **name**="*variable-name*",
nfd=*integer*,

nfl=*integer*
}, {...}},
where="*where-expression*"
}
transpose=TRUE | FALSE
width=*64-bit-integer*
;

Parameter Descriptions

* **casOut**={*casouttable*}

specifies the settings for an output table.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

groupChannels=TRUE | FALSE

when set to True, groups the channels of the images together (all the blue pixels, then all the green pixels, and then all the red pixels) before writing them to the output table.

Default FALSE

height=*64-bit-integer*

specifies the height of the output images.

image="*variable-name*"

specifies the name of the column that contains image binaries.

* **table**={*castable*}

specifies the table name, caslib, and other common parameters.

Alias imageTable

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that

are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

transpose=TRUE | FALSE

when set to True, transposes the images before writing them to the output table.

Default FALSE

width=*64-bit-integer*

specifies the width of the output images.

Image Action Set: Syntax

loadImages Action

CASL Syntax

image.loadImages <**result**=*results*> <**status**=*rc*> /

addColumns={"CHANNELCOUNT", "CHANNELTYPE", "DEPTH", "HEIGHT", "ORIENTATION", "POSITION", "SPACING", "SPAN", "WIDTH"} | {

dicomAttributes={

keywords={*"string-1"* <, *"string-2"*, ...>}

tags={*"string-1"* <, *"string-2"*, ...>}

},

general={"CHANNELCOUNT", "CHANNELTYPE", "DEPTH", "HEIGHT", "ORIENTATION", "POSITION", "SPACING", "SPAN", "WIDTH"}

}

***required parameter** **casOut**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

caslib="*string*"

decode=TRUE | FALSE

distribution={**type**="RANDOM" | "ROUNDROBIN", *type-specific-parameters*}

labelLevels=*integer*

***required parameter** **path**="*string*"

pathIsList=TRUE | FALSE

quiet=TRUE | FALSE

recurse=TRUE | FALSE

repeat=TRUE | FALSE

series={

dicom=TRUE | FALSE,

reverse=TRUE | FALSE

}

;

Parameter Descriptions

addColumns={"CHANNELCOUNT", "CHANNELTYPE", "DEPTH", "HEIGHT", "ORIENTATION", "POSITION", "SPACING", "SPAN", "WIDTH"} | {*extendedAddColumnsParms*}

specifies additional metadata columns to be added to the output table.

The extendedAddColumnsParms value can be one or more of the following:

dicomAttributes={*dicomAttributesParms*}

specifies parameters related to DICOM attributes.

The dicomAttributesParms value can be one or more of the following:

keywords={*"string-1"* <, *"string-2"*, ...>}

specifies DICOM keywords.

Requirement The specified values must be unique.

tags={*"string-1"* <, *"string-2"*, ...>}

specifies DICOM tags.

Requirement The specified values must be unique.

general={"CHANNELCOUNT", "CHANNELTYPE", "DEPTH", "HEIGHT", "ORIENTATION", "POSITION", "SPACING", "SPAN", "WIDTH"}

specifies general columns applicable across image formats.

Range 1–9 values

Requirement The specified values must be unique.

* **casOut**={*casouttable*}

specifies the settings for an output table.

For more information about specifying the casOut parameter, see the common casouttable

parameter (Appendix A: Common Parameters).

caslib="*string*"

specifies the name of the caslib to load images from.

decode=TRUE | FALSE

when set to True, writes decompressed images and metadata to the output table.

Alias decompress

Default FALSE

distribution={type="RANDOM" | "ROUNDROBIN", *type-specific-parameters*}

specifies the algorithm and parameters to be used to distribute loaded images to worker nodes.

The value that you specify for type determines the other parameters that apply.

Default ROUNDROBIN

labelLevels=*integer*

specifies the maximum number of directory levels to include in the label.

Default 0

* **path**="*string*"

specifies the file or directory path to load images from.

Alias source

pathIsList=TRUE | FALSE

when set to True, the path parameter specifies a list of paths to image files.

Default FALSE

quiet=TRUE | FALSE

when set to True, suppresses notes and warnings for successful operations.

Alias silent

Default FALSE

recurse=TRUE | FALSE

when set to True, loads images in subdirectories recursively.

Default FALSE

repeat=TRUE | FALSE

when set to True, duplicates identical table rows on all worker nodes.

Alias duplicate

Default FALSE

series={*loadSeriesParms*}

specifies the parameters to be used to load directory trees as image series.

dicom=TRUE | FALSE

when set to True, loads directory trees as DICOM series.

Default FALSE

reverse=TRUE | FALSE

when set to True, reverses the order of the image series.

Default FALSE

Parameters for type="RANDOM"

seed=*integer*

specifies the seed for the pseudorandom number generator.

Default TKZ_NEW_SEED

Parameters for type="ROUNDROBIN"

No parameters apply when you specify ROUNDROBIN

Image Action Set: Syntax

matchImages Action

CASL Syntax

image.matchImages <**result**=*results*> <**status**=*rc*> /

*__required parameter__ **casOut**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

caslib="*string*"

decode=TRUE | FALSE

highlight=TRUE | FALSE

image="*string*"

methodOptions={**method**="DESCRIPTORMATCH" | "TEMPLATEMATCH", *method-specific-parameters*}

*__required parameter__ **table**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

singlePass=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

```
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      where="where-expression"
  }
  threshold=double
;
```

Parameter Descriptions

*** casOut**={*casouttable*}

specifies the settings for an output table.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

decode=TRUE | FALSE

when set to True, decompresses images before writing them to the output table.

Alias decompress

Default FALSE

highlight=TRUE | FALSE

when set to True, highlights detections by using a rectangle.

Default FALSE

image="*string*"

specifies the template image.

methodOptions={**method**="DESCRIPTORMATCH" | "TEMPLATEMATCH", *method-specific-parameters*}

specifies the method that is used for matching: either matching by template or matching by descriptor.

The value that you specify for method determines the other parameters that apply.

* **table**={*castable*}

specifies the table name, caslib, and other common parameters.

Alias imageTable

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

threshold=*double*

specifies the threshold value for eliminating low-confidence matches.

Parameters for method="DESCRIPTORMATCH"

descType="BRISK" | "ORB" | "ORB_BRISK" | *64-bit-integer*

specifies the type of descriptor to be computed on the key points.

thresholdRatio=*double*

specifies the threshold ratio for eliminating weak matches in descriptor pairs.

Default 0.8

Parameters for method="TEMPLATEMATCH"

No parameters apply when you specify TEMPLATEMATCH

Image Action Set: Syntax

processImages Action

CASL Syntax

image.processImages <**result**=*results*> <**status**=*rc*> /

*__required parameter__ **casOut**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

decode=TRUE | FALSE

image="*variable-name*"

imageFunctions={{

functionOptions={**functionType**="ADD_CONSTANT" | "BILATERAL_FILTER" | "BOX_FILTER" | "BUILD_PYRAMID" | "CANNY_EDGE" | "CONTOURS" | "CONVERT_COLOR" | "CUSTOM_FILTER" | "GAUSSIAN_FILTER" | "GET_PATCH" | "HIST_EQUALIZATION" | "LAPLACIAN" | "LINEAR_TRANSFORMATION" | "MEDIAN_FILTER" | "MORPHOLOGY" | "MUTATIONS" | "NORMALIZE" | "RESCALE" | "RESIZE" | "SOBEL" | "THRESHOLD", *functionType-specific-parameters*}

}, {...}}

*__required parameter__ **table**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

singlePass=TRUE | FALSE,

```
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
;
```

Parameter Descriptions

*** casOut**={*casouttable*}

specifies the settings for an output table.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the variables to be copy from the input table to the output table.

Alias copyVar

decode=TRUE | FALSE

when set to True, writes decompressed images and metadata to the output table.

Alias decompress

Default FALSE

image="*variable-name*"

specifies the name of the column that contains image binaries.

imageFunctions={{*imcorefunctiondesc-1*} <, {*imcorefunctiondesc-2*}, ...>}

lists image functions.

Alias steps

functionOptions={functionType="ADD_CONSTANT" | "BILATERAL_FILTER" | "BOX_FILTER" | "BUILD_PYRAMID" | "CANNY_EDGE" | "CONTOURS" | "CONVERT_COLOR" | "CUSTOM_FILTER" | "GAUSSIAN_FILTER" | "GET_PATCH" | "HIST_EQUALIZATION" | "LAPLACIAN" | "LINEAR_TRANSFORMATION" |

"MEDIAN_FILTER" | "MORPHOLOGY" | "MUTATIONS" | "NORMALIZE" | "RESCALE" | "RESIZE" | "SOBEL" | "THRESHOLD", *functionType-specific-parameters*}

Alias options, stepParameters

The value that you specify for functionType determines the other parameters that apply.

* **table**={*castable*}

specifies the table name, caslib, and other common parameters.

Alias imageTable

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

Parameters for functionType="ADD_CONSTANT"

b=*double*

specifies a constant value for the B channel.

Default 0

g=*double*

specifies a constant value for the G channel.

Default 0

r=*double*

specifies a constant value for the R channel.

Default 0

Parameters for functionType="BILATERAL_FILTER"

diameter=*64-bit-integer*

specifies the diameter of each pixel neighborhood.

sigmaColor=*double*

specifies the filter sigma in the color space. Larger values affect the farthest colors.

sigmaSpace=*double*

specifies the filter sigma in the coordinate space. Larger values affect the farthest colors.

Parameters for functionType="BOX_FILTER"

kernelHeight=*64-bit-integer*

specifies the height of a kernel (number of rows).

Alias h

kernelWidth=*64-bit-integer*

specifies the width of a kernel (number of columns).

Alias w

Parameters for functionType="BUILD_PYRAMID"

pyrUpDown="PYR_DOWN" | "PYR_UP" | *64-bit-integer*

specifies the type of the pyramid operation, downward or upward.

Parameters for functionType="CANNY_EDGE"

highThreshold=*double*

specifies the upper bound for the threshold.

kernelSize=*64-bit-integer*

specifies the size of the square kernel.

Alias size

Default 3

lowThreshold=*double*

specifies the lower bound for the threshold.

Parameters for functionType="CONTOURS"

approximate=TRUE | FALSE

when set to True, applies approximation to contours.

Default FALSE

boundingBox=TRUE | FALSE

when set to True, creates bounding boxes on contours.

Default FALSE

groupBoxes=TRUE | FALSE

when set to True, groups bounding boxes.

Default FALSE

Parameters for functionType="CONVERT_COLOR"

type="BGR2HSV" | "BGR2RGB" | "BGR2YUV" | "COLOR2GRAY" | "GRAY2COLOR" | "HSV2BGR" | "RGB2BGR" | "YUV2BGR" | *64-bit-integer*

specifies the color conversion code.

Parameters for functionType="CUSTOM_FILTER"

border="BORDER_BLACK" | "BORDER_REFLECT" | "BORDER_REFLECT101" | "BORDER_REPLICATE" | *64-bit-integer*

specifies the border method in the convolution.

kernelHeight=*64-bit-integer*

specifies the height of a kernel (number of rows).

Alias h

kernelValues={*double-1* <, *double-2*, ...>}

specifies values for the custom filter.

kernelWidth=*64-bit-integer*

specifies the width of a kernel (number of columns).

Alias w

Parameters for functionType="GAUSSIAN_FILTER"

kernelHeight=*64-bit-integer*

specifies the height of a kernel (number of rows).

Alias h

kernelWidth=*64-bit-integer*

specifies the width of a kernel (number of columns).

Alias w

Parameters for functionType="GET_PATCH"

height=*64-bit-integer*

specifies the height of an image (number of rows).

Alias h

width=*64-bit-integer*

specifies the width of an image (number of columns).

Alias w

x=*64-bit-integer*

specifies the X location.

Default 0

y=*64-bit-integer*

specifies the Y location.

Default 0

Parameters for functionType="HIST_EQUALIZATION"

method="ADAPTIVE" | "GLOBAL" | *64-bit-integer*

specifies the histogram equalization method.

Parameters for functionType="LAPLACIAN"

delta=*double*

specifies the delta value.

Default 0

kernelSize=*64-bit-integer*

specifies the size of the square kernel.

Alias size

scale=*double*

specifies the scale value.

Default 1

Parameters for functionType="LINEAR_TRANSFORMATION"

method="LOCAL_CONTRAST_NORM" | "STANDARDIZATION" | "WHITENING_PCA" | "WHITENING_ZCA" | *64-bit-integer*

specifies the linear transformation method.

Parameters for functionType="MEDIAN_FILTER"

kernelSize=*64-bit-integer*

specifies the size of the square kernel.

Alias size

Parameters for functionType="MORPHOLOGY"

kernelHeight=*64-bit-integer*

specifies the height of a kernel (number of rows).

Alias h

kernelWidth=*64-bit-integer*

specifies the width of a kernel (number of columns).

Alias w

method="BLACKHAT" | "CLOSE" | "DILATE" | "ERODE" | "GRADIENT" | "HITMISS" | "OPEN" | "TOPHAT"

specifies the operation to be used in the morphology transformation.

shape="CROSS" | "ELLIPSE" | "RECT"

specifies the shape of the kernel used in the morphology transformation.

Parameters for functionType="MUTATIONS"

type="COLOR_JITTERING" | "COLOR_SHIFTING" | "DARKEN" | "HORIZONTAL_FLIP" | "INVERT_PIXELS" | "LIGHTEN" | "ROTATE_LEFT" | "ROTATE_RIGHT" | "SHARPEN" | "VERTICAL_FLIP" | *64-bit-integer*

specifies the mutation type.

Parameters for functionType="NORMALIZE"

alpha=*double*

specifies the alpha value.

Default 1

beta=*double*

specifies the beta value.

Default 0

type="HAMMING" | "HAMMING2" | "INF" | "L1" | "L2" | "L2SQR" | "MINMAX" | "RELATIVE" | *64-bit-integer*

specifies the function used in normalization.

Parameters for functionType="RESCALE"

alpha=*double*

specifies the alpha value.

Default 1

beta=*double*

specifies the beta value.

Default 0

type="TO_32F" | "TO_64F" | "TO_8U"

specifies the code for determining rescale.

Parameters for functionType="RESIZE"

height=*64-bit-integer*

specifies the height of an image (number of rows).

Alias h

width=*64-bit-integer*

specifies the width of an image (number of columns).

Alias w

Parameters for functionType="SOBEL"

delta=*double*

specifies the delta value.

Default 0

dx=*64-bit-integer*

specifies the order of the derivative X.

dy=*64-bit-integer*

specifies the order of the derivative Y.

kernelSize="SIZE1" | "SIZE3" | "SIZE5" | "SIZE7" | *64-bit-integer*

specifies the size of the square kernel.

Alias size

scale=*double*

specifies the scale value.

Default 1

Parameters for functionType="THRESHOLD"

type="BINARY" | "BINARY_INVERSE" | "OTSU" | "TO_ZERO" | "TO_ZERO_INVERSE" | "TRIANGLE" | "TRUNCATE" | *64-bit-integer*

specifies the function used in thresholding.

value=*double*

specifies the threshold value.

Image Action Set: Syntax

saveImages Action

CASL Syntax

image.saveImages <**result**=*results*> <**status**=*rc*> /

caslib="*string*"

convert=TRUE | FALSE

***required parameter images**={

bioMedDimension="*variable-name*",

bioMedId="*variable-name*",

dimension="*variable-name*",

id="*variable-name*",

image="*variable-name*",

imageFormat="*variable-name*",

label="*variable-name*",

orientation="*variable-name*",

path="*variable-name*",

position="*variable-name*",

resolution="*variable-name*",

size="*variable-name*",

sliceIndex="*variable-name*",

spacing="*variable-name*",

*required parameter **table**={

caslib="*string*"

computedOnDemand=TRUE | FALSE

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

```
            nfl=integer
        }, {...}}
        computedVarsProgram="string"
        dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2,
          ...>}
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" |
          "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-
          specific-parameters}
      *required parameter name="table-name"
        onDemand=TRUE | FALSE
        singlePass=TRUE | FALSE
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}}
        where="where-expression"
    },
    type="variable-name"
}
labelLevels=integer
overwrite=TRUE | FALSE
prefix="string"
series={
    digits=integer,
    start=integer
}
subdirectory="string"
suffix="string"
type="string"
;
```

Parameter Descriptions

caslib="*string*"

specifies the name of the caslib to save images to.

convert=TRUE | FALSE

when set to True, allows image depth to be changed to 8 bits for saving.

Default TRUE

* **images**={*biomedimageTableIn*}

specifies the list of parameters that describe the input image table.

For more information about specifying the images parameter, see the common biomedimageTableIn parameter (Appendix A: Common Parameters).

labelLevels=*integer*

specifies the maximum number of directory levels to save from the label.

Default 0

overwrite=TRUE | FALSE

when set to True, allows existing files to be overwritten.

Default TRUE

prefix="*string*"

specifies the string to be added as a prefix to filenames.

series={*saveSeriesParms*}

specifies the parameters to be used to save images as series.

digits=*integer*

specifies the minimum number of digits in the numeric part of the series filenames.

Default 4

Range 1–BMICORE_SERIES_MAX_DIGITS

start=*integer*

specifies the starting index for the numeric part of the series filenames.

Default 0

Minimum value 0

subdirectory="*string*"

specifies the name of the subdirectory relative to the caslib root to which to write images.

suffix="*string*"

specifies the string to be added as a suffix to file names.

type="*string*"

specifies the type of images to be written.

Image Action Set: Syntax

summarizeImages Action

CASL Syntax

image.summarizeImages <**result**=*results*> <**status**=*rc*> /

image="*variable-name*"

***required parameter** **table**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

***required parameter** **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

***required parameter** **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

***required parameter** **name**="*table-name*",

```
        onDemand=TRUE | FALSE,
        orderBy={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        singlePass=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
    }
;
```

Parameter Descriptions

image="*variable-name*"
specifies the name of the column that contains image binaries.

* **table**={*castable*}
specifies the table name, caslib, and other common parameters.

Alias imageTable

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

caslib="*string*"
specifies the caslib that contains the table that you want to use with the action. By

default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

Details

About the Image Action Set

The Image action set offers the ability to load, process, and save images. It enables images to be used as a basic data type in SAS Viya. The load and save actions support all common image formats such as JPG for photographic images, and Digital Imaging and Communications in Medicine (DICOM) for biomedical images. The process action supports all images loaded from a photographic format.

The augmentImages Action

The **augmentImages** action enables you to create patches (by using either sliding windows or coordinates of a rectangle) and to augment images by using a number of methods, including rotating them, flipping them, and adjusting their contrast. This action can process more than one patch or augmentation command in a single run.

For example, you can use the sliding window method on input images and separately flip input images. The output CAS table will contain the resulting images. In addition, the **writeRandomly** parameter enables you to randomly apply augmentations to input images. This is very important for machine learning methods that are based on deep neural networks.

Order of Parameters

Setting both the **sweepImage** and **useWholeImage** parameters to True is contradictory. Therefore, the **useWholeImage** parameter overrides the **sweepImage** parameter.

The compareImages Action

The **compareImages** action compares two sets of images (submitted in two CAS input tables) in order to identify pairs of similar images. The first image in each pair belongs to one set, and the second image belongs to the other set. You can provide a criterion for pairing images based on metadata. (For example, you could pair images that have the same filename.) The action also provides an unrestricted pairing option, which pairs each image from the first table with each image from the second table.

The action supports the following similarity measures, which you specify in the **method** parameter:

SSIM	reports a structural similarity index whose value is a real number that lies in the bounded set [0, 1], where 1 indicates an identical image pair.
PSNR	reports a peak signal-to-noise ratio whose value is a nonnegative real number, where the higher the PSNR value, the more similar the images.

The output of the action is a table that contains the following columns:

_source_id_	Value of the pairing variable (default is filename) in the first table
_reference_id_	Value of the pairing variable (default is filename) in the second table
channel<*i*>	Similarity index for image channel *i*

The condenseImages Action

The **condenseImages** action provides the opposite functionality of the **flattenImageTable** action. In other words, the **condenseImages** action converts a "wide" format CAS table (that is, one pixel per column) into a CAS image table so that other image actions can process it. This action takes the following parameters:

- **width** specifies the width of the images
- **height** specifies the height of the images
- **numberOfChannels** specifies whether the image is in gray-scale or color
- **depth** specifies the depth of the images
- **groupedChannels** specifies whether the input pixels are grouped (all the blue pixels, then all the green pixels, and then all the red pixels) or not grouped (one blue pixel, one green pixel, and one red pixel; then another blue pixel, green pixel, and red pixel; and so on)

If the images are color images and the **depth** parameter is set to 8-bit, then the resulting images will be compressed images. Otherwise, they will be decoded images.

The extractDetectedObjects Action

The **extractDetectedObjects** action enables you to extract the object detection results from the SAS Deep Learning Toolkit. The extraction can take place in two ways: visualizing detections in input images and cropping detections from input images. This action supports common formats, including the COCO (Common Objects in Context) data set.

Details of Visualization

Visualization of object detections includes drawing a bounding box around the detected objects. The object names are also written above the bounding boxes. If the input table contains probability values of detections, then the **extractDetectedObjects** action adds them. Note that if many objects are detected in an image, then probability values are skipped. The thickness of the lines of the bounding boxes and the font size of the object name text are reduced for better visibility.

Details of Cropping

In addition to enabling you to visualize an object detection, the **extractDetectedObjects** action enables you to crop the detections and save them in an output table. An image can have multiple detections. The action provides two options for saving those cropped detections in the output table. The first option is to save detections of an image in a single row and multiple columns. This means that there will be multiple

image columns for an input image. The second option is to store detections of an image in multiple rows and a single column.

Supported Formats

A number of formats are available in the object detection literature. The **extractDetectedObjects** action supports the commonly used formats. The action can visualize or crop detections when they are in the following formats:

- COCO: Specifies a bounding box by using the X and Y coordinates of its top left corner along with X and Y coordinates of its bottom right corner. This format takes its name from a data set known as COCO (common objects in context).
- RECT: Specifies a bounding box by using the X and Y coordinates of its top left corner along with width and height values.
- YOLO: Specifies a bounding box by using the X and Y coordinates of its center along with width and height values. Note that in this format, values are normalized by the width and height of the input image. Therefore, values should be between 0 and 1. This format takes its name from a recent object detection algorithm known as YOLO (you only look once).

The fetchImages Action

The **fetchImages** action normalizes image tables (compressed or decompressed) in a table that can be parsed by clients without requiring additional dependencies. This action takes a CAS table as an input and produces a results table.

The **fetchImages** action is as a lightweight version of the **fetch** action for image tables; it supports a number of functionalities from the **fetch** action such as **to**, **from**, and **sort-by**.

The flattenImageTable action

In actions that work with images, the images are stored in columns (one image per column) so that the number of columns does not change when the image changes. This approach has two problems:

- Input images must all be the same size.
- It is not feasible to store very large images (for example, a high-definition image with a resolution of 1,920 × 1,080 requires more than six million columns).

The **flattenImageTable** action converts image formats into a one-column-per-pixel format. The **flattenImageTable** action accepts the following parameters:

- **outputWidth**
- **outputHeight**

- **transpose**
- **groupChannels**

The **outputWidth** and **outputHeight** parameters specify the resolution of the resulting images. The **transpose** parameter transposes an image before flattening it. If the **groupChannels** parameter is set to True, then the channels are grouped together so that all B pixels come before G and R pixels. By default, images in BGR (blue, green, red) format are supported.

Order of Parameters

The order of operations is prioritized as follows:

1. Resize
2. Transpose
3. Group the channels

The loadImages Action

The **loadImages** action reads images and associated metadata into memory from a variety of sources, including these:

- photographic image files
- biomedical image files, including those in Digital Imaging and Communications in Medicine (DICOM) format
- ZIP files containing photographic images
- a directory that contains image files or ZIP files
- a directory that contains one or more series of photographic or biomedical images
- a directory tree that contains any number of the preceding items
- URLs
- lists that contain combinations of the sources in the preceding items

The output contains images in a CAS table in encoded or decoded format, depending on the value of the **decode** input parameter. When this parameter is set to False (default value), the output table has the following columns:

_id_	An integer that is unique for each image
path	The full pathname of the image
image	A binary large object that contains the entire image file

size	The byte length of the binary large object in the _image_ column
type	A three-character string that specifies the format of the image (for example, **jpg** for JPG images)

When the **decode** parameter is set to True, the _image_ column contains decompressed image data, and the output table has these additional columns:

dimension	Number of dimensions of the image (for example, 2 for two-dimensional images, 3 for three-dimensional images)
resolution	Size of the image in each dimension (for example, {100, 200} for an array for a two-dimensional image that is 100 pixels wide and 200 pixels high)
imageFormat	Integer that represents the organization of data in the _image_ column. The most common values are 0, 16, 2, and 18, corresponding to 8-bit gray scale, 8-bit RGB, 16-bit gray scale, and 16-bit RGB images, respectively.

Other input parameters that affect the output table format are **labelLevels** and **addColumns**. If **labelLevels** is set to a nonzero integer *n*, the output table will include an additional column called _label_ whose type is string. If *n* is positive, each image in a directory will have a _label_ column that contains *n* top-level subdirectories of the root directory, separated by slashes. Similarly, if *n* is negative, the column will contain –*n* bottom-level subdirectories.

The **addColumns** parameter is a list that contains two subparameters, **general** and **dicomAttributes**. You can specify the following values for the **general** subparameter, each of which produces an additional metadata column in the output table:

CHANNELCOUNT	produces an additional metadata column named _channelCount_, which contains the number of channels in each pixel.
CHANNELTYPE	produces an additional metadata column named _channelType_, which contains a string that indicates the number of bits in each channel and indicates whether the value is signed or unsigned.
DEPTH	produces an additional metadata column named _depth_, which contains the number of 2-D slices in a 3-D biomedical image.
HEIGHT	produces an additional metadata column named _height_, which contains the number of pixels along the height of the image.
ORIENTATION	produces an additional metadata column named _orientation_, which contains vectors that describe how a biomedical image is oriented in space.
POSITION	produces an additional metadata column named _position_, which contains the coordinates of the origin of a biomedical image (in millimeters).
SPACING	produces an additional metadata column named _spacing_, which contains the physical dimensions of a pixel in a biomedical image (in millimeters).
SPAN	produces an additional metadata column named _span_, which contains the number of time frames in a 4-D biomedical image.
WIDTH	produces an additional metadata column named _width_, which contains the number of pixels along the width of the image.

The **dicomAttributes** subparameter of the **addColumns** parameter supports the loading of all public DICOM attributes for which encoding is known. This parameter is in turn a list that contains two subparameters, **tags** and **keywords**, where you can specify strings that represent DICOM attribute tags and keywords, respectively. When DICOM attributes are specified, the output table has an additional column that contains strings representing the values of each specified attribute. A complete list of DICOM attributes, keywords, and tags is available elsewhere (National Electrical Manufacturers Association 2018).

A parameter in the **loadImages** action that is of particular interest in biomedical image processing is **series**, a parameter list. If the **series** parameter is specified, the action reads only image series from directories, where the series are recognized on the basis of the subparameter **dicom**. If the **dicom** subparameter is set to True, only DICOM series are read, and each image is assigned a unique path that is based on the DICOM Unique Identifier (UID) of the series. If the **dicom** subparameter is set to False (the default), the action reads all images in a directory and attempts to assemble them into one higher-dimensional image. Only those series that constitute a 3-D image are fully supported by the **series** parameter.

As an important disk access control update to the **loadImages** action, beginning in SAS Viya 3.4, the use of an absolute path as an input to the **path** parameter of the action is reserved for users who can assume an administrator role. If you have existing code that invokes the action by using absolute paths, either you must be granted the ability to assume an administrator role for that code to continue running without errors, or you must convert that code to use paths relative to CAS libraries. Even if you can assume an administrator role, the use of absolute paths is deprecated, and it might be discontinued in future releases. Note that CAS libraries can be specified in the action by using the **caslib** parameter. By default, the **caslib** parameter assumes the name of the active CAS library in the session.

The matchImages Action

The **matchImages** action enables you to search for an image in a large collection of images. For example, you can search a license plate against a large surveillance system. You can query by either of the following methods:

- Template matching: The query image is slid through an input image and a relevance score is computed.
- Descriptor matching: Keypoints are first detected and then descriptors are computed on those keypoints. Finally, descriptors of the query image and an input image are matched and a relevance score is computed on those matches. Descriptors enable you to have rotation and scale invariant matches. The following descriptors are supported:
  - ORB (oriented features from accelerated segment test and rotated binary robust independent elementary features)
  - BRISK (binary robust invariant scalable keypoints)
  - ORB+BRISK

Notes on the matchImages action

- Scores from template matching are higher than scores from descriptor matching.
- The ORB descriptor is faster than the BRISK descriptor.

- The BRISK descriptor is better than the ORB descriptor for handling scale variance.
- For descriptor matching, X and Y locations are averaged on the matched descriptors; therefore, they are not exact but are only approximations.
- If no match is reported in descriptor matching, then either there is no good match between descriptors or the images are too small to detect key points.
- For template matching, the query image should be smaller than the search images.
- In descriptor matching, it is difficult to visualize matching descriptors.

The processImages Action

The **processImages** action performs single or multiple image processing functions on input images. The input of this action is a CAS table that contains images and an array of functions (with their specific parameters). The output is a CAS table that contains the resulting images after the specified functions are applied. You can specify the following functions:

RESIZE — resizes an input image based on the **width** and **height** parameters. The parameters must be specified as positive integers.

GET_PATCH — creates a patch based on a rectangular region (using **x**, **y**, **width**, and **height** parameters) from an input image. The **x** and **y** parameters must be specified as integers that are greater than or equal to 0. The **width** and **height** parameters must be specified as positive integers. Further, if **x** + **width** is greater than the specified width of the input image, then the width of the rectangular region is reduced to the input image's specified width. The same is true for the **y** and **height** parameters.

CANNY_EDGE — detects edges in input images by using the Canny edge detection algorithm. This function accepts three parameters: **lowThreshold**, **highThreshold**, and **kernelSize**.

LAPLACIAN — applies the Laplace operator to input images. The Laplace operator is a second-order differential operator in the n-dimensional Euclidean space. When the value of the **kernelSize** parameter is 1, the resulting value is approximated by convolving the following kernel with input images:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Otherwise, the value is approximated by summing the second x and y derivatives, which are calculated using the Sobel operator.

SOBEL — uses the Sobel operator to calculate the first, second, third, or mixed image derivatives. Derivatives are calculated by convolving the image with a kernel. Kernels must be square. The size can only be one of the following numbers: 1, 3, 5, and 7. When the kernel size is 1, a 1×3 kernel is used for the first or second derivative of x, and a 3×1 kernel is used for the first or second derivative of y. For the other

kernel sizes, the Sobel operator considers Gaussian smoothing and differentiation together (handling noise). The following kernel is used when the **`kernelSize`**, **`dx`**, and **`dy`** parameters have the values 3, 1, and 0, respectively:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

The following kernel is used when they have the values 3, 0, and 1, respectively:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

NORMALIZE — normalizes the value range of an image. You can specify the following normalization functions in the **`type`** parameter: INF, L1, L2, L2SQR, HAMMING, HAMMING2, RELATIVE, and MINMAX. The MINMAX function is commonly used to normalize an image between alpha and beta values.

THRESHOLD — applies a threshold value to each pixel in an image. You can specify the following threshold types: BINARY, BINARY_INVERSE, TRUNCATE, TO_ZERO, TO_ZERO_INVERSE, OTSU, and TRIANGLE.

CONVERT_COLOR — converts an image's color space. Currently available conversions are COLOR2GRAY, GRAY2COLOR, BGR2RGB, and RGB2BGR. The default color space for an image is BGR. COLOR2GRAY means BGR2GRAY.

RESCALE — changes an image's depth. You can specify the following types of rescaling: TO_8U (depth is changed to 8-bit, where a pixel is represented by an unsigned integer), TO_32F (depth is changed to 32-bit, where a pixel is represented by a floating number), and TO_64F (depth is changed to 64-bit, where a pixel is represented by a double-precision number). You can scale the values based on the **`alpha`** and **`beta`** parameters.

MORPHOLOGY — performs a morphological transformation on images. You can specify the following types of transformation: ERODE, DILATE, OPEN, CLOSE, GRADIENT, TOPHAT, BLACKHAT, and HITMISS. Each of these operations can be in RECT (rectangular), CROSS, and ELLIPSE shapes.

BOX_FILTER — blurs an image by using a normalized box filter. The **`kernelWidth`** and **`kernelHeight`** parameters shape the filter. The anchor is assumed to be at the kernel's center.

GAUSSIAN_FILTER — blurs an image by using a Gaussian filter. A convolution kernel is created based on the **`kernelWidth`** and **`kernelHeight`** parameters, which must be positive and odd.

BILATERAL_FILTER — applies a bilateral filter to images. A bilateral filter is a nonlinear, edge-preserving, noise-reducing smoothing filter. The idea behind this filter is to consider pixels close if they are in spatially nearby locations, and similar if they have nearby values (in color space).

MEDIAN_FILTER — blurs an image by using the median filter. This filter uses a square matrix of the size specified in the **`kernelSize`** parameter, which must be greater than 1 and odd.

BUILD_PYRAMID blurs an image and samples it down (PYR_DOWN) or up (PYR_UP). A 5×5 Gaussian kernel is used for blurring.

CONTOURS applies a number of preprocessing steps for more accurate results. This function accepts only one-channel images (for example, gray scale images). It provides better results when the input image is a binarized image (for example, pixel values are either black or white). If you provide a gray scale image rather than a binarized image, you will still see results but they might be inaccurate or unexpected (for example, a contour could look like a frame that contains the whole picture).

CUSTOM_FILTER creates a custom filter and uses it in convolution. This operation takes the width, height, and values of a filter.

ADD_CONSTANT adds or subtracts a constant value from the pixel intensity values input images. This function also makes sure that the resulting values are in the permitted range that is determined by the depth of a pixel. For example, if an image's pixel depth is unsigned 8-bit (which means that the pixel values can be between 0 and 255 only), then this function makes sure that the resulting values are within the range of 0 and 255 (no overflows and negative values are permitted in this case). Furthermore, if the image is a one-channel image, then the function uses only the constant value of the first input.

HIST_EQUALIZATION aims to stretch out the intensity range. This function maps the intensity distribution of an input image to another distribution that is wider and has a more uniform distribution of intensity values. Two versions are supported: global and adaptive. In the global version, the mapping is applied by considering the whole image. In the adaptive version, the mapping is performed in local patches of input images. Therefore, the adaptive version is more robust to images where contrast is quite different at different regions of those images.

LINEAR_TRANSFORMATION performs linear transformation in one of the following ways, based on the value of the **`method`** parameter:

- STANDARDIZATION creates an image that has zero mean and unit variance.
- WHITENING_PCA uses principal component analysis to create an image that has an identity covariance.
- WHITENING_ZCA uses a ZCA transformation (also called a Mahalanobis transformation) to create an image that has an identity covariance.
- LOCAL_CONTRAST_NORM is similar to STANDARDIZATION except that it convolves a 9×9 Gaussian image with the input image.

MUTATIONS mutates images by using different augmentation techniques: flipping vertically, flipping horizontally, scaling up and down with pyramids, changing the contrast (darkening or lightening), sharpening, and rotating to the left or right. This function is useful for increasing the variety of the input images, which is a major preprocessing step for training methods that are based on deep neural networks.

The saveImages Action

The **saveImages** action writes images and associated metadata from a CAS table in memory onto a disk in a desired format, which can be either a photographic or biomedical image format. The action can be used to convert images between different formats, including between biomedical and photographic image formats. Note that the metadata saved by the action are limited by both the output of the **loadImages** action and the format to which the images are written.

The paths to which images are saved depend on parameters **caslib**, **subdirectory**, and **labelLevels**. If **labelLevels** is set to a nonzero integer *n*, the action will save images in subdirectories whose names are based on the contents of the column specified by the subparameter **label** of the **images** parameter list. If *n* is positive, each image is saved in a subdirectory whose path ends with the leftmost *n* subdirectories in the label column of that image. Similarly, if *n* is negative, each image is saved in a subdirectory whose path ends with the rightmost *n* subdirectories in the label column of that image.

The names of the saved images depend on the input table column that contains the image paths, as specified by the subparameter **path** of the **images** parameter list. If **path** is specified, the names of the output images are derived from the filenames contained in that path column. If image paths are not provided, the action automatically generates the image names and saves the images in the format that you specify. You can also specify a prefix or suffix for the names of saved images using parameters **prefix** and **suffix**, respectively.

A **saveImages** parameter that is of particular interest to biomedical image processing is **series**, a parameter list. This parameter can be used to write a 3D biomedical image as a series of 2D slices in a desired format, e.g., to write a 3D image loaded from a DICOM series as a sequence of JPEG images. If the saved images are in a photographic format, they can then be loaded back and processed using actions such as **processImages**, **compareImages**, and **flattenImages**.

The summarizeImages Action

The **summarizeImages** action extracts the following information about an input image table:

- minimum, maximum, and average width and height values
- minimum, maximum, and average pixel intensity values.

In addition, this action reports the distribution of image types. For example, jpg:4, and png:3 denote that there are four images in the input table with an extension of *jpg* and three with an extension of *png*. Only the types that exist in the table are reported.

Examples

Example 13.1: Use Sliding Window without Mutating Using the augmentImages Action

This example is not available for the CAS language.

Example 13.2: Use Sliding Window and Mutate Using the augmentImages Action

This example is not available for the CAS language.

Example 13.3: Create a Patch Using the augmentImages Action

This example is not available for the CAS language.

Example 13.4: Create a Patch and Mutate Using the augmentImages Action

This example is not available for the CAS language.

Example 13.5: Mutate without Creating a Patch Using the augmentImages Action

This example is not available for the CAS language.

Example 13.6: Introduce Randomness Using the augmentImages Action

This example is not available for the CAS language.

Example 13.7: Comparing Images

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For

more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

The following statements use the **compareImages** action to load images from the *SourceImages* and *ReferenceImages* directories, compare each image in the *SourceImages* directory with each image in the *ReferenceImages* directory, and output the similarity indices and filenames for each image pair. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
proc cas;
   loadactionset "image";
   image.loadImages / path="SourceImages"
                      casout={name="source", replication=0, replace=true};

   image.loadImages / path="ReferenceImages"
                      casout={name="reference", replication=0, replace=true};

   image.compareImages / sourceImages={table="source"}
                         referenceImages={table="reference"}
                         casOut={name="mycas.comparisons", replication=0, replace=true}
pairAll=TRUE;
run;

data mycomp; set mycas.comparisons; run;
proc print data=mycomp; run;

quit;
```

If each of the *SourceImages* and *ReferenceImages* directories contains three images named *1.bmp*, *2.bmp*, and *3.bmp*, the output would look like Output 13.7.1.

Output 13.7.1 compareImages Output

Obs	_channel1_	_channel2_	_channel3_	_channel4_	_source_id_	_reference_id_
1	0.92434	0.98846	0.97123	1	3.bmp	3.bmp
2	0.00000	0.00000	0.00000	0	3.bmp	1.bmp
3	0.00000	0.00000	0.00000	0	3.bmp	2.bmp
4	0.99179	0.99794	0.99652	1	1.bmp	1.bmp
5	0.00000	0.00000	0.00000	0	1.bmp	3.bmp
6	0.27950	0.28518	0.28413	1	1.bmp	2.bmp
7	0.99159	0.99797	0.99669	1	2.bmp	2.bmp
8	0.27909	0.28517	0.28425	1	2.bmp	1.bmp
9	0.00000	0.00000	0.00000	0	2.bmp	3.bmp

Example 13.8: Condense Images

This example is not available for the CAS language.

Example 13.9: Condense Images Using the groupedChannels Parameter

This example is not available for the CAS language.

Example 13.10: Condense Images Using the depth Parameter

This example is not available for the CAS language.

Example 13.11: Condense Images Using the numberOfChannels Parameter

This example is not available for the CAS language.

Example 13.12: Condense Images Using the inputs Parameter

This example is not available for the CAS language.

Example 13.13: Visualize Detections

This example is not available for the CAS language.

Example 13.14: Visualize Only One Detection

This example is not available for the CAS language.

Example 13.15: Visualize Detections in the COCO Format

This example is not available for the CAS language.

Example 13.16: Visualize Detections in the YOLO Format

This example is not available for the CAS language.

Example 13.17: Cropping Detections and Saving Them in Multiple Columns Using the extractDetectedObjects Action

This example is not available for the CAS language.

Example 13.18: Cropping Detections and Saving Them in Multiple Rows Using the extractDetectedObjects Action

This example is not available for the CAS language.

Example 13.19: Fetch Images

This example is not available for the CAS language.

Example 13.20: Fetch Only Two Images

This example is not available for the CAS language.

Example 13.21: Copying Variables While Fetching Images

This example is not available for the CAS language.

Example 13.22: Process Images and Then Fetch Them

This example is not available for the CAS language.

Example 13.23: Fetch Images Using SortBy

This example is not available for the CAS language.

Example 13.24: Simple Flattening

This example is not available for the CAS language.

Example 13.25: Transpose While Flattening an Image Table

This example is not available for the CAS language.

Example 13.26: Group Channels While Flattening an Image Table

This example is not available for the CAS language.

Example 13.27: Loading an Image

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates how to invoke the **loadImages** action using PROC CAS. In the example, the *list.txt* file could contain a list like this:

```
/.../1.bmp
/.../tenImages.zip
http://.../imageFarAway.jpg
/.../2.jpg
/.../3.png
```

The following statements read the images in the *list.txt* file and load them into a table named mycas.list:

```
proc cas;
loadactionset "image";
image.loadImages / path="list.txt"
                   decode=TRUE
                   pathIsList=TRUE


casout={name="mycas.list", replication=0, replace=true};
run;
quit;
```

Example 13.28: Loading a Directory Tree and Metadata

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example executes the **loadImages** action in order to recursively load images and useful metadata from a directory tree whose root is *MyImages* in the session's active CAS library:

```
proc cas;
loadactionset "image";
image.loadImages / path="MyImages"
                   casout={name="mycas.tree", replication=0, replace=TRUE}
                   recurse=TRUE
                   decode=TRUE
                   labelLevels=-2
                   addColumns={"WIDTH", "HEIGHT"};
run;
quit;
```

In the output table, the images at the subdirectory level are identified by the _label_ column that corresponds to those images. For example, *MyImages/A/1.jpg* and *MyImages/A/B/C/2.jpg* are labeled A and B/C, respectively.

Example 13.29: Recursively Loading All DICOM Series under a Directory

This example is not available for the CAS language.

Example 13.30: Loading and Displaying a 3-D Image

This example is not available for the CAS language.

Example 13.31: Loading DICOM Attributes Along with DICOM Images

This example is not available for the CAS language.

Example 13.32: Template Matching

This example is not available for the CAS language.

Example 13.33: Descriptor Matching

This example is not available for the CAS language.

Example 13.34: Resizing Images Using the processImages Action

This example is not available for the CAS language.

Example 13.35: Creating a Patch from an Input Image Using the processImages Action

This example is not available for the CAS language.

Example 13.36: Calculating Image Derivatives Using the processImages Action

This example is not available for the CAS language.

Example 13.37: Detecting Edges with Canny Using the processImages Action

This example is not available for the CAS language.

Example 13.38: Applying the Laplace Operator Using the processImages Action

This example is not available for the CAS language.

Example 13.39: Normalizing Images Using the processImages Action

This example is not available for the CAS language.

Example 13.40: Applying a Threshold Using the processImages Action

This example is not available for the CAS language.

Example 13.41: Converting Color Spaces Using the processImages Action

This example is not available for the CAS language.

Example 13.42: Performing a Morphological Transformation Using the processImages Action

This example is not available for the CAS language.

Example 13.43: Blurring Images Using the processImages Action

This example is not available for the CAS language.

Example 13.44: Reducing Noise and Detecting Edges Using the processImages Action

This example is not available for the CAS language.

Example 13.45: Creating a Custom Filter and Applying It Using the processImages Action

This example is not available for the CAS language.

Example 13.46: Performing Histogram Equalization Using the processImages Action

This example is not available for the CAS language.

Example 13.47: Linear Transformation Using the processImages Action

This example is not available for the CAS language.

Example 13.48: Saving Images

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

The following statements use the **`saveImage`** action to load JPG images from one directory and save them in BMP format into another directory:

```
proc cas;
loadactionset "image";
image.loadImages / path="/.../MyJpegImages"
                   casout={name="mycas.list", replication=0, replace=true};

image.saveImages / images={table="mycas.list", path="_path_"}
                   dimension="myDimensionColumn"}
                   caslib="mycaslib"
                   subdirectory="MyBmpImages/"
                   prefix="image_"
                   type="bmp";
run;
quit;
```

Example 13.49: Saving Image Labels

This example is not available for the CAS language.

Example 13.50: Converting DICOM Series Into Sequences of JPEG Images

This example is not available for the CAS language.

Example 13.51: Summarize an Image Table

This example is not available for the CAS language.

Example 13.52: Summarize an Image Table with Custom Image Column

This example is not available for the CAS language.

References

National Electrical Manufacturers Association (2018). "Digital Imaging and Communications in Medicine (DICOM) Standard." http://medical.nema.org/.

Chapter 14
Language Model Action Set

Contents

Syntax **473**
The lmDecode Action 474
The lmImport Action 482
Details **489**
Overview of the Language Model Action Set 489
The lmDecode Action 489
The lmImport Action 489
Examples **489**
Example 14.1: Decoding RNN Scores 489
Example 14.2: Decoding RNN Scores with Labels Specified 489
Example 14.3: Importing a Language Model 489
Example 14.4: Importing a Language Model with Mapping between Characters 490
Example 14.5: Training an RNN Acoustic Model 490
Example 14.6: End-to-End Speech-to-Text Application 490

Language Model Action Set: Syntax

lmDecode Action

CASL Syntax

```
langModel.lmDecode <result=results> <status=rc> /
    alpha=double
    beta=double
    blankLabel="string"
  *required parameter casOut={
        caslib="string",
        compress=TRUE | FALSE,
        indexVars={"variable-name-1" <, "variable-name-2", ...>},
        label="string",
        maxMemSize=64-bit-integer,
        name="table-name",
        onDemand=TRUE | FALSE,
        promote=TRUE | FALSE,
        replace=TRUE | FALSE,
        replication=integer,
        timeStamp="string",
        where={"string-1" <, "string-2", ...>}
  }
    columnMap={"string-1" <, "string-2", ...>}
    copyVars={"variable-name-1" <, "variable-name-2", ...>}
  *required parameter langModelTable={
        caslib="string",
        computedVars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
```

```
            nfd=integer,
            nfl=integer
        }, {...}},
      *required parameter name="table-name"
}
  maxPathSize=64-bit-integer
  ngramsOrder=64-bit-integer
  spaceLabel="string"
*required parameter table={
        caslib="string",
        computedOnDemand=TRUE | FALSE,
        computedVars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        computedVarsProgram="string",
        dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
        groupBy={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        groupByMode="NOSORT" | "REDISTRIBUTE",
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
```

```
    *required parameter name="table-name",
     onDemand=TRUE | FALSE,
     orderBy={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
     }, {...}},
     singlePass=TRUE | FALSE,
     vars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
     }, {...}},
     where="where-expression"
  }
;
```

Parameter Descriptions

alpha=*double*

specifies tunable parameter that controls the trade-off between the acoustic model and the language model. The larger 'alpha' is, the more the language model will count while ranking all candidates of the final guess.

Default 1

beta=*double*

specifies tunable parameter that controls the trade-off between the acoustic model and the length of the sentence. The larger 'beta' is, the more the length of sentence will count while ranking all candidates of the final guess.

Default 0

blankLabel="*string*"

specifies string used to indicate 'blank' label.

Default "_"

* **casOut**={*casouttable*}

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

columnMap={*"string-1"* <, *"string-2"*, ...>}

specifies labels the score columns in each time frame represent, which must follow the same order.

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the variables to transfer from the input table to the output table.

Alias copyVar

* **langModelTable**={*castable*}

specifies the N-gram language model to use in decoding process.

Long form langModelTable={name="*table-name*"}

Shortcut form langModelTable="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

* **name**="*table-name*"

specifies the name of the table to use.

maxPathSize=*64-bit-integer*

specifies maximum number of paths kept as candidates of final results during decoding process.

Default 100

ngramsOrder=*64-bit-integer*

specifies highest order of N-grams to use during decoding process.

Default 3

Minimum value 1

spaceLabel="*string*"

specifies string used to indicate 'space' label.

Default " "

* **table**={*castable*}

specifies RNN scores returned by the acoustic model.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that

are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

Language Model Action Set: Syntax

lmImport Action

CASL Syntax

langModel.lmImport <**result**=*results*> <**status**=*rc*> /

*required parameter **casOut**={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

labelMapTable={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

```
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
*required parameter table={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
```

```
        singlePass=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
    }
;
```

Parameter Descriptions

* **casOut**={*casouttable*}

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

labelMapTable={*castable*}

specifies mapping between labels used by the language model and labels to use in speech-to-text system.

Long form labelMapTable={name="*table-name*"}

Shortcut form labelMapTable="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this

parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

* **table**={*castable*}

specifies the N-gram language model.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

Details

Overview of the Language Model Action Set

The **langModel** action set provides actions for importing and using N-gram language models in speech-to-text applications.

For detailed examples that show how you can use the actions in this action set for speech-to-text applications, see Example 14.5 and Example 14.6.

The lmDecode Action

The **lmDecode** action applies an imported language model to recurrent neural network (RNN) scores to compute the text transcript of the speech waveform.

For examples that show how you can decode RNN scores by using an imported N-gram language model, see Example 14.1 and Example 14.2.

The lmImport Action

The **lmImport** action imports an N-gram language model from an external source.

For examples that show how you can import an N-gram language model, see Example 14.3 and Example 14.4.

Examples

Example 14.1: Decoding RNN Scores

This example is not available for the CAS programming language.

Example 14.2: Decoding RNN Scores with Labels Specified

This example is not available for the CAS programming language.

Example 14.3: Importing a Language Model

This example is not available for the CAS programming language.

Example 14.4: Importing a Language Model with Mapping between Characters

This example is not available for the CAS programming language.

Example 14.5: Training an RNN Acoustic Model

This example is not available for the CAS programming language.

Example 14.6: End-to-End Speech-to-Text Application

This example is not available for the CAS programming language.

Chapter 15
Machine Learning Tools Action Set

Contents

Syntax **491**
The crossValidateML Action 492
Details **494**
Machine Learning Tools Action Set 494
The crossValidateML Action 494
Examples **496**
Example 15.1: Cross Validation of a Forest Model 496

Machine Learning Tools Action Set: Syntax

crossValidateML Action

CASL Syntax

mlTools.crossValidateML <**result**=*results*> <**status**=*rc*> /

casOut={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

kFolds=*integer*

logLevel=*integer*

modelType="BNET" | "DECISIONTREE" | "FACTMAC" | "FOREST" | "GRADBOOST" | "NEURALNET" | "SVM"

nSubsessionWorkers=*integer*

parallelFolds=TRUE | FALSE

seed=*integer*

targetEvent="*string*"

***required parameter** **trainOptions**={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

;

Parameter Descriptions

casOut={*casouttable*}

specifies the score output table name and details.

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

kFolds=*integer*

specifies the number of folds to use for cross validation.

Default 5

Minimum value 2

logLevel=*integer*

specifies the level of log messages to be written: no logs (0), initialization and completion logs (1), setup summary logs added (2), fold begin and complete logs added (3).

Default 3

Range 0–3

modelType="BNET" | "DECISIONTREE" | "FACTMAC" | "FOREST" | "GRADBOOST" | "NEURALNET" | "SVM"

specifies the model type to which cross validation is applied.

Default DECISIONTREE

nSubsessionWorkers=*integer*

specifies the number of worker nodes for each subsession to use for parallel fold evaluation.

parallelFolds=TRUE | FALSE

when set to True, evaluates folds in parallel.

Default TRUE

seed=*integer*

specifies the seed to use for fold sampling for cross validation.

Default 0

targetEvent="*string*"

specifies the name of the nominal target event to use for model assessment.

* **trainOptions**={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies a list of parameters for the model training action to use in the cross validation process.

Details

Machine Learning Tools Action Set

The **mltools** action set provides utility actions for machine learning. Currently the action set contains a single action, **crossValidateML**, for performing cross validation to assess the accuracy of different model types.

The crossValidateML Action

The **crossValidateML** action performs cross validation over specified machine learning actions to assess the accuracy of a model. In *k*-fold cross validation, the data are partitioned into *k* subsets (folds). For each fold, a new model is trained on all folds except the selected (holdout) fold (that is, it is trained on *k*–1 folds) and then validated and assessed using the selected (holdout) fold. The assessment metrics are then averaged across all the holdout folds.

Table 15.1 lists the model types that are currently supported by the **modelType** parameter in the **crossValidateML** action.

Table 15.1 Model Types Supported by the **crossValidateML** Action

modelType Value	Model Type Name
bnet	Bayesian network classifier
decisionTree	Decision tree
factmac	Factorization machine
forest	Forest
gradBoost	Gradient boosting
neuralNet	Neural network
svm	Support vector machine

The **crossValidateML** action performs the following steps for the specified model type:

1. Determines the number of folds that can be evaluated in parallel, as described in the section "Determining the Number of Parallel Evaluations" on page 495.
2. Partitions the data table into *k* folds by using the **kfold** action in the **sampling** action set.
3. For each fold (potentially in parallel):
   a) Evaluates the train action for the specified model type on all folds except the selected (holdout) fold (that is, it is trained on *k*–1 folds).
   b) Evaluates the score action for the specified model type on the holdout fold.
   c) Evaluates the **assess** action in the **percentile** action set on the holdout fold.

4. Appends the score tables that are generated for each fold to a single new score table, including the _FoldInd_ column and the target column.

5. Generates a "Cross Validation Fit Statistics" table that contains one row for each fold and a final Average row that displays all assess fit statistics averaged over all folds.

For more information about the **kfold** action in the **sampling** action set, see *SAS Visual Statistics: Programming Guide*. For more information about the **score** action in the **astore** action set, see *SAS Visual Data Mining and Machine Learning: Programming Guide*. For more information about the **assess** action in the **percentile** action set, see *SAS Visual Statistics: Procedures Guide*.

Determining the Number of Parallel Evaluations

The number of parallel fold evaluations is determined as follows:

1. First the number of worker nodes to use in parallel subsessions is determined:

   a) If you specify a value greater than 0 for the number of workers to use in parallel subsessions by using the **nSubsessionWorkers** parameter, that value is used for each subsession.

   b) If you specify a negative value for the **nSubsessionWorkers** parameter, the number of workers for each subsession is the same as the number used in the parent session.

   c) If you do not specify a value for the **nSubsessionWorkers** parameter, or if you specify the default value of 0, the number of workers to use in each subsession defaults to the number of workers in the parent session. If the number of workers that are used in the parent session times the number of folds is greater than the number of workers available in the server, the number of workers for each subsession is reduced to the number of workers in the server divided by the number of folds. The number of workers in each subsession is then increased if necessary, according to the size of the data table, as follows: **nSubsessionWorkers** = 1 + *nDataRows* * *nDataColumns* / 50,000,000 (the number of worker nodes in each subsession is at least one node per 50 million values).

2. The number of parallel evaluations is then limited by the server configuration:

   a) In single-machine mode, if the number of folds is greater than 16 and not specified in the **nParallel** parameter, it is limited to 16.

   b) In distributed mode, the upper limit for the number of parallel evaluations is calculated as *2*W/n*, where *W* is the number of workers that are connected to the server and *n* is the number of workers in the parallel subsessions.

Examples

Example 15.1: Cross Validation of a Forest Model

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example shows how to use the **crossValidateML** action to perform *k*-fold cross validation of a forest model. The following DATA step loads the JunkMail data set from the Sashelp library into a data table named mycas.JunkMail. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.JunkMail;
   set sashelp.JunkMail;
run;
```

The following PROC CAS code uses the **crossValidateML** action to automatically perform *k*-fold cross validation of a forest model that is trained on the JunkMail data table:

```
proc cas noqueue;
   mltools.crossValidateML /
      modelType='forest'
      kFolds=3
      seed=12345
      casOut='junkmail_forest_score'
      trainOptions={
         table={
            name='junkmail',
            where='Class NE .'
         },
         inputs={
            "Address",     "Addresses",  "All",       "Bracket",
            "Business",    "CS",         "CapAvg",    "CapLong",
            "CapTotal",    "Conference", "Credit",    "Data",
            "Direct",      "Dollar",     "Edu",       "Email",
            "Exclamation", "Font",       "Free",      "George",
            "HP",          "HPL",        "Internet",  "Lab",
            "Labs",        "Mail",       "Make",      "Meeting",
            "Money",       "Order",      "Original",  "Our",
            "Over",        "PM",         "Paren",     "Parts",
            "People",      "Pound",      "Project",   "RE",
            "Receive",     "Remove",     "Semicolon", "Table",
```

```
            "Technology",  "Telnet",     "Will",       "You",
            "Your",        "_000",       "_85",        "_415",
            "_650",        "_857",       "_1999",      "_3D"
         },
         target='Class',
         nominals={
            {name='Class'}
         },
         casOut={
            name='forest_junkmail_model', replace=true
         }
      }
   ;
   run;
quit;
```

The "Cross Validation Fit Statistics" table in Output 15.1.1 displays the fit statistics for each fold and the fit statistic values averaged across all folds.

Output 15.1.1 Cross Validation Fit Statistics Output

Results from mlTools.crossValidateML

Cross Validation Fit Statistics

Fold	Sum of Frequencies	Average Squared Error	Divisor for ASE	Root Average Squared Error	Misclassification (MCE)	Multi-Class Log Loss
Fold 1	1535.00	0.1312	1535.00	0.3622	0.1713	0.4306
Fold 2	1533.00	0.1293	1533.00	0.3595	0.1650	0.4253
Fold 3	1533.00	0.1318	1533.00	0.3630	0.1657	0.4310
Average	1533.67	0.1307	1533.67	0.3616	0.1674	0.4289

Chapter 16
The Network Action Set

Contents

Syntax **500**
The biconnectedComponents Action 501
The centrality Action 513
The clique Action 529
The community Action 543
The connectedComponents Action 560
The core Action 573
The cycle Action 585
The nodeSimilarity Action 599
The path Action 617
The patternMatch Action 635
The reach Action 655
The readGraph Action 672
The shortestPath Action 684
The summary Action 701
The transitiveclosure Action 714
Details **726**
The network Action Set 726
The biconnectedComponents Action 726
The centrality Action 727
The clique Action 727
The community Action 727
The connectedComponents Action 727
The core Action 728
The cycle Action 728
The nodeSimilarity Action 728
The path Action 728
The patternMatch Action 729
The reach Action 729
The readGraph Action 729
The shortestPath Action 729
The summary Action 729
The transitiveClosure Action 730
Examples **730**
Example 16.1: Finding the Biconnected Components of a Simple Undirected Graph . 730
Example 16.2: Calculating the Centrality of a Simple Directed Graph 734

Example 16.3: Finding the Maximal Cliques of a Simple Undirected Graph 737
Example 16.4: Detecting Communities for a Simple Undirected Graph 740
Example 16.5: Finding the Connected Components of a Simple Undirected Graph . . 744
Example 16.6: Calculating the Core Decomposition of a Simple Undirected Graph . . 746
Example 16.7: Enumerating the Cycles of a Simple Directed Graph 750
Example 16.8: Node Similarity in an Undirected Graph 754
Example 16.9: Enumerating the Paths of a Simple Directed Graph 758
Example 16.10: Pattern Matching of a Simple Directed Graph 762
Example 16.11: Finding the Reach Networks of a Simple Directed Graph 766
Example 16.12: Reading a Simple Directed Graph 769
Example 16.13: Finding the Shortest Paths for All Source-Sink Pairs 772
Example 16.14: Calculating the Summary Statistics of a Simple Directed Graph . . . 777
Example 16.15: Calculating the Transitive Closure of a Simple Directed Graph 781

Network Action Set: Syntax

biconnectedComponents Action

CASL Syntax

network.biconnectedComponents <**result**=*results*> <**status**=*rc*> /

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

```
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
      *required parameter name="table-name",
        onDemand=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
}
linksVar={
        auxWeight="variable-name",
        from="variable-name",
        id="variable-name",
        to="variable-name",
        vars={"variable-name-1" <, "variable-name-2", ...>},
        weight="variable-name"
}
logFreqTime=integer
logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"
nThreads=integer
```

```
nodes={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
nodesVar={
    node="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
```

outLinks={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outNodes={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*},

...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

standardizedLabels=TRUE | FALSE

;

Parameter Descriptions

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias logFrequencyTime

Default 5

Minimum value 1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

numArticulationPoints

indicates the number of articulation points found by the algorithm. The result is an integer. You can access the value from *results.numArticulationPoints*.

numComponents

indicates the number of biconnected components found by the algorithm. The result is an integer. You can access the value from *results.numComponents*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

centrality Action

CASL Syntax

network.centrality <**result**=*results*> <**status**=*rc*> /

auth="BOTH" | "UNWEIGHT" | "WEIGHT"

between="BOTH" | "UNWEIGHT" | "WEIGHT"

betweenNorm=TRUE | FALSE

close="BOTH" | "UNWEIGHT" | "WEIGHT"

closeNoPath="DIAMETER" | "HARMONIC" | "NNODES" | "ZERO"

clusteringCoef=TRUE | FALSE

degree=TRUE | FALSE

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

eigen="BOTH" | "UNWEIGHT" | "WEIGHT"

eigenAlgorithm="AUTOMATIC" | "JACOBIDAVIDSON" | "POWER"

eigenMaxIters=*integer*

hub="BOTH" | "UNWEIGHT" | "WEIGHT"

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

influence="BOTH" | "UNWEIGHT" | "WEIGHT"

internalFormat="FULL" | "THIN"

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

```
    }, {...}},
    where="where-expression"
}
linksVar={
    auxWeight="variable-name",
    from="variable-name",
    id="variable-name",
    to="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
logFreqTime=integer
logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"
nThreads=integer
nodes={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
```

```
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
}
nodesVar={
        node="variable-name",
        vars={"variable-name-1" <, "variable-name-2", ...>},
        weight="variable-name"
}
outLinks={
        caslib="string",
        compress=TRUE | FALSE,
        indexVars={"variable-name-1" <, "variable-name-2", ...>},
        label="string",
        maxMemSize=64-bit-integer,
        name="table-name",
        onDemand=TRUE | FALSE,
        promote=TRUE | FALSE,
        replace=TRUE | FALSE,
        replication=integer,
        timeStamp="string",
        where={"string-1" <, "string-2", ...>}
}
outNodes={
        caslib="string",
        compress=TRUE | FALSE,
        indexVars={"variable-name-1" <, "variable-name-2", ...>},
        label="string",
```

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

pageRank="BOTH" | "UNWEIGHT" | "WEIGHT"

pageRankAlpha=*double*

pageRankTolerance=*double*

standardizedLabels=TRUE | FALSE

;

Parameter Descriptions

auth="BOTH" | "UNWEIGHT" | "WEIGHT"

specifies which type of authority centrality to calculate.

BOTH

calculates the metric based on both weighted and unweighted graphs.

UNWEIGHT

calculates the metric based on the unweighted graph.

WEIGHT

calculates the metric based on the weighted graph.

between="BOTH" | "UNWEIGHT" | "WEIGHT"

specifies which type of betweenness centrality to calculate.

BOTH

calculates the metric based on both weighted and unweighted graphs.

UNWEIGHT

calculates the metric based on the unweighted graph.

WEIGHT

calculates the metric based on the weighted graph.

betweenNorm=TRUE | FALSE

when set to True, normalizes the betweenness centrality calculation.

Default TRUE

close="BOTH" | "UNWEIGHT" | "WEIGHT"

specifies which type of closeness centrality to calculate.

BOTH

calculates the metric based on both weighted and unweighted graphs.

UNWEIGHT

calculates the metric based on the unweighted graph.

WEIGHT

calculates the metric based on the weighted graph.

closeNoPath="DIAMETER" | "HARMONIC" | "NNODES" | "ZERO"

specifies a method of accounting for the shortest path distance between two nodes when a path does not exist (disconnected nodes).

Default DIAMETER

DIAMETER

uses the graph diameter (plus 1) as the shortest path distance between disconnected nodes.

HARMONIC

uses the harmonic formula for calculating closeness centrality.

NNODES

uses the number of nodes as the shortest path distance between disconnected nodes. You cannot specify this value when the value of the close parameter is WEIGHT or BOTH.

ZERO

uses zero as the shortest path distance between disconnected nodes.

clusteringCoef=TRUE | FALSE

when set to True, calculates the node clustering coefficient.

Default FALSE

degree=TRUE | FALSE

when set to True, calculates the degree centrality.

Default FALSE

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

eigen="BOTH" | "UNWEIGHT" | "WEIGHT"

specifies which type of eigenvector centrality to calculate.

BOTH

calculates the metric based on both weighted and unweighted graphs.

UNWEIGHT

calculates the metric based on the unweighted graph.

WEIGHT

calculates the metric based on the weighted graph.

eigenAlgorithm="AUTOMATIC" | "JACOBIDAVIDSON" | "POWER"

specifies the algorithm to use in calculating centrality metrics that require solving eigensystems - that is, when the eigen, hub, or auth parameter (or some combination) is specified.

Default AUTOMATIC

AUTOMATIC

automatically determines which eigensolver to use.

JACOBIDAVIDSON

uses a variant of the Jacobi-Davidson algorithm for solving eigensystems.

Alias JD

POWER

uses the power method to calculate eigenvectors.

eigenMaxIters=*integer*

specifies the maximum number of iterations to use for eigenvector calculations in order to limit the computation time when convergence is slow.

Default 10000

Minimum value 1

hub="BOTH" | "UNWEIGHT" | "WEIGHT"

specifies which type of hub centrality to calculate.

BOTH

calculates the metric based on both weighted and unweighted graphs.

UNWEIGHT

calculates the metric based on the unweighted graph.

WEIGHT

calculates the metric based on the weighted graph.

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

influence="BOTH" | "UNWEIGHT" | "WEIGHT"

specifies which type of influence centrality to calculate.

BOTH

calculates the metric based on both weighted and unweighted graphs.

UNWEIGHT

calculates the metric based on the unweighted graph.

WEIGHT

calculates the metric based on the weighted graph.

internalFormat="FULL" | "THIN"

specifies the internal graph format to use for the graph data structure.

Default FULL

FULL

stores the graph in standard (adjacency-list-based) format.

THIN

stores the graph in thin (simple list of links) format. This parameter can improve performance in some cases both by reducing memory and by simplifying the construction of the internal data structures. It causes the action to skip the removal of duplicate links when it reads in the graph.

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias logFrequencyTime

Default 5

Minimum value 1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

*** name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

*** name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

pageRank="BOTH" | "UNWEIGHT" | "WEIGHT"

specifies which type of PageRank centrality to calculate.

BOTH

calculates the metric based on both weighted and unweighted graphs.

UNWEIGHT

calculates the metric based on the unweighted graph.

WEIGHT

calculates the metric based on the weighted graph.

pageRankAlpha=*double*

specifies the damping factor for the PageRank algorithm.

Default 0.85

Range 0–1

pageRankTolerance=*double*

specifies the convergence tolerance for the PageRank algorithm.

Default 1e-09

Range 0–1

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

clique Action

CASL Syntax

network.clique <**result**=*results*> <**status**=*rc*> /

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

```
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
      *required parameter name="table-name",
        onDemand=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
}
linksVar={
        auxWeight="variable-name",
        from="variable-name",
        id="variable-name",
        to="variable-name",
        vars={"variable-name-1" <, "variable-name-2", ...>},
        weight="variable-name"
}
logFreqTime=integer
logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"
maxCliques=integer | "ALL"
```

maxLinkWeight=*double*

maxNodeWeight=*double*

maxSize=*integer*

maxTime=*double*

minLinkWeight=*double*

minNodeWeight=*double*

minSize=*integer*

nThreads=*integer*

nodes={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

```
        nfl=integer
    }, {...}},
    where="where-expression"
}
nodesVar={
    node="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
out={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
    replace=TRUE | FALSE,
    replication=integer,
    timeStamp="string",
    where={"string-1" <, "string-2", ...>}
}
outLinks={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
    replace=TRUE | FALSE,
```

```
      replication=integer,
      timeStamp="string",
      where={"string-1" <, "string-2", ...>}
   }
   outNodes={
      caslib="string",
      compress=TRUE | FALSE,
      indexVars={"variable-name-1" <, "variable-name-2", ...>},
      label="string",
      maxMemSize=64-bit-integer,
      name="table-name",
      onDemand=TRUE | FALSE,
      promote=TRUE | FALSE,
      replace=TRUE | FALSE,
      replication=integer,
      timeStamp="string",
      where={"string-1" <, "string-2", ...>}
   }
   outputTables={
      groupByVarsRaw=TRUE | FALSE,
      includeAll=TRUE | FALSE,
      names={"string-1" <, "string-2", ...>} | {key-1={casouttable-1} <, key-2={casouttable-2},
        ...>},
      repeated=TRUE | FALSE,
      replace=TRUE | FALSE
   }
   standardizedLabels=TRUE | FALSE
   timeType="CPU" | "REAL"
;
```

Parameter Descriptions

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table

from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias logFrequencyTime

Default 5

Minimum value 1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

maxCliques=*integer* | "ALL"

specifies the maximum number of cliques to return during clique enumeration. You can specify either an integer (which can be any 32-bit integer greater than or equal to 1) or ALL (which represents the maximum that can be represented by a 32-bit integer).

all represents the maximum that can be represented by a 32-bit integer.

maxLinkWeight=*double*

specifies the maximum sum of link weights in a clique. Any clique whose sum of link weights is greater than this value is removed from the results. The default is the largest number that can be represented by a double, which causes no cliques to be removed from the results.

maxNodeWeight=*double*

specifies the maximum sum of node weights in a clique. Any clique whose sum of node weights is greater than this value is removed from the results. The default is the largest number that can be represented by a double, which causes no cliques to be removed from the results.

maxSize=*integer*

specifies the maximum number of nodes in a clique. Any clique whose size is greater than this value is removed from the results. The default is the largest number that can be represented by a 32-bit integer, which causes no cliques to be removed from the results.

Minimum value 1

maxTime=*double*

specifies the maximum amount of time for the algorithm to spend.

Minimum value (exclusive) 0

minLinkWeight=*double*

specifies the minimum sum of link weights in a clique. Any clique whose sum of link weights is less than this value is removed from the results. The default is the largest (in magnitude) negative number that can be represented by a double, which causes no cliques to be removed from the results.

minNodeWeight=*double*

specifies the minimum sum of node weights in a clique. Any clique whose sum of node weights is less than this value is removed from the results. The default is the largest (in magnitude) negative number that can be represented by a double, which causes no cliques to be removed from the results.

minSize=*integer*

specifies the minimum number of nodes in a clique. Any clique that has fewer nodes than this value is removed from the results. By default, no cliques are removed from the results.

Default 1

Minimum value 1

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

out={*casouttable*}

specifies the output data table to contain the maximal cliques.

For more information about specifying the out parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

timeType="CPU" | "REAL"

specifies whether to use CPU time or real time for the maximum time limit.

Default REAL

CPU

specifies units of CPU time. The time restriction is applied per processing machine (not across all machines).

REAL

specifies units of real time.

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

numCliques

indicates the number of cliques found by the algorithm. The result is an integer. You can access the value from *results.numCliques*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

community Action

CASL Syntax

network.community <**result**=*results*> <**status**=*rc*> /

algorithm="LABELPROP" | "LOUVAIN" | "PARALLELLABELPROP"

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

internalFormat="FULL" | "THIN"

linkRemovalRatio=*integer*

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

```
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
linksVar={
    auxWeight="variable-name",
    from="variable-name",
    id="variable-name",
    to="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
```

logFreqTime=*integer*

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

maxIters=*integer*

nThreads=*integer*

nodes={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

where="*where-expression*"

}

nodesVar={

node="*variable-name*",

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

weight="*variable-name*"

}

outCommLinks={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outCommunity={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outLevel={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outLinks={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outNodes={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outOverlap={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

recursive={

maxCommSize=*integer*,

maxDiameter=*integer*,

relation="AND" | "OR"

}

resolutionList={*double-1* <, *double-2*, ...>}

standardizedLabels=TRUE | FALSE

tolerance=*double*

;

Parameter Descriptions

algorithm="LABELPROP" | "LOUVAIN" | "PARALLELLABELPROP"

specifies the algorithm to use for community detection.

Default LOUVAIN

LABELPROP

uses the label propagation algorithm.

LOUVAIN

uses the Louvain algorithm.

PARALLELLABELPROP

uses the parallel (distributed and threaded) label propagation algorithm. You cannot specify this value when the value of the internalFormat parameter is FULL. This algorithm is experimental in this release.

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

internalFormat="FULL" | "THIN"

specifies the internal graph format to use for the graph data structure.

Default THIN

FULL

stores the graph in standard (adjacency-list-based) format.

THIN

stores the graph in thin (simple list of links) format. This parameter can improve performance in some cases both by reducing memory and by simplifying the construction of the internal data structures. It causes the action to skip the removal of duplicate links when it reads in the graph.

linkRemovalRatio=*integer*

specifies the percentage of small-weight links to be removed around each node neighborhood.

Default 10

Range 0–100

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias	logFrequencyTime
Default	5
Minimum value	1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

maxIters=*integer*

specifies the maximum number of iterations that the algorithm can run. The default value of the maxIters parameter is 20 when the value of the algorithm parameter is LOUVAIN or 100 when the value of the algorithm parameter is LABELPROP or PARALLELLABELPROP.

Minimum value 1

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

outCommLinks={*casouttable*}

specifies the output data table that describes how communities are connected.

For more information about specifying the outCommLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outCommunity={*casouttable*}

specifies the output data table that describes the number of nodes in each community.

For more information about specifying the outCommunity parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outLevel={*casouttable*}

specifies the output data table that describes the number of communities and their corresponding modularity values at various resolution levels.

For more information about specifying the outLevel parameter, see the common

casouttable parameter (Appendix A: Common Parameters).

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outOverlap={*casouttable*}

specifies the output data table that describes the intensity of each node. At the end of community detection, a node could have links that connect to multiple communities. The intensity of a node is computed as the sum of the link weights that connect to nodes in the specified community divided by the total link weights of the node. This data table is computationally expensive to produce, and it requires a large amount of disk space. Therefore, it is not produced if the value of the algorithm parameter is PARALLELLABELPROP together with multiple resolution values in the resolutionList parameter. However, if the value of the algorithm parameter is LOUVAIN, the data table is produced and will contain only results corresponding to the smallest value of the resolutionList parameter.

For more information about specifying the outOverlap parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

recursive={*recursive*}

breaks down large communities into smaller ones until the specified conditions are satisfied. This parameter starts with the keyword recursive followed by any combination of three subparameters enclosed in parentheses.

maxCommSize=*integer*

specifies the maximum number of nodes to be contained in any community. The default is the largest number that can be represented by a 32-bit integer.

Minimum value 1

maxDiameter=*integer*

specifies the maximum number of links on the shortest path between any pair of nodes in any community. The maxDiameter parameter is ignored when the value of the algorithm parameter is PARALLELLABELPROP. The default is the largest number that can be represented by a 32-bit integer.

Minimum value 1

relation="AND" | "OR"

specifies the relationship between the values of the maxCommSize and maxDiameter parameters.

Default OR

AND

the recursive splitting continues until both the maximum community size and the maximum diameter conditions are satisfied.

OR

the recursive splitting continues until either the maximum community size or the maximum diameter condition is satisfied.

resolutionList={*double-1* <, *double-2*, ...>}

specifies a list of resolution values (positive numbers) that are separated by spaces (for example, 4.3 2.1 1.0 0.6 0.2).

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

tolerance=*double*

specifies the tolerance value for when to stop iterations. When the value of the algorithm parameter is LOUVAIN, the algorithm stops iterations when the percentage modularity gain between two consecutive iterations is less than this value. When the value of the algorithm parameter is LABELPROP or PARALLELLABELPROP, the algorithm stops iterations when the percentage of label changes for all nodes in the graph is less than this value. The valid range is strictly between 0 and 1. By default, the tolerance is 0.001 when the value of the algorithm parameter is LOUVAIN or LABELPROP, or 0.01 when the value of the algorithm parameter is PARALLELLABELPROP.

Alias modularity

Range (0, 1)

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

connectedComponents Action

CASL Syntax

network.connectedComponents <**result**=*results*> <**status**=*rc*> /

algorithm="AUTOMATIC" | "DFS" | "PARALLEL" | "UNIONFIND"

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

internalFormat="FULL" | "THIN"

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

```
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
linksVar={
    auxWeight="variable-name",
    from="variable-name",
    id="variable-name",
    to="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
logFreqTime=integer
```

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

nThreads=*integer*

nodes={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>*},

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

where="*where-expression*"

}

nodesVar={

node="*variable-name*",

vars={*"variable-name-1" <, "variable-name-2", ...>*},

weight="*variable-name*"
}
outLinks={
caslib="*string*",
compress=TRUE | FALSE,
indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},
label="*string*",
maxMemSize=*64-bit-integer*,
name="*table-name*",
onDemand=TRUE | FALSE,
promote=TRUE | FALSE,
replace=TRUE | FALSE,
replication=*integer*,
timeStamp="*string*",
where={*"string-1"* <, *"string-2"*, ...>}
}
outNodes={
caslib="*string*",
compress=TRUE | FALSE,
indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},
label="*string*",
maxMemSize=*64-bit-integer*,
name="*table-name*",
onDemand=TRUE | FALSE,
promote=TRUE | FALSE,
replace=TRUE | FALSE,
replication=*integer*,
timeStamp="*string*",
where={*"string-1"* <, *"string-2"*, ...>}
}
outputTables={
groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

standardizedLabels=TRUE | FALSE

;

Parameter Descriptions

algorithm="AUTOMATIC" | "DFS" | "PARALLEL" | "UNIONFIND"

specifies the algorithm to use for calculating connected components.

Default AUTOMATIC

AUTOMATIC

automatically determines the algorithm to use for calculating connected components. By default, the value of the algorithm parameter is UNIONFIND for undirected graphs and DFS for directed graphs.

DFS

uses the depth-first search algorithm for calculating connected components.

PARALLEL

uses the distributed parallel union-find algorithm for calculating connected components. You can use this algorithm only for undirected graphs.

UNIONFIND

uses the union-find algorithm for calculating connected components. You can use this algorithm only for undirected graphs.

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no

direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

internalFormat="FULL" | "THIN"

specifies the internal graph format to use for the graph data structure.

Default THIN

FULL

stores the graph in standard (adjacency-list-based) format.

THIN

stores the graph in thin (simple list of links) format. This parameter can improve performance in some cases both by reducing memory and by simplifying the construction of the internal data structures. It causes the action to skip the removal of duplicate links when it reads in the graph.

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias logFrequencyTime

Default 5

Minimum value 1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the

value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

numComponents

indicates the number of connected components found by the algorithm. The result is an integer. You can access the value from *results.numComponents*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

core Action

CASL Syntax

network.core <**result**=*results*> <**status**=*rc*> /

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

```
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
      *required parameter name="table-name",
        onDemand=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
}
linksVar={
        auxWeight="variable-name",
        from="variable-name",
        id="variable-name",
        to="variable-name",
        vars={"variable-name-1" <, "variable-name-2", ...>},
        weight="variable-name"
}
logFreqTime=integer
logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"
maxTime=double
```

```
nThreads=integer
nodes={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
nodesVar={
    node="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
```

}

outLinks={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outNodes={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

standardizedLabels=TRUE | FALSE

timeType="CPU" | "REAL"

;

Parameter Descriptions

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For

example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias	logFrequencyTime
Default	5
Minimum value	1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

maxTime=*double*

specifies the maximum amount of time for the algorithm to spend.

Minimum value (exclusive) 0

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

timeType="CPU" | "REAL"

specifies whether to use CPU time or real time for the maximum time limit.

Default REAL

CPU

specifies units of CPU time. The time restriction is applied per processing machine (not across all machines).

REAL

specifies units of real time.

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

cycle Action

CASL Syntax

network.cycle <**result**=*results*> <**status**=*rc*> /

algorithm="BACKTRACK" | "BUILD"

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

```
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
linksVar={
    auxWeight="variable-name",
    from="variable-name",
    id="variable-name",
    to="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
logFreqTime=integer
logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"
```

maxCycles=*integer* | "ALL"

maxLength=*integer*

maxLinkWeight=*double*

maxNodeWeight=*double*

maxTime=*double*

minLength=*integer*

minLinkWeight=*double*

minNodeWeight=*double*

nThreads=*integer*

nodes={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>*},

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

```
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
nodesVar={
    node="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
out={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
    replace=TRUE | FALSE,
    replication=integer,
    timeStamp="string",
    where={"string-1" <, "string-2", ...>}
}
outLinks={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
```

```
        replace=TRUE | FALSE,
        replication=integer,
        timeStamp="string",
        where={"string-1" <, "string-2", ...>}
    }
    outNodes={
        caslib="string",
        compress=TRUE | FALSE,
        indexVars={"variable-name-1" <, "variable-name-2", ...>},
        label="string",
        maxMemSize=64-bit-integer,
        name="table-name",
        onDemand=TRUE | FALSE,
        promote=TRUE | FALSE,
        replace=TRUE | FALSE,
        replication=integer,
        timeStamp="string",
        where={"string-1" <, "string-2", ...>}
    }
    outputTables={
        groupByVarsRaw=TRUE | FALSE,
        includeAll=TRUE | FALSE,
        names={"string-1" <, "string-2", ...>} | {key-1={casouttable-1} <, key-2={casouttable-2},
          ...>},
        repeated=TRUE | FALSE,
        replace=TRUE | FALSE
    }
    standardizedLabels=TRUE | FALSE
    timeType="CPU" | "REAL"
;
```

Parameter Descriptions

algorithm="BACKTRACK" | "BUILD"

specifies which algorithm to use in enumerating cycles. By default, the value of the algorithm parameter is BACKTRACK when the value of the maxLength parameter is greater than 20; otherwise, the value of the maxLength parameter is BUILD.

BACKTRACK

uses a backtracking algorithm.

BUILD

uses a building algorithm.

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias logFrequencyTime

Default 5

Minimum value 1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

maxCycles=*integer* | "ALL"

specifies the maximum number of cycles to return during cycle enumeration. You can specify either an integer (which can be any 32-bit integer greater than or equal to 1) or ALL (which represents the maximum that can be represented by a 32-bit integer).

all represents the maximum that can be represented by a 32-bit integer.

maxLength=*integer*

specifies the maximum number of links in a cycle. Any cycle whose length is greater than this value is removed from the results. The default is the largest number that can be represented by a 32-bit integer, which causes no cycles to be removed from the results.

Minimum value 1

maxLinkWeight=*double*

specifies the maximum sum of link weights in a cycle. Any cycle whose sum of link weights is greater than this value is removed from the results. The default is the largest number that can be represented by a double, which causes no cycles to be removed from the results.

maxNodeWeight=*double*

specifies the maximum sum of node weights in a cycle. Any cycle whose sum of node weights is greater than this value is removed from the results. The default is the largest number that can be represented by a double, which causes no cycles to be removed from the results.

maxTime=*double*

specifies the maximum amount of time for the algorithm to spend.

Minimum value (exclusive) 0

minLength=*integer*

specifies the minimum number of links in a cycle. Any cycle that has fewer links than this value is removed from the results. By default, no cycles are removed from the results.

Default 1

Minimum value 1

minLinkWeight=*double*

specifies the minimum sum of link weights in a cycle. Any cycle whose sum of link weights is less than this value is removed from the results. The default is the largest (in magnitude) negative number that can be represented by a double, which causes no cycles to be removed from the results.

minNodeWeight=*double*

specifies the minimum sum of node weights in a cycle. Any cycle whose sum of node weights is less than this value is removed from the results. The default is the largest (in magnitude) negative number that can be represented by a double, which causes no cycles to be removed from the results.

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

out={*casouttable*}

specifies the output data table to contain the cycles found.

For more information about specifying the out parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

timeType="CPU" | "REAL"

specifies whether to use CPU time or real time for the maximum time limit.

Default REAL

CPU

specifies units of CPU time. The time restriction is applied per processing machine (not across all machines).

REAL

specifies units of real time.

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

numCycles

indicates the number of cycles found by the algorithm. The result is an integer. You can access the value from *results.numCycles*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

nodeSimilarity Action

CASL Syntax

network.nodeSimilarity <**result**=*results*> <**status**=*rc*> /

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

internalFormat="FULL" | "THIN"

jaccard=TRUE | FALSE

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

```
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
linksVar={
    auxWeight="variable-name",
    from="variable-name",
    id="variable-name",
    to="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
logFreqTime=integer
```

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

maxScore=*double*

minScore=*double*

nDimensions=*integer*

nSamples=*integer*

nThreads=*integer*

negativeSampleFactor=*integer*

nodes={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

```
    where="where-expression"
}
nodesSubset={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
nodesSubsetVar={
    node="variable-name",
    sink="variable-name",
```

source="*variable-name*"
}
nodesVar={
node="*variable-name*",
vars={*"variable-name-1"* <, *"variable-name-2"*, ...>},
weight="*variable-name*"
}
outLinks={
caslib="*string*",
compress=TRUE | FALSE,
indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},
label="*string*",
maxMemSize=*64-bit-integer*,
name="*table-name*",
onDemand=TRUE | FALSE,
promote=TRUE | FALSE,
replace=TRUE | FALSE,
replication=*integer*,
timeStamp="*string*",
where={*"string-1"* <, *"string-2"*, ...>}
}
outNodes={
caslib="*string*",
compress=TRUE | FALSE,
indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},
label="*string*",
maxMemSize=*64-bit-integer*,
name="*table-name*",
onDemand=TRUE | FALSE,
promote=TRUE | FALSE,
replace=TRUE | FALSE,
replication=*integer*,

```
        timeStamp="string",
        where={"string-1" <, "string-2", ...>}
    }
    outSimilarity={
        caslib="string",
        compress=TRUE | FALSE,
        indexVars={"variable-name-1" <, "variable-name-2", ...>},
        label="string",
        maxMemSize=64-bit-integer,
        name="table-name",
        onDemand=TRUE | FALSE,
        promote=TRUE | FALSE,
        replace=TRUE | FALSE,
        replication=integer,
        timeStamp="string",
        where={"string-1" <, "string-2", ...>}
    }
    outputTables={
        groupByVarsRaw=TRUE | FALSE,
        includeAll=TRUE | FALSE,
        names={"string-1" <, "string-2", ...>} | {key-1={casouttable-1} <, key-2={casouttable-2},
          ...>},
        repeated=TRUE | FALSE,
        replace=TRUE | FALSE
    }
    proximityOrder="FIRST" | "SECOND"
    sink="string" | double
    source="string" | double
    standardizedLabels=TRUE | FALSE
    vector=TRUE | FALSE
;
```

Parameter Descriptions

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

internalFormat="FULL" | "THIN"

specifies the internal graph format to use for the graph data structure.

Default THIN

FULL

stores the graph in standard (adjacency-list-based) format.

THIN

stores the graph in thin (simple list of links) format. This parameter can improve performance in some cases both by reducing memory and by simplifying the construction of the internal data structures. It causes the action to skip the removal of duplicate links when it reads in the graph.

jaccard=TRUE | FALSE

when set to True, calculates Jaccard node similarity values.

Default TRUE

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias logFrequencyTime

Default 5

Minimum value 1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

maxScore=*double*

specifies the maximum similarity value in the outSimilarity table.

Default 1

Range 0–1

minScore=*double*

specifies the minimum similarity value in the outSimilarity table.

Default 0

Range 0–1

nDimensions=*integer*

specifies the number of dimensions to use for vector space node representation.

Default 100

Minimum value 1

nSamples=*integer*

specifies the total number of edges to use in training the model.

Minimum value 1

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

negativeSampleFactor=*integer*

specifies a multiplier for the number of negative training samples for each input edge sample for the vector algorithm.

Default 5

Minimum value 0

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesSubset={*castable*}

specifies the input data table that contains the graph nodes subset information.

Long form nodesSubset={name="*table-name*"}

Shortcut form nodesSubset="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesSubsetVar={*nodesSubsetVarOptT*}

specifies the data variable names for the nodes subset table.

Long form nodesSubsetVar={node="*variable-name*"}

Shortcut form nodesSubsetVar="*variable-name*"

The nodesSubsetVarOptT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

sink="*variable-name*"

specifies the data variable name for the sink indicator.

Default "sink"

source="*variable-name*"

specifies the data variable name for the source indicator.

Default "source"

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results

from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outSimilarity={*casouttable*}

specifies the output data table that describes the similarity values for each pair of nodes.

For more information about specifying the outSimilarity parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

proximityOrder="FIRST" | "SECOND"

specifies the order of the proximity measure of a node to another node.

Default SECOND

FIRST

specifies first-order proximity in a vector node similarity algorithm.

SECOND

specifies second-order proximity in a vector node similarity algorithm.

sink="*string*" | *double*

specifies the sink node for node similarity calculations.

source="*string*" | *double*

specifies the source node for node similarity calculations.

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

vector=TRUE | FALSE

when set to True, calculates vector node similarity values.

Default FALSE

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

path Action

CASL Syntax

network.path <**result**=*results*> <**status**=*rc*> /

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

internalFormat="FULL" | "THIN"

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

```
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
linksVar={
    auxWeight="variable-name",
    from="variable-name",
    id="variable-name",
    to="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
logFreqTime=integer
logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"
```

maxLength=*integer*

maxLinkWeight=*double*

maxNodeWeight=*double*

maxTime=*double*

minLength=*integer*

minLinkWeight=*double*

minNodeWeight=*double*

nThreads=*integer*

nodes={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

```
        nfl=integer
    }, {...}},
    where="where-expression"
}
nodesSubset={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
nodesSubsetVar={
    node="variable-name",
```

sink="*variable-name*",

source="*variable-name*"

}

nodesVar={

node="*variable-name*",

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

weight="*variable-name*"

}

outLinks={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outNodes={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outPathsLinks={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outPathsNodes={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

sink="*string*" | *double*

source="*string*" | *double*

standardizedLabels=TRUE | FALSE

timeType="CPU" | "REAL"

;

Parameter Descriptions

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

internalFormat="FULL" | "THIN"

specifies the internal graph format to use for the graph data structure.

Default THIN

FULL

stores the graph in standard (adjacency-list-based) format.

THIN

stores the graph in thin (simple list of links) format. This parameter can improve performance in some cases both by reducing memory and by simplifying the construction of the internal data structures. It causes the action to skip the removal of duplicate links when it reads in the graph.

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias logFrequencyTime

Default 5

Minimum value 1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

maxLength=*integer*

specifies the maximum number of links in a path. Any path whose length is greater than this value is removed from the results. The default is the largest number that can be represented by a 32-bit integer, which causes no paths to be removed from the results.

Minimum value 1

maxLinkWeight=*double*

specifies the maximum sum of link weights in a path. Any path whose sum of link weights is greater than this value is removed from the results. The default is the largest number that can be represented by a double, which causes no paths to be removed from the results.

maxNodeWeight=*double*

specifies the maximum sum of node weights in a path. Any path whose sum of node weights is greater than this value is removed from the results. The default is the largest number that can be represented by a double, which causes no paths to be removed from the results.

maxTime=*double*

specifies the maximum amount of time for the algorithm to spend.

Minimum value (exclusive) 0

minLength=*integer*

specifies the minimum number of links in a path. Any path that has fewer links than this value is removed from the results. By default, no paths are removed from the results.

Default 1

Minimum value 1

minLinkWeight=*double*

specifies the minimum sum of link weights in a path. Any path whose sum of link weights is less than this value is removed from the results. The default is the largest (in magnitude) negative number that can be represented by a double, which causes no paths to be removed from the results.

minNodeWeight=*double*

specifies the minimum sum of node weights in a path. Any path whose sum of node weights is less than this value is removed from the results. The default is the largest (in magnitude) negative number that can be represented by a double, which causes no paths to be removed from the results.

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesSubset={*castable*}

specifies the input data table that contains the graph nodes subset information.

Long form nodesSubset={name="*table-name*"}

Shortcut form nodesSubset="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesSubsetVar={*nodesSubsetVarOptT*}

specifies the data variable names for the nodes subset table.

Long form nodesSubsetVar={node="*variable-name*"}

Shortcut form nodesSubsetVar="*variable-name*"

The nodesSubsetVarOptT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

sink="*variable-name*"

specifies the data variable name for the sink indicator.

Default "sink"

source="*variable-name*"

specifies the data variable name for the source indicator.

Default "source"

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outPathsLinks={*casouttable*}

specifies the output data table to contain the path links.

For more information about specifying the outPathsLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outPathsNodes={*casouttable*}

specifies the output data table to contain the path nodes.

For more information about specifying the outPathsNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

sink="*string*" | *double*

specifies the sink node for path calculations. This setting overrides the use of the variable 'sink' in the data table that is specified in the nodesSubset parameter.

source="*string*" | *double*

specifies the source node for path calculations. This setting overrides the use of the variable 'source' in the data table that is specified in the nodesSubset parameter.

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

timeType="CPU" | "REAL"

specifies whether to use CPU time or real time for the maximum time limit.

Default REAL

CPU

specifies units of CPU time. The time restriction is applied per processing machine (not across all machines).

REAL

specifies units of real time.

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

numPaths

indicates the number of paths found by the algorithm. The result is a 64-bit integer. You can access the value from *results.numPaths*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

patternMatch Action

CASL Syntax

network.patternMatch <**result**=*results*> <**status**=*rc*> /

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

```
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
        parameters},
    *required parameter name="table-name",
      onDemand=TRUE | FALSE,
      vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
      where="where-expression"
}
linksQuery={
      caslib="string",
      computedOnDemand=TRUE | FALSE,
      computedVars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
```

```
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
linksQueryVar={
    from="variable-name",
    to="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>}
}
linksVar={
    auxWeight="variable-name",
    from="variable-name",
```

```
    id="variable-name",
    to="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
logFreqTime=integer
logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"
nThreads=integer
nodes={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
```

```
    }, {...}},
    where="where-expression"
}
nodesQuery={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
nodesQueryVar={
    node="variable-name",
```

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}
}
nodesVar={
node="*variable-name*",
vars={*"variable-name-1"* <, *"variable-name-2"*, ...>},
weight="*variable-name*"
}
outLinks={
caslib="*string*",
compress=TRUE | FALSE,
indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},
label="*string*",
maxMemSize=*64-bit-integer*,
name="*table-name*",
onDemand=TRUE | FALSE,
promote=TRUE | FALSE,
replace=TRUE | FALSE,
replication=*integer*,
timeStamp="*string*",
where={*"string-1"* <, *"string-2"*, ...>}
}
outMatchLinks={
caslib="*string*",
compress=TRUE | FALSE,
indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},
label="*string*",
maxMemSize=*64-bit-integer*,
name="*table-name*",
onDemand=TRUE | FALSE,
promote=TRUE | FALSE,
replace=TRUE | FALSE,
replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outMatchNodes={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outNodes={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

standardizedLabels=TRUE | FALSE

;

Parameter Descriptions

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksQuery={*castable*}

specifies the input data table that contains the query graph link information.

Long form linksQuery={name="*table-name*"}

Shortcut form linksQuery="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksQueryVar={*linksQueryVarT*}

specifies the data variable names for the links table to query.

Long form linksQueryVar={from="*variable-name*"}

Shortcut form linksQueryVar="*variable-name*"

The linksQueryVarT value can be one or more of the following:

from="*variable-name*"

specifies the data variable name for the 'from' nodes.

Alias fromVar

Default "from"

to="*variable-name*"

specifies the data variable name for the 'to' nodes.

Alias toVar

Default "to"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for link attributes.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias logFrequencyTime

Default 5

Minimum value 1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesQuery={*castable*}

specifies the input data table that contains the query graph node information.

Long form nodesQuery={name="*table-name*"}

Shortcut form nodesQuery="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the

computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesQueryVar={*nodesQueryVarT*}

specifies the data variable names for the nodes table to query.

Long form nodesQueryVar={node="*variable-name*"}

Shortcut form nodesQueryVar="*variable-name*"

The nodesQueryVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outMatchLinks={*casouttable*}

specifies the links output data table for matching subgraphs.

For more information about specifying the outMatchLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outMatchNodes={*casouttable*}

specifies the nodes output data table for matching subgraph mappings.

For more information about specifying the outMatchNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

numMatches

indicates the number of matches found by the algorithm. The result is a 64-bit integer. You can access the value from *results.numMatches*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

reach Action

CASL Syntax

network.reach <**result**=*results*> <**status**=*rc*> /

digraph=TRUE | FALSE

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

eachSource=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

```
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
linksVar={
    auxWeight="variable-name",
    from="variable-name",
    id="variable-name",
    to="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
logFreqTime=integer
```

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

maxReach=*integer*

nThreads=*integer*

nodes={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

where="*where-expression*"

}

nodesSubset={

caslib="*string*",

```
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
nodesSubsetVar={
    node="variable-name",
    reach="variable-name"
}
nodesVar={
    node="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
```

```
    weight="variable-name"
}
outCounts={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
    replace=TRUE | FALSE,
    replication=integer,
    timeStamp="string",
    where={"string-1" <, "string-2", ...>}
}
outLinks={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
    replace=TRUE | FALSE,
    replication=integer,
    timeStamp="string",
    where={"string-1" <, "string-2", ...>}
}
outNodes={
    caslib="string",
```

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outReachLinks={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outReachNodes={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

standardizedLabels=TRUE | FALSE

;

Parameter Descriptions

digraph=TRUE | FALSE

when set to True, calculates the directed reach counts when you are computing the reach networks and includes the directed counts in the resulting output data table, which is specified in the outCounts parameter. This parameter is ignored unless the maxReach parameter is set to 1.

Default FALSE

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

eachSource=TRUE | FALSE

treats each node as a source and calculates a reach network from each one.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias logFrequencyTime

Default 5

Minimum value 1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

maxReach=*integer*

specifies the maximum number of links from each source node in a reach network.

Default 1

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesSubset={*castable*}

specifies the input data table that contains the graph nodes subset information.

Long form nodesSubset={name="*table-name*"}

Shortcut form nodesSubset="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesSubsetVar={*nodesSubsetVarT*}

specifies the data variable names for the nodes subset table.

Long form nodesSubsetVar={node="*variable-name*"}

Shortcut form nodesSubsetVar="*variable-name*"

The nodesSubsetVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

reach="*variable-name*"

specifies the data variable name for the reach index.

Default "reach"

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

outCounts={*casouttable*}

specifies the output data table that describes the number of nodes in each reach network for each set of source nodes.

For more information about specifying the outCounts parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results

from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outReachLinks={*casouttable*}

specifies the output data table that describes the links in each reach network that are found from each set of source nodes. Output of the reach network links can sometimes be more computationally expensive than calculating only the nodes or counts in the reach networks.

For more information about specifying the outReachLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outReachNodes={*casouttable*}

specifies the output data table that describes the nodes in each reach network that are found from each set of source nodes.

For more information about specifying the outReachNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

readGraph Action

CASL Syntax

network.readGraph <**result**=*results*> <**status**=*rc*> /

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

internalFormat="FULL" | "THIN"

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

```
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
linksVar={
    auxWeight="variable-name",
    from="variable-name",
    id="variable-name",
    to="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
logFreqTime=integer
logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"
```

```
nThreads=integer
nodes={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
nodesVar={
    node="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
```

}

outLinks={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outNodes={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

standardizedLabels=TRUE | FALSE

;

Parameter Descriptions

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

internalFormat="FULL" | "THIN"

specifies the internal graph format to use for the graph data structure.

Default FULL

FULL

stores the graph in standard (adjacency-list-based) format.

THIN

stores the graph in thin (simple list of links) format. This parameter can improve performance in some cases both by reducing memory and by simplifying the construction of the internal data structures. It causes the action to skip the removal of duplicate links when it reads in the graph.

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For

more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias logFrequencyTime

Default 5

Minimum value 1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results

from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

solutionStatusCode

indicates the solution status code of the selected problem type (algorithm class). The result is a 64-bit integer. You can access the value from *results.solutionStatusCode*.

Network Action Set: Syntax

shortestPath Action

CASL Syntax

network.shortestPath <**result**=*results*> <**status**=*rc*> /

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

```
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
      *required parameter name="table-name",
        onDemand=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
}
linksVar={
        auxWeight="variable-name",
        from="variable-name",
        id="variable-name",
        to="variable-name",
        vars={"variable-name-1" <, "variable-name-2", ...>},
        weight="variable-name"
}
logFreqTime=integer
logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"
maxPathWeight=double
```

nThreads=*integer*

nodes={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

where="*where-expression*"

}

nodesSubset={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

```
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        computedVarsProgram="string",
        dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
      *required parameter name="table-name",
        onDemand=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
}
nodesSubsetVar={
        node="variable-name",
        sink="variable-name",
        source="variable-name"
}
nodesVar={
        node="variable-name",
        vars={"variable-name-1" <, "variable-name-2", ...>},
        weight="variable-name"
}
```

outLinks={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outNodes={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outPaths={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outWeights={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

sink="*string*" | *double*

source="*string*" | *double*

standardizedLabels=TRUE | FALSE

;

Parameter Descriptions

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias	logFrequencyTime
Default	5
Minimum value	1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

maxPathWeight=*double*

specifies the maximum path weight. Any shortest path whose sum of link weights is greater than this value is removed from the results. The default is the largest number that can be represented by a double, which causes no paths to be removed from the results.

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesSubset={*castable*}

specifies the input data table that contains the graph nodes subset information.

Long form nodesSubset={name="*table-name*"}

Shortcut form nodesSubset="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesSubsetVar={*nodesSubsetVarOptT*}

specifies the data variable names for the nodes subset table.

Long form nodesSubsetVar={node="*variable-name*"}

Shortcut form nodesSubsetVar="*variable-name*"

The nodesSubsetVarOptT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

sink="*variable-name*"

specifies the data variable name for the sink indicator.

Default "sink"

source="*variable-name*"

specifies the data variable name for the source indicator.

Default "source"

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results

from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outPaths={*casouttable*}

specifies the output data table to contain the shortest paths.

For more information about specifying the outPaths parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outWeights={*casouttable*}

specifies the output data table to contain the shortest path summaries.

For more information about specifying the outWeights parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

sink="*string*" | *double*

specifies the sink node for shortest path calculations. The value of this parameter overrides the use of the variable 'sink' in the data table that you specify in the nodesSubset parameter.

source="*string*" | *double*

specifies the source node for shortest path calculations. The value of this parameter overrides the use of the variable 'source' in the data table that you specify in the nodesSubset parameter.

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

numPaths

indicates the number of shortest paths found by the algorithm. The result is a 64-bit integer. You can access the value from *results.numPaths*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

summary Action

CASL Syntax

network.summary <**result**=*results*> <**status**=*rc*> /

biconnectedComponents=TRUE | FALSE

connectedComponents=TRUE | FALSE

diameterApprox="BOTH" | "UNWEIGHT" | "WEIGHT"

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

```
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
linksVar={
    auxWeight="variable-name",
    from="variable-name",
    id="variable-name",
    to="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
```

logFreqTime=*integer*

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

nThreads=*integer*

nodes={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

where="*where-expression*"

}

nodesVar={

node="*variable-name*",

```
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
out={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
    replace=TRUE | FALSE,
    replication=integer,
    timeStamp="string",
    where={"string-1" <, "string-2", ...>}
}
outLinks={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
    replace=TRUE | FALSE,
    replication=integer,
    timeStamp="string",
    where={"string-1" <, "string-2", ...>}
}
outNodes={
```

```
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
    replace=TRUE | FALSE,
    replication=integer,
    timeStamp="string",
    where={"string-1" <, "string-2", ...>}
  }
  outputTables={
    groupByVarsRaw=TRUE | FALSE,
    includeAll=TRUE | FALSE,
    names={"string-1" <, "string-2", ...>} | {key-1={casouttable-1} <, key-2={casouttable-2},
      ...>},
    repeated=TRUE | FALSE,
    replace=TRUE | FALSE
  }
  shortestPath="BOTH" | "UNWEIGHT" | "WEIGHT"
  standardizedLabels=TRUE | FALSE
;
```

Parameter Descriptions

biconnectedComponents=TRUE | FALSE

calculates information about biconnected components. The graph must be undirected to use this parameter.

Default FALSE

connectedComponents=TRUE | FALSE

calculates information about connected components.

Default FALSE

diameterApprox="BOTH" | "UNWEIGHT" | "WEIGHT"

calculates information about the approximate diameter and specifies which type of calculation to perform. Use this parameter when calculating the exact diameter (by calculating all shortest paths) is too computationally expensive.

BOTH

calculates the metric based on both weighted and unweighted graphs.

UNWEIGHT

calculates the metric based on the unweighted graph.

WEIGHT

calculates the metric based on the weighted graph.

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For

example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias	logFrequencyTime
Default	5
Minimum value	1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

out={*casouttable*}

specifies the output data table to contain the summary results.

For more information about specifying the out parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

shortestPath="BOTH" | "UNWEIGHT" | "WEIGHT"

calculates information about shortest paths and specifies which type of calculation to perform.

BOTH

calculates the metric based on both weighted and unweighted graphs.

UNWEIGHT

calculates the metric based on the unweighted graph.

WEIGHT

calculates the metric based on the weighted graph.

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Network Action Set: Syntax

transitiveClosure Action

CASL Syntax

network.transitiveClosure <**result**=*results*> <**status**=*rc*> /

direction="DIRECTED" | "UNDIRECTED"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

distributed=TRUE | FALSE

includeSelfLink=TRUE | FALSE

indexOffset=*integer*

links={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

```
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
      *required parameter name="table-name",
        onDemand=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
}
linksVar={
        auxWeight="variable-name",
        from="variable-name",
        id="variable-name",
        to="variable-name",
        vars={"variable-name-1" <, "variable-name-2", ...>},
        weight="variable-name"
}
logFreqTime=integer
logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"
nThreads=integer
```

```
nodes={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
nodesVar={
    node="variable-name",
    vars={"variable-name-1" <, "variable-name-2", ...>},
    weight="variable-name"
}
```

out={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outLinks={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outNodes={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

standardizedLabels=TRUE | FALSE

;

Parameter Descriptions

direction="DIRECTED" | "UNDIRECTED"

specifies whether to consider the input graph as directed or undirected.

Default UNDIRECTED

DIRECTED

specifies the graph as directed. In a directed graph, each link (i,j) has a direction that defines how something (such as information) can flow over that link. In link (i,j), the flow is from node i to node j. Node i is called the source (tail) node, and node j is called the sink (head) node.

UNDIRECTED

specifies the input graph as undirected. In an undirected graph, each link {i,j} has no direction, and the flow can be in either direction. That is, {i,j} = {j,i}.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

distributed=TRUE | FALSE

when set to True, uses a distributed graph.

Default FALSE

includeSelfLink=TRUE | FALSE

when set to True, includes self-links when an input graph is read.

Default FALSE

indexOffset=*integer*

specifies the index offset for identifiers in the log and results output data tables. For example, if three entities are found, they are labeled entity 1, 2, and 3 by default. If the value of indexOffset is 4, the entities are labeled entity 4, 5, and 6.

Default 0

Minimum value 0

links={*castable*}

specifies the input data table that contains the graph link information.

Long form links={name="*table-name*"}

Shortcut form links="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

linksVar={*linksVarT*}

specifies the data variable names for the links table.

For more information about specifying the linksVar parameter, see the common linksVarT parameter (Appendix A: Common Parameters).

logFreqTime=*integer*

controls the frequency, in number of seconds, for displaying iteration logs. This parameter

is useful for computationally intensive algorithms. Setting this value too low can hurt algorithm performance. The value of the parameter can be any integer greater than or equal to 1.

Alias logFrequencyTime

Default 5

Minimum value 1

logLevel="AGGRESSIVE" | "BASIC" | "MODERATE" | "NONE"

controls the amount of information that is displayed in the SAS log.

Default BASIC

AGGRESSIVE

displays a more detailed summary of the input, output, and algorithmic processing.

BASIC

displays a brief summary of the algorithmic processing.

MODERATE

displays a moderately detailed summary of the input, output, and algorithmic processing.

NONE

turns off all action-related messages in the SAS log.

nThreads=*integer*

specifies the maximum number of threads to use for multithreaded processing.

Range 1–256

nodes={*castable*}

specifies the input data table that contains the graph node information.

Long form nodes={name="*table-name*"}

Shortcut form nodes="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nodesVar={*nodesVarT*}

specifies the data variable names for the nodes table.

Long form nodesVar={node="*variable-name*"}

Shortcut form nodesVar="*variable-name*"

The nodesVarT value can be one or more of the following:

node="*variable-name*"

specifies the data variable name for the nodes.

Default "node"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for node attributes.

weight="*variable-name*"

specifies the data variable name for the node weights.

Default "weight"

out={*casouttable*}

specifies the output data table to contain the transitive closure results.

For more information about specifying the out parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outLinks={*casouttable*}

specifies the output data table to contain the graph link information along with any results from the algorithms that calculate metrics on links.

For more information about specifying the outLinks parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outNodes={*casouttable*}

specifies the output data table to contain the graph node information along with any results from the algorithms that calculate metrics on nodes.

For more information about specifying the outNodes parameter, see the common casouttable parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

standardizedLabels=TRUE | FALSE

when set to True, specifies that the input graph data are in a standardized format.

Default FALSE

Result Descriptions

ProblemSummary

contains a basic summary of the graph input. The result is a table. You can access the value from *results.ProblemSummary*.

SolutionSummary

contains a basic solution summary for the algorithm. The result is a table. You can access the value from *results.SolutionSummary*.

solutionStatus

indicates the solution status of the selected problem type (algorithm class). The result is a string. You can access the value from *results.solutionStatus*.

Details

The network Action Set

The **network** action set includes a number of graph theory and network analysis algorithms that can augment data mining and machine learning approaches. In many practical applications of data mining and machine learning models, pairwise interaction between the entities of interest in the model often plays an important role. For example, when you are modeling churn in a telecommunications network to support a retention campaign, the influence of individual customers on the other customers—such as friends and acquaintances that they regularly interact with—might contribute to the propensity of other customers to churn. You could likewise imagine a customer being able to influence the propensity of his or her acquaintances to acquire new products. Social networks such as Facebook and Twitter are obvious examples of networks that represent such interactions between individuals.

Networks also appear explicitly and implicitly in many other application contexts. Networks are often constructed from certain natural co-occurrence types of relationships—such as relationships among researchers who coauthor articles, actors who appear in the same movie, words or topics that occur in the same document, items that appear together in a shopping basket, terrorism suspects who travel together or are seen in the same location, and so on. In these types of relationship, the strength or frequency of interaction is modeled as weights on the links of the resulting network.

To support the myriad ways in which networks appear in data mining, the **network** action set makes no assumptions about the context or application from which the network arises. It provides a number of network analysis algorithms that take an abstract graph or network as input and help explain network structure and compute important network measures. Depending on the application, this type of network analysis can stand on its own and provide independent value, or it can support machine learning models—for example, by providing additional features that are derived from network measures such as node centrality.

As input, the actions in the **network** action set expects *graph* $G = (N, A)$, which is defined over a set N of nodes and a set A of arcs. A *node* is an abstract representation of some entity (or object), and an *arc* defines the relationship (or connection) between two nodes. The terms *node* and *vertex* are interchangeable in describing an entity. The term *arc* is interchangeable with the term *edge* or *link* in describing a connection. Similarly, the terms *graph* and *network* are interchangeable.

For more information about network analysis algorithms and the capabilities of the **network** action set, see the section "Details: NETWORK Procedure" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The biconnectedComponents Action

A *biconnected component* of a graph $G = (N, A)$ is a connected subgraph that you cannot break into disconnected pieces by deleting any single node (and its incident links). An *articulation point* of a graph is a node whose removal would cause an increase in the number of connected components. Articulation points can be important when you analyze any graph that represents a communications network. Consider an articulation point $i \in N$ that, if removed, breaks the graph into two components, C^1 and C^2. All paths in G between some nodes in C^1 and some nodes in C^2 must pass through node i. In this sense, articulation points

are critical to communication. Examples where articulation points are important include airline hubs, electric circuits, network wires, protein bonds, traffic routers, and many other industrial applications.

For more information about biconnected components and the capabilities of the **biconnectedComponents** action, see the section "Biconnected Components and Articulation Points" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The centrality Action

In general terms, the *centrality* of a node or link in a graph indicates its relative importance within the graph. In the field of network analysis, many different types of centrality metrics are used to better understand levels of prominence.

For more information about centrality and the capabilities of the **centrality** action, see the section "Centrality" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The clique Action

A *clique* of a graph $G = (N, A)$ is an induced subgraph that is a complete graph. Every node in a clique is connected to every other node in that clique. A *maximal clique* is a clique that is not a subset of the nodes of any larger clique. That is, it is a set C of nodes such that every pair of nodes in C is connected by a link and every node not in C is missing a link to at least one node in C. The number of maximal cliques in a particular graph can be very large and can grow exponentially with every node that is added. Finding cliques in graphs has applications in many industries, including bioinformatics, social networks, electrical engineering, and chemistry.

For more information about cliques and the capabilities of the **clique** action, see the section "Clique Enumeration" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The community Action

Community detection partitions a graph into communities such that the nodes within the community subgraphs are more densely connected than the nodes from different communities.

For more information about community detection and the capabilities of the **community** action, see the section "Community Detection" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The connectedComponents Action

A *connected component* of a graph is a set of nodes that are all reachable from each other. That is, if two nodes are in the same component, then there is a path between them. For a directed graph, there are two types of components: a *strongly connected* component has a directed path between any two nodes, and a *weakly connected* component ignores direction and requires only that a path exist between any two nodes.

For more information about connected components and the capabilities of the **connectedComponents** action, see the section "Connected Components" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The core Action

An alternative to community detection for finding cohesive subgroups is a method of extracting *k-cores* known as *core decomposition*. Although this method is generally not as powerful as community detection for extracting a detailed community structure, it can provide a coarse approximation of cohesive structure at a very low computational cost. Let $G = (N, A)$ define a graph that contains nodes N and links A, and let $G_S = (S, A_S)$ be an induced subgraph on nodes S. The subgraph G_S is a *k-core* if and only if for every node $v \in S$, the degree of v is greater than or equal to k, and G_S is the maximum subgraph with this property. By definition, the cores are nested. That is, if G_{S_k} is a k-core of size k, then $G_{S_{k+1}}$ is contained in G_{S_k}.

For more information about core decomposition and the capabilities of the **core** action, see the section "Core Decomposition" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The cycle Action

A *path* in a graph is a sequence of nodes, each of which has a link to the next node in the sequence. An *elementary cycle* is a path in which the starting node and the ending node are the same and no other node appears more than once in the sequence.

For more information about cycle detection and the capabilities of the **cycle** action, see the section "Cycle Enumeration" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The nodeSimilarity Action

Node similarity is a measure of how similar the neighborhoods of two nodes are.

For more information about node similarity and the capabilities of the **nodeSimilarity** action, see the section "Node Similarity" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The path Action

A *path* in a graph is a sequence of nodes, each of which has a link to the next node in the sequence. An *elementary path* is a path in which no node appears more than once in the sequence. A path between two nodes, u and v, in a graph is a path that starts at u and ends at v. The starting node is called the *source node*, and the ending node is called the *sink node*.

For more information about path enumeration and the capabilities of the **path** action, see the section "Path Enumeration" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The patternMatch Action

Given two graphs, G (input) and Q (query), *subgraph isomorphism* is the problem of finding all subgraphs Q' of G that are isomorphic to Q. In other words, each such subgraph Q' has the same topology as graph Q. *Pattern matching* addresses the analogous problem in the presence of node and link attributes. It is the problem of finding all subgraphs Q' of G isomorphic to graph Q such that all node and link attributes that are defined in Q are preserved in Q' under the isomorphism map.

For more information about pattern matching and the capabilities of the **patternMatch** action, see the section "Pattern Matching" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The reach Action

The *reach network* of a graph $G = (N, A)$ is a graph $G_L^R = (N_L^R, A_L^R)$ that is defined as the induced subgraph over the set of nodes N_L^R that are reachable in L steps (or hops) from a set S of nodes, called the *source nodes*. In the context of social networks, reach networks are often referred to as *ego networks*, because they focus on the neighbors of one particular individual (or more than one).

For more information about reach networks and the capabilities of the **reach** action, see the section "Reach (Ego) Network" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The readGraph Action

As input, the actions in the **network** action set expect *graph* $G = (N, A)$, which is defined over a set N of nodes and a set A of arcs. A *node* is an abstract representation of some entity (or object), and an *arc* defines the relationship (or connection) between two nodes. The **readGraph** action simply reads a links data table or a nodes data table (or both) that describes the graph.

For more information about the format of the graph input data tables, see the section "Graph Input Data" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The shortestPath Action

A *shortest path* between two nodes, u and v, in a graph is a path that starts at u and ends at v and has the lowest total link weight. The starting node is called the *source node*, and the ending node is called the *sink node*.

For more information about shortest paths and the capabilities of the **shortestPath** action, see the section "Shortest Path" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The summary Action

You can calculate various summary statistics for a graph and its nodes by using the **summary** action.

For more information about graph summary statistics and the capabilities of the **summary** action, see the section "Summary Statistics" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

The transitiveClosure Action

The *transitive closure* of a graph G is a graph $G^T = (N, A^T)$ such that for all $i, j \in N$ there is a link $(i, j) \in A^T$ if and only if there is a path from i to j in G.

For more information about transitive closure and the capabilities of the **transitiveClosure** action, see the section "Transitive Closure" (Chapter 2, *SAS Visual Data Mining and Machine Learning: The NETWORK Procedure*).

Examples

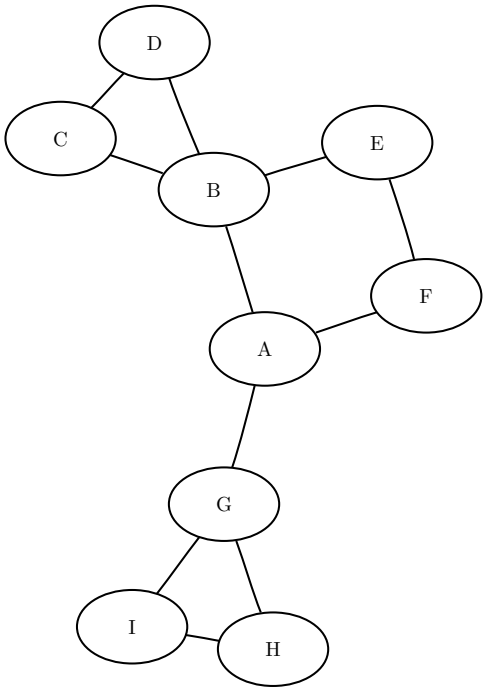
Example 16.1: Finding the Biconnected Components of a Simple Undirected Graph

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the biconnected components algorithm on the simple undirected graph G shown in Figure 16.1.

Figure 16.1 A Simple Undirected Graph G



The undirected graph G can be represented by the following links data set, LinkSetIn:

```
data LinkSetIn;
   input from $ to $ @@;
   datalines;
A B  A F  A G  B C  B D
B E  C D  E F  G I  G H
H I
;
```

The following DATA step loads the LinkSetIn data set into a CAS data table named mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements calculate the biconnected components and articulation points for G and output the results in the data tables LinkSetOut and NodeSetOut:

```
proc cas;
   loadactionset "network";
   action network.biconnectedComponents result=r status=s /
      indexOffset = 1
      links       = {name = "LinkSetIn"}
      outNodes    = {name = "NodeSetOut", replace="true"}
      outLinks    = {name = "LinkSetOut", replace="true"};
   run;
   print r.ProblemSummary; run;
```

```
   print r.SolutionSummary; run;
   action table.fetch / table = "LinkSetOut" sortBy = "biconcomp"; run;
   action table.fetch / table = "NodeSetOut"; run;
quit;
```

The problem summary output from this action is presented in Output 16.1.1.

Output 16.1.1 Problem Summary

Problem Summary	
Number of Nodes	9
Number of Links	11
Graph Direction	Undirected

The solution summary output from this action is presented in Output 16.1.2.

Output 16.1.2 Solution Summary

Solution Summary	
Problem Type	Biconnected Components
Solution Status	OK
Number of Components	4
Number of Articulation Points	3
CPU Time	0.00
Real Time	0.00

The output data table LinkSetOut contains the biconnected components of the input graph, as shown in Output 16.1.3.

Output 16.1.3 Biconnected Components of a Simple Undirected Graph

Selected Rows from Table LINKSETOUT			
Index	**from**	**to**	**biconcomp**
1	A	F	1
2	A	B	1
3	B	E	1
4	E	F	1
5	A	G	2
6	C	D	3
7	B	C	3
8	B	D	3
9	H	I	4
10	G	H	4
11	G	I	4

The output data table NodeSetOut contains the articulation points of the input graph, as shown in Output 16.1.4.

Output 16.1.4 Articulation Points of a Simple Undirected Graph

Selected Rows from Table NODESETOUT		
Index	node	artpoint
1	A	1
2	B	1
3	D	0
4	G	1
5	I	0
6	F	0
7	E	0
8	H	0
9	C	0

The biconnected components are shown graphically in Figure 16.2 and Figure 16.3.

Figure 16.2 Biconnected Components C^1 and C^2

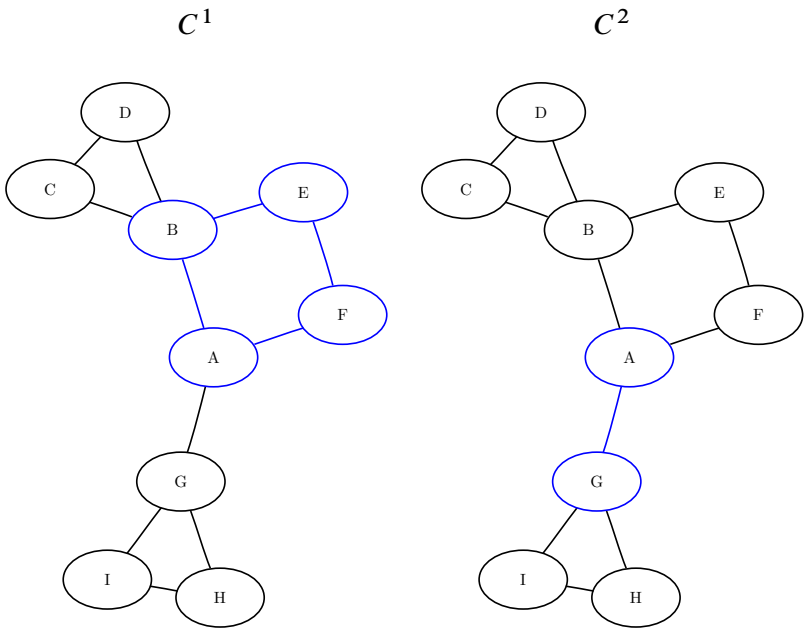
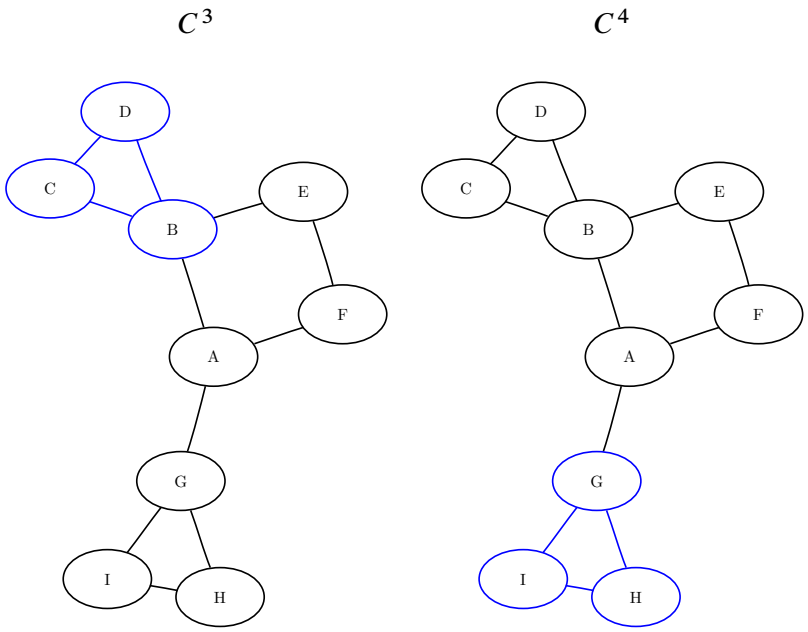


Figure 16.3 Biconnected Components C^3 and C^4



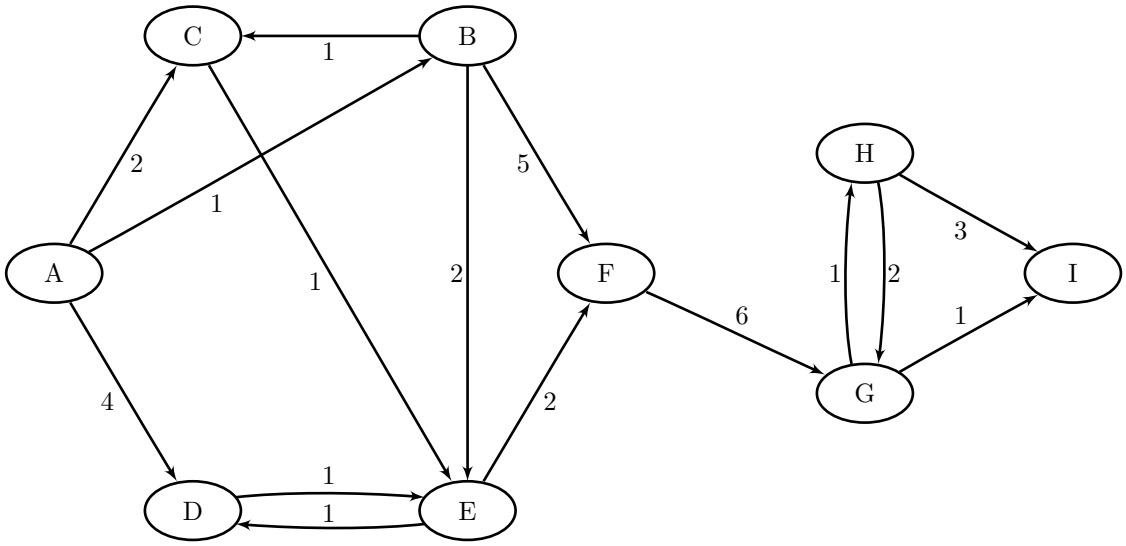
Example 16.2: Calculating the Centrality of a Simple Directed Graph

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the **`centrality`** action on the simple directed graph *G* shown in Figure 16.4.

Figure 16.4 A Simple Directed Graph



The directed graph *G* can be represented by the following links data set, LinkSetIn:

```
data LinkSetIn;
   input from $ to $ weight @@;
   datalines;
A B 1  A C 2  A D 4  B C 1  B E 2
B F 5  C E 1  D E 1  E D 1  E F 2
F G 6  G H 1  G I 1  H G 2  H I 3
;
```

The following DATA step loads the LinkSetIn data set into a CAS data table named mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements calculate various centrality metrics for *G* and output the results in the data table NodeSetOut:

```
proc cas;
   loadactionset "network";
   action network.centrality result=r status=s /
      direction = "directed"
      links     = {name = "LinkSetIn"}
      degree    = true
      influence = "weight"
      close     = "weight"
      between   = "weight"
      hub       = "weight"
      auth      = "weight"
      outNodes  = {name = "NodeSetOut", replace=true};
  run;
```

```
   print r.ProblemSummary; run;
   print r.SolutionSummary; run;
   action table.fetch / table = "NodeSetOut"; run;
quit;
```

The problem summary output from this action is presented in Output 16.2.1.

Output 16.2.1 Problem Summary

Problem Summary	
Number of Nodes	9
Number of Links	15
Graph Direction	Directed

The solution summary output from this action is presented in Output 16.2.2.

Output 16.2.2 Solution Summary

Solution Summary	
Problem Type	Centrality
Solution Status	OK
CPU Time	0.00
Real Time	0.01

The output data table NodeSetOut contains the centrality metrics of the input graph, as shown in Output 16.2.3.

Output 16.2.3 Centrality Metrics of a Simple Directed Graph

Selected Rows from Table NODESETOUT							
Index	node	centr_degree_in	centr_degree_out	centr_degree	centr_hub_wt	centr_auth_wt	centr_close_wt
1	A	0	3	3	0	0	0.3883495146
2	B	1	3	4	0	0	0.5513377926
3	C	2	1	3	0	0	0.3826021465
4	D	2	1	3	0	0	0.4011305349
5	E	3	2	5	0	0	0.4596188126
6	F	2	1	3	1	0	0.443572327
7	G	2	2	4	0.0329217416	1	0.56401766
8	H	1	2	3	0.4296988036	0.0047995092	0.4166311739
9	I	2	0	2	0	0.1927309405	0.3022670025

Selected Rows from Table NODESETOUT				
centr_close_in_wt	centr_close_out_wt	centr_between_wt	centr_influence1_wt	centr_influence2_wt
0	0.7766990291	0	0.7777777778	1.1111111111
0.3	0.8026755853	0.0714285714	0.8888888889	1.1111111111
0.3404255319	0.4247787611	0	0.1111111111	0.3333333333
0.4120171674	0.3902439024	0.0178571429	0.1111111111	0.3333333333
0.4343891403	0.4848484848	0.1785714286	0.3333333333	0.7777777778
0.5031446541	0.384	0.2678571429	0.6666666667	0.2222222222
0.7947019868	0.3333333333	0.2142857143	0.2222222222	0.5555555556
0.4828973843	0.3503649635	0	0.5555555556	0.2222222222
0.604534005	0	0	0	0

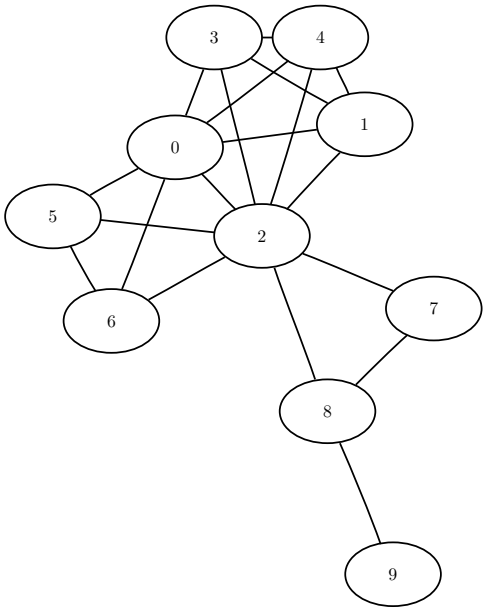
Example 16.3: Finding the Maximal Cliques of a Simple Undirected Graph

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the **clique** action on the simple undirected graph *G* shown in Figure 16.5.

Figure 16.5 A Simple Undirected Graph *G*



The undirected graph *G* can be represented by the following links data set, LinkSetIn:

```
data LinkSetIn;
   input from to @@;
   datalines;
0 1  0 2  0 3  0 4  0 5
0 6  1 2  1 3  1 4  2 3
2 4  2 5  2 6  2 7  2 8
3 4  5 6  7 8  8 9
;
```

The following DATA step loads the LinkSetIn data set into a CAS data table named mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements calculate the maximal cliques and output the results in the data table Cliques:

```
proc cas;
   loadactionset "network";
   action network.clique result=r status=s /
      indexOffset = 1
      links       = {name = "LinkSetIn"}
      out         = {name = "Cliques", replace=true}
      maxCliques  = "all";
   run;
   print r.ProblemSummary; run;
   print r.SolutionSummary; run;
   action table.fetch / table = "Cliques" sortBy = "clique"; run;
quit;
```

The problem summary output from this action is presented in Output 16.3.1.

Output 16.3.1 Problem Summary

Problem Summary	
Number of Nodes	10
Number of Links	19
Graph Direction	Undirected

The solution summary output from this action is presented in Output 16.3.2.

Output 16.3.2 Solution Summary

Solution Summary	
Problem Type	Clique
Solution Status	OK
Number of Cliques	4
CPU Time	0.00
Real Time	0.00

The output data table Cliques now contains the maximal cliques of the input graph, as shown in Output 16.3.3.

Output 16.3.3 Maximal Cliques of a Simple Undirected Graph

Selected Rows from Table CLIQUES		
Index	clique	node
1	1	3
2	1	4
3	1	1
4	1	0
5	1	2
6	2	2
7	2	0
8	2	6
9	2	5
10	3	7
11	3	8
12	3	2
13	4	9
14	4	8

The maximal cliques are shown graphically in Figure 16.6 and Figure 16.7.

Figure 16.6 Maximal Cliques C^1 and C^2

$C^1 = \{0, 1, 2, 3, 4\}$ $C^2 = \{0, 2, 5, 6\}$

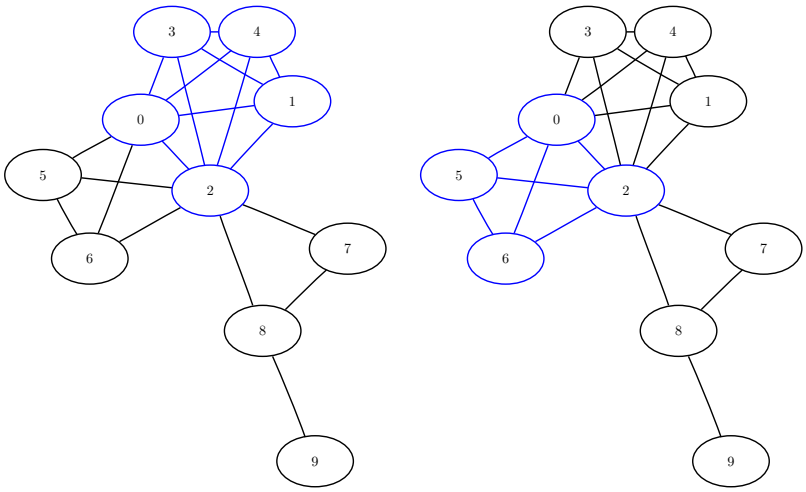
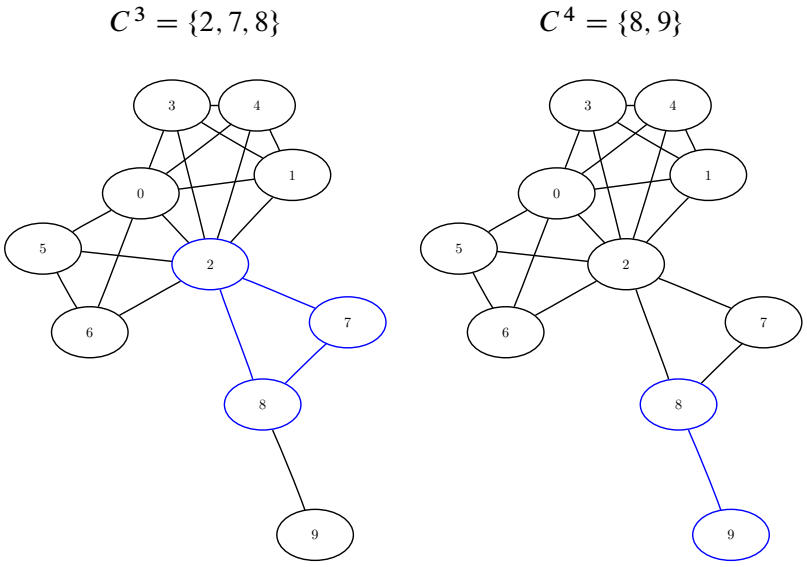


Figure 16.7 Maximal Cliques C^3 and C^4



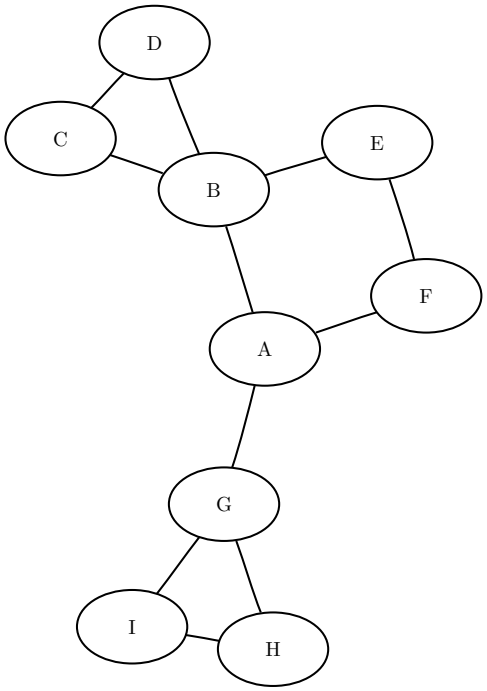
Example 16.4: Detecting Communities for a Simple Undirected Graph

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the **`community`** action on the simple undirected graph G shown in Figure 16.8.

Figure 16.8 A Simple Undirected Graph G



The undirected graph G can be represented by the following links data set, LinkSetIn:

```
data LinkSetIn;
   input from $ to $ @@;
   datalines;
A B  A F  A G  B C  B D
B E  C D  E F  G I  G H
H I
;
```

The following DATA step loads the LinkSetIn data set into a CAS data table named mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements perform community detection and output the results in the specified data tables:

```
proc cas;
   loadactionset "network";
   action network.community result=r status=s /
      indexOffset    = 1
      links          = {name = "LinkSetIn"}
      outNodes       = {name = "NodeSetOut", replace=true}
      outLevel       = {name = "CommLevelOut", replace=true}
      outCommunity   = {name = "CommOut", replace=true}
      outOverlap     = {name = "CommOverlapOut", replace=true}
      outCommLinks   = {name = "CommLinksOut", replace=true}
```

```
      resolutionList = {1.0, 0.5};
   run;
   print r.ProblemSummary; run;
   print r.SolutionSummary; run;
   action table.fetch / table = "NodeSetOut"; run;
   action table.fetch / table = "CommLevelOut" sortBy = "level"; run;
   action table.fetch / table = "CommOut" sortBy = {"community","level"}; run;
   action table.fetch / table = "CommOverlapOut" sortBy = "community"; run;
   action table.fetch / table = "CommLinksOut" sortBy = "level"; run;
quit;
```

The problem summary output from this action is presented in Output 16.4.1.

Output 16.4.1 Problem Summary

Problem Summary	
Number of Nodes	9
Number of Links	11
Graph Direction	Undirected

The solution summary output from this action is presented in Output 16.4.2.

Output 16.4.2 Solution Summary

Solution Summary	
Problem Type	Community Detection
Solution Status	OK
CPU Time	0.00
Real Time	0.00

The output data table NodeSetOut contains the community identifier of each node, as shown in Output 16.4.3.

Output 16.4.3 Community Detection on an Undirected Simple Graph: Nodes Output

Selected Rows from Table NODESETOUT			
Index	node	community_1	community_2
1	A	1	1
2	B	2	1
3	F	1	1
4	G	3	2
5	C	2	1
6	D	2	1
7	E	1	1
8	H	3	2
9	I	3	2

The output data table CommLevelOut contains summary information at each resolution level, as shown in Output 16.4.4.

Output 16.4.4 Community Detection on an Undirected Simple Graph: Level Output

Selected Rows from Table COMMLEVELOUT				
Index	level	resolution	communities	modularity
1	1	1	3	0.3925619835
2	2	0.5	2	0.3429752066

The output data table CommOut contains the number of nodes in each community, as shown in Output 16.4.5.

Output 16.4.5 Community Detection on an Undirected Simple Graph: Community Summary

Selected Rows from Table COMMOUT				
Index	level	resolution	community	nodes
1	1	1	1	3
2	2	0.5	1	6
3	1	1	2	3
4	2	0.5	2	3
5	1	1	3	3

The output data table CommOverlapOut contains community overlap information, as shown in Output 16.4.6.

Output 16.4.6 Community Detection on an Undirected Simple Graph: Community Overlap

Selected Rows from Table COMMOVERLAPOUT			
Index	node	community	intensity
1	G	1	0.3333333333
2	E	1	1
3	F	1	1
4	A	1	0.6666666667
5	B	1	1
6	C	1	1
7	D	1	1
8	G	2	0.6666666667
9	I	2	1
10	H	2	1
11	A	2	0.3333333333

The output data table CommLinksOut describes how the communities are connected, as shown in Output 16.4.7.

Output 16.4.7 Community Detection on an Undirected Simple Graph: Intercommunity Links

Selected Rows from Table COMMLINKSOUT					
Index	level	resolution	from_community	to_community	link_weight
1	1	1	1	3	1
2	1	1	1	2	2
3	2	0.5	1	2	1

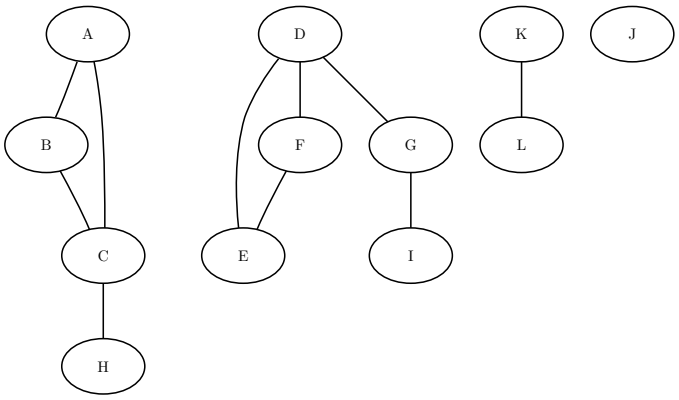
Example 16.5: Finding the Connected Components of a Simple Undirected Graph

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the **connectedComponents** action on the simple undirected graph *G* shown in Figure 16.9.

Figure 16.9 A Simple Undirected Graph *G*



The undirected graph *G* can be represented by the following links data set, LinkSetIn:

```
data LinkSetIn;
   input from $ to $ @@;
   datalines;
A B  A C  B C  C H  D E  D F  D G  F E  G I  K L
;
```

The following DATA step loads the LinkSetIn data set into a CAS data table named mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements find the connected components and output the results in the data table NodeSetOut:

```
proc cas;
   loadactionset "network";
   action network.connectedComponents result=r status=s /
      indexOffset = 1
      links       = {name = "LinkSetIn"}
      outNodes    = {name = "NodeSetOut", replace=true};
   run;
   print r.ProblemSummary; run;
   print r.SolutionSummary; run;
   action table.fetch / table = "NodeSetOut" sortBy = "concomp"; run;
quit;
```

The problem summary output from this action is presented in Output 16.5.1.

Output 16.5.1 Problem Summary

Problem Summary	
Number of Nodes	11
Number of Links	10
Graph Direction	Undirected

The solution summary output from this action is presented in Output 16.5.2.

Output 16.5.2 Solution Summary

Solution Summary	
Problem Type	Connected Components
Solution Status	OK
Number of Components	3
CPU Time	0.00
Real Time	0.00

The output data table NodeSetOut contains the connected components of the input graph, as shown in Figure 16.5.3.

Output 16.5.3 Connected Components of a Simple Undirected Graph

Selected Rows from Table NODESETOUT		
Index	**node**	**concomp**
1	H	1
2	B	1
3	A	1
4	C	1
5	F	2
6	D	2
7	I	2
8	G	2
9	E	2
10	L	3
11	K	3

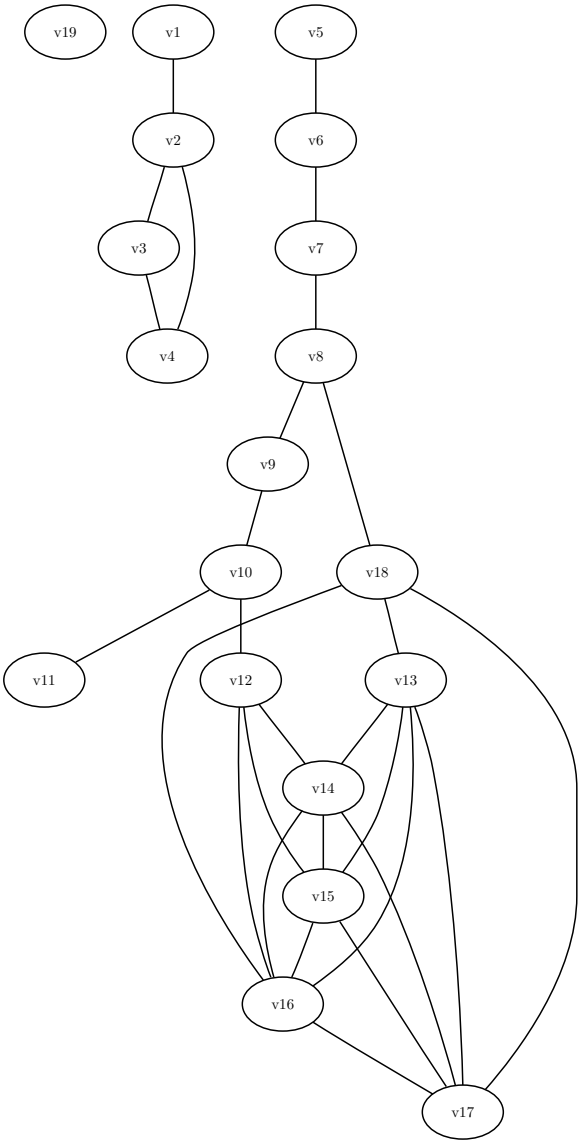
Example 16.6: Calculating the Core Decomposition of a Simple Undirected Graph

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the core decomposition algorithm on the simple undirected graph *G* shown in Figure 16.10.

Figure 16.10 A Simple Undirected Graph



The undirected graph G can be represented using the following nodes data set, NodeSetIn, and links data set, LinkSetIn:

```
data NodeSetIn;
   input node $ @@;
   datalines;
v1    v2   v3   v4   v5
v6    v7   v8   v9  v10
v11  v12  v13  v14  v15
v16  v17  v18  v19
;

data LinkSetIn;
   input from $ to $ @@;
   datalines;
```

```
v1   v2   v5  v6   v6  v7   v7  v8  v10 v11
v2   v3   v3  v4   v2  v4   v8  v9   v9 v10
v8  v18  v10 v12  v13 v14  v13 v15  v13 v16
v13 v17  v14 v15  v14 v16  v14 v17  v15 v16
v15 v17  v16 v17  v18 v13  v18 v17  v18 v16
v12 v14  v12 v15  v12 v16
;
```

The following DATA steps load the NodeSetIn and LinkSetIn data sets into CAS data tables named mycas.NodeSetIn and mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.NodeSetIn;
   set NodeSetIn;
run;
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements calculate the core decomposition and output the results in the data table NodeSetOut:

```
proc cas;
   loadactionset "network";
   action network.core result=r status=s /
      links    = {name = "LinkSetIn"}
      nodes    = {name = "NodeSetIn"}
      outNodes = {name = "NodeSetOut", replace=true};
   run;
   print r.ProblemSummary; run;
   print r.SolutionSummary; run;
   action table.fetch / table = "NodeSetOut" sortBy = "core_out"; run;
quit;
```

The problem summary output from this action is presented in Output 16.6.1.

Output 16.6.1 Problem Summary

Problem Summary	
Number of Nodes	19
Number of Links	28
Graph Direction	Undirected

The solution summary output from this action is presented in Output 16.6.2.

Output 16.6.2 Solution Summary

Solution Summary	
Problem Type	Core Decomposition
Solution Status	OK
CPU Time	0.00
Real Time	0.00

The nodes output data table NodeSetOut contains the core number (variable core_out) for each node, as

shown in Output 16.6.3.

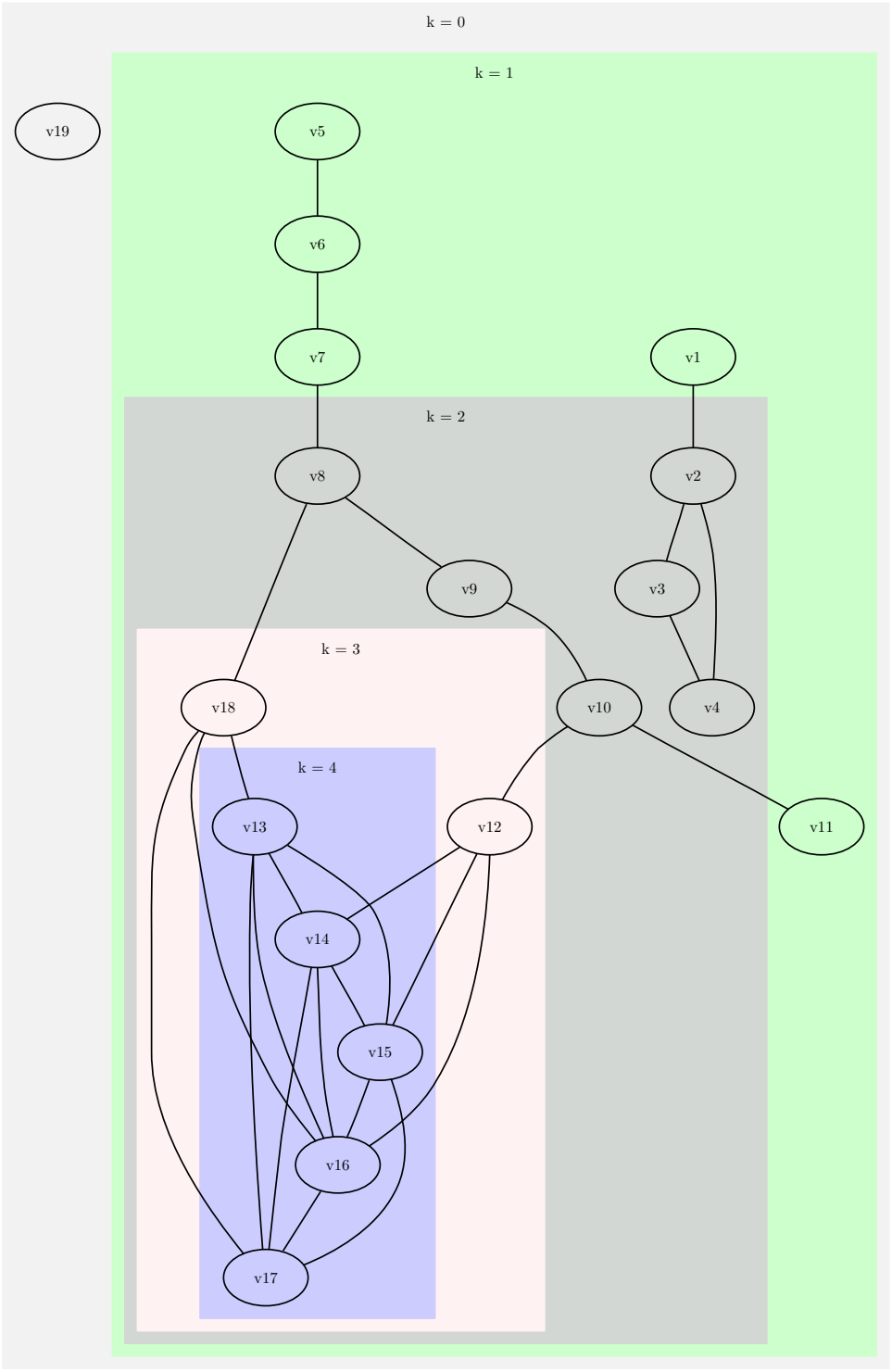
Output 16.6.3 Core Decomposition of a Simple Undirected Graph

Selected Rows from Table NODESETOUT

Index	node	core_out
1	v19	0
2	v5	1
3	v1	1
4	v11	1
5	v6	1
6	v7	1
7	v4	2
8	v3	2
9	v2	2
10	v9	2
11	v10	2
12	v8	2
13	v18	3
14	v12	3
15	v15	4
16	v14	4
17	v13	4
18	v17	4
19	v16	4

Figure 16.11 shows the graph layered by its core number.

Figure 16.11 Core Decomposition



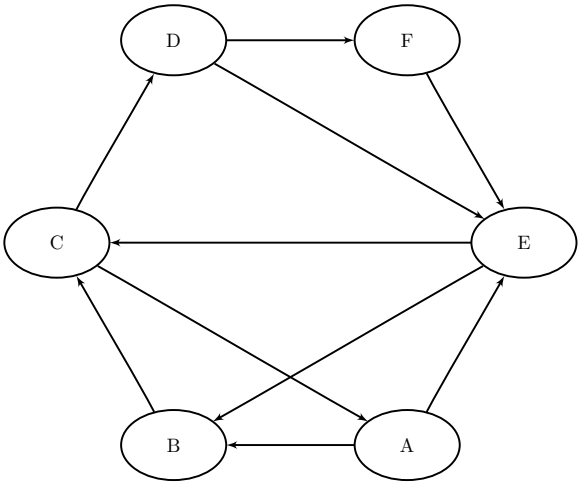
Example 16.7: Enumerating the Cycles of a Simple Directed Graph

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example provides a simple example of using the cycle detection algorithm on the simple directed graph G shown in Figure 16.12.

Figure 16.12 A Simple Directed Graph G



The directed graph G can be represented by the following links data set, LinkSetIn:

```
data LinkSetIn;
   input from $ to $ @@;
   datalines;
A B  A E  B C  C A  C D
D E  D F  E B  E C  F E
;
```

The following DATA step loads the LinkSetIn data set into a CAS data table named mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements find all the cycles in the graph and output the results in the data table Cycles:

```
proc cas;
   loadactionset "network";
   action network.cycle result=r status=s /
      indexOffset = 1
      direction   = "directed"
      links       = {name = "LinkSetIn"}
```

```
      out         = {name = "Cycles", replace=true}
      maxCycles   = "all";
   run;
   print r.ProblemSummary; run;
   print r.SolutionSummary; run;
   action table.fetch / table = "Cycles" to=1000 sortBy = {"cycle","order"}; run;
quit;
```

The problem summary output from this action is presented in Output 16.7.1.

Output 16.7.1 Problem Summary

Problem Summary	
Number of Nodes	6
Number of Links	10
Graph Direction	Directed

The solution summary output from this action is presented in Output 16.7.2.

Output 16.7.2 Solution Summary

Solution Summary	
Problem Type	Cycle
Solution Status	OK
Number of Cycles	7
CPU Time	0.00
Real Time	0.00

The output data table Cycles now contains all the cycles in the input graph, as shown in Output 16.7.3.

Output 16.7.3 All Cycles in a Simple Directed Graph

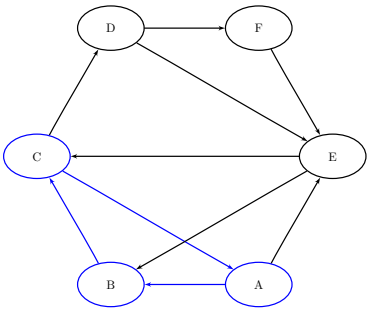
Selected Rows from Table CYCLES

Index	cycle	order	node
1	1	1	A
2	1	2	B
3	1	3	C
4	1	4	A
5	2	1	A
6	2	2	E
7	2	3	B
8	2	4	C
9	2	5	A
10	3	1	A
11	3	2	E
12	3	3	C
13	3	4	A
14	4	1	B
15	4	2	C
16	4	3	D
17	4	4	E
18	4	5	B
19	5	1	B
20	5	2	C
21	5	3	D
22	5	4	F
23	5	5	E
24	5	6	B
25	6	1	E
26	6	2	C
27	6	3	D
28	6	4	E
29	7	1	E
30	7	2	C
31	7	3	D
32	7	4	F
33	7	5	E

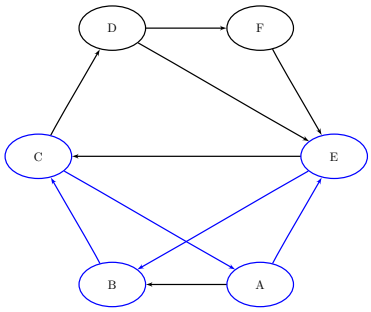
The cycles are shown graphically in Figure 16.13.

Figure 16.13 Cycles

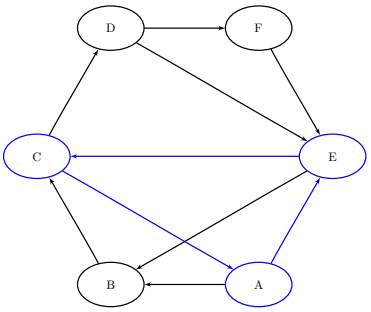
$A \rightarrow B \rightarrow C \rightarrow A$



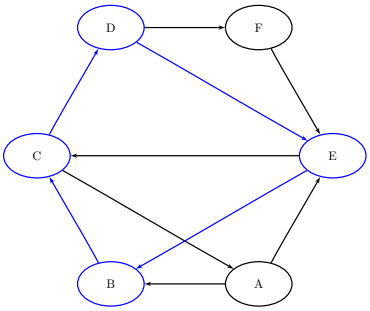
$A \rightarrow E \rightarrow B \rightarrow C \rightarrow A$



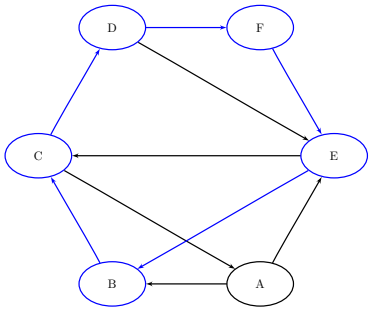
$A \rightarrow E \rightarrow C \rightarrow A$



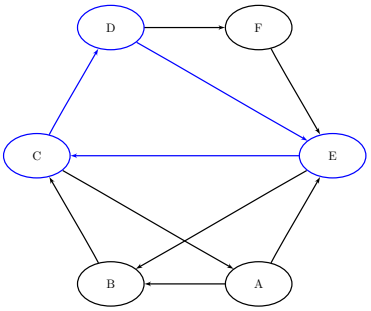
$B \rightarrow C \rightarrow D \rightarrow E \rightarrow B$



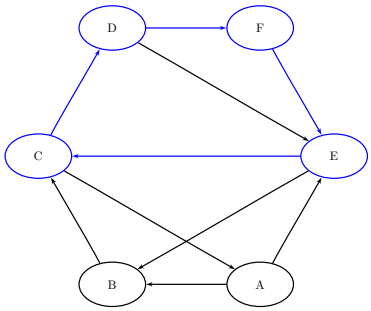
$B \rightarrow C \rightarrow D \rightarrow F \rightarrow E \rightarrow B$



$E \rightarrow C \rightarrow D \rightarrow E$



$E \rightarrow C \rightarrow D \rightarrow F \rightarrow E$



Example 16.8: Node Similarity in an Undirected Graph

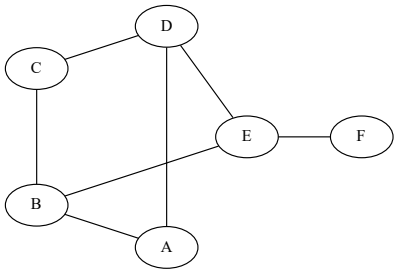
This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud*

Analytic Services: CAS Procedure Programming Guide and Reference, and *SAS Viya: System Programming Guide*.

This section provides a simple example of calculating node similarity in the undirected graph *G* that is shown in Figure 16.14.

Figure 16.14 A Simple Undirected Graph *G*



The undirected graph *G* can be represented by the links data set, LinkSetIn, that is created by the following DATA step:

```
data LinkSetIn;
   input from $ to $ weight;
   datalines;
A B 1
A D 1
C B 1
C D 1
E B 1
E D 1
E F 1
;
```

The following DATA step loads the LinkSetIn data set into a CAS data table named mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements find all node pairs that have a similarity score greater than or equal to 0.7:

```
proc cas;
   loadactionset "network";
   action nodeSimilarity result=r status=s /
      links          = {name = "LinkSetIn"}
      vector         = true
      nDimensions    = 10
      proximityOrder = "second"
      minScore       = 0.7
      outSimilarity  = {name = "NodeSim", replace=true};
```

```
   run;
   print r.ProblemSummary; run;
   print r.SolutionSummary; run;
   action table.fetch / table = "NodeSim" sortBy = {"source","sink"}; run;
quit;
```

The problem summary output from this action is presented in Output 16.8.1.

Output 16.8.1 Problem Summary

Problem Summary	
Number of Nodes	6
Number of Links	7
Graph Direction	Undirected

The solution summary output from this action is presented in Output 16.8.2.

Output 16.8.2 Solution Summary

Solution Summary	
Problem Type	Node Similarity
Solution Status	OK
CPU Time	3.26
Real Time	1.05

The output data table NodeSim, as shown in Output 16.8.3, contains the pairs of nodes that have either a Jaccard or vector similarity score at or above the specified **minScore** parameter value of 0.7.

Output 16.8.3 Node Pairs with Similarity Scores Greater Than 0.7

Selected Rows from Table NODESIM				
Index	source	sink	jaccard	vector
1	A	A	1	1
2	A	C	1	0.9993204255
3	A	E	0.6666666667	0.8037895053
4	B	B	1	1
5	B	D	1	0.9940500316
6	B	F	0.3333333333	0.8331655259
7	C	A	1	0.9993204255
8	C	C	1	1
9	C	E	0.6666666667	0.8097860746
10	D	B	1	0.9940500316
11	D	D	1	1
12	D	F	0.3333333333	0.7726934599
13	E	A	0.6666666667	0.8037895053
14	E	C	0.6666666667	0.8097860746
15	E	E	1	1
16	F	B	0.3333333333	0.8331655259
17	F	D	0.3333333333	0.7726934599
18	F	F	1	1

This example illustrates the intuition behind node similarity values. The nodes A and C have the same set of nearest neighbors (see Figure 16.14), B and D, resulting in the Jaccard similarity score of 1 and a vector similarity score very close to 1; thus the (A,C) pair is present in the output. The pair (B,F) appears in the output because the vector similarity score is greater than 0.7, whereas the Jaccard similarity score is lower than the **minScore** parameter threhold value. In this case, although B and F share only one neighbor out of four total, they do have common neighbors of the nearest neighbors; this results in a greater vector similarity value but not in a greater value of Jaccard similarity, which captures only the similarity of nearest-neighbor structures.

The first-order proximity can be calculated by the following statements:

```
proc cas;
   loadactionset "network";
   action nodeSimilarity result=r status=s /
      links          = {name = "LinkSetIn"}
      jaccard        = false
      vector         = true
      nDimensions    = 10
      proximityOrder = "first"
      minScore       = 0.7
      outSimilarity  = {name = "NodeSim", replace=true}
      outNodes       = {name = "NodeVecs", replace=true};
   run;
   action table.fetch / table = "NodeSim" sortBy = {"source","sink"}; run;
   action table.fetch / table = "NodeVecs" sortBy = "node"; run;
quit;
```

The output data table NodeSim, as shown in Output 16.8.4, contains the pairs of nodes and similarity scores for the first-order proximity. The example also demonstrates the intuition behind similarity based on first-order proximity and how it is different from the Jaccard-based proximity and the vector for the second-order proximity. Intuitively, the first-order vector similarity value between the nodes that are directly connected is higher than the first-order vector similarity value between the nodes that are separated by a longer path in a graph, whereas the Jaccard similarity and second-order vector similarity values are higher between the nodes that share more neighbors. The first-order vector similarity is more closely related to results that are produced by label propagation community detection algorithms, in the sense that the pairs of nodes that have high first-order vector similarity scores between them can be placed in one community.

Output 16.8.4 Node Pairs with Similarity Scores Greater Than 0.7

Selected Rows from Table NODESIM			
Index	source	sink	vector
1	A	A	1
2	A	B	0.757278421
3	A	D	0.9800626127
4	B	A	0.757278421
5	B	B	1
6	B	C	0.9307237114
7	B	D	0.7741068941
8	C	B	0.9307237114
9	C	C	1
10	C	D	0.7627349306
11	D	A	0.9800626127
12	D	B	0.7741068941
13	D	C	0.7627349306
14	D	D	1
15	E	E	1
16	E	F	0.9999504081
17	F	E	0.9999504081
18	F	F	1

The output data table NodeVecs contains node-embedding vectors, as shown in Output 16.8.5.

Output 16.8.5 Node-Embedding Vectors

Selected Rows from Table NODEVECS								
Index	node	vec_0	vec_1	vec_2	vec_3	vec_4	vec_5	vec_6
1	A	-0.070340221	-0.400010105	-0.214410412	-0.32031008	-0.16938333	0.151205169	-0.09202592
2	B	0.1625498556	-0.373941288	-0.372776841	0.055958009	0.3697669008	0.2621612106	0.2837873096
3	C	0.3693920902	-0.298292118	-0.321636119	0.0586087847	0.1585155527	0.4008413282	0.2659550706
4	D	0.0621557611	-0.395800609	-0.226534926	-0.308582922	-0.24036634	0.2573960761	-0.067743237
5	E	-0.134047062	0.4351021199	0.321094088	0.1830317219	0.0098514615	-0.304418697	-0.083028316
6	F	-0.13322379	0.4347304518	0.3232978095	0.1784336972	0.0000792602	-0.304805824	-0.087913821

Selected Rows from Table NODEVECS		
vec_7	vec_8	vec_9
0.0249410893	-0.335962398	-0.716211227
0.3557387832	-0.443461713	-0.287144414
0.5952034098	-0.132761869	-0.198143004
0.1965312881	-0.213483577	-0.692906487
-0.316510289	0.3228688599	0.5933219585
-0.316280448	0.3291680825	0.5898138807

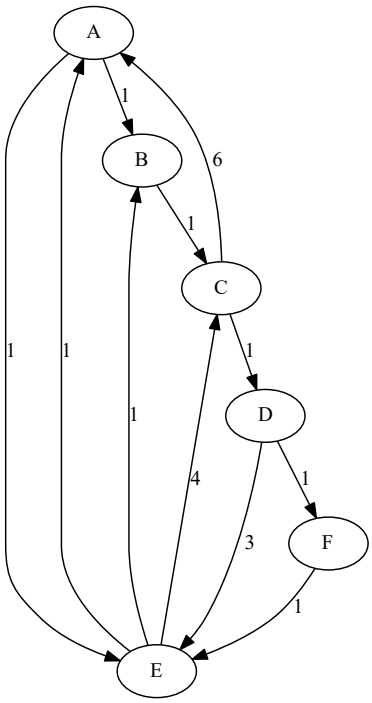
Example 16.9: Enumerating the Paths of a Simple Directed Graph

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This section provides a simple example of using the path enumeration algorithm on the simple directed graph *G* shown in Figure 16.9.1 to find all paths between one source-sink pair by using the SOURCE= and SINK= options.

Output 16.9.1 A Simple Directed Graph *G*



The directed graph *G* can be represented by the following links data set, LinkSetIn:

```
data LinkSetIn;
   input from $ to $ weight @@;
   datalines;
A B 1 A E 1 B C 1 C A 6 C D 1
D E 3 D F 1 E B 1 E C 4 F E 1
E A 1
;
```

The following DATA step loads the LinkSetIn data set into a CAS data table named mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements find all paths between node *D* and node *A* whose path link weight is less than or equal to 10. They output the results in the data tables PathLinks and PathNodes.

```
proc cas;
   loadactionset "network";
   action network.path result=r status=s /
      indexOffset   = 1
      direction     = "directed"
      links         = {name = "LinkSetIn"}
      source        = "D"
      sink          = "A"
      maxLinkWeight = 10
      outPathsLinks = {name = "PathLinks", replace=true}
      outPathsNodes = {name = "PathNodes", replace=true};
   run;
   print r.ProblemSummary; run;
   print r.SolutionSummary; run;
   action table.fetch / table = "PathLinks" to=1000
                        sortBy = {"source","sink","path","order"}; run;
   action table.fetch / table = "PathNodes" to=1000
                        sortBy = {"source","sink","path","order"}; run;
quit;
```

The problem summary output from this action is presented in Output 16.9.2.

Output 16.9.2 Problem Summary

Problem Summary	
Number of Nodes	6
Number of Links	11
Graph Direction	Directed

The solution summary output from this action is presented in Output 16.9.3.

Output 16.9.3 Solution Summary

Solution Summary	
Problem Type	Path
Solution Status	OK
Number of Paths	3
CPU Time	0.00
Real Time	0.00

The output data table PathLinks contains the links of the three paths from *D* to *A* whose path link weight is less than or equal to 10, as shown in Output 16.9.4.

Output 16.9.4 Links for All (Short) Paths in a Simple Directed Graph

Selected Rows from Table PATHLINKS							
Index	source	sink	path	order	from	to	weight
1	D	A	1	1	D	E	3
2	D	A	1	2	E	A	1
3	D	A	2	1	D	F	1
4	D	A	2	2	F	E	1
5	D	A	2	3	E	A	1
6	D	A	3	1	D	F	1
7	D	A	3	2	F	E	1
8	D	A	3	3	E	B	1
9	D	A	3	4	B	C	1
10	D	A	3	5	C	A	6

The output data table PathNodes contains the nodes of the three paths, as shown in Output 16.9.5.

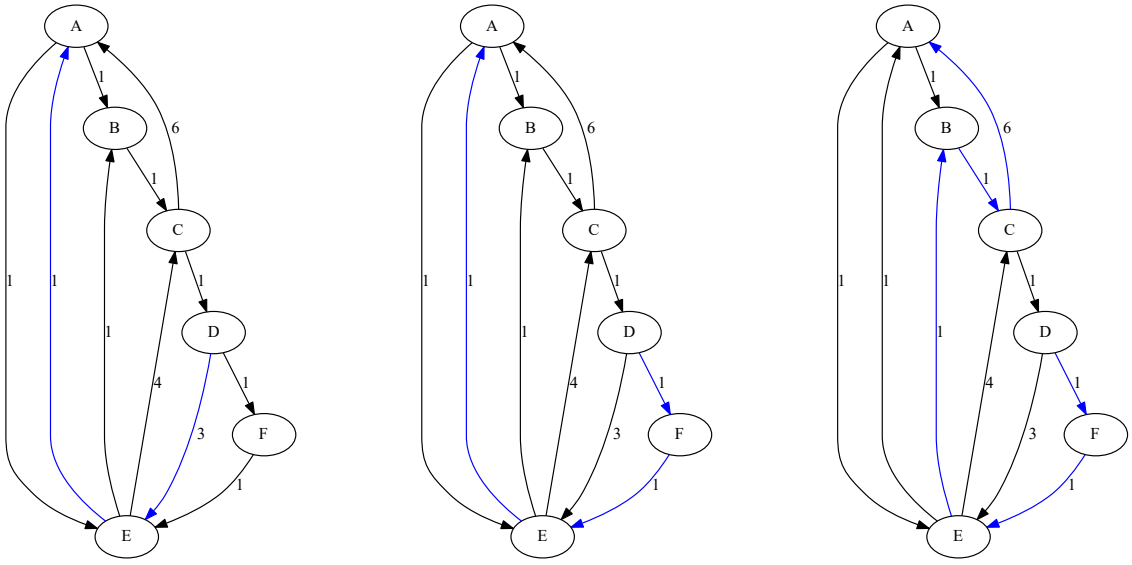
Output 16.9.5 Nodes for All (Short) Paths in a Simple Directed Graph

Selected Rows from Table PATHNODES					
Index	source	sink	path	order	node
1	D	A	1	1	D
2	D	A	1	2	E
3	D	A	1	3	A
4	D	A	2	1	D
5	D	A	2	2	F
6	D	A	2	3	E
7	D	A	2	4	A
8	D	A	3	1	D
9	D	A	3	2	F
10	D	A	3	3	E
11	D	A	3	4	B
12	D	A	3	5	C
13	D	A	3	6	A

The three (short) paths are shown graphically in Figure 16.15.

Figure 16.15 Paths

$D \rightarrow E \rightarrow A$ $D \rightarrow F \rightarrow E \rightarrow A$ $D \rightarrow F \rightarrow E \rightarrow B \rightarrow C \rightarrow A$



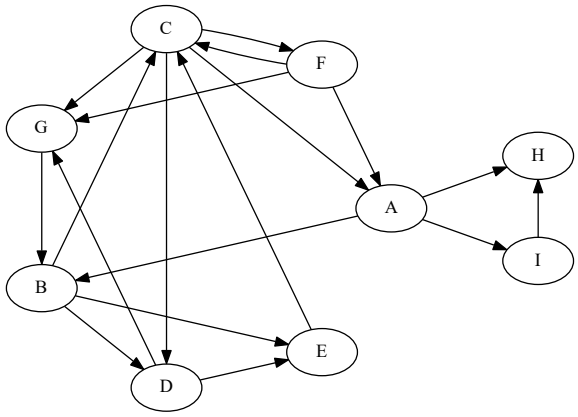
Example 16.10: Pattern Matching of a Simple Directed Graph

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the pattern matching algorithm on the simple directed graph *G* shown in Figure 16.16.

Figure 16.16 Simple Directed Graph *G*



The directed graph *G* can be represented using the links data set, Links, and nodes data set, Nodes, that are created by the following DATA steps:

```
data Links;
   input from $ to $ @@;
   datalines;
A B A I A H
B D B E B C
C F C G C A
F G F A G B
E C I H D G
F C C D D E
;
data Nodes;
   input node $ @@;
   label=node;
   datalines;
A B C D E F G H I
;
```

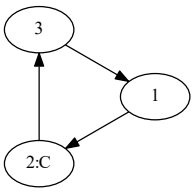
The following DATA steps load the Links and Nodes data sets into CAS data tables named mycas.Links and mycas.Nodes. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.Links;
   set Links;
run;
data mycas.Nodes;
   set Nodes;
run;
```

Each node has a node attribute label that is defined as the name of the node.

In this example, you want to find all subgraphs that are directed cycles of length 3 that pass through node C. The query graph *Q* that defines the pattern to search for is shown in Figure 16.17.

Figure 16.17 Query Graph Q



The query graph Q can be represented using the links data set, LinksQuery, and nodes data set, NodesQuery, that are created by the following DATA steps:

```
data LinksQuery;
   input from to;
   datalines;
1 2
2 3
3 1
;
data NodesQuery;
   input node label $;
   datalines;
2 C
;
```

The following DATA steps load the LinksQuery and NodesQuery data sets into CAS data tables named mycas.LinksQuery and mycas.NodesQuery. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinksQuery;
   set LinksQuery;
run;
data mycas.NodesQuery;
   set NodesQuery;
run;
```

You can use the following statements to find all subgraphs that have the specified pattern:

```
proc cas;
   loadactionset "network";
   action patternMatch result=r status=s /
      direction      = "directed"
      links          = {name = "Links"}
      nodes          = {name = "Nodes"}
      linksQuery     = {name = "LinksQuery"}
      nodesQuery     = {name = "NodesQuery"}
      nodesVar       = {vars = "label"}
      nodesQueryVar  = {vars = "label"}
      outMatchNodes  = {name = "OutMatchNodes", replace=true}
      outMatchLinks  = {name = "OutMatchLinks", replace=true};
   run;
```

```
   print r.ProblemSummary; run;
   print r.SolutionSummary; run;
   action table.fetch / table = "OutMatchNodes" sortBy = {"match","nodeQ","node"}; run;
   action table.fetch / table = "OutMatchLinks" sortBy = {"match","from","to"}; run;
quit;
```

The problem summary output from this action is presented in Output 16.10.1.

Output 16.10.1 Problem Summary

Problem Summary	
Number of Nodes	9
Number of Links	18
Graph Direction	Directed

The solution summary output from this action is presented in Output 16.10.2.

Output 16.10.2 Solution Summary

Solution Summary	
Problem Type	Pattern Match
Solution Status	OK
Number of Matches	3
CPU Time	0.01
Real Time	0.01

The output data table OutMatchNodes now contains the mapping from nodes in the query graph to nodes in the input graph for each pattern match, as shown in Output 16.10.3.

Output 16.10.3 Node Mappings for Pattern Matches

Selected Rows from Table OUTMATCHNODES				
Index	**match**	**nodeQ**	**node**	**label**
1	0	1	B	B
2	0	2	C	C
3	0	3	G	G
4	1	1	B	B
5	1	2	C	C
6	1	3	A	A
7	2	1	E	E
8	2	2	C	C
9	2	3	D	D

The output data table OutMatchLinks now contains the subgraphs for each pattern match, as shown in Output 16.10.4.

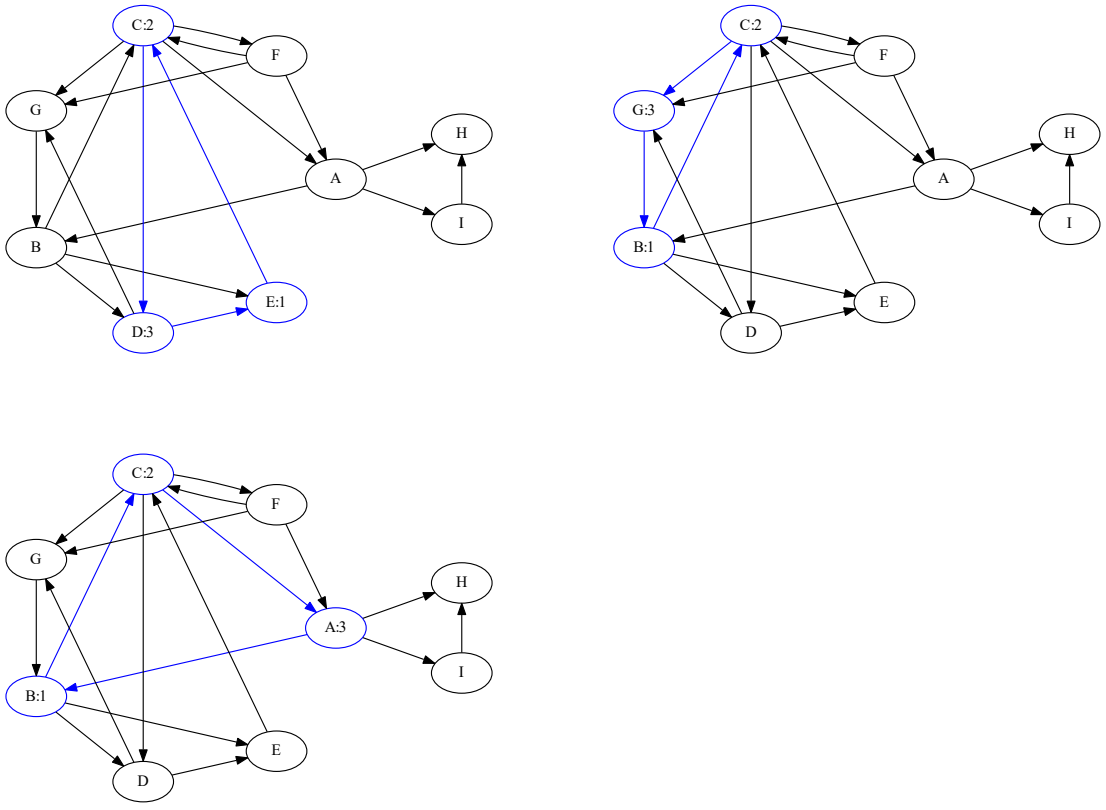
Output 16.10.4 Subgraphs for Pattern Matches

Selected Rows from Table OUTMATCHLINKS

Index	match	from	to
1	0	B	C
2	0	C	G
3	0	G	B
4	1	A	B
5	1	B	C
6	1	C	A
7	2	C	D
8	2	D	E
9	2	E	C

The results are displayed graphically in Figure 16.18.

Figure 16.18 Subgraphs



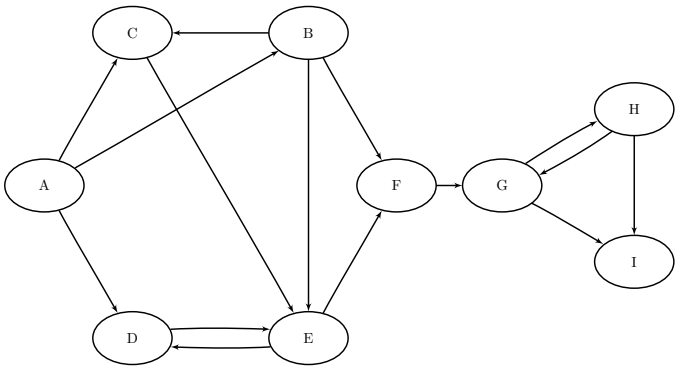
Example 16.11: Finding the Reach Networks of a Simple Directed Graph

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This section illustrates the use of the **reach** action on the simple directed graph G shown in Figure 16.19.

Figure 16.19 A Simple Directed Graph G



The directed graph G can be represented using the following links data set, LinkSetIn:

```
data LinkSetIn;
   input from $ to $ @@;
   datalines;
A B  A C  A D  B C  B E
B F  C E  D E  E D  E F
F G  G H  G I  H G  H I
;
```

Consider two sets of source nodes, $S_1 = \{C\}$ and $S_2 = \{A, H\}$. These source node sets can be defined together as follows:

```
data NodeSubSetIn;
   input node $ reach;
   datalines;
A 2
C 1
H 2
;
```

The following DATA steps load the LinkSetIn and NodeSubSetIn data sets into CAS data tables named mycas.LinkSetIn and mycas.NodeSubSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinkSetIn;
   set LinkSetIn;
run;
data mycas.NodeSubSetIn;
```

```
   set NodeSubSetIn;
run;
```

You can use the following statements to find the reach network that is restricted by a hop limit of 1 from both sets of source nodes:

```
proc cas;
   loadactionset "network";
   action reach result=r status=s /
      direction     = "directed"
      links         = {name = "LinkSetIn"}
      nodessubset   = {name = "NodeSubSetIn"}
      outReachNodes = {name = "ReachNodes", replace=true}
      outReachlinks = {name = "ReachLinks", replace=true}
      outCounts     = {name = "ReachCounts", replace=true}
      maxreach      = 1;
   run;
   print r.ProblemSummary; run;
   print r.SolutionSummary; run;
   action table.fetch / table = "ReachNodes" sortBy = "reach"; run;
   action table.fetch / table = "ReachLinks" sortBy = "reach"; run;
   action table.fetch / table = "ReachCounts" sortBy = "reach"; run;
quit;
```

The problem summary output from this action is presented in Output 16.11.1.

Output 16.11.1 Problem Summary

Problem Summary	
Number of Nodes	9
Number of Links	15
Graph Direction	Directed

The solution summary output from this action is presented in Output 16.11.2.

Output 16.11.2 Solution Summary

Solution Summary	
Problem Type	Reach
Solution Status	OK
CPU Time	0.00
Real Time	0.00

The output data tables ReachNodes, ReachLinks, and ReachCounts now contain the nodes, links, and counts of the reach networks, respectively, that come from S_1 and S_2. They are shown in Output 16.11.3 through Output 16.11.5.

Output 16.11.3 Reach Networks (Nodes) for $S_1 = \{C\}$ and $S_2 = \{A, H\}$ with Hop Limit of 1

Selected Rows from Table REACHNODES		
Index	reach	node
1	1	E
2	1	C
3	2	B
4	2	C
5	2	D
6	2	G
7	2	H
8	2	I
9	2	A

Output 16.11.4 Reach Networks (Links) for $S_1 = \{C\}$ and $S_2 = \{A, H\}$ with Hop Limit of 1

Selected Rows from Table REACHLINKS			
Index	reach	from	to
1	1	C	E
2	2	A	C
3	2	A	D
4	2	B	C
5	2	G	H
6	2	H	G
7	2	G	I
8	2	H	I
9	2	A	B

Output 16.11.5 Reach Networks (Counts) for $S_1 = \{C\}$ and $S_2 = \{A, H\}$ with Hop Limit of 1

Selected Rows from Table REACHCOUNTS				
Index	reach	node	count	count_not
1	1	C	2	7
2	2	H	7	2
3	2	A	7	2

Example 16.12: Reading a Simple Directed Graph

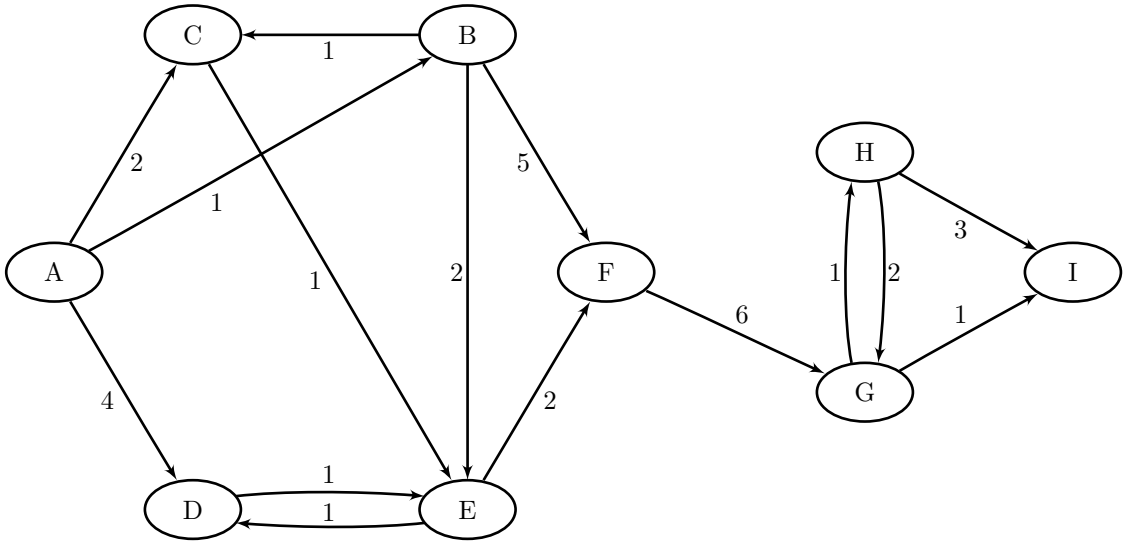
This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud*

Analytic Services: CAS Procedure Programming Guide and Reference, and *SAS Viya: System Programming Guide*.

This example describes how to input a graph by using the **readGraph** action. Let $G = (N, A)$ define a graph that contains a set N of nodes and a set A of links. Consider the directed graph shown in Figure 16.20.

Figure 16.20 A Simple Directed Graph



The directed graph G can be represented by the following links data set, LinkSetIn:

```
data LinkSetIn;
   input from $ to $ weight @@;
   datalines;
A B 1  A C 2  A D 4  B C 1  B E 2
B F 5  C E 1  D E 1  E D 1  E F 2
F G 6  G H 1  G I 1  H G 2  H I 3
;
```

The following DATA step loads the LinkSetIn data set into a CAS data table named mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements read in this graph, declare it as a directed graph, and output the resulting links and nodes data tables. These statements do not run any algorithms, so the resulting output contains only the input graph.

```
proc cas;
   loadactionset "network";
   action network.readGraph result=r status=s /
      direction = "directed"
      links     = {name = "LinkSetIn"}
      outNodes  = {name = "NodeSetOut", replace=true}
```

```
      outLinks  = {name = "LinkSetOut", replace=true};
   run;
   print r.ProblemSummary; run;
   action table.fetch / table = "NodeSetOut"; run;
   action table.fetch / table = "LinkSetOut"; run;
quit;
```

The problem summary output from this action is presented in Output 16.12.1.

Output 16.12.1 Problem Summary

Problem Summary	
Number of Nodes	9
Number of Links	15
Graph Direction	Directed

The output data table NodeSetOut, shown in Figure 16.12.2, now contains the nodes that are read from the input links data table. The variable node shows the label that is associated with each node.

Output 16.12.2 Nodes Data Table of a Simple Directed Graph

Selected Rows from Table NODESETOUT	
Index	**node**
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I

The output data table LinkSetOut, shown in Figure 16.12.3, contains the links that were read from the input links data table. The variables from and to show the associated node labels.

Output 16.12.3 Links Data Table of a Simple Directed Graph

Selected Rows from Table LINKSETOUT			
Index	from	to	weight
1	A	B	1
2	A	C	2
3	A	D	4
4	B	C	1
5	B	E	2
6	B	F	5
7	C	E	1
8	D	E	1
9	E	D	1
10	E	F	2
11	F	G	6
12	G	H	1
13	H	G	2
14	G	I	1
15	H	I	3

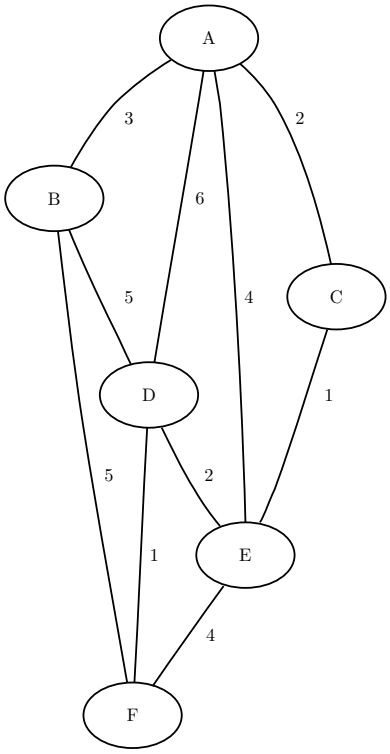
Example 16.13: Finding the Shortest Paths for All Source-Sink Pairs

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the shortest path algorithm for all source-sink pairs on the simple undirected graph *G* shown in Figure 16.21.

Figure 16.21 A Simple Undirected Graph G



The undirected graph G can be represented by the following links data set, LinkSetIn:

```
data LinkSetIn;
   input from $ to $ weight @@;
   datalines;
A B 3  A C 2  A D 6  A E 4  B D 5
B F 5  C E 1  D E 2  D F 1  E F 4
;
```

The following DATA step loads the LinkSetIn data set into a CAS data table named mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements find the shortest paths for all source-sink pairs:

```
proc cas;
   loadactionset "network";
   action network.shortestPath result=r status=s /
      indexOffset = 1
      links       = {name = "LinkSetIn"}
      outWeights  = {name = "ShortPathW", replace=true}
      outPaths    = {name = "ShortPathP", replace=true};
   run;
   print r.ProblemSummary; run;
```

```
   print r.SolutionSummary; run;
   action table.fetch / table  = "ShortPathP" to=1000
                        sortBy = {"source","sink","order"}; run;
   action table.fetch / table = "ShortPathW" to=1000; run;
quit;
```

The problem summary output from this action is presented in Output 16.13.1.

Output 16.13.1 Problem Summary

Problem Summary	
Number of Nodes	6
Number of Links	10
Graph Direction	Undirected

The solution summary output from this action is presented in Output 16.13.2.

Output 16.13.2 Solution Summary

Solution Summary	
Problem Type	Shortest Path
Solution Status	OK
Number of Paths	30
CPU Time	0.01
Real Time	0.01

The output data table ShortPathP contains the shortest paths, as shown in Output 16.13.3.

Output 16.13.3 All-Pairs Shortest Paths

Selected Rows from Table SHORTPATHP

Index	source	sink	order	from	to	weight
1	A	B	1	A	B	3
2	A	C	1	A	C	2
3	A	D	1	A	C	2
4	A	D	2	C	E	1
5	A	D	3	E	D	2
6	A	E	1	A	C	2
7	A	E	2	C	E	1
8	A	F	1	A	C	2
9	A	F	2	C	E	1
10	A	F	3	E	D	2
11	A	F	4	D	F	1
12	B	A	1	B	A	3
13	B	C	1	B	A	3
14	B	C	2	A	C	2
15	B	D	1	B	D	5
16	B	E	1	B	A	3
17	B	E	2	A	C	2
18	B	E	3	C	E	1
19	B	F	1	B	F	5
20	C	A	1	C	A	2
21	C	B	1	C	A	2
22	C	B	2	A	B	3
23	C	D	1	C	E	1
24	C	D	2	E	D	2
25	C	E	1	C	E	1
26	C	F	1	C	E	1
27	C	F	2	E	D	2
28	C	F	3	D	F	1
29	D	A	1	D	E	2
30	D	A	2	E	C	1
31	D	A	3	C	A	2
32	D	B	1	D	B	5
33	D	C	1	D	E	2
34	D	C	2	E	C	1
35	D	E	1	D	E	2
36	D	F	1	D	F	1
37	E	A	1	E	C	1
38	E	A	2	C	A	2
39	E	B	1	E	C	1
40	E	B	2	C	A	2
41	E	B	3	A	B	3
42	E	C	1	E	C	1
43	E	D	1	E	D	2
44	E	F	1	E	D	2
45	E	F	2	D	F	1
46	F	A	1	F	D	1
47	F	A	2	D	E	2

Output 16.13.3 *continued*

Selected Rows from Table SHORTPATHP

Index	source	sink	order	from	to	weight
48	F	A	3	E	C	1
49	F	A	4	C	A	2
50	F	B	1	F	B	5
51	F	C	1	F	D	1
52	F	C	2	D	E	2
53	F	C	3	E	C	1
54	F	D	1	F	D	1
55	F	E	1	F	D	1
56	F	E	2	D	E	2

The output data table ShortPathW contains the path weights of the shortest paths of each source-sink pair, as shown in Output 16.13.4.

Output 16.13.4 All-Pairs Shortest Paths Summary

Selected Rows from Table SHORTPATHW			
Index	source	sink	path_weight
1	C	A	2
2	C	B	5
3	C	D	3
4	C	E	1
5	C	F	4
6	D	A	5
7	D	B	5
8	D	C	3
9	D	E	2
10	D	F	1
11	F	A	6
12	F	B	5
13	F	C	4
14	F	D	1
15	F	E	3
16	A	B	3
17	A	C	2
18	A	D	5
19	A	E	3
20	A	F	6
21	E	A	3
22	E	B	6
23	E	C	1
24	E	D	2
25	E	F	3
26	B	A	3
27	B	C	5
28	B	D	5
29	B	E	6
30	B	F	5

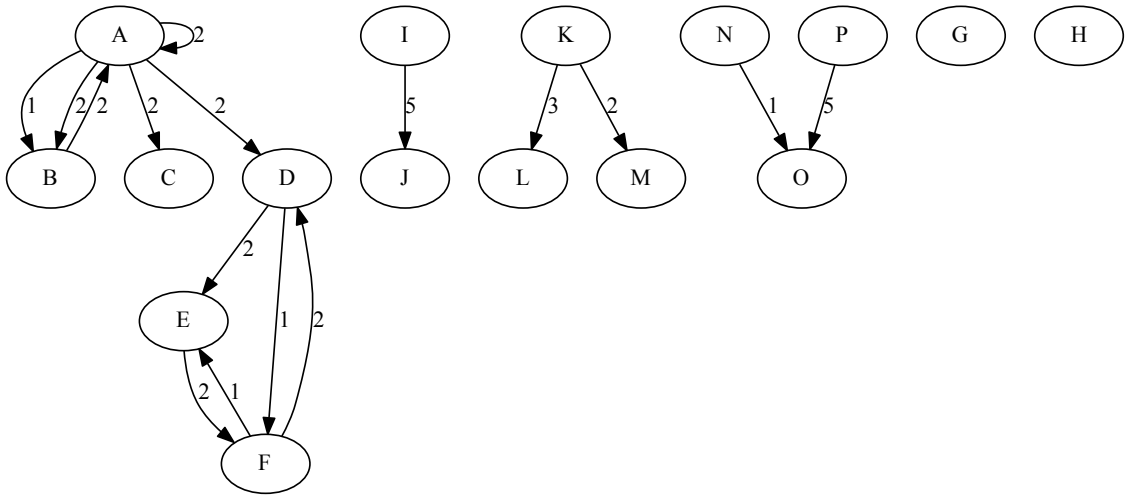
Example 16.14: Calculating the Summary Statistics of a Simple Directed Graph

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This section illustrates the calculation of summary statistics on the simple directed graph *G* shown in Figure 16.22.

Figure 16.22 A Simple Directed Graph *G*



You can represent the directed graph *G* by using the following nodes data set, NodeSetIn, and links data set, LinkSetIn:

```
data NodeSetIn;
  input node $ @@;
  datalines;
A B C D E F G H I J K L M N O P
;
data LinkSetIn;
   input from $ to $ weight @@;
   datalines;
A B 1 A C 2 A D 2 B A 2 D E 2
D F 1 E F 2 F D 2 F E 1 A A 2
A B 2 I J 5 K L 3 K M 2 N O 1
P O 5
;
```

The following DATA steps load the NodeSetIn and LinkSetIn data sets into CAS data tables named mycas.NodeSetIn and mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.NodeSetIn;
   set NodeSetIn;
run;
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements calculate the default summary statistics and produce information about the connectedness of the graph. They output the results in the data tables NodeSetOut and Summary.

```
proc cas;
   loadactionset "network";
   action network.summary result=r status=s /
      direction   = "directed"
      indexOffset = 1
      nodes       = {name = "NodeSetIn"}
      links       = {name = "LinkSetIn"}
      outNodes    = {name = "NodeSetOut", replace=true}
      out         = {name = "Summary", replace=true}
      connectedComponents = true;
  run;
  print r.ProblemSummary; run;
  print r.SolutionSummary; run;
  action table.fetch / table = "NodeSetOut"; run;
  action table.fetch / table = "Summary"; run;
quit;
```

The problem summary output from this action is presented in Output 16.14.1.

Output 16.14.1 Problem Summary

Problem Summary	
Number of Nodes	16
Number of Links	14
Graph Direction	Directed

The solution summary output from this action is presented in Output 16.14.2.

Output 16.14.2 Solution Summary

Solution Summary	
Problem Type	Summary
Solution Status	OK
CPU Time	0.00
Real Time	0.00

The output data table NodeSetOut contains the node-level summary metrics of the input graph, as shown in Output 16.14.3.

Output 16.14.3 Node-Level Graph Summary and Connectedness Statistics of a Simple Directed Graph

Selected Rows from Table NODESETOUT							
Index	node	leaf_node	singleton_node	neighbor_leaf_nodes	isolated_star_out	isolated_star_in	isolated_pair
1	A	0	0	1	.	.	.
2	B	0	0	0	.	.	.
3	C	1	0	0	.	.	.
4	D	0	0	0	.	.	.
5	E	0	0	0	.	.	.
6	F	0	0	0	.	.	.
7	G	0	1	0	.	.	.
8	H	0	1	0	.	.	.
9	I	0	0	1	.	.	1
10	J	1	0	0	.	.	1
11	K	0	0	2	1	.	.
12	L	1	0	0	1	.	.
13	M	1	0	0	1	.	.
14	N	0	0	1	.	1	.
15	O	1	0	0	.	1	.
16	P	0	0	1	.	1	.

Selected Rows from Table NODESETOUT
sum_in_and_out_wt
7
3
2
7
5
6
0
0
5
5
5
3
2
1
6
5

The output data table Summary contains the summary statistics of the input graph, as shown in Figure 16.14.4.

Output 16.14.4 Graph Summary and Connectedness Statistics of a Simple Directed Graph

Selected Rows from Table SUMMARY							
Index	nodes	links	avg_links_per_node	density	self_links_ignored	dup_links_ignored	leaf_nodes
1	16	14	0.875	0.0583333333	1	1	5

Selected Rows from Table SUMMARY				
singleton_nodes	concomp	isolated_pairs	isolated_stars_out	isolated_stars_in
2	13	1	1	1

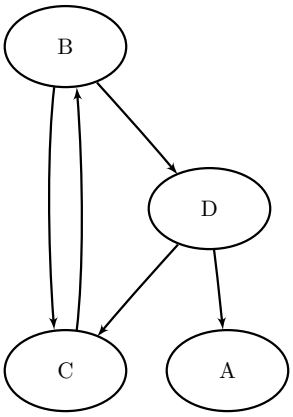
Example 16.15: Calculating the Transitive Closure of a Simple Directed Graph

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example illustrates the use of the transitive closure algorithm on the simple directed graph *G* shown in Figure 16.23.

Figure 16.23 A Simple Directed Graph *G*



The directed graph *G* can be represented by the following links data set, LinkSetIn:

```
data LinkSetIn;
   input from $ to $ @@;
   datalines;
B C  B D  C B  D A  D C
;
```

The following DATA step loads the LinkSetIn data set into a CAS data table named mycas.LinkSetIn. These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.LinkSetIn;
   set LinkSetIn;
run;
```

The following statements calculate the transitive closure and output the results in the data table TransClosure:

```
proc cas;
   loadactionset "network";
   action network.transitiveClosure result=r status=s /
      direction = "directed"
      links     = {name = "LinkSetIn"}
      out       = {name = "TransClosure", replace=true};
   run;
   print r.ProblemSummary; run;
   print r.SolutionSummary; run;
   action table.fetch / table = "TransClosure"; run;
quit;
```

The problem summary output from this action is presented in Output 16.15.1.

Output 16.15.1 Problem Summary

Problem Summary	
Number of Nodes	4
Number of Links	5
Graph Direction	Directed

The solution summary output from this action is presented in Output 16.15.2.

Output 16.15.2 Solution Summary

Solution Summary	
Problem Type	Transitive Closure
Solution Status	OK
CPU Time	0.00
Real Time	0.00

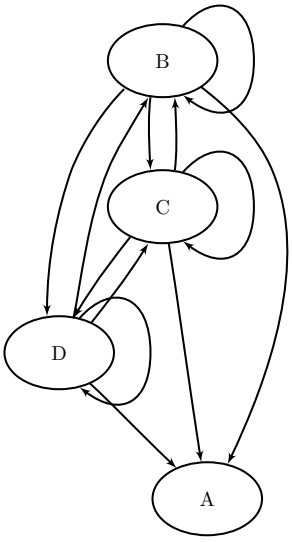
The output data table TransClosure contains the transitive closure of G, as shown in Output 16.15.3.

Output 16.15.3 Transitive Closure of a Simple Directed Graph

Selected Rows from Table TRANSCLOSURE		
Index	from	to
1	B	C
2	C	B
3	B	D
4	D	C
5	D	A
6	B	B
7	D	B
8	C	C
9	C	D
10	D	D
11	B	A
12	C	A

The transitive closure of G is shown graphically in Figure 16.24.

Figure 16.24 Transitive Closure of G



Chapter 17
Neural Network Action Set

Contents

Syntax **785**
The annCode Action 786
The annScore Action 791
The annTrain Action 800
Details **820**
Working with Artificial Neural Networks 820
Using a Neural Network to Approximate a Function 820
Examples **821**
Example 17.1: Generate Stand-Alone SAS Score Code for a Neural Network Model . 821
Example 17.2: Score Input Data with a Neural Network Model and the annScore Action 822
Example 17.3: Create and Train a Neural Network 823
References **829**

Neural Network Action Set: Syntax

annCode Action

CASL Syntax

neuralNet.annCode <**result**=*results*> <**status**=*rc*> /

code={

casOut={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

comment=TRUE | FALSE,

fmtWdth=*integer*,

indentSize=*integer*,

labelId=*integer*,

lineSize=*integer*,

noTrim=TRUE | FALSE,

tabForm=TRUE | FALSE

}

listNode="ALL" | "HIDDEN" | "INPUT" | "OUTPUT"

modelId="*string*"

***required parameter** **modelTable**={

caslib="*string*",

```
        computedOnDemand=TRUE | FALSE,
        computedVars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        computedVarsProgram="string",
        dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
       *required parameter name="table-name",
        onDemand=TRUE | FALSE,
        singlePass=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
    }
;
```

Parameter Descriptions

code={*codegen*}

requests that the action produce SAS score code. Specify additional parameters.

For more information about specifying the code parameter, see the common codegen parameter (Appendix A: Common Parameters).

listNode="ALL" | "HIDDEN" | "INPUT" | "OUTPUT"

specifies the nodes to be included in the scored output table. When the autoencoding of input nodes is requested, the default is HIDDEN. This value is particularly useful when autoencoding is applied to reduce the dimension of the input nodes. By reusing the node output values, machine learning algorithms such as neural networks, clustering, decision tree, and forests can use the newly encoded vectors as input.

ALL specifies to include all the nodes in the scored output table.

HIDDEN specifies to include the hidden nodes only.

INPUT specifies to include the input nodes only.

OUTPUT specifies to include the output nodes only.

modelId="*string*"

specifies a model ID variable name that is included in the generated DATA step scoring code. By default, this variable name is the target variable name with ANN_ set as the prefix.

* **modelTable**={*castable*}

specifies the table that contains the artificial neural network model. You can use this model to score a table or generate SAS scoring code.

Alias model

Long form modelTable={name="*table-name*"}

Shortcut form modelTable="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

Neural Network Action Set: Syntax

annScore Action

CASL Syntax

neuralNet.annScore <**result**=*results*> <**status**=*rc*> /

assess=TRUE | FALSE

assessOneRow=TRUE | FALSE

casOut={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

impute=TRUE | FALSE

includeMissing=TRUE | FALSE

listNode="ALL" | "HIDDEN" | "INPUT" | "OUTPUT"

modelId="*string*"

***required parameter** **modelTable**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

```
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        computedVarsProgram="string",
        dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
      *required parameter name="table-name",
        onDemand=TRUE | FALSE,
        singlePass=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
}
*required parameter table={
        caslib="string",
        computedOnDemand=TRUE | FALSE,
        computedVars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
```

```
        nfl=integer
      }, {...}},
      computedVarsProgram="string",
      dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
        parameters},
     *required parameter name="table-name",
      onDemand=TRUE | FALSE,
      singlePass=TRUE | FALSE,
      vars={{
        format="string",
        formattedLength=integer,
        label="string",
       *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
      }, {...}},
      where="where-expression"
  }
  target="variable-name"
;
```

Parameter Descriptions

assess=TRUE | FALSE

when set to True, predicted probabilities are added to the result table for the event levels. You can use these predicted probabilities with the assess action.

Default FALSE

assessOneRow=TRUE | FALSE

when set to True, predicted probabilities are added to the result table for the event levels. All event probabilities are included as separate columns and are named with the prefix _NN_P_. You can use these predicted probabilities with the assess action.

Default FALSE

casOut={*casouttable*}

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the variables to transfer from the input table to the output table.

Alias copyVar

impute=TRUE | FALSE

when set to True, observations with a nonmissing value for the target variable are used as the predicted values. That is, the observed value is assumed to be known without error. Only observations with missing values for the target variable are scored, based on their values for the input variables.

Default FALSE

includeMissing=TRUE | FALSE

by default, observations with missing values are included. When set to False, any observations with missing values for the variables used in the model are not included.

Default TRUE

listNode="ALL" | "HIDDEN" | "INPUT" | "OUTPUT"

specifies the nodes to be included in the scored output table. When the autoencoding of input nodes is requested, the default is HIDDEN. This value is particularly useful when autoencoding is applied to reduce the dimension of the input nodes. By reusing the node output values, machine learning algorithms such as neural networks, clustering, decision tree, and forests can use the newly encoded vectors as input.

ALL specifies to include all the nodes in the scored output table.

HIDDEN specifies to include the hidden nodes only.

INPUT specifies to include the input nodes only.

OUTPUT specifies to include the output nodes only.

modelId="*string*"

specifies the model ID variable name to use when generating the scored table. By default, the variable name is _NN_PredName_ for classifications and _NN_Pred_ for regressions.

* **modelTable**={*castable*}

specifies the table that contains the artificial neural network model. You can use this model to score a table or generate SAS scoring code.

Alias model

Long form modelTable={name="*table-name*"}

Shortcut form modelTable="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

target="*variable-name*"

specifies the target variable when scoring a data set. If the target variable name in the model is the same in the scored table, then this option is not required.

Neural Network Action Set: Syntax

annTrain Action

CASL Syntax

neuralNet.annTrain <**result**=*results*> <**status**=*rc*> /

acts={"EXP", "IDENTITY", "LOGISTIC", "RECTIFIER", "SIN", "SOFTPLUS", "TANH"}

arch="DIRECT" | "GLIM" | "MLP"

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

bias=*double*

casOut={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

code={

casOut={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

comment=TRUE | FALSE,

fmtWdth=*integer*,

indentSize=*integer*,

labelId=*integer*,

lineSize=*integer*,

noTrim=TRUE | FALSE,

tabForm=TRUE | FALSE

}

combs={"ADD", "LINEAR", "RADIAL"}

delta=*double*

dropOut=*double*

dropOutInput=*double*

errorFunc="ENTROPY" | "GAMMA" | "NORMAL" | "POISSON"

fullWeights=TRUE | FALSE

hiddens={*64-bit-integer-1* <, *64-bit-integer-2*, ...>}

includeBias=TRUE | FALSE

***required parameter** **inputs**={{

format="*string*",

formattedLength=*integer*,

```
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
}, {...}}
  inversePriors=TRUE | FALSE
  listNode="ALL" | "HIDDEN" | "INPUT" | "OUTPUT"
  missing="MAX" | "MEAN" | "MIN" | "NONE"
  modelId="string"
  modelTable={
      caslib="string",
      computedOnDemand=TRUE | FALSE,
      computedVars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      computedVarsProgram="string",
      dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
        parameters},
    *required parameter name="table-name",
      onDemand=TRUE | FALSE,
      singlePass=TRUE | FALSE,
      vars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
```

nfd=*integer*,

nfl=*integer*

}, {...}},

where="*where-expression*"

}

nAnns=*64-bit-integer*

nTries=*64-bit-integer*

nloOpts={

algorithm="HF" | "LBFGS" | "SGD",

lbfgsOpt={

lineSearchMethod="ARMIJO" | "HYBRID" | "MORETHUENTE" | "STRONGWOLFE" | "WOLFE"

numCorrections=*64-bit-integer*

},

optmlOpt={

clipWeightMaxNorm=*double*

fConv=*double*

fConvWindow=*64-bit-integer*

gTol=*double*

maxEvals=*64-bit-integer*

maxIters=*64-bit-integer*

maxTime=*double*

regL1=*double*

regL2=*double*

},

printOpt={

logLevel=*64-bit-integer*

printFreq=*64-bit-integer*

printLevel="PRINTBASIC" | "PRINTDETAIL" | "PRINTNONE"

},

sgdOpt={

adaptiveRate=TRUE | FALSE

annealingRate=*double*

commFreq=*64-bit-integer*

learningRate=*double*

miniBatchSize=*64-bit-integer*

momentum=*double*

seed=*64-bit-integer*

useLocking=TRUE | FALSE

},

state={

checkpointFreq=*64-bit-integer*

saveBest=TRUE | FALSE

table={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

}

},

validate={

frequency=*64-bit-integer*

goal=*double*

stagnation=*64-bit-integer*

}

}

nominals={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

randDist="CAUCHY" | "MSRA" | "NORMAL" | "UNIFORM" | "XAVIER"

resume=TRUE | FALSE

samplingRate=*double*

scaleInit=*64-bit-integer*

seed=*double*

std="MIDRANGE" | "NONE" | "STD"

step=*double*

t=*double*

***required parameter table**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>*},

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

singlePass=TRUE | FALSE,

```
      vars={{
           format="string",
           formattedLength=integer,
           label="string",
          *required parameter name="variable-name",
           nfd=integer,
           nfl=integer
      }, {...}},
      where="where-expression"
}
 target="variable-name"
 targetAct="EXP" | "IDENTITY" | "LOGISTIC" | "SIN" | "SOFTMAX" | "TANH"
 targetComb="ADD" | "LINEAR" | "RADIAL"
 targetMissing="MAX" | "MEAN" | "MIN" | "NONE"
 targetStd="MIDRANGE" | "NONE" | "STD"
 validTable={
      caslib="string",
      computedOnDemand=TRUE | FALSE,
      computedVars={{
           format="string",
           formattedLength=integer,
           label="string",
          *required parameter name="variable-name",
           nfd=integer,
           nfl=integer
      }, {...}},
      computedVarsProgram="string",
      dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
        parameters},
     *required parameter name="table-name",
      onDemand=TRUE | FALSE,
```

```
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
  }
  weight="variable-name"
;
```

Parameter Descriptions

acts={"EXP", "IDENTITY", "LOGISTIC", "RECTIFIER", "SIN", "SOFTPLUS", "TANH"}

specifies the activation function for the neurons on each hidden layer.

Alias act

arch="DIRECT" | "GLIM" | "MLP"

specifies the network architecture to be trained.

DIRECT specifies to use an architecture that is an extension of MLP with direct connections between the input layer and the output layer.

GLIM specifies to use the generalized linear model architecture. This uses a two-layer perceptron (one is the input layer and the other is the output layer) without hidden layers or units.

MLP specifies to use a multilayer perceptron with one or more hidden layers.

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies temporary attributes, such as a format, to apply to input variables.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

bias=*double*

specifies a fixed bias value for all the hidden and output neurons. In this case, the bias parameters are fixed and not optimized.

casOut={*casouttable*}

For more information about specifying the casOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

code={*codegen*}

requests that the action produce SAS score code. Specify additional parameters.

For more information about specifying the code parameter, see the common codegen parameter (Appendix A: Common Parameters).

combs={"ADD", "LINEAR", "RADIAL"}

specifies the combination function for the neurons on each hidden layer.

Alias comb

delta=*double*

specifies the annealing parameter when performing a simulated annealing (SA) global optimization. Without this value, the step size and the temperature are used to perform a Monte Carlo (MC) global optimization. When you specify a value, the optimization becomes SA where the temperature is scaled by delta*t at every MC step.

dropOut=*double*

specifies the dropout ratio for the hidden layers. This parameter is valid when SGD is used for network layer optimization only and all the connections use the linear combination function.

Range [0–1)

dropOutInput=*double*

specifies the dropout ratio for the input layers. This parameter is valid when SGD is used for network layer optimization only and all the connections use the linear combination function.

Range [0–1)

errorFunc="ENTROPY" | "GAMMA" | "NORMAL" | "POISSON"

specifies the error function to train the network. If you do not specify this parameter, then the ENTROPY function is used for nominal variables. The NORMAL function is used for interval variables.

fullWeights=TRUE | FALSE

Generates the full weight model for LBFGS

Default FALSE

hiddens={*64-bit-integer-1* <, *64-bit-integer-2*, ...>}

specifies the number of hidden neurons for each hidden layer in the feedforward model. For example, hiddens={5, 3} specifies two hidden layers: one with 5 hidden neurons and the other with 3 hidden neurons. When you specify this parameter, the default architecture is multi-layer perceptron (MLP).

Alias hidden

includeBias=TRUE | FALSE

by default, bias parameters are included for the hidden and output units. When set to False, these parameters are not included.

Default TRUE

* **inputs**={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the input variables to use in the analysis.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

inversePriors=TRUE | FALSE

calculates the weight applied to the prediction error of each nominal target variable as the total number of observations divided by the number of observations whose target class is the same as the current observation.

Default FALSE

listNode="ALL" | "HIDDEN" | "INPUT" | "OUTPUT"

specifies the nodes to be included in the output table that is generated by the DATA step scoring code. When the autoencoding of input nodes is requested, the default is HIDDEN. This value is particularly useful when autoencoding is applied to reduce the dimension of the input nodes. By reusing the node output values, machine learning algorithms such as neural networks, clustering, decision tree, and forests can use the newly encoded vectors as input.

ALL specifies to include all the nodes in the scored output table.

HIDDEN specifies to include the hidden nodes only.

INPUT specifies to include the input nodes only.

OUTPUT specifies to include the output nodes only.

missing="MAX" | "MEAN" | "MIN" | "NONE"

specifies how to impute missing values for the input or target variables. If you do not specify this parameter or the parameter is NONE, then observations with missing values are ignored. For nominal variables, a new category is created for the missing values.

MAX specifies to replace missing values for each variable with its maximum value.

MEAN specifies to replace missing values for each variable with its mean value.

MIN specifies to replace missing values for each variable with its minimum value.

NONE specifies to exclude the observations with missing values

modelId="*string*"

specifies a model ID variable name that is included in the generated DATA step scoring code. By default, this variable name is the target variable name with ANN_ set as the prefix.

modelTable={*castable*}

specifies the table that contains the artificial neural network model. The weights in this table are loaded to initialize the neural network.

Alias model

Long form modelTable={name="*table-name*"}

Shortcut form modelTable="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

nAnns=*64-bit-integer*

specifies the number of networks to select out of the specified number of tries. The networks with the smallest errors are selected as a set of optimal networks. When data is scored, the most frequent predicted values among the selected networks are used to make the final predictions. Note that you must specify a value to perform Monte Carlo or simulated annealing optimizations which also use the delta, step, and t parameters (experimental for this release).

Alias	numAnn
Default	0
Minimum value	0

nTries=*64-bit-integer*

specifies the number of tries when training networks with random initial weights. The network with the smallest error is chosen as the optimal network. Note that you must specify a value to perform Monte Carlo or simulated annealing global optimizations which also use the delta, step, and t parameters.

nloOpts={*casOptml*}

specifies the optimization options.

For more information about specifying the nloOpts parameter, see the common casOptml parameter (Appendix A: Common Parameters).

nominals={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the nominal input and target variables to use in the analysis.

For more information about specifying the nominals parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

randDist="CAUCHY" | "MSRA" | "NORMAL" | "UNIFORM" | "XAVIER"

specifies the distributions for randomly generating the initial network connection weights. All the weights are in the range [-1.0, 1.0]. The initial bias values are zero. When XAVIER or MSRA is specified, the scaleinit option will be ignored.

resume=TRUE | FALSE

Resumes a training optimization using weights obtained from previous training. The initial weights for resuming the optimization are read from a temporary table with the modelTable= option. The specified framework for the model must be the same as the previous framework.

Default FALSE

samplingRate=*double*

specifies the fraction of the data to use for building a neural network.

Range (0–1]

scaleInit=*64-bit-integer*

specifies how to scale the initial weights. If you specify 1, then the range is scaled to [-1.0 / sqrt(n), 1.0 / sqrt(n)], where n is the number of units in the previous layer. If you specify 2, then the range is scaled to [-6.0 / sqrt(n + n1), 6.0 / sqrt(n + n1)], where n1 is the number of units in the current layer.

seed=*double*

specifies the random number seed for generating random numbers to initialize the network weights.

Maximum value MACINT

std="MIDRANGE" | "NONE" | "STD"

specifies the standardization to use on the interval variables.

MIDRANGE specifies to scale the variables to a midrange of 0 and a half-range of 1.

NONE specifies not to alter the variables.

STD specifies to scale the variables to a mean of 0 and a standard deviation of 1.

step=*double*

specifies a step size for perturbations on the network weights when performing Monte Carlo or simulated annealing global optimizations.

t=*double*

specifies the artificial temperature parameter when performing Monte Carlo or simulated annealing global optimizations.

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

target="*variable-name*"

specifies the target or response variable for training. If you do not specify a target, then the artificial neural network is trained for autoencoding.

targetAct="EXP" | "IDENTITY" | "LOGISTIC" | "SIN" | "SOFTMAX" | "TANH"

specifies the activation function for the neurons on the output layer. If you do not specify this parameter, then SOFTMAX is used for nominal variables. The IDENTITY function is used for interval variables. If the target variable is not provided, for the purposes of encoding the input nodes, then the SOFTMAX function is used.

targetComb="ADD" | "LINEAR" | "RADIAL"

specifies the combination function for the neurons on the target output nodes.

ADD adds all the incoming values without using any weights or biases.

LINEAR uses a linear combination of the incoming values and weights.

RADIAL uses a radial basis function with equal heights and unequal widths for all units in the layer.

targetMissing="MAX" | "MEAN" | "MIN" | "NONE"

specifies how to impute missing values for the target variable. If you specify NONE for this parameter, then observations with missing target values are ignored. For nominal variables, a new category is created for the missing values.

MAX specifies to replace missing values for each variable with its maximum value.

MEAN specifies to replace missing values for each variable with its mean value.

MIN specifies to replace missing values for each variable with its minimum value.

NONE specifies to exclude the observations with missing values

targetStd="MIDRANGE" | "NONE" | "STD"

specifies the standardization to use on the interval variables.

MIDRANGE specifies to scale the variables to a midrange of 0 and a half-range of 1.

NONE specifies not to alter the variables.

STD specifies to scale the variables to a mean of 0 and a standard deviation of 1.

validTable={*castable*}

specifies the table with the validation data. Using a validation table enables the early stopping of the iteration process with the nloOpts parameter. The validation table must have the same columns and data types as the training table.

Long form validTable={name="*table-name*"}

Shortcut form validTable="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

weight="*variable-name*"

specifies a variable to weight the prediction errors (the difference between the output of the network and the target value specified in the input data set) for each observation during training.

Details

Working with Artificial Neural Networks

What Is a Neural Network?

Artificial neural networks were originally developed by researchers who were trying to mimic the neurophysiology of the human brain. By combining many simple computing elements (neurons or units) into a highly interconnected system, these researchers hoped to produce complex phenomena such as intelligence. Neural networks are a class of flexible nonlinear regression, discriminant, and data reduction models. By detecting complex nonlinear relationships in data, neural networks can help make predictions about real-world problems.

Neural networks are especially useful for prediction problems in which the following conditions exist:

- No mathematical formula is known that relates inputs to outputs.
- Prediction is more important than explanation.
- There is a lot of training data.

Common applications of neural networks include credit risk assessment, direct marketing, and sales prediction.

The **neuralNet** action set is based on a multilayer perceptron (MLP), which has the following characteristics:

- has any number of inputs
- uses linear combination functions in the hidden and output layers
- uses sigmoid activation functions in the hidden layers
- has one or more hidden layers that contain any number of units

Using a Neural Network to Approximate a Function

The **neuralNet** action set trains a network by minimizing an objective function. The objective function is the sum of a total error function and a penalty function, divided by the total frequency. The total error function is a summation over all the cases in the training data and all the target variables of an individual error function applied to each target value and corresponding output. The penalty function is the product of the weight decay constant and the sum of all network weights other than output biases.

When you develop a neural network, you need to make many choices: the number of inputs to use, which basic network architecture to use, the number of hidden layers to use, the number of units per hidden layer, which activation and combination functions to use, and so on. If you have considerable prior information about the function to be learned, you might be able to make some of these choices based on theoretical considerations.

You might not need any hidden layers at all. Linear and generalized linear models are useful in a wide variety of applications. And even if the function that you want to learn is mildly nonlinear, you might obtain better generalization by using a simple linear model than by using a complicated nonlinear model if there are too little data or too much noise to estimate the nonlinearities accurately. The simplest approach is to begin with a network that has no hidden units and add one hidden unit at a time. Then estimate the generalization error of each network. Stop adding hidden units when the generalization error increases.

If there are enough data, enough hidden units, and enough training time, an MLP with just one hidden layer can learn to approximate virtually any function to any degree of accuracy. A statistical analogy is approximating a function with *n*th-order polynomials. MLPs are known as universal approximators and can be used when you have little prior knowledge of the relationship between inputs and targets.

Examples

Example 17.1: Generate Stand-Alone SAS Score Code for a Neural Network Model

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about PROC CAS, see *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*).

After you train and validate a neural network model, you can use the model to score new data. You can score new data in more than one way. One way is to submit new data and then run the model to score the data by using your data mining environment via SAS Enterprise Miner or SAS Visual Data Mining and Machine Learning to produce scored output.

This example shows how to use the **neuralNet.annCode** action to generate stand-alone SAS scoring code for an ANN model. The SAS score code can be run in a SAS environment that does not have a SAS data mining solution license.

```
proc cas;
  neuralNet.anncode /
    code={labelId=1,comment=TRUE},
    modelTable={name="nnet_train_model"}
    listNode="OUTPUT";
 run;
```

Example 17.2: Score Input Data with a Neural Network Model and the annScore Action

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about PROC CAS, see *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*).

After you train and validate a neural network model, you can use the model to score new data. The most common technique is to submit new data and run the model to score the new data, using your data mining environment via SAS Enterprise Miner or SAS Visual Data Mining and Machine Learning to produce scored output.

To successfully run this example, you must first run the code from the example in the section "Example 17.3: Create and Train a Neural Network" on page 823, which uses the **annTrain** action to create and train a neural network. After you have a trained neural network, you can use that neural network model and the **annScore** action to score new input data, as follows:

```
proc cas;

trnTable = {name="iris_partitioned",where="0=_partind_"}; /*1*/
vldTable = {name="iris_partitioned",where="1=_partind_"}; /*2*/

neuralNet.annScore /                                      /*3*/
  table=trnTable
  modelTable="Nnet_train_model";

neuralNet.annScore /                                      /*4*/
  table=vldTable
  modelTable="Nnet_train_model";
run;
```

1. Identify the training data table. The training data are the observations in the iris_partitioned table that have a value of 0 in the partition indicator column (_partind_).

2. Identify the validation data table. The validation data are the observations in the iris_partitioned table that have a value of 1 in the partition indicator column (_partind_).

3. Invoke the **annScore** action to score the training data. The **annScore** action submits input data to be scored by a trained neural network model. Because the data that are being scored in this block of code are the model training data, you should expect the score code to read all 105 observations and predict the target variable value with a 0% misclassification error. Model training data contain known target values, so you should always expect model training data to have a misclassification error of 0% when they are scored.

4. Invoke the **annScore** action to score the validation data. The **annScore** action submits input data to be scored by a trained neural network model in a SAS data mining environment. Validation data contain known target values, but the training algorithm does not read validation data. The scoring algorithm predicts target values for each observation in the validation data and then compares the predicted values to the known values. The misclassification error percentage is calculated by subtracting the percentage of correctly predicted classifications from 1. Lower misclassification error percentages usually indicate better model performance.

The validation data consist of 30% of the original input data observations, stratified by the target variable Species. The original source data contained 50 observations per iris species; the validation data (at 30%) contain a proportional 15 observations for each of three iris species, for a total of 45 observations. If 42 of 45 observations in the validation data are correctly classified, then the misclassification error for such a model would be 6.67%.

Example 17.3: Create and Train a Neural Network

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

The **annTrain** action creates and trains an artificial neural network (ANN) to approximate unknown functions for classification, regression, or autoencoding tasks.

This example uses the Iris data set in the Sashelp library to create a multilayer perceptron (MLP) neural network for a nominal target. The iris data published by Fisher (1936) contain 150 observations. The sepal length, sepal width, petal length, and petal width are measured in millimeters on 50 iris specimens from each of three species: *Iris setosa, I. versicolor, and I. virginica*. The four measurement types become input variables. The species name becomes the nominal target variable. The objective is to predict the species of an iris flower from measurements of its petal and sepal dimensions.

You can load the Sashelp.Iris data set into your CAS session by naming your CAS engine libref in the first statement of the following DATA step. This DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.iris;
   set sashelp.iris;
run;
```

There are no missing values in the iris data. This is significant, because the **annTrain** action excludes observations that contain missing data from model training. If the input data that you want to use for neural network analysis contain a significant number of observations with missing values, you should replace or impute missing values before you perform model training. Because the iris data contain no missing values, the example does not perform variable replacement.

The example uses the **annTrain** action to create and train a neural network. The neural network approximates a function that predicts an iris flower species based on inputs of the length and width of its sepals and petals (measured in millimeters).

```
proc cas;
      sampling.Stratified /
    table={name="iris",groupby={"species"}}                    /*1*/
    partInd=True                                               /*2*/
    samppct=30,                                                /*3*/
    seed=12345,                                                /*4*/
    output={
      casout={name="iris_partitioned", replace=true},          /*5*/
      copyVars="all"                                           /*6*/
       };
run;



/* Use the map data in the newly added partition column to create */
/* separate CAS tables for Neural Net training and validation.    */


proc cas;

/* data explorations */
table.columninfo/table='iris_partitioned';
table.fetch/table='iris_partitioned' to=5;
simple.freq/table='iris_partitioned' inputs='species';
run;

trnTable = {name="iris_partitioned",where="0=_partind_"};      /*7*/
vldTable = {name="iris_partitioned",where="1=_partind_"};      /*8*/


/* Use the annTrain action to create and train a MLP neural network */
/* for a nominal target SPECIES.                                    */


neuralNet.annTrain /
  table=trnTable,
  target="species"                                              /*9 */
  inputs={"sepallength","sepalwidth","petallength","petalwidth"} /*10*/
  nominals={"species"}                                          /*11*/
  hiddens={2}                                                   /*12*/
  maxiter=1000                                                  /*13*/
  seed=12345,                                                   /*14*/
  randDist="UNIFORM",                                           /*15*/
  scaleInit=1,                                                  /*16*/
  combs={"LINEAR"},                                             /*17*/
  targetAct="SOFTMAX",                                          /*18*/
  errorFunc="ENTROPY",                                          /*19*/
  std="MIDRANGE",                                               /*20*/
  validTable=vldTable,                                          /*21*/
```

```
  casout={name="Nnet_train_model", replace=TRUE}                    /*22*/
  nloOpts={                                                         /*23*/
   optmlOpt={maxIters=250, fConv=1e-10},                            /*24*/
   lbfgsOpt={numCorrections=6},                                     /*25*/
   validate={frequency=1}                                           /*26*/
   };
run;
```

1. Use the **sampling.Stratified** action to partition the Iris input data by the target variable Species.
2. Add the partition indicator column _Partind_ to the output table. The _Partind_ column contains integer values that map to data partitions.
3. Create a sampled partition consisting of 30% of table observations by Species. The remaining 70% of table observations form the second partition.
4. Specify a random seed value of **12345** to be used for sampling function.
5. Name the output table that the **sampling_stratified** action created (with a new partition information column), iris_partitioned. If a table by that name exists in CAS memory, the existing table is overwritten by the new iris_partitioned table content.
6. Specify all the variables in the source table to be transferred to the sampled CAS table.
7. Use the map data in the newly added partition column to create separate CAS tables for neural network training and validation. Let the CAS training table trnTable be a subset of all observations in the iris_partitioned table where the integer value of the column _Partind_ is equal to 1.
8. Use the map data in the newly added partition column to create separate CAS tables for neural network training and validation. Let the CAS validation table vldTable be a subset of all observations in the iris_partitioned table where the integer value of the column _Partind_ is equal to 0.
9. Use the **annTrain** action to create and train a MLP neural network by using the table trnTable with the target variable Species.
10. Specify the four input variables to be used as analysis variables for the ANN analysis.
11. Request that the target variable Species be treated as a nominal variable for analysis.
12. Specify the number of hidden neurons for each hidden layer in the neural network feedforward model. For example, **hiddens={2}** specifies one hidden layer with two hidden neurons.
13. Specify the maximum number of iterations to be performed while seeking objective function convergence.
14. Specify the random seed to use to perform sampling and partitioning tasks.
15. Request that a UNIFORM distribution be used for randomly generating the initial neural network connection weights.
16. Specify a scaling factor for connection weights relative to the number of units in the previous layer. The default value for the **scaleInit** parameter is 1. Setting the value of the **scaleInit** parameter to 2 increases the scale of the connection weights.

17. Specify the LINEAR combination function for the neurons in each hidden layer.
18. Specify the activation function for the neurons in the output layer. The SOFTMAX function is used by default for nominal variables.
19. Specify the error function to train the network. ENTROPY is the default setting for nominal targets.
20. Specify the standardization to use on the interval variables. When the value of the **`std`** parameter is MIDRANGE, the variables are scaled to a midrange of 0 and a half-range of 1. Variables will have a minimum value of –1 and a maximum value of 1.
21. Specify the CAS input table name for the validation table to operate on. This enables early stopping of the iteration process by using the **`optmlOpt`** parameter. Validation and training tables must share the same layout.
22. Specify the Nnet_train_model as the output CAS table.
23. Enable neural algorithm solver optimization tools.
24. Specify 250 maximum iterations for optimization, and also specify 1E–10 as the threshold stopping value for the objective function.
25. Enable the LBFGS solver, using up to six corrections in the LBFGS update. LBFGS is an optimization algorithm in the quasi-Newton method family that approximates the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm by using a limited amount of computer memory.
26. Set the validation option by using only the frequency parameter. When the value of the **`frequency`** parameter is 1, validation will occur every epoch. When **`frequency`** is 0, no validation will occur.

Output 17.3.1 displays an overview of the Fisher's iris data used for the exercise. If you ran a **`table.columnInfo`** command on the Iris input data table in CAS, this is the input train and validation data structure.

Output 17.3.1 Column Information

Results from table.columnInfo

Column Information for IRIS_PARTITIONED in Caslib CASUSERHDFS(xianhu)

Column	Label	Id	Type	Length	Formatted Length
Species	Iris Species	**1**	char	10	10
SepalLength	Sepal Length (mm)	**2**	double	8	12
SepalWidth	Sepal Width (mm)	**3**	double	8	12
PetalLength	Petal Length (mm)	**4**	double	8	12
PetalWidth	Petal Width (mm)	**5**	double	8	12
PartInd	Partition Indicator	**6**	double	8	12

If you use the **`table.fetch`** command on the input table in CAS, you can look at example data rows, displayed in Output 17.3.2.

Output 17.3.2 Fetched Rows

Results from table.fetch

Selected Rows from Table IRIS_PARTITIONED						
Index	Iris Species	Sepal Length (mm)	Sepal Width (mm)	Petal Length (mm)	Petal Width (mm)	Partition Indicator
1	Setosa	50	33	14	2	0
2	Setosa	51	33	17	5	0
3	Setosa	52	34	14	2	0
4	Setosa	50	35	16	6	0
5	Setosa	48	30	14	3	0

If you use the **simple.freq** command on the input table in CAS, you can verify that there are 50 observations from each of three iris species, for a total of 150 observations in the input data table, as displayed in Output 17.3.3.

Output 17.3.3 Frequency for Species

Results from simple.freq

Frequency for IRIS_PARTITIONED				
Column	Character Value	Formatted Value	Level	Frequency
Species	Setosa	Setosa	1	50
Species	Versicolor	Versicolor	2	50
Species	Virginica	Virginica	3	50

When you successfully complete the training process for the input table Iris by using **neuralNet.annTrain**, your results will show a training data iteration history with Objective Function, Loss, and Validation Error columns, as displayed in Output 17.3.4.

Output 17.3.4 Optimization Iteration History

Results from neuralNet.annTrain

Iteration History			
Progress	Objective Function	Loss	Valid Error
1	3.6297981903	3.6297981903	0.377778
2	2.7791013693	2.7791013693	0.333333
3	2.5782926363	2.5782926363	0.333333
4	1.7282928986	1.7282928986	0.222222
5	1.437374815	1.437374815	0.111111
6	1.2779658463	1.2779658463	0.111111
7	0.9175803534	0.9175803534	0.044444
8	0.607647857	0.607647857	0.044444
9	0.2499541848	0.2499541848	0.044444
10	0.1826930557	0.1826930557	0.044444
11	0.1540036188	0.1540036188	0.044444
12	0.1360973228	0.1360973228	0.044444
13	0.1155067965	0.1155067965	0.044444
14	0.0788647659	0.0788647659	0.044444
15	0.051841513	0.051841513	0.044444
16	0.0362879979	0.0362879979	0.044444
17	0.0257927722	0.0257927722	0.044444
18	0.0222274724	0.0222274724	0.044444
19	0.0149660461	0.0149660461	0.044444
20	0.0070598441	0.0070598441	0.044444
21	0.0021743625	0.0021743625	0.044444
22	0.00092302	0.00092302	0.044444
23	0.0003978182	0.0003978182	0.044444
24	0.0001794401	0.0001794401	0.044444
25	0.0000819114	0.0000819114	0.044444
26	0.0000379579	0.0000379579	0.044444
27	0.0000177397	0.0000177397	0.044444

Below the "Iteration History" table, you should see a "Convergence Status" table. For a successful neural network model, the "Convergence Status" should report, "The optimization converged," as displayed in Output 17.3.5.

Output 17.3.5 Convergence Status

Convergence Status
The optimization converged.

A successful model training session includes summary results for the output model, as shown in Output 17.3.6.

Output 17.3.6 Model Information

Neural Net Model Info for IRIS_PARTITIONED	
Model	Neural Net
Number of Observations Used	105
Number of Observations Read	105
Target/Response Variable	Species
Number of Nodes	9
Number of Input Nodes	4
Number of Output Nodes	3
Number of Hidden Nodes	2
Number of Hidden Layers	1
Number of Weight Parameters	14
Number of Bias Parameters	5
Architecture	MLP
Number of Neural Nets	1
Objective Value	0.0000177397
Misclassification Error for Validation (%)	4.4444444444

These results reiterate critical model building factors, such as model type; observations read; target variable; summary of the neural network model input, hidden, and output nodes; weight and bias parameters; final objective value; and the misclassification error for the scored validation data set.

At the bottom of the table, you see the final misclassification error percentage as determined by the validation data. If you were to use this neural network model as a predictive function, and your data came from a pool that had the same data distributions as the Iris validation table, you could expect that 93%–94% of the species predictions would be correct.

References

Fisher, R. A. (1936). "The Use of Multiple Measurements in Taxonomic Problems." *Annals of Eugenics* 7:179–188.

Chapter 18
Nonparametric Bayes Action Set

Contents

Syntax **831**
The gmm Action 832
Details **841**
The gmm Action 841
Displayed Output 841
CAS Table Output 842
Gaussian Mixture Model 843
Dirichlet Process 844
Variational Bayes 844
Inference 844
Results 845
Example **845**
Example 18.1: Clustering with the gmm Action 845
References **852**

Nonparametric Bayes Action Set: Syntax

gmm Action

CASL Syntax

nonParametricBayes.gmm <**result**=*results*> <**status**=*rc*> /

alpha=*double*

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

clusterCovOut={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

clusterSumOut={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

```
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
    replace=TRUE | FALSE,
    replication=integer,
    timeStamp="string",
    where={"string-1" <, "string-2", ...>}
}
display={
    caseSensitive=TRUE | FALSE,
    exclude=TRUE | FALSE,
    excludeAll=TRUE | FALSE,
    keyIsPath=TRUE | FALSE,
    names={"string-1" <, "string-2", ...>},
    pathType="LABEL" | "NAME",
    traceNames=TRUE | FALSE
}
freq="variable-name"
inference={method="VB", method-specific-parameters}
inputs={{
    format="string",
    formattedLength=integer,
    label="string",
  *required parameter name="variable-name",
    nfd=integer,
    nfl=integer
}, {...}}
maxClusters=integer
nominals={{
    format="string",
```

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

output={

*required parameter **casOut**={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

copyVars="ALL" | "ALL_MODEL" | "ALL_NUMERIC" | {*"variable-name-1"* <, *"variable-name-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

saveState={

```
      caslib="string",
      name="table-name",
      promote=TRUE | FALSE,
      replace=TRUE | FALSE
  }
  seed=double
*required parameter table={
      caslib="string",
      computedOnDemand=TRUE | FALSE,
      computedVars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      computedVarsProgram="string",
      dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
        parameters},
     *required parameter name="table-name",
      onDemand=TRUE | FALSE,
      singlePass=TRUE | FALSE,
      vars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
```

```
        where="where-expression"
    }
;
```

Parameter Descriptions

alpha=*double*

specifies the concentration parameter for the Dirichlet process.

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

changes the attributes of variables that are used in this action.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

clusterCovOut={*casouttable*}

creates a table on the server that contains the covariance matrix of each cluster.

For more information about specifying the clusterCovOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

clusterSumOut={*casouttable*}

creates a table on the server that contains the summary of the clustering results including the size, neighbor, and mean of each cluster.

For more information about specifying the clusterSumOut parameter, see the common casouttable parameter (Appendix A: Common Parameters).

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

freq="*variable-name*"

names the numeric variable that contains the frequency of occurrence for each observation.

inference={**method**="VB", *method-specific-parameters*}

specifies the inference method to use in the analysis.

Alias infer

The value that you specify for method determines the other parameters that apply.

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies variables to use for analysis.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

maxClusters=*integer*

specifies the maximum number of clusters.

Alias nClusters

nominals={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies nominal variables to use for analysis.

For more information about specifying the nominals parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

output={*outputStatement*}

creates a table on the server that contains the predicted cluster as well as the probability distribution over all obtained clusters for each observation.

For more information about specifying the output parameter, see the common outputStatement parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

saveState={*casouttable*}

specifies to the table in which to save the model state for future model prediction.

Long form saveState={name="*table-name*"}

Shortcut form saveState="*table-name*"

caslib="*string*"

specifies the name of the caslib to use.

name="*table-name*"

specifies the name to associate with the table.

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

seed=*double*

specifies a double to use to start the pseudorandom number generator for initialization.

Default 0

Minimum value 0

* **table**={*castable*}

specifies the input data table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

Parameters for method="VB"

covariance="DIAGONAL" | "FULL"

specifies the covariance matrix type of the Gaussian mixtures.

Default DIAGONAL

maxVbIter=*64-bit-integer*

specifies the number of iterations for the variational Bayesian (VB) inference.

threshold=*double*

specifies the threshold of the difference between the current and previous likelihoods.

Details

The gmm Action

The **gmm** action performs clustering—a common step in data exploration—on quantitative data in SAS Viya.

This action performs cluster analysis by using the Gaussian mixture model (GMM), which is a probabilistic model that assumes that all the data points are generated from a mixture of Gaussian distributions. This model can be regarded as generalizing *k*-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.

The **gmm** action has the following basic features:

- performs soft clustering, which provides not only the predicted cluster score but also the probability distribution over the clusters for each observation, as a benefit of the probabilistic modeling. For more information, see the section "Gaussian Mixture Model" on page 843.
- learns the best number of clusters during the clustering process, which is supported by the Dirichlet process. This is a nonparametric Bayesian machine learning method. For more information, see the section "Dirichlet Process" on page 844.
- uses a parallel variational Bayes (VB) method as the model inference method. VB is an inference method for probabilistic models that approximates the (intractable) posterior distribution, then iteratively updates the model parameters until it reaches convergence. For more information, see the section "Variational Bayes" on page 844.

Displayed Output

The **gmm** action displays various tables for use in cluster analysis. The following sections describe the output tables in the order of their appearance when the related parameters are specified.

Number of Observations

The "Number of Observations" table displays the number of observations that are read from the input data table and the number of observations that are used in the clustering. The Number of Observations Read column displays the count of raw data that are read from the input data table, and the Number of Observations Used column displays the count of data that are actually used in the clustering. The count in the Number of Observations Used column can be smaller if you exclude the observations (rows) in the input data table that contain missing values.

Descriptive Statistics

The "Descriptive Statistics" table displays the mean, standard deviation, minimum, and maximum for each interval variable that is specified in the **input** parameter. The calculation of these statistics is based on all observations (rows) that do not contain missing values in the input data table.

Any input variable whose standard deviation is zero is a constant variable (column) in the input data table. Such constant variables do not contribute to the clustering, and they are not used in the Gaussian mixture model.

Model Information

The "Model Information" table provides basic information about the parameters that are used in the Gaussian mixture model. This information includes the clustering algorithm, inference method, random number seed, maximum number of clusters, Dirichlet process mass, covariance matrix type, number of variational Bayes (VB) iterations, VB threshold, VB convergence, and number of clusters discovered.

CAS Table Output

The **gmm** action produces results of the cluster analysis in the following four output CAS data tables.

Cluster Score Table

The cluster score table is specified in the **output** parameter. This table shows the soft-clustering results that contain the cluster membership and the probability distribution over all clusters for each observation in the input data table. When the probability distributions over all clusters for all observations are given, the **gmm** action not only predicts the cluster scores but also provides a sense of how it makes the predictions. The cluster score table includes the variables that are specified in the **copyVars** parameter and the following variables:

- _CLUSTER_1_ to _CLUSTER_K_ (the probability distribution over all the K clusters)
- _PREDICTED_CLUSTER_ (the predicted cluster ID)

Cluster Summary Table

The cluster summary table is specified in the **clusterSumOut** parameter. This table contains the Gaussian means of the clusters. It also contains the number of observations in each cluster and the neighbor cluster of each cluster. The cluster summary table includes all the variables that are used in the Gaussian mixture model to indicate the Gaussian means (the constant variables with a standard deviation of zero are excluded) and the following variables to indicate the statistics of the clusters:

- _CLUSTER_ID_ (the cluster ID)
- _SIZE_ (the number of observations in the cluster specified by _CLUSTER_ID_)
- _NEIGHBOR_ (the cluster closest to the cluster specified by _CLUSTER_ID_)

Cluster Covariance Table

The cluster covariance table is specified in the **clusterCovOut** parameter. This table contains the Gaussian covariance matrices of the clusters, which indicate the shapes of the clusters. This contrasts with other clustering algorithms, such as the *k*-means clustering algorithm, where the only statistics that the clustering algorithm knows are the distances of the observations to the centroids of their clusters.

The **gmm** action can capture the variations among the cluster shapes and discover the correct clusters. The cluster covariance table contains all the variables used that the Gaussian mixture model uses to indicate the Gaussian covariances (the constant variables with a standard deviation of zero are excluded) and the _CLUSTER_ID_ variable, which identifies the rows that have the same cluster ID that form the covariance matrix of a cluster.

Analytic Store Table

The analytic store table is specified in the **saveState** parameter. This data table contains the saved Gaussian mixture model that is generated by the **gmm** action. This saved Gaussian mixture model keeps the model status after the clustering is finished, such as the discovered number of clusters, cluster means, and cluster covariances. You can use this model, which is saved in the analytic store table, directly on the data on which you want to perform clustering analysis.

Gaussian Mixture Model

The Gaussian mixture model is a probabilistic model that assumes that all the data points are generated from a mixture of Gaussian distributions; see Rasmussen (1999). The Gaussian mixture model can be regarded as generalizing the *k*-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians. In probabilistic modeling, the probability distribution over all the discovered clusters is inferred for each observation. Thus the Gaussian mixture model provides soft clustering, which is different from the hard clustering that is provided by *k*-means, hierarchical, and density-based spatial clustering.

In the **gmm** action, the Gaussian mixture model is Bayesian: the parameters in the Gaussian mixture model (the means and covariances of the Gaussian distributions and the proportion of the Gaussian mixture) are generated along with their prior distributions; conditioned on these parameters, the observations are generated along with the likelihood distributions; and the parameters in the model are estimated by their posterior distributions. To be specific, the prior and likelihood distributions are listed as follows, where μ is the Gaussian mean; Σ is the Gaussian covariance matrix; π is the proportion of the Gaussian mixture; K is the number of clusters discovered; N is the total number of observations in the input data table; and α_0 and H_0 are the mass and base measure, respectively, of the Dirichlet process H:

- Priors:

  $\mu_k \sim N(\mu_0, \Sigma_0), \quad k = 1, 2, \ldots, K$
  $\Sigma_k^{-1} \sim \text{Wishart}(k_0 \Sigma_0^{-1}, m_0), \quad k = 1, 2, \ldots, K$
  $\pi \sim \text{DP}(\alpha_0, H_0)$

- Likelihood:

  $x_i \sim \sum_{k=1}^{K} \pi_k N(\mu_k, \Sigma_k), \quad i = 1, 2, \ldots, N$

Dirichlet Process

The **gmm** action uses the Dirichlet process in the Gaussian mixture model to find the best number of clusters among the input data. The nonparametric Bayes approach uses stochastic processes instead of probability distributions in probabilistic models to achieve better model flexibility and adaptivity.

In the **gmm** action, the Dirichlet process uses the Gaussian mixture model, with its ability to add new clusters and remove existing clusters during the clustering process, thus finding the best number of clusters adaptively. In the **gmm** action, the Dirichlet process serves as the prior for the proportion of the Gaussian mixture. The likelihood is simply a multinomial distribution over the clusters. To be specific, the prior and posterior for the proportion of the Gaussian mixture are listed as follows, where K is the discovered number of clusters; $n_1, n_2, \ldots, n_K$ is the number of observations in the K clusters; $N = \sum_k^K n_k$ is the total number of observations in the input data table; and α_0 and H_0 are the mass and base measure, respectively, of the Dirichlet process:

- Priors:

  $H \sim \mathrm{DP}(\alpha_0, H_0)$

- Posterior:

  $H|n_1, n_2, \ldots, n_K \sim \mathrm{DP}(\alpha_0 + N, \frac{\alpha_0}{\alpha_0+N} H_0 + \sum_k \frac{n_k}{\alpha_0+N})$

Variational Bayes

The **gmm** action uses a parallel variational Bayesian (VB) method of model inference. Variational Bayesian methods are a family of techniques for approximating the intractable integrals that arise in the posterior distributions of Bayesian models that have computable alternative distributions. The VB algorithm that is applied in the **gmm** action is the mean-field VB algorithm, where the approximation criterion is to minimize the Kullback-Leibler distance to the true model posterior distribution. To be specific, the mean-field principle is as follows, where p is the true posterior distribution and q is the mean-field approximated distribution:

$$q = \mathrm{argmin}_q KL(q||p) = \mathrm{argmin}_q \int q \log(\frac{q}{p})$$

For the details of how to set up the VB inference of the **gmm** action, see the section "Inference" on page 844.

For a general introduction to VB inference, see Bishop (2006). For a specific VB inference for the Gaussian mixture model, see Blei and Jordan (2006).

Inference

The **inference** parameter specifies the inference method that the **gmm** action uses. You can use the **method** subparameter to specify VB as the inference method. For now, the **gmm** action supports only VB inference.

The **covariance** subparameter specifies the covariance matrices types in the Gaussian distributions.

The **maxVbIter** subparameter specifies the maximum number of VB iterations. The VB iterations stop either when the number of iterations reaches the specified number or when the change in the Kullback-Leibler distance between the approximated distribution *q* and the true posterior distribution *p* between two consecutive VB iterations is lower than the threshold specified in the **threshold** subparameter.

The **threshold** subparameter specifies the threshold that defines the convergence of the VB iterations. When the VB iterations stop with the threshold criterion satisfied, the VB algorithm reaches convergence.

Results

The results produced by the **logistic** action are stored in tables. You can access these tables by their names, which are shown in Table 18.1. You use these table names when you specify the **display** parameter.

Table 18.1 Results Tables Produced by the **gmm** Action

Table Name	Description	Parameter
DescStats	Descriptive statistics for interval input variables	Input
ModelInfo	Model information	Default
NObs	Number of observations	Default

Example

Example 18.1: Clustering with the gmm Action

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example uses the Fish data set in the Sashelp library to demonstrate how to use the **gmm** action to perform cluster analysis, and how to use the **store** action in the **aStore** action set to save the trained model and use the saved model for future clustering.

The Fish data set contains 159 observations and seven variables. Among these variables, Species contains different fish species and the other six numeric variables contain statistics of the fish: height, weight, width, and three measures of length.

The following DATA step loads the Sashelp.Fish data set into your CAS session by naming your CAS engine libref in the first statement. The DATA step divides the mycas.Fish data into training and testing data tables. The **gmm** action uses the first 150 observations in the mycas.Fish data for clustering and saves the model, and it uses the saved model and the last 9 observations directly for cluster score prediction.

```
data mycas.fish_train;
   title "Using gmm Action for Clustering";
   set sashelp.fish (obs=150);
run;
data mycas.fish_test;
   set sashelp.fish (firstobs=151);
run;
```

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately named CAS engine libref.

The following PROC CAS statements use the **gmm** action to perform clustering on the training data:

```
proc cas;
   action nonParametricBayes.gmm
      table={name="fish_train"},
      inputs={"Height", "Weight", "Width",
      "Length1", "Length2", "Length3"},
      seed=1234567890,
      nThreads=32,
      maxClusters=100,
      alpha=1,
      inference={method="VB", maxVbIter=1000,
      covariance="DIAGONAL", threshold=0.001},
      output={casOut={name="score", replace=TRUE},
      copyVars={"Height", "Weight", "Width", "Length1",
      "Length2", "Length3", "Species"}},
      clusterSumOut={name="clustersum", replace="TRUE"},
      clusterCovOut={name="clustercov", replace="TRUE"},
      display={names={"NObs", "DescStats", "ModelInfo"}},
      saveState={name="astore", replace="TRUE"};
run;
```

The **table** parameter names the input data table to be analyzed. The **inputs** parameter specifies the input variables to use in clustering. The **seed** parameter specifies the random seed to use in the cluster assignment initialization. The **nThreads** parameter specifies the number of threads to use in the parallel computation. The **maxClusters** parameter specifies the maximum number of possible clusters. The **alpha** parameter specifies the mass parameter of the Dirichlet process. The **inference** parameter specifies the model inference setting. The **method** parameter, which is set to VB, indicates that the parallel variational Bayes (VB) method is used for the model inference. The **maxVbIter** parameter specifies the maximum number of VB iterations; the **covariance** parameter specifies the type of the covariance matrices for the Gaussian mixture model; and the **threshold** parameter specifies the threshold to indicate convergence of the VB iterations. The **output** parameter outputs the clustering scores to the mycas.score data table. The **clusterSumOut** parameter outputs the clusters summary to the mycas.clustersum data table. The **clusterCovOut** parameter outputs the clusters covariance to the mycas.clustercov data table. The **saveState** parameter saves the trained Gaussian mixture model to the mycas.astore data table for future testing.

The **gmm** action generates three ODS tables, which are shown in Output 18.1.1 through Output 18.1.3.

Output 18.1.1 Number of Observations

Using gmm Action for Clustering

Results from nonParametricBayes.gmm

Number of Observations Read	150
Number of Observations Used	149

Output 18.1.2 Descriptive Statistics

Input Variable Statistics

Variable	Mean	Standard Deviation	Min	Max
Height	9.337331	4.080024	1.728400	18.957000
Weight	422.000671	356.628288	0	1650.000000
Width	4.594358	1.564004	1.047600	8.142000
Length1	27.091946	9.660294	7.500000	59.000000
Length2	29.344295	10.316750	8.400000	63.400000
Length3	32.239597	11.138215	8.800000	68.000000

Output 18.1.3 Model Information

Model Information

Clustering Algorithm	Gaussian Mixture Model
Inference Method	Variational Bayesian
Random Number Seed	1234567890
Clusters Maximum	100
Dirichlet Process Mass	1
Covariance Matrix Type	Diagonal
VB Iterations	17
VB Threshold	0.001
VB Converge	Yes
Clusters Discovered	8

In this example, the **gmm** action generates the cluster score table mycas.score. The following statements print the first 15 rows in the table in Output 18.1.4.

Because the statistics of fish in different species overlap, the clusters that the **gmm** action discovers do not exactly match the fish species in the Species variable. However, the **gmm** action provides a good analysis of the inhomogeneity among the data.

```
proc print noobs data=mycas.score (obs=15);
run;
```

Output 18.1.4 Cluster Score Table

Using gmm Action for Clustering

Height	Weight	Width	Length1	Length2	Length3	Species	_CLUSTER_1_	_CLUSTER_2_	_CLUSTER_3_
11.5200	242	4.0200	23.2	25.4	30.0	Bream	3.0238628E-7	0.9999716372	1.1956256E-7
12.7300	363	4.4555	26.3	29.0	33.5	Bream	0.2333574448	0.7666338586	8.6917474E-6
14.1795	500	5.2785	26.8	29.7	34.5	Bream	0.9999708165	0.0000286	1.110233E-12
14.2266	500	4.9594	28.5	30.7	36.2	Bream	0.9999959889	2.2401229E-6	6.1225365E-9
13.7592	500	4.3680	29.1	31.5	36.4	Bream	0.9999902944	8.2562544E-6	3.0905788E-7
15.4380	600	5.5800	29.4	32.0	37.2	Bream	0.9997692534	1.110223E-12	1.110223E-12
15.6330	610	5.1338	30.9	33.5	38.6	Bream	0.9989377925	1.110223E-12	1.110223E-12
15.9936	685	5.3704	31.4	34.0	39.2	Bream	0.978354239	1.110223E-12	1.110223E-12
16.2405	700	5.5890	31.9	35.0	40.5	Bream	0.7151578801	1.110223E-12	1.110223E-12
16.5170	714	5.8515	32.7	36.0	41.5	Bream	0.0838816178	1.110223E-12	1.110223E-12
18.0369	920	6.3063	35.0	38.5	44.1	Bream	1.5272591E-7	1.110223E-12	1.110223E-12
18.6354	975	6.7473	37.4	41.0	45.9	Bream	1.110223E-12	1.110223E-12	1.110223E-12
5.2983	69	2.8217	16.5	18.2	20.3	Roach	1.110223E-12	1.110223E-12	1.110223E-12
6.2160	120	3.5742	18.6	20.0	22.2	Roach	1.110223E-12	2.1547817E-8	1.110223E-12
6.1146	120	3.2943	19.4	21.0	23.7	Roach	1.110223E-12	9.5470428E-8	1.110223E-12

_CLUSTER_4_	_CLUSTER_5_	_CLUSTER_6_	_CLUSTER_7_	_CLUSTER_8_	_PREDICTED_CLUSTER_
1.110286E-12	0.0000279409	1.110286E-12	1.110286E-12	1.110286E-12	2
2.6968608E-9	1.111333E-12	2.1180419E-9	1.111333E-12	1.111333E-12	2
5.8349996E-7	1.110233E-12	1.110233E-12	1.110233E-12	1.110233E-12	1
1.7648349E-6	1.110227E-12	1.110227E-12	1.110227E-12	1.110227E-12	1
1.1402932E-6	1.110245E-12	1.110245E-12	1.110245E-12	1.110245E-12	1
0.0002307466	1.110223E-12	1.110223E-12	1.110223E-12	1.110223E-12	1
0.0010622075	1.110223E-12	1.110223E-12	1.110223E-12	1.110223E-12	1
0.021645761	1.110223E-12	1.110223E-12	1.110223E-12	1.110223E-12	1
0.2848421199	1.110223E-12	1.110223E-12	1.110223E-12	1.110223E-12	1
0.9161183822	1.110223E-12	1.110223E-12	1.110223E-12	1.110223E-12	4
0.9999998473	1.110223E-12	1.110223E-12	1.110223E-12	1.110223E-12	4
1	1.110223E-12	1.110223E-12	1.110223E-12	1.110223E-12	4
1.110223E-12	0.9857152018	1.110223E-12	0.0142847982	1.110223E-12	5
1.110223E-12	0.9999847372	1.110223E-12	0.0000152413	1.110223E-12	5
1.110223E-12	0.999991702	1.110223E-12	8.2025386E-6	1.110223E-12	5

The **gmm** action generates the cluster summary table mycas.clusterSum. The following statements print the table in Output 18.1.5. In this table, you can see that the **gmm** action performs well in assigning the observations to the correct clusters and also finds the correct cluster centers.

```
proc print noobs data=mycas.clustersum;
run;
```

Output 18.1.5 Cluster Summary Table

Using gmm Action for Clustering

_CLUSTER_ID_	_SIZE_	_NEIGHBOR_	Height	Weight	Width	Length1	Length2	Length3
1	23	5	13.6608	546.34	5.30368	29.4112	31.9805	36.6365
2	21	2	8.9104	278.32	4.40867	24.8094	26.9320	29.5571
3	6	1	6.6428	327.35	4.15171	32.5208	34.8979	37.8324
4	30	0	13.6202	853.52	6.70364	35.5933	38.5077	42.0681
5	46	6	6.5562	135.28	3.46895	19.7435	21.5307	23.4985
6	6	0	7.6604	536.13	4.80355	39.5262	42.2591	45.1938
7	12	4	3.9082	57.73	2.01089	12.8912	13.8975	15.2347
8	5	3	9.8259	1236.79	6.25261	49.7294	53.4046	57.1702

The **gmm** action also generates the cluster covariance table mycas.clusterCov. The following statements print the table in Output 18.1.6. In this table, you can see that the **gmm** action successfully captures the variation in the shapes of the clusters.

```
proc print noobs data=mycas.clustercov;
run;
```

Output 18.1.6 Cluster Covariance Table

Using gmm Action for Clustering

_CLUSTER_ID_	Height	Weight	Width	Length1	Length2	Length3
1	4.12480	0.00	0.00000	0.0000	0.0000	0.0000
1	0.00000	12721.52	0.00000	0.0000	0.0000	0.0000
1	0.00000	0.00	0.31445	0.0000	0.0000	0.0000
1	0.00000	0.00	0.00000	5.6988	0.0000	0.0000
1	0.00000	0.00	0.00000	0.0000	6.4155	0.0000
1	0.00000	0.00	0.00000	0.0000	0.0000	8.3510
2	3.64653	0.00	0.00000	0.0000	0.0000	0.0000
2	0.00000	6925.14	0.00000	0.0000	0.0000	0.0000
2	0.00000	0.00	0.15945	0.0000	0.0000	0.0000
2	0.00000	0.00	0.00000	4.9918	0.0000	0.0000
2	0.00000	0.00	0.00000	0.0000	5.6081	0.0000
2	0.00000	0.00	0.00000	0.0000	0.0000	6.6596
3	2.26844	0.00	0.00000	0.0000	0.0000	0.0000
3	0.00000	13841.82	0.00000	0.0000	0.0000	0.0000
3	0.00000	0.00	0.29499	0.0000	0.0000	0.0000
3	0.00000	0.00	0.00000	13.1712	0.0000	0.0000
3	0.00000	0.00	0.00000	0.0000	14.6120	0.0000
3	0.00000	0.00	0.00000	0.0000	0.0000	15.7617
4	8.00257	0.00	0.00000	0.0000	0.0000	0.0000
4	0.00000	23809.43	0.00000	0.0000	0.0000	0.0000
4	0.00000	0.00	0.49086	0.0000	0.0000	0.0000
4	0.00000	0.00	0.00000	10.3403	0.0000	0.0000
4	0.00000	0.00	0.00000	0.0000	11.3305	0.0000
4	0.00000	0.00	0.00000	0.0000	0.0000	11.7454
5	1.85494	0.00	0.00000	0.0000	0.0000	0.0000
5	0.00000	5567.13	0.00000	0.0000	0.0000	0.0000
5	0.00000	0.00	0.17524	0.0000	0.0000	0.0000
5	0.00000	0.00	0.00000	5.7835	0.0000	0.0000
5	0.00000	0.00	0.00000	0.0000	6.4344	0.0000
5	0.00000	0.00	0.00000	0.0000	0.0000	7.8727
6	1.72722	0.00	0.00000	0.0000	0.0000	0.0000
6	0.00000	17045.07	0.00000	0.0000	0.0000	0.0000
6	0.00000	0.00	0.27282	0.0000	0.0000	0.0000
6	0.00000	0.00	0.00000	24.6844	0.0000	0.0000
6	0.00000	0.00	0.00000	0.0000	27.3717	0.0000
6	0.00000	0.00	0.00000	0.0000	0.0000	29.0909
7	4.70924	0.00	0.00000	0.0000	0.0000	0.0000
7	0.00000	15230.12	0.00000	0.0000	0.0000	0.0000
7	0.00000	0.00	0.76042	0.0000	0.0000	0.0000
7	0.00000	0.00	0.00000	20.5925	0.0000	0.0000
7	0.00000	0.00	0.00000	0.0000	24.3943	0.0000
7	0.00000	0.00	0.00000	0.0000	0.0000	29.4208
8	1.77457	0.00	0.00000	0.0000	0.0000	0.0000
8	0.00000	115875.94	0.00000	0.0000	0.0000	0.0000
8	0.00000	0.00	0.66091	0.0000	0.0000	0.0000
8	0.00000	0.00	0.00000	70.6799	0.0000	0.0000

Output 18.1.6 *continued*

Using gmm Action for Clustering

_CLUSTER_ID_	Height	Weight	Width	Length1	Length2	Length3
8	0.00000	0.00	0.00000	0.0000	80.1632	0.0000
8	0.00000	0.00	0.00000	0.0000	0.0000	87.8886

The **gmm** action also generates the analytic store table by using the **saveState** parameter. This table saves the trained model and uses it for testing, as shown in the following code for the **score** action in the **aStore** action set:

```
proc cas;
   action aStore.score
      table={name='fish_test'},
      out={name='newscore'},
      rstore={name='astore'};
run;
```

The following statements print the prediction for the testing data in the mycas.newscore CAS table shown in Output 18.1.7. In this table, you can see that all nine fish that belong to the smelt species are assigned (very confidently) to the correct cluster, just as they are in the Fish training data.

```
proc print noobs data=mycas.newscore (obs=9);
run;
```

Output 18.1.7 Cluster Scores for Testing Data

Using gmm Action for Clustering

_CLUSTER_1_	_CLUSTER_2_	_CLUSTER_3_	_CLUSTER_4_	_CLUSTER_5_	_CLUSTER_6_
1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12
1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12
1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12
1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12
1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12
1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	.000000179	1.1102E-12
1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12
1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12
1.1102E-12	1.1102E-12	1.1102E-12	1.1102E-12	.000000466	1.1102E-12

_CLUSTER_7_	_CLUSTER_8_	_PREDICTED_CLUSTER_
1.00000	1.1102E-12	7
1.00000	1.1102E-12	7
1.00000	1.1102E-12	7
1.00000	1.1102E-12	7
1.00000	1.1102E-12	7
1.00000	1.1102E-12	7
1.00000	1.1102E-12	7
1.00000	1.1102E-12	7
1.00000	1.1102E-12	7

References

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. New York: Springer.

Blei, D. M., and Jordan, M. I. (2006). "Variational Inference for Dirichlet Process Mixtures." *Bayesian Analysis* 1:121–144.

Rasmussen, C. E. (1999). "The Infinite Gaussian Mixture Model." In *NIPS '99: Proceedings of the 12th International Conference on Neural Information Processing Systems*, 554–560. Cambridge, MA: MIT Press.

Chapter 19
Robust PCA Action Set

Contents

Syntax **853**
The mwpca Action 854
The robustpca Action 867
Details **895**
The mwpca Action 895
The robustpca Action 896
Examples **899**
Example 19.1: Monitoring Operation of Four Wind Turbines 899
Example 19.2: Decomposing Data 902
References **905**

Robust PCA Action Set: Syntax

mwpca Action

CASL Syntax

robustPca.mwpca <**result**=*results*> <**status**=*rc*> /

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

center=TRUE | FALSE

cumEigPctTol=*double*

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

fixedMu=TRUE | FALSE

***required parameter** **id**="*variable-name*"

inputs={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

```
      nfl=integer
}, {...}}
lambda=double
lambdaWeight=double
maxIter=integer
method="ALM" | "APG"
mu=double
nThreads=integer
onRpca=TRUE | FALSE
output={
   *required parameter casOut={
         caslib="string"
         compress=TRUE | FALSE
         indexVars={"variable-name-1" <, "variable-name-2", ...>}
         label="string"
         maxMemSize=64-bit-integer
         name="table-name"
         onDemand=TRUE | FALSE
         promote=TRUE | FALSE
         replace=TRUE | FALSE
         replication=integer
         timeStamp="string"
         where={"string-1" <, "string-2", ...>}
   },
    nPC=integer,
    standardPc=TRUE | FALSE,
    winsummary={
         caslib="string"
         compress=TRUE | FALSE
         indexVars={"variable-name-1" <, "variable-name-2", ...>}
         label="string"
         maxMemSize=64-bit-integer
```

```
        name="table-name"
        onDemand=TRUE | FALSE
        promote=TRUE | FALSE
        replace=TRUE | FALSE
        replication=integer
        timeStamp="string"
        where={"string-1" <, "string-2", ...>}
    }
}
outputTables={
    groupByVarsRaw=TRUE | FALSE,
    includeAll=TRUE | FALSE,
    names={"string-1" <, "string-2", ...>} | {key-1={casouttable-1} <, key-2={casouttable-2},
      ...>},
    repeated=TRUE | FALSE,
    replace=TRUE | FALSE
}
*required parameter stepSize=integer
svdMaxRank=integer
svdMethod="EIGEN" | "ITERATIVE" | "RANDOM"
*required parameter table={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
```

```
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    orderBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
  }
   tolerance=double
  *required parameter windowSize=integer
;
```

Parameter Descriptions

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

changes the attributes of variables that are used in this action.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

center=TRUE | FALSE

when set to True, centers the numeric variables by the mean of each column.

Alias centering

Default FALSE

cumEigPctTol=*double*

specifies the significance level of the eigenvalues that determine the rank of the low-rank matrix.

Default 1

Range (0–1]

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

fixedMu=TRUE | FALSE

when set to True, fixes mu in each iteration of the accelerated proximal gradient method. Otherwise, mu is dynamically updated in each iteration.

Default FALSE

* **id**="*variable-name*"

specifies the name of a numeric variable that identifies observations in the input data table. The values need to be a series of numbers that have a common difference. The action stops if this column contains a missing value.

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the numeric variables to be analyzed. If this parameter is omitted, all numeric variables that are not specified in other parameters are analyzed.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

lambda=*double*

specifies the value of lambda, which is multiplied by the L1 norm of the sparse matrix in the objective function.

Default -1

Range (0–10000000000]

lambdaWeight=*double*

specifies the weight of lambda.

Default 1

Range (0–10000000000]

maxIter=*integer*

specifies the maximum number of iterations for robust principal component analysis algorithms.

Default 1000

Minimum value 0

method="ALM" | "APG"

specifies the method to use to perform the robust principal component analysis.

Default ALM

ALM

uses the augmented Lagrange multiplier method.

APG

uses the accelerated proximal gradient method.

mu=*double*

specifies an initial value of mu in the objective function for the accelerated proximal gradient method.

Default 0.001

Range 0–10000000000

nThreads=*integer*

specifies the maximum number of threads to use on each computation node.

Default 16

Minimum value 1

onRpca=TRUE | FALSE

when set to True, performs the robust principal component analysis method at each window.

Default FALSE

output={*mwpcaOutput*}

specifies a list of parameters for the output table.

* **casOut**={*casouttable*}

specifies the name of the output table.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

nPC=*integer*

specifies the number of principal components to be displayed for each window in the output table.

Default 1

Range 1–100

standardPc=TRUE | FALSE

when set to True, standardizes the principal components in the output table.

Default FALSE

winsummary={*casouttable*}

specifies the name of the output table that contains the solution summary for each window.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

* **stepSize**=*integer*

specifies the length of the step size. This parameter is required.

Default 100

Minimum value 1

svdMaxRank=*integer*

specifies the maximum value for rank to be considered in the singular value decomposition

solver.

Default 0

Minimum value 1

svdMethod="EIGEN" | "ITERATIVE" | "RANDOM"

specifies the type of the singular value decomposition solver.

Default EIGEN

EIGEN

uses the eigenvalue decomposition method.

ITERATIVE

uses the iterative singular value decomposition method.

RANDOM

uses the randomized singular value decomposition method.

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

tolerance=*double*

specifies the convergence criterion for the robust principal component analysis algorithms.

Alias	stopcriterion
Default	1e-07
Minimum value	1e-10

* **windowSize**=*integer*

specifies the length of the window. This parameter is required.

Default	5000
Minimum value	2

Robust PCA Action Set: Syntax

robustpca Action

CASL Syntax

robustPca.robustpca <**result**=*results*> <**status**=*rc*> /

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

center=TRUE | FALSE

code={

casOut={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

comment=TRUE | FALSE,

fmtWdth=*integer*,

indentSize=*integer*,

labelId=*integer*,

lineSize=*integer*,

noTrim=TRUE | FALSE,

projectionType="LRS" | "PCA",

tabForm=TRUE | FALSE

}

colStatistics={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

cumEigPctTol=*double*

decomp="NONE" | "PCA" | "SVD"

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

fixedMu=TRUE | FALSE

```
id={"variable-name-1" <, "variable-name-2", ...>}
inputs={{
    format="string",
    formattedLength=integer,
    label="string",
   *required parameter name="variable-name",
    nfd=integer,
    nfl=integer
}, {...}}
lambda=double
lambdaWeight=double
maxIter=integer
method="ALM" | "APG"
mu=double
nThreads=integer
outMat={
    errMat={
        caslib="string"
        compress=TRUE | FALSE
        indexVars={"variable-name-1" <, "variable-name-2", ...>}
        label="string"
        maxMemSize=64-bit-integer
        name="table-name"
        onDemand=TRUE | FALSE
        promote=TRUE | FALSE
        replace=TRUE | FALSE
        replication=integer
        timeStamp="string"
        where={"string-1" <, "string-2", ...>}
    },
    lowRankMat={
        caslib="string"
```

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

sparseMat={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

}

}

outPca={

pcLoadings={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

pcScores={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

}

}

outSvd={

svdDiag={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

svdLeft={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

svdRight={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

pcPrefix="*string*"

saveState={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

scale=TRUE | FALSE

svdMaxRank=*integer*

svdMethod="EIGEN" | "ITERATIVE" | "RANDOM"

svdRand={

power=*integer*,

randSeed=*integer*
}
***required parameter table**={
caslib="*string*",
computedOnDemand=TRUE | FALSE,
computedVars={{
format="*string*",
formattedLength=*integer*,
label="*string*",
***required parameter **name**="*variable-name*",
nfd=*integer*,
nfl=*integer*
}, {...}},
computedVarsProgram="*string*",
dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},
importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},
***required parameter **name**="*table-name*",
onDemand=TRUE | FALSE,
orderBy={{
format="*string*",
formattedLength=*integer*,
label="*string*",
***required parameter **name**="*variable-name*",
nfd=*integer*,
nfl=*integer*
}, {...}},
singlePass=TRUE | FALSE,
vars={{
format="*string*",
formattedLength=*integer*,
label="*string*",

```
        *required parameter name="variable-name",
         nfd=integer,
         nfl=integer
      }, {...}},
      where="where-expression"
   }
   tolerance=double
;
```

Parameter Descriptions

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

changes the attributes of variables that are used in this action.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

center=TRUE | FALSE

when set to True, centers the numeric variables by the mean of each column.

Alias centering

Default FALSE

code={*rpcaCodegen*}

produces SAS score code.

casOut={*casouttable*}

specifies the settings for an output table.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should

allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

comment=TRUE | FALSE

when set to True, adds comments to the DATA step code.

Default FALSE

fmtWdth=*integer*

specifies the width to use for formatting derived numbers such as parameter estimates in the DATA step code.

Alias fmtWidth

Default 20

Range 0–32

indentSize=*integer*

specifies the number of spaces to indent the DATA step code for each indent level.

Default 3

Range 0–10

labelId=*integer*

specifies the label ID to use in array names and statement labels in the DATA step code. By default, a random positive integer is used.

lineSize=*integer*

specifies the line size for the generated code.

Default 120

Range 64–254

noTrim=TRUE | FALSE

requests that the comparison of variables with formatted values be based on the full format width, with padding. By default, leading and trailing blanks are removed from the formatted values.

Default FALSE

projectionType="LRS" | "PCA"

specifies the type of scoring.

Default PCA

LRS

projects the scoring observations in the low-rank space.

PCA

projects the scoring observations onto the principal components.

tabForm=TRUE | FALSE

when set to True, generates the code in way that is appropriate for storing in a table.

Alias tableForm

Default FALSE

colStatistics={*casouttable*}

specifies the name of the output table to contain simple statistics for the variables of the input data set.

For more information about specifying the colStatistics parameter, see the common casouttable parameter (Appendix A: Common Parameters).

cumEigPctTol=*double*

specifies the significance level of the eigenvalues that determine the rank of the low-rank matrix.

Default 1

Range (0–1]

decomp="NONE" | "PCA" | "SVD"

specifies the decomposition method for the low-rank matrix. If the value of the maxiter parameter is 0, decomposition is applied to the original input data instead of to the low-rank matrix.

Default NONE

NONE

performs neither analysis.

PCA

performs principal component analysis.

SVD

performs singular value decomposition.

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

fixedMu=TRUE | FALSE

when set to True, fixes mu in each iteration of the accelerated proximal gradient method. Otherwise, mu is dynamically updated in each iteration.

Default FALSE

id={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the variables to use as record identifiers.

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the numeric variables to be analyzed. If this parameter is omitted, all numeric variables that are not specified in other parameters are analyzed.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

lambda=*double*

specifies the value of lambda, which is multiplied by the L1 norm of the sparse matrix in the objective function.

Default -1

Range (0–10000000000]

lambdaWeight=*double*

specifies the weight of lambda.

Default 1

Range (0–10000000000]

maxIter=*integer*

specifies the maximum number of iterations for robust principal component analysis algorithms.

Default 1000

Minimum value 0

method="ALM" | "APG"

specifies the method to use to perform the robust principal component analysis.

Default ALM

ALM

uses the augmented Lagrange multiplier method.

APG

uses the accelerated proximal gradient method.

mu=*double*

specifies an initial value of mu in the objective function for the accelerated proximal gradient method.

Default 0.001

Range 0–10000000000

nThreads=*integer*

specifies the maximum number of threads to use on each computation node.

Default 16

Minimum value 1

outMat={*outRpcaTabs*}

specifies a list of parameters for the output tables of the robust principal component analysis method.

errMat={*casouttable*}

specifies the name of the output table for the error matrix.

Alias outError

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

lowRankMat={*casouttable*}

specifies the name of the output table for the low-rank matrix.

Alias outLowRank

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

sparseMat={*casouttable*}

specifies the name of the output table for the sparse matrix.

Alias outSparse

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

outPca={*outPcaTabs*}

specifies a list of parameters for the output tables of the principal component analysis.

pcLoadings={*casouttable*}

specifies the name of the output table for the principal component loadings.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

pcScores={*casouttable*}

specifies the name of the output table for the principal component scores.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high

availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

outSvd={*outSvdTabs*}

specifies a list of parameters for the output tables of the singular value decomposition.

svdDiag={*casouttable*}

specifies the name of the output table for the diagonal vector of the rectangular diagonal matrix.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

svdLeft={*casouttable*}

specifies the name of the output table for the left-singular vectors.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

svdRight={*casouttable*}

specifies the name of the output table for the right-singular vectors.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

pcPrefix="*string*"

specifies a prefix for naming the principal components.

Default "Prin"

saveState={*casouttable*}

specifies the output data table in which to save the scoring results to be used in the score action of the aStore action set. You can specify the RPCA_PROJECTION_TYPE subparameter in the options parameter in the score action: the value 0 projects the scoring observations onto the principal component space, and the value 1 projects the scoring observations onto the low-rank subspace.

For more information about specifying the saveState parameter, see the common casouttable parameter (Appendix A: Common Parameters).

scale=TRUE | FALSE

when set to True, scales the numeric variables by the standard deviation of each column.

Alias scaling

Default FALSE

svdMaxRank=*integer*

specifies the maximum value for rank to be considered in the singular value decomposition solver.

Default 0

Minimum value 1

svdMethod="EIGEN" | "ITERATIVE" | "RANDOM"

specifies the type of the singular value decomposition solver.

Default EIGEN

EIGEN

uses the eigenvalue decomposition method.

ITERATIVE

uses the iterative singular value decomposition method.

RANDOM

uses the randomized singular value decomposition method.

svdRand={*randomizedSvd*}

svdRand={*randomizedSvd*}

specifies a list of parameters to use when the value of the svdMethod parameter is RANDOM.

power=*integer*

specifies the parameter power.

Default 0

Minimum value 0

randSeed=*integer*

specifies the seed value.

Default 0

Minimum value 1

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode

parameter is set to REDISTRIBUTE.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

tolerance=*double*

specifies the convergence criterion for the robust principal component analysis algorithms.

Alias	stopcriterion
Default	1e-07
Minimum value	1e-10

Details

The mwpca Action

Principal component analysis (PCA) is a statistical approach that converts a set of correlated variables to a set of linearly uncorrelated variables known as principal components. Because the first few principal components (which are associated with the largest eigenvalues) usually capture most of the variability in the data, they can be tracked over time to assess whether any changes have taken place in the subspace that is spanned by the data. Moving windows PCA (MWPCA) is designed to track whether any such changes occur. MWPCA is useful in monitoring systems that contain many correlated measures for which the correlations hold over time in a properly functioning system. A significant change in the elements of the most important principal components indicates that there has been a change in the data structure. MWPCA is particularly useful in situations where similar machines or machine parts are operating in a similar environment and thus should be functioning similarly as the environment changes. Wind turbines on the same wind farm illustrate a useful application of MWPCA. Although each turbine might produce different output depending on its specific location in the farm, changes in wind conditions should result in highly correlated changes in output from the turbines. In this case, only the elements of the first principal component need to be tracked. If any one of the loading factors in the first principal component of one turbine starts to behave differently than the same loading factor of the other turbines, you should probably investigate whether the associated machine performance is deviating from the others and needs attention.

MWPCA performs better than the Hotelling T^2 approach in detecting degradations when it is applied to systems that have nonstationary measures and systems that have measures that are affected by seasonality. Also, MWPCA can give you a relatively clear indication of which part of the system is getting out of control.

The optimal choice for window size and step size varies by application and frequency of sampling.

Results

The results that the **mwpca** action produces are stored in tables. You can access these tables by their names, which are shown in Table 19.1. For more information about the contents of these tables, see the section "Displayed Output" in the chapter "The MWPCA Procedure" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*).

Table 19.1 Results Tables Produced by the **mwpca** Action

Table Name	Description	Action	Parameter
Modelinfo	Model information	**mwpca**	Default
Dimensions	Dimensions of the input table	**mwpca**	Default
Summary	Summary of the results	**mwpca**	Default

The robustpca Action

Robust principal component analysis (RPCA) is a matrix decomposition algorithm that decomposes an input matrix M into a low-rank matrix L_0 and a sparse matrix S_0, where $M = L_0 + S_0$. This decomposition is obtained by solving a convex programming problem called principal component pursuit (PCP).

Let $||L||_* := \sum_i \rho_i(L)$ denote the nuclear norm of the matrix L (that is, the sum of the singular values of L), and let $||S||_1 = \sum_{ij} |S_{ij}|$ denote the l_1 norm of S. In this case, PCP can be formulated as follows:

$$\begin{aligned} &\text{minimize} && ||L||_* + \lambda||S||_1 \\ &\text{subject to} && L + S = M \end{aligned}$$

For more information about this formulation, see Candès et al. (2011).

RPCA is robust because the principal components are computed from the low-rank matrix. The sparse matrix includes outlier values.[1] As a result, many applications of RPCA focus on the sparse matrix.

The **`robustpca`** action implements one of two algorithms: augmented Lagrange multiplier or accelerated proximal gradient.

Augmented Lagrange Multiplier Method

In general, the augmented Lagrange method is used to solve nonlinear constrained optimization problems. In the case of PCP, an augmented Lagrange function is used to reformulate the PCP problem as the following nonlinear unconstrained optimization problem:

$$\text{minimize} \quad l(L, S, Y) = ||L||_* + \lambda||S||_1 + < Y, M - L - S > + \frac{\mu}{2}||M - L - S||_F^2$$

Candès et al. (2011) use the ALM method to find the solution to the preceding optimization problem. The basic idea is to update S, L, and Y iteratively. At iteration k, given L_k and Y_k, the first step is to find S_{k+1} by minimizing $l(L_k, S, Y_k)$. In the second step, L_{k+1} is obtained by the singular value thresholding operator, which minimizes $l(L, S_{k+1}, Y_k)$.[2] Next the Lagrange multiplier Y_{k+1} is updated. For more information, see Candès et al. (2011).

Accelerated Proximal Gradient Method

Another common relaxation of PCP is the following unconstrained optimization formulation, which contains two terms in the objective function. The first term is the original objective function weighted by μ, and the second term is the penalty term of the constraint:

[1] The sparse matrix can also include noise. The sparsity of the sparse matrix depends highly on the settings of the λ and λ_{weight} parameters.

[2] The singular value thresholding operator is based on the singular value decomposition.

$$\text{minimize} \quad \mu||L||_* + \mu\lambda||S||_1 + \frac{1}{2}||M - L - S||_F^2$$

Lin et al. (2009) apply the accelerated proximal gradient (APG) algorithm to solve the preceding relaxed model. In each iteration of their proposed algorithm, L_{k+1} and S_{k+1} are obtained by applying singular value decomposition to the updated matrix, which is computed based on the values of L and S in the last two iterations (that is, L_{k-1}, L_k, S_{k-1}, and S_k).

The APG algorithm is generally slower than the ALM algorithm. Zhou et al. (2010) suggest that the APG algorithm be used when the observations are significantly corrupted by noise.

Scaling and Centering

If the **`center`** or **`scale`** parameter (or both) is specified, the original data M are standardized as

$$M(i,j) = \frac{M(i,j) - \mu_j}{\sigma_j}$$

where μ_j and σ_j are the mean and standard deviation of the jth column in M, respectively. Therefore, the low-rank matrix (L), the sparse matrix (S), and the error matrix (N) are updated as follows:

$$L(i,j) = L(i,j) * (\sigma_j) + \mu_j$$
$$S(i,j) = S(i,j) * (\sigma_j)$$
$$N(i,j) = N(i,j) * (\sigma_j)$$

(The error matrix N is generated only if the value of the **`method`** parameter is APG.)

Cumulative Eigenvalue Percentage Tolerance

Sometimes when the input data set has noise, the rank that is reported in the ODS summary table from running the **`robustpca`** action (or equivalently, the rank that is reported in the winsummary table that resulted from running the **`mwpca`** action) is higher than the true rank. This occurs because the existence of noise in the input data set can increase the number of eigenvalues that is reported in the svdDiag table. To address this issue, you can eliminate the insignificant eigenvalues by specifying a tolerance on the cumulative percentage of eigenvalues.

Suppose that the eigenvalues that are computed for the low-rank matrix are 1,000, 100, and 1. If CUMEIGPCTTOL=1, then the rank of the low-rank matrix is reported as 3. However, the third eigenvalue is much smaller than the other two eigenvalues. If CUMEIGPCTTOL=0.99, then the first two eigenvalues cumulatively contribute to more than 99.90% of the sum of the eigenvalues. In this case, the smallest eigenvalue can be ignored, and the rank of the low-rank matrix is 2.

Results

The results that the **robustpca** action produces are stored in tables. You can access these tables by their names, which are shown in Table 19.2. For more information about the contents of these tables, see the section "Displayed Output" in the chapter "The RPCA Procedure" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*).

Table 19.2 Results Tables Produced by the **robustpca** Action

Table Name	Description	Action	Parameter
Modelinfo	Model information	**robustpca**	Default
Dimensions	Dimensions of the input table	**robustpca**	Default
Summary	Summary of the results	**robustpca**	Default

Output Data Tables

The **robustpca** action creates output tables for the low-rank matrix (specified in the **lowRankMat** parameter), the sparse matrix (specified in the **sparseMat** parameter), the error matrix (specified in the **errMat** parameter), and the column statistics (specified in the **colStatistics** parameter).

Also, depending on the value of the **decomp** parameter, the low-rank matrix is decomposed into SVD tables (specified in the **svdLeft**, **svdDiag**, and **svdRight** parameters) or PCA tables (specified in the **pcLoadings** and **pcScores** parameters). Table 19.3 lists details about these data tables.

Table 19.3 Output Data Tables Produced by the **robustpca** Action

Parameter	Content	Dimensions
lowRankMat	Low-rank matrix	Number of observations × number of variables
sparseMat	Sparse matrix	Number of observations × number of variables
errMat	Error matrix	Number of observations × number of variables
svdLeft	Left-singular vectors	Number of observations × number of singular values
svdDiag	Singular values in the decreasing order	Number of singular values
svdRight	Right-singular vectors (and an extra column for variable names)	Number of variables × (number of singular values + 1)
pcLoadings	Right-singular vectors (and an extra column for variable names)	Number of variables × (number of singular values + 1)
pcScores	Product of matrix of left-singular vectors and diagonal matrix of singular values	Number of observations × number of singular values

Table 19.3 *continued*

Parameter	Content	Dimensions
`colStatistics`	Mean and standard deviation for each variable of the input data table	Number of variables × 3

Examples

Example 19.1: Monitoring Operation of Four Wind Turbines

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

The following simple example shows how to use the **mwpca** action to monitor the operation of four wind turbines. The data, which are simulated, are the hourly energy (in kilowatts) that each turbine produces.

The following DATA step creates the Turbines data set:

```
data turbines;
   input time Turbine1 Turbine2 Turbine3 Turbine4;
   datalines;
1   3241.250445   4079.490559   3505.854227   3392.565663
2   2395.881828   3410.130913   2783.411447   2694.114311
3   3628.878452   4543.677282   3818.729737   3701.380754
4   2823.608204   3806.751908   3221.480613   3029.188791
5   3743.196614   4632.571843   3859.732421   3817.242149
6   1807.899427   2536.174173   1698.306145   1619.159414
7   2240.006055   3271.723656   2477.616826   2210.066151

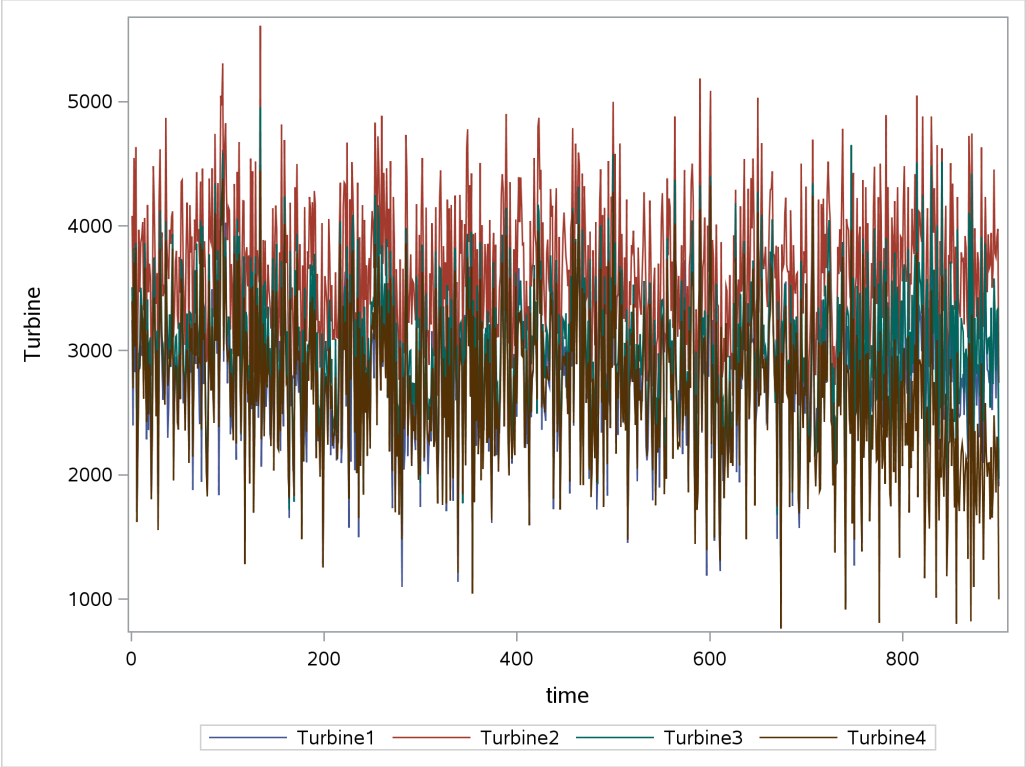
... more lines ...
```

The following statements plot the amount of energy that each turbine produces over time, and Output 19.1.1 shows the resulting plot.

```
proc sgplot data=turbines;
    series x=time y=Turbine1/legendlabel="Turbine1" lineattrs=(thickness=0.1);
    series x=time y=Turbine2/legendlabel="Turbine2" lineattrs=(thickness=0.1);
    series x=time y=Turbine3/legendlabel="Turbine3" lineattrs=(thickness=0.1);
```

```
    series x=time y=Turbine4/legendlabel="Turbine4" lineattrs=(thickness=0.1);
    yaxis label="Turbine";
run;
```

Output 19.1.1 Energy Generated in (Kilowatts)



You can load the Turbines data set into your CAS session by specifying your CAS engine libref in the following DATA step. These statements assumes your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.turbines;
    set turbines;
run;
```

The following statements run the **mwpca** action:

```
ods trace on;
proc cas;
    loadactionset "tkrpca";
    action mwpca /
        table={name="turbines"}
        windowsize=200
        stepsize=1
        id="time"
        output={casout={name="windowpcs" replace=True} standardPc=True}
    ;
run;
```

Note that in the output statement **standardpc** is specified to true. Using this option is recommended.

Output 19.1.2 displays the "Model Information," "Dimensions," and "Results Summary" results tables. In the "Model Information" table, you can see the **windowsize** and **stepsize** parameters and some other parameters that are set to their default values. A window size of 200 and step size of 1 indicates that the procedure starts with the first 200 observations in the data table, calculates the principal components, moves ahead one observation and uses observations 2–201 to calculate the principal components, and continues in this manner until the entire data table has been used.

In the "Dimensions" table, you can see the number of observations and variables in the input table. The **mwpca** action ignores observations that have missing values; however, the ID column cannot be missing for any observation.

In the "Results Summary" table, you can see that the windowpcs data table contains 701 windows, each of which has 200 observations. You can also see that the solution status is optimal; that is, the algorithm ended successfully for all windows.

Output 19.1.2 Results Tables

Model Information	
Data Source	TURBINES
Window Size	200
Step Size	1
RPCA Method	Augmented Lagrange Multiplier
Lambda	0.0707107
Lambda Weight	1
SVD Method	Eigenvalue Decomposition
Cumulative Eigenvalue Percent Tolerance	1

Number of Observations	900
Number of Variables	4

Results Summary	
Number of Windows	701
Solution Status	Optimal
Run Time (Seconds)	0.48

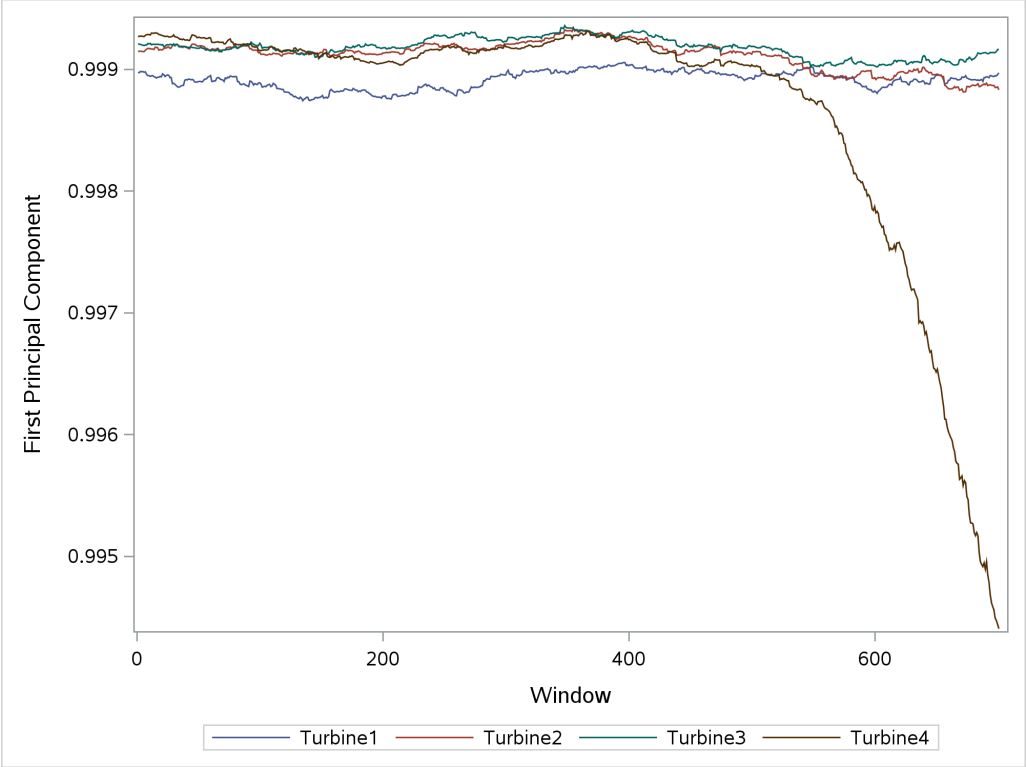
You can see the output table windowpcs in your CAS engine library (mycas). This table contains the first principal component for each turbine and information about how it changes over time. The following statements plot the value of the first principal component over time for each turbine. Output 19.1.3 shows the resulting plot.

```
data windowpcs;
    set mycas.windowpcs;
run;

proc sgplot data =mycas.windowpcs;
    series x= window_id y= Turbine1/legendlabel = "Turbine1" lineattrs = (thickness = 1);
    series x= window_id y= Turbine2/legendlabel = "Turbine2" lineattrs = (thickness = 1);
    series x= window_id y= Turbine3/legendlabel = "Turbine3" lineattrs = (thickness = 1);
    series x= window_id y= Turbine4/legendlabel = "Turbine4" lineattrs = (thickness = 1);
    yaxis label="First Principal Component";
    xaxis label="Window";
run;
```

As you can see in Output 19.1.3, the first principal component displays the same pattern of changes for the first three turbines. However, toward the end, the first principal component for the fourth turbine starts to behave differently from the other three turbines, indicating an abnormality in the operation of the fourth turbine.

Output 19.1.3 First Principal Component



Example 19.2: Decomposing Data

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example shows how to use the **robustpca** action to decompose the input data set into a low-rank matrix and a sparse matrix. It also demonstrates how to decompose the low-rank matrix to obtain the principal components. In this example, the input table rpcaData has 50 observations and nine variables, X1 through X9. The data table is created by adding a rank-1 matrix (L_0) and a sparse matrix (S_0). The rank-1 matrix, L_0, is generated as uv^T, where u and v are a 50×1 vector and a 9×1 vector with iid $N(0, 1)$ entries, respectively. The sparse matrix, S_0, consists of 10% of independently distributed nonzero elements, where each nonzero value is uniformly distributed in $[-10, 10]$.

The following DATA step creates the rpcaData table:

```
data rpcaData;
    input X1 X2 X3 X4 X5 X6 X7 X8 X9;
    datalines;
-0.01913 -0.02729 -0.03017 -0.01974 0.02374 -0.00704 -0.05185 0.01644 -0.00886
-0.97601 -1.39281 -1.53941 -1.00709 1.21127 -0.35943 -2.64598 0.83917 -0.45222
-0.14506 -0.20700 -0.22879 -0.14968 10.18002 -0.05342 -0.39325 0.12472 -0.06721
9.66046 -0.48454 -0.53554 -0.35035 0.42139 -0.12504 -0.92051 0.29194 -0.15732
-0.75473 -1.07704 -1.19040 -0.77877 0.93666 -0.27794 -2.04610 0.64892 -0.34970
-9.97829 10.03099 0.03425 0.02240 -0.02695 0.00800 10.05887 -0.01867 -9.98994
-0.28766 -0.41050 -0.45371 -0.29682 0.35699 -0.10593 -0.77984 0.24733 -0.13328
-0.38761 -0.55314 -0.61136 -0.39995 0.48104 -0.14274 -1.05082 0.33327 -0.17959
-0.73243 -1.04521 -1.15523 -0.75576 10.90898 -0.26973 -1.98564 0.62974 -0.33936
-11.81488 -2.58993 -2.86253 -1.87268 2.25236 -0.66836 -4.92020 -8.43956 -0.84091
-0.21801 -0.31111 -0.34386 -0.22496 0.27056 -0.08029 -0.59104 0.18745 -0.10101
0.28987 0.41365 -9.54281 0.29910 -0.35974 0.10675 0.78584 -0.24923 0.13431
-10.93952 -1.34075 -1.48187 -0.96945 1.16599 -0.34600 -12.54707 0.80780 -0.43532
-0.10800 -0.15412 -0.17034 -0.11144 0.13403 -0.03977 -0.29278 0.09286 -0.05004
0.15719 0.22431 0.24792 0.16219 -0.19508 -9.94211 0.42614 -0.13515 0.07283
-10.30990 -0.44224 -0.48879 -0.31977 0.38460 -0.11413 9.15985 0.26645 -0.14359
-0.31643 9.54843 -0.49910 -0.32651 10.39271 -0.11653 -0.85786 0.27207 -0.14662
-0.98880 -1.41106 -1.55958 8.97971 1.22714 -0.36414 -2.68065 0.85017 -0.45815
-2.20730 -3.14992 -3.48147 -2.27759 -7.26064 -0.81287 -5.98404 11.89783 -1.02273
0.66099 0.94326 1.04254 0.68204 -0.82031 -9.75658 1.79195 -0.56831 0.30626
-0.44430 -0.63403 -0.70077 -0.45845 0.55139 -0.16362 -1.20450 0.38201 -0.20586
-1.35575 -1.93472 -2.13836 -1.39893 1.68255 -0.49928 -3.67548 1.16567 -0.62817
0.10227 0.14594 0.16130 0.10553 -0.12692 0.03766 0.27725 -0.08793 0.04738
0.75583 1.07861 1.19214 -9.22009 -0.93803 0.27835 2.04909 -0.64987 0.35021
-0.83888 -1.19713 -1.32313 -0.86560 -8.95891 9.69107 -2.27424 0.72127 -0.38869
-0.10527 -0.15023 -0.16604 -0.10862 0.13065 -0.03877 -0.28540 0.09051 -0.04878
9.29007 -1.01310 -1.11973 -0.73254 0.88105 -0.26144 -1.92463 0.61039 -0.32894
1.12268 1.60212 1.77076 1.15844 -11.39330 0.41345 3.04362 -0.96528 0.52018
-9.15403 1.20723 1.33430 0.87291 -1.04988 0.31154 2.29344 -0.72736 0.39197
0.39655 0.56590 0.62546 -9.59082 -0.49214 0.14604 1.07506 -0.34095 0.18374
0.74998 1.07026 1.18291 0.77386 -0.93076 10.27619 2.03322 -0.64483 0.34749
-1.12661 -1.60773 -1.77696 -1.16249 1.39818 -0.41489 -3.05428 0.96866 9.47800
-1.46105 -2.08499 -2.30445 8.49242 1.81324 -0.53806 -3.96095 1.25621 -0.67696
-0.93088 -1.32840 8.53177 -0.96052 1.15526 -0.34281 -2.52363 10.80037 -0.43131
1.04589 1.49253 1.64963 1.07920 -1.29800 0.38517 2.83543 -0.89926 0.48460
0.35235 0.50282 0.55575 0.36357 -0.43729 0.12976 0.95524 -0.30295 0.16326
-9.78876 0.30144 0.33317 10.21796 -0.26215 0.07779 0.57267 -10.18162 0.09787
0.50092 0.71484 10.79008 0.51687 -0.62167 0.18447 1.35801 -0.43069 0.23210
0.05180 0.07392 0.08170 0.05345 -0.06428 0.01908 0.14042 -0.04454 0.02400
1.01292 1.44549 1.59763 1.04518 -1.25708 0.37302 2.74605 9.12909 0.46933
0.81896 1.16870 1.29171 0.84504 -1.01637 0.30160 2.22022 -0.70414 0.37946
-1.50085 -2.14179 -2.36722 -1.54865 1.86263 -0.55271 -4.06885 1.29043 -10.69540
8.00772 -2.84308 -3.14233 -2.05573 2.47251 -0.73369 -5.40112 1.71296 -0.92310
0.19119 0.27283 0.30155 10.19728 9.76273 0.07041 0.51831 -0.16438 0.08858
0.24158 0.34474 0.38103 0.24927 -0.29981 0.08896 0.65492 -0.20771 0.11193
0.98543 1.40625 11.55427 1.01681 -1.22296 -9.63710 2.67152 -0.84727 0.45659
-1.16989 -1.66948 -1.84521 -1.20714 1.45188 -0.43083 -3.17159 1.00587 -0.54205
-2.52645 -3.60536 -3.98486 -12.60691 3.13544 -0.93041 -6.84927 12.17224 -1.17060
-0.86476 -1.23406 -1.36395 -0.89230 1.07321 -0.31846 -2.34439 0.74352 -0.40068
-1.37591 -1.96350 -2.17017 -1.41973 1.70757 -0.50670 -3.73014 1.18301 9.36249
;
run;
```

You can load rpcaData into your CAS session by naming your CAS engine libref in the first statement of the following DATA step:

```
data mycas.rpcaData;
    set rpcaData;
run;
```

This statement assumes your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

The following statements run the **robustpca** action and output the decomposition results:

```
ods trace on;
proc cas;
    loadactionset "tkrpca";
    action robustpca /
        table={name="rpcaData"}
        method="ALM"
        decomp="svd"
        outmat={lowrankmat={name="lowrankmat" replace=True},
                sparsemat={name="sparsemat" replace=True}}
        outsvd={svdleft={name="svdleft" replace=True},
                svddiag={name="svddiag" replace=True},
                svdright={name="svdright" replace=True}}
    ;
run;
```

These statements produce the lowrankmat and sparsemat tables, which are the decompositions of the rpcaData input table. Because the **decomp** parameter specifies singular value decomposition (SVD), the statements also produce the svdleft, svddiag, and svdright tables, which are the decompositions of the lowrankmat table.

Output 19.2.1 Results Tables

Model Information	
Data Source	RPCADATA
RPCA Method	Augmented Lagrange Multiplier
SVD Method	Eigenvalue Decomposition
Lambda	0.1414214
Lambda Weight	1
Cumulative Eigenvalue Percent Tolerance Specified	1
Cumulative Eigenvalue Percent Tolerance Applied	1

Number of Observations	50
Number of Variables	9
Number of Observations Used	50
Number of Observations with Missing Values	0

Output 19.2.1 *continued*

Results Summary	
Sparsity of Sparse Matrix	0.2022
Rank of Low Rank Matrix	1
Number of Iterations	36
Solution Status	Optimal
Run Time (Seconds)	0.26

Output 19.2.1 displays the "Model Information," "Dimensions," and "Results Summary" tables. The "Model Information" table shows the specified method for the robust principal component analysis (RPCA), the default method for SVD, and the default values of the **lambda** and **lambdaWeight** parameters. The "Dimensions" table shows the number of observations and variables in the input table and the number of observations that have missing values. (The **robustpca** action ignores observations that have missing values.) The "Results Summary" table shows that the solution status is optimal; that is, the RPCA algorithm converged based on the tolerance value (this example uses the default value of 10^{-7}) within the maximum number of iterations (this example uses the default value of 1,000). In the "Results Summary" table, you can also see that the algorithm converged after 36 iterations. Furthermore, you can observe in this table that the rank of the low-rank matrix is 1, which is the same rank as in the original low-rank matrix, L_0. Note that the sparsity value is 0.2022, which indicates that the sparse matrix contains many nonzero values. In this example, most of the nonzero values are so small that they are ignored, because the values in the DATA step are rounded at the sixth decimal point. The default value of the **lambdaWeight** parameter is 1. As you increase the value of the **lambdaWeight** parameter, the sparsity of the sparse matrix increases.

References

Candès, E. J., Li, X., Ma, Y., and Wright, J. (2011). "Robust Principal Component Analysis?" *Journal of the Association for Computing Machinery* 58:1–37.

Lin, Z., Ganesh, A., Wright, J., Wu, L., Chen, M., and Ma, Y. (2009). "Fast Convex Optimization Algorithms for Exact Recovery of a Corrupted Low-Rank Matrix." In *IEEE International Workshop on Computational Advances in Multi-sensor Adaptive Processing*. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Zhou, Z., Li, X., Wright, J., Candès, E. J., and Ma, Y. (2010). "Stable Principal Component Pursuit." In *IEEE International Symposium on Information Theory Proceedings*, 1518–1522. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Chapter 20
Support Vector Data Description Action Set

Contents

Syntax **907**
The svddTrain Action 908
Details **920**
Examples **920**
Example 20.1: Refrigerant Data 920
References **925**

Support Vector Data Description Action Set: Syntax

svddTrain Action

CASL Syntax

svDataDescription.svddTrain <**result**=*results*> <**status**=*rc*> /

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

bw=*double*

bwList={*double-1* <, *double-2*, ...>}

bwListGen={

beg=*double*,

by=*double*,

end=*double*

}

centerTol=*double*

code={

casOut={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

comment=TRUE | FALSE,

fmtWdth=*integer*,

iProb=TRUE | FALSE,

indentSize=*integer*,

intoCutPt=*double*,

labelId=*integer*,

lineSize=*integer*,

noTrim=TRUE | FALSE,

pCatAll=TRUE | FALSE,

tabForm=TRUE | FALSE

}

delta=*double*

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

fraction=*double*

id={*"variable-name-1"* <, *"variable-name-2"*, ...>}

inputs={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

maxIter=*integer*

maxTime=*double*

maxsv=*double*

nbins=*integer*

nmatchIter=*integer*

nominals={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

output={

*required parameter **casOut**={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

copyVars="ALL" | "ALL_MODEL" | "ALL_NUMERIC" | {*"variable-name-1"* <, *"variable-name-2"*, ...>}

}

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

radiusTol=*double*

sTol=*double*

sampleSize=*integer*

saveState={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

score=TRUE | FALSE

seed=*double*

solver="ACTSET" | "FASTINC" | "STOCHS" | *64-bit-integer*

standardize=TRUE | FALSE

***required parameter** **table**={

caslib="*string*",

```
      computedOnDemand=TRUE | FALSE,
      computedVars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      computedVarsProgram="string",
      dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
        parameters},
     *required parameter name="table-name",
      onDemand=TRUE | FALSE,
      singlePass=TRUE | FALSE,
      vars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      where="where-expression"
  }
   tol=double
   tuneMethod="MEAN" | "MEAN2" | 64-bit-integer
   useConstCol=TRUE | FALSE
   weight="variable-name"
;
```

Parameter Descriptions

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies variable attributes.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

bw=*double*

specifies the radial basis function (RBF) kernel bandwidth value.

bwList={*double-1* <, *double-2*, ...>}

specifies the list of radial basis function (RBF) kernel bandwidth values.

bwListGen={*listGenParm*}

specifies the list generation parameters for radial basis function (RBF) kernel bandwidth values.

beg=*double*

specifies the beginning value for list generation for RBF kernel bandwidth parameters.

Minimum value 1e-06

by=*double*

specifies the step-size value for list generation for RBF kernel bandwidth parameters.

Minimum value 1e-06

end=*double*

specifies the end value for list generation for RBF kernel bandwidth parameters.

Minimum value 1e-06

centerTol=*double*

specifies the convergence criteria for center for the STOCHS solver. Nonpositive values are replaced by the default value.

Alias cTol

Default -1

Minimum value 0

code={*aircodegen*}

produces SAS score code.

For more information about specifying the code parameter, see the common aircodegen parameter (Appendix A: Common Parameters).

delta=*double*

specifies the delta value for tuning

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

fraction=*double*

specifies the expected outlier fraction value.

Alias frac, outlier_fraction

Default 0

Range (0, 1)

id={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the variables to use as record identifiers and to transfer to the state output table.

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to be used in the training.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

maxIter=*integer*

specifies the maximum number of iterations for the STOCHS solver.

Minimum value (exclusive) 0

maxTime=*double*

specifies the maximum amount of time to spend in optimization.

Default 36000

Minimum value 1

maxsv=*double*

specifies the maximum number of observations in the master set support vectors for the STOCHS solver. Values smaller than 5 are ignored.

Default -1

Minimum value 5

nbins=*integer*

specifies the size of the bins for the histogram.

Default 20

Minimum value 2

nmatchIter=*integer*

specifies the nmatch value for the STOCHS solver. Nonpositive values are replaced by the default value.

Alias nMatch

Default -1

Minimum value 0

nominals={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the nominal variables to be used in the training.

For more information about specifying the nominals parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

output={*outputStatement*}

specifies the output data table in which to save support vectors.

For more information about specifying the output parameter, see the common outputStatement parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

radiusTol=*double*

specifies the convergence criteria for the threshold R square for the STOCHS solver. Nonpositive values are replaced by the default value.

Alias rTol

Default -1

Minimum value 0

sTol=*double*

specifies the tolerance value for the STOCHS solver. Nonpositive values are replaced by the default value.

Default -1

Minimum value 1e-06

sampleSize=*integer*

specifies the sample size for the STOCHS solver. Nonpositive values are replaced by the default value.

saveState={*casouttable*}

specifies the output data table in which to save the state of the support vector data description for future scoring.

For more information about specifying the saveState parameter, see the common casouttable parameter (Appendix A: Common Parameters).

score=TRUE | FALSE

specifies the option for scoring training data.

Default FALSE

seed=*double*

specifies the seed value for random number generation.

Minimum value 0

solver="ACTSET" | "FASTINC" | "STOCHS" | *64-bit-integer*

specifies the optimization solver.

standardize=TRUE | FALSE

specifies the standardization option.

Alias std

Default FALSE

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

tol=*double*

specifies the tolerance value for the ACTSET solver. Nonpositive values are replaced by the default value.

Default -1

Minimum value 0

tuneMethod="MEAN" | "MEAN2" | *64-bit-integer*

specifies the method of tuning to use in the action. "MEAN" indicates the mean criterion method, and "MEAN2" indicates the modified mean criterion method.

Alias tm

useConstCol=TRUE | FALSE

when set to True, uses a constant column in the analysis.

Alias ucc

Default FALSE

weight="*variable-name*"

specifies the weight variable.

Details

The **svddTrain** action implements the support vector data description (SVDD) algorithm (Tax and Duin 2004). SVDD is a one-class classification technique that is useful in applications where data that belong to one class are abundant but data about any other class are scarce or missing. Fraud detection, equipment health monitoring, and process control are examples of applications where the majority of data belong to one class. You can use SVDD to model such one-class data and subsequently use the model to perform outlier detection.

In its simplest form, you obtain an SVDD model by building a minimum-radius hypersphere around the one-class training data. The hypersphere provides a compact spherical description of the training data. You can use this training data description to determine whether a new observation is similar to the training data observations. The distance from any new observation to the hypersphere center is computed and compared to the hypersphere radius. If this distance is greater than the radius, the observation is designated as an outlier. Using kernel functions in SVDD formulation provides a more flexible description of training data. Such a description is nonspherical and conforms to the geometry of the data.

SVDD is useful for obtaining a geometric description of data and, in most applications, also for detecting outliers. SVDD is used in areas where the majority of the data belong to one class. Some applications of SVDD follow:

- multivariate process control (Sun and Tsung 2003; Sukchotrat, Kim, and Tsung 2009)
- equipment prognostics and health management (Benkedjouh et al. 2012; Tax, Ypma, and Duin 1999a, b)
- cybersecurity and intrusion detection (Kang, Jeong, and Kong 2012)
- fraud identification (Jeong, An, and Nam 2016; Juszczak et al. 2008; Jeong, An, and Nam 2016)
- hyperspectral image analysis (Banerjee, Burlina, and Diehl 2006; Banerjee, Burlina, and Meth 2007; Sakla et al. 2011)
- analysis of health data from wearable devices (Shin, Lee, and Park 2011; Yang et al. 2010)

Examples

Example 20.1: Refrigerant Data

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud*

Analytic Services: CAS Procedure Programming Guide and Reference, and *SAS Viya: System Programming Guide*.

This example demonstrates how you can use the SVDD procedure to obtain a description of two-variable data and perform outlier detection. The training data in this example are refrigerant analysis data (Heckert and Filliben 2000). The data contain 360 observations and 31 variables. The variables represent various process measurements that are taken during the refrigerant analysis. This example uses the variables y5 (inlet water temperature 1) and y4 (pressure drop 6) to perform SVDD training. These variables are selected to highlight the ability of the SVDD algorithm to obtain an accurate description of the training data.

The following DATA step creates the data set refrigerant (which is stored in the work library by default):

```
data refrigerant;
infile datalines delimiter=',';
input y4 y5 ;
label y4="Pressure Drop 6" y5="Inlet Water Temperature 1";
datalines;
2.0562,25.0904
2.04147,25.1676
2.04922,25.6362

   ... more lines ...
```

The following statements standardize the variables y4 and y5:

```
 proc sql noprint;
 select min(y4), min(y5), max(y4), max(y5)
 into :miny4, :miny5, :maxy4, :maxy5
 from  Refrigerant;
 quit;

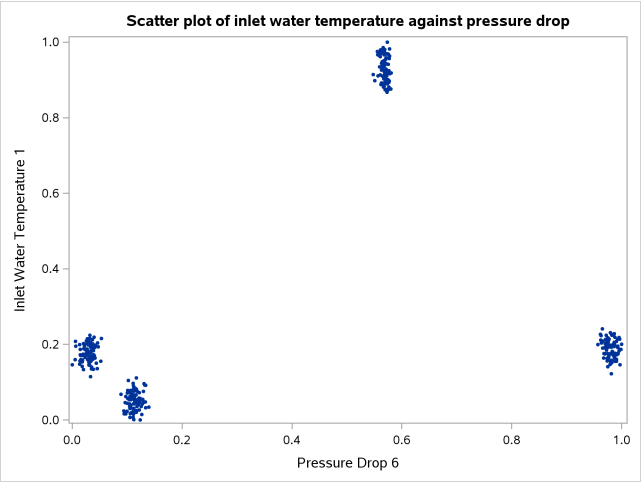
data refrigerant;
   set refrigerant;
   y4=(y4-&miny4)/%sysevalf(&maxy4-&miny4);
   y5=(y5-&miny5)/%sysevalf(&maxy5-&miny5);
run;
```

The following statements plot inlet water temperature 1 against pressure drop 6:

```
 proc sgplot data=Refrigerant;
    title "Refrigerant Analysis";
    title1 "Scatter plot of inlet water temperature against pressure drop";
    scatter x=y4 y=y5/markerattrs=(size=3 symbol=circlefilled);;
 run;
```

Output 20.1.1 shows the results. The scatter plot indicates four distinct clusters.

Output 20.1.1 Scatter Plot of Inlet Water Temperature against Pressure Drop



.

You can load the work.refrigerant data set into your CAS session by specifying your CAS engine libref in the second statement in the following DATA step:

```
data mycas.refrigerant;
   set refrigerant;
run;
```

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

The following statements run the SVDD algorithm on the mycas.refrigerant data table and produce Figure 20.1.2 through Figure 20.1.5:

```
 proc cas;
    loadactionset "svdd";
 run;
    action svddTrain result=r /table= {name='Refrigerant'}
    id={"y4","y5"}
    gauss= 0.021
    input={{name="y4"},{name="y5"}}
    solver = "ACTSET"
    savestate= {name="state"}
    ;
run;
   print r;
quit;
```

The INPUT statement defines the input variables y4 and y5 as interval variables. The KERNEL statement specifies the kernel function as a radial basis function (RBF) and specifies a value of 0.021 for the bandwidth parameter. The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in the mycas.state data table. You can use the analytic store later in the ASTORE procedure to score new data. The “Model Information” table in Output 20.1.2 summarizes the key options and the input data variables.

Output 20.1.2 Refrigerant Analysis Model Information

Model Information	
Optimization Method	Active set
Kernel Type	RBF
RBF Kernel Bandwidth	0.021
Bandwidth Relative Scale	0.1663083933
Expected Outlier Fraction	1E-6
Optimization Tolerance	0.0001
Number of Interval Variables	2
Number of Nominal Variables	0

The "Training Results" table in Output 20.1.3 shows the number of support vectors and the R^2 threshold value.

Output 20.1.3 Refrigerant Analysis Training Results

Training Results	
Number of Support Vectors	38
Number of Support Vectors on Boundary	38
Number of Dropped Observations	0
Threshold R Square Value	0.94341
Constant (C_r) Value	0.05659
Run Time (Seconds)	0.01087

Output 20.1.4 displays the number of observations in the training data set.

Output 20.1.4 Refrigerant Analysis Training Observations

Number of Observations Read	360
Number of Observations Used	360

The "Optimization Summary" table in Output 20.1.5 shows whether the solution is optimal, the number of iterations that were required, and the objective function value.

Output 20.1.5 Refrigerant Analysis Optimization Summary

Optimization Summary	
Number of Iterations	1
Objective Value	0.0565942724
Infeasibility	0.0000295791
Optimization Status	Optimal
Degenerate	No

To evaluate the quality of the training results, scoring is performed on a 200×200 data grid. The following DATA step creates a data set named scoreds to contain the scoring observations:

```
data scoreds;
   do i=0 to 1 by 0.005;
      do  j=0 to 1 by 0.005;
         output;
      end;
   end;
rename i=y5 j=y4;
run;
```

You can load the scoreds data set into your CAS session by specifying your CAS engine libref in the second statement in the following DATA step:

```
data mycas.scoreds;
   set scoreds;
run;
```

The following statements score the mycas.scoreds data table using the score analytic store that was created during training:

```
proc cas  ;
action aStore.score / table={name='scoreds'}
out={name='score_results', compress=false,replace=true, replication=1, promote=false}
rstore={name='state'};
run;
quit;
```

The following statements copy the score_results into the work library.

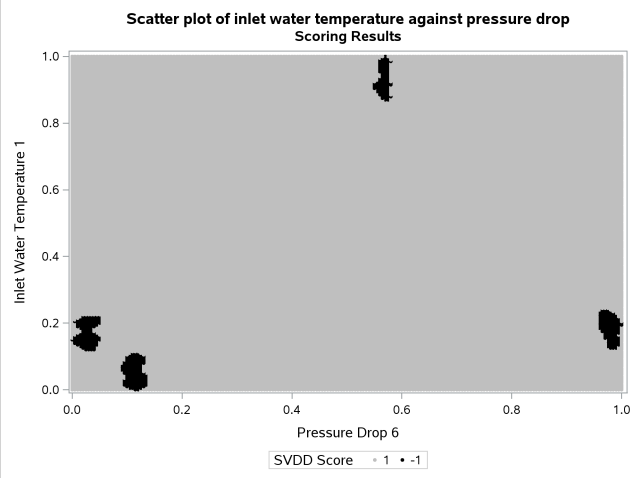
```
data work.score_results;
   set mycas.score_results;
run;
```

The following statements plot the scoring results:

```
proc sgplot data=score_results(where=(_svddscore_ in (-1,1)));
styleattrs datacontrastcolors=(ligr black) datasymbols=(circlefilled);
title "Refrigerant Analysis";
title1 "Scatter plot of inlet water temperature against pressure drop";
title2 "Scoring Results";
scatter x=y4 y=y5/group=_svddscore_ markerattrs=(size=3);
run;
```

Output 20.1.6 shows the scatter plot of the scoring data set. The blue area indicates the observations that are identified as outliers, and the red area indicates observations that are identified as inliers. Comparison between Output 20.1.1 and the training data in Output 20.1.6 indicates that SVDD has correctly identified observations outside the training data as outliers. In this example, the SVDD procedure described four disjoint clusters. Any observation not part of these four clusters was identified as an outlier during scoring.

Output 20.1.6 Scoring Results Scatter Plot



.

References

Banerjee, A., Burlina, P., and Diehl, C. (2006). "A Support Vector Method for Anomaly Detection in Hyperspectral Imagery." *IEEE Transactions on Geoscience and Remote Sensing* 44:2282–2291.

Banerjee, A., Burlina, P., and Meth, R. (2007). "Fast Hyperspectral Anomaly Detection via SVDD." In *Proceedings of the 2007 IEEE International Conference on Image Processing*, vol. 4, 101–104. Piscataway, NJ: IEEE.

Benkedjouh, T., Medjaher, K., Zerhouni, N., and Rechak, S. (2012). "Fault Prognostic of Bearings by Using Support Vector Data Description." In *IEEE Conference on Prognostics and Health Management (PHM)*, 1–7. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Heckert, N. A., and Filliben, J. J. (2000). "Dataplot." Accessed January 17, 2017. http://www.itl.nist.gov/div898/software/dataplot/.

Jeong, M.-K., An, S.-H., and Nam, K. (2016). "SVDD-Based Financial Fraud Detection Method through Respective Learnings of Normal/Abnormal Behaviors." *International Journal of Security and Its Applications* 10:429–438.

Juszczak, P., Adams, N. M., Hand, D. J., Whitrow, C., and Weston, D. J. (2008). "Off-the-Peg and Bespoke Classifiers for Fraud Detection." *Computational Statistics and Data Analysis* 52:4521–4532.

Kang, I., Jeong, M. K., and Kong, D. (2012). "A Differentiated One-Class Classification Method with Applications to Intrusion Detection." *Expert Systems with Applications* 39:3899–3905.

Sakla, W., Chan, A., Ji, J., and Sakla, A. (2011). "An SVDD-Based Algorithm for Target Detection in Hyperspectral Imagery." *IEEE Geoscience and Remote Sensing Letters* 8:384–388.

Shin, J. H., Lee, B., and Park, K. S. (2011). "Detection of Abnormal Living Patterns for Elderly Living Alone Using Support Vector Data Description." *IEEE Transactions on Information Technology in Biomedicine* 15:438–448.

Sukchotrat, T., Kim, S. B., and Tsung, F. (2009). "One-Class Classification-Based Control Charts for Multivariate Process Monitoring." *IIE Transactions* 42:107–120.

Sun, R., and Tsung, F. (2003). "A Kernel-Distance-Based Multivariate Control Chart Using Support Vector Methods." *International Journal of Production Research* 41:2975–2989.

Tax, D. M. J., and Duin, R. P. W. (2004). "Support Vector Data Description." *Machine Learning* 54:45–66.

Tax, D. M. J., Ypma, A., and Duin, R. P. W. (1999a). "Pump Failure Detection Using Support Vector Data Descriptions." In *Advances in Intelligent Data Analysis: Third International Symposium, IDA-99*, edited by D. J. Hand, J. N. Kok, and M. R. Berthold, 415–426. Berlin: Springer-Verlag.

Tax, D. M. J., Ypma, A., and Duin, R. P. W. (1999b). "Support Vector Data Description Applied to Machine Vibration Analysis." In *Proceedings of the Fifth Annual Conference of the Advanced School for Computing and Imaging*, 398–405. Delft, Netherlands: ASCI.

Yang, J., Wang, S., Chen, N., Chen, X., and Shi, P. (2010). "Wearable Accelerometer Based Extendable Activity Recognition System." In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA)*, 3641–3647. Piscataway, NJ: IEEE.

Chapter 21
Support Vector Machine Action Set

Contents

Syntax **927**
The svmTrain Action 928
Details **938**
Displayed Output 938
Results 940
Example **940**
Example 21.1: German Credit Benchmark Data 940

Support Vector Machine Action Set: Syntax

svmTrain Action

CASL Syntax

svm.svmTrain <**result**=*results*> <**status**=*rc*> /

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

c=*double*

code={

casOut={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

comment=TRUE | FALSE,

fmtWdth=*integer*,

iProb=TRUE | FALSE,

indentSize=*integer*,

intoCutPt=*double*,

labelId=*integer*,

lineSize=*integer*,

noTrim=TRUE | FALSE,

pCatAll=TRUE | FALSE,

tabForm=TRUE | FALSE

}

degree=*integer*

id={*"variable-name-1"* <, *"variable-name-2"*, ...>}

includeMissing=TRUE | FALSE

inputs={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

iterationReport=TRUE | FALSE

maxiter=*integer*

nominals={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

noprint=TRUE | FALSE

noscale=TRUE | FALSE

output={

*required parameter **casOut**={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

copyVars="ALL" | "ALL_MODEL" | "ALL_NUMERIC" | {*"variable-name-1"* <, *"variable-name-2"*, ...>}

}

partbyfrac={

seed=*integer*,

test=*double*,

validate=*double*

}

partbyvar={

*required parameter **name**="*variable-name*",

test="*string*",

train="*string*",

validate="*string*"

}

printtarget=TRUE | FALSE

savestate={

caslib="*string*",

name="*table-name*",

promote=TRUE | FALSE,

```
      replace=TRUE | FALSE
   }
  *required parameter table={
      caslib="string",
      computedOnDemand=TRUE | FALSE,
      computedVars={{
         format="string",
         formattedLength=integer,
         label="string",
        *required parameter name="variable-name",
         nfd=integer,
         nfl=integer
      }, {...}},
      computedVarsProgram="string",
      dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
        parameters},
     *required parameter name="table-name",
      onDemand=TRUE | FALSE,
      singlePass=TRUE | FALSE,
      vars={{
         format="string",
         formattedLength=integer,
         label="string",
        *required parameter name="variable-name",
         nfd=integer,
         nfl=integer
      }, {...}},
      where="where-expression"
  }
   target="variable-name"
   tolerance=double
;
```

Parameter Descriptions

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

alters attributes on variables used in this action.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

c=*double*

specifies the penalty.

Default 1

Minimum value (exclusive) 0

code={*aircodegen*}

produces SAS score code.

For more information about specifying the code parameter, see the common aircodegen parameter (Appendix A: Common Parameters).

degree=*integer*

specifies the degree of the polynomial kernel.

Default 1

Minimum value 1

id={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the variables to transfer to the generated table.

includeMissing=TRUE | FALSE

when set to True, includes missing values in the training.

Default FALSE

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies variables to use for analysis.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

iterationReport=TRUE | FALSE

when set to True, calculates the accuracy for each iteration.

Alias iterations

Default FALSE

maxiter=*integer*

specifies the maximum number of iterations.

Default 25

nominals={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies nominal variables to use for analysis.

For more information about specifying the nominals parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

noprint=TRUE | FALSE

when set to True, ignores the ODS tables.

Default FALSE

noscale=TRUE | FALSE

when set to True, does not scale the interval variables.

Default FALSE

output={*outputStatement*}

For more information about specifying the output parameter, see the common outputStatement parameter (Appendix A: Common Parameters).

partbyfrac={*partByFracStatement*}

The partByFracStatement value can be one or more of the following:

seed=*integer*

specifies the seed to use in the random number generator that is used for partitioning the data.

Default 0

test=*double*

randomly assigns the specified proportion of observations in the input table to the testing role. The sum of the fractions that are specified in the test and validate parameters must be less than 1.

Range 0–1

validate=*double*

randomly assigns the specified proportion of observations in the input table to the validation role. The sum of the fractions that are specified in the test and validate parameters must be less than 1.

Alias valid

Range 0–1

partbyvar={*partByVarStatement*}

Long form partbyvar={name="*variable-name*"}

Shortcut form partbyvar="*variable-name*"

The partByVarStatement value can be one or more of the following:

* **name**="*variable-name*"

  names the variable in the input table whose values are used to assign rows to each observation.

test="*string*"

specifies the formatted value of the variable that is used to assign observations to the testing role.

train="*string*"

specifies the formatted value of the variable that is used to assign observations to the training role. If you do not specify the train parameter, then all observations whose roles are not determined by the test and validate parameters are assigned to training.

validate="*string*"

specifies the formatted value of the variable that is used to assign observations to the validation role.

Alias valid

printtarget=TRUE | FALSE

when set to True, generates the table for the target variable.

Default FALSE

savestate={*casouttable*}

specifies to the table in which to save the model for future scoring.

Long form savestate={name="*table-name*"}

Shortcut form savestate="*table-name*"

caslib="*string*"

specifies the name of the caslib to use.

name="*table-name*"

specifies the name to associate with the table.

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

target="*variable-name*"

specifies the target variable to use for analysis.

tolerance=*double*

specifies the tolerance.

Default 1e-06

Details

The **svmTrain** action uses linear or nonlinear kernels to compute support vector machine (SVM) learning classifiers for the binary pattern recognition problem.

In the linear kernel case, the **svmTrain** action computes the parameters w and β that define the model function,

$$f(x) = w^T x + \beta$$

by solving the quadratic optimization problem

$$\begin{aligned} \underset{w,\beta,z}{\text{minimize}} \quad & \frac{1}{2} w^T w + C e^T z \\ \text{subject to} \quad & \mathbf{DX}w + \beta d \geq e - z \\ & z \geq 0 \end{aligned}$$

where $\mathbf{X}$ denotes the $m \times n$ matrix whose rows correspond to the observations, $\mathbf{D}$ denotes a diagonal matrix whose diagonals are 1 or –1, z denotes the slack variables that relax the classification constraints, and C denotes the penalty term. The corresponding dual optimization problem is

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & \frac{1}{2} \alpha^T \mathbf{Q} \alpha - e^T \alpha \\ \text{subject to} \quad & d^T \alpha = 0 \\ & 0 \leq \alpha \leq C \end{aligned}$$

where $\mathbf{Q} = \mathbf{DXX}^T\mathbf{D}$ and α is the Lagrange multiplier. For a more general discussion about duality in the context of quadratic programming, see the chapter "The OPTQP Procedure" in *SAS/OR User's Guide: Mathematical Programming*. At the solution, w is related to α by the equation $w = \mathbf{X}^T\mathbf{D}\alpha$. Observations that correspond to nonzero entries in α are called support vectors. Observations that correspond to entries in α that are active at their upper bound are called support vectors on the margin.

In the nonlinear case, the dual optimization problem satisfies $Q_{ij} = d_i d_j k(x_i, x_j)$, where $k(x, y)$ denotes the selected kernel function, which is defined in Chapter 21, "The SVMACHINE Procedure" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). The corresponding model function is defined in terms of β, α, and the support vectors as follows:

$$f(x) = \beta + \sum_{\alpha_i > 0} \alpha_i d_i k(x, x_i).$$

When nonlinear kernels are used, the dimension of the corresponding primal problem can be prohibitively large or infinite. Polynomial kernels are a special case in that the primal problem definition can be formed explicitly; in this case, the matrix $\mathbf{X}$ corresponds to explicitly projected observations.

Displayed Output

The following sections describe the output that the **svmTrain** action produces by default. The output is organized into various tables, which are discussed in the order of their appearance.

Model Information

The "Model Information" table contains the initial training settings, such as task type, optimization technique, and kernel function type. If the kernel function type is polynomial, then the kernel degree is also displayed.

Number of Observations

The "Number of Observations" table contains the number of observations and the number of observations used.

Training Results

The "Training Results" table shows the model information. It includes but is not limited to the inner product of weights, bias, and the number of support vectors.

Iteration History

The "Iteration History" table contains the number of iterations, the complementarity, and the feasibility. The complementarity is controlled by the **`tolerance`** parameter (which specifies the minimal absolute tolerance at which an iteration stops) and the **`maxiter`** parameter (which controls the number of iterations).

Misclassification Matrix

The "Misclassification Matrix" table contains the target information, both observed and predicted. The columns include the observed target, predicted event, predicted nonevent, and total numbers of events or nonevents for the training data.

Fit Statistics

The "Fit Statistics" table contains the model accuracy information, which includes accuracy, error, sensitivity, and specificity. The statistics are calculated from the "Misclassification Matrix" table.

Iteration Report

The "Iteration Report" table contains the model accuracy value for each iteration step. If the PARTITION statement is specified, then the specified validation or testing data accuracy is also reported.

Predicted Probability Variables

The "Predicted Probability Variables" table contains the target variable and the predicted probability variables. The names of the predicted probability variables begin with P_.

Predicted Target Variable

The "Predicted Target Variable" table contains the predicted target variable, whose name begins with I_.

Results

The results that the **svmTrain** action produces are stored in tables. You can access these tables by their names, which are shown in Table 21.1. For more information about the contents of these tables, see the section "Displayed Output" (Chapter 21, *SAS Visual Data Mining and Machine Learning: Procedures Guide*).

Table 21.1 Results Tables Produced by the **svmTrain** Action

Table Name	Description	Action	Parameter
FitStatistics	Accuracy information about the training	**svmTrain**	Default
IterationReport	Accuracy report for each iteration	**svmTrain**	**iterationreport**
IterHistory	Iteration history	**svmTrain**	Default
Misclassification	Misclassification matrix table	**svmTrain**	Default
ModelInfo	Basic model information for the training	**svmTrain**	Default
NObs	Observation information about the input data	**svmTrain**	Default
PredIntoName	Predicted target variable	**svmTrain**	**printtarget**
PredProbName	Predicted probability variables	**svmTrain**	**printtarget**
TrainingResult	Displays the training results	**svmTrain**	Default

Example

Example 21.1: German Credit Benchmark Data

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example trains the model by using German credit benchmark data, which are available in the sampsio.dmagecr data set. This data set contains 1,000 observations, each of which contains an applicant's information, including the applicant's credit rating (GOOD or BAD). The binary target is named GOOD_BAD. Other input variables are Checking, Duration, History, and so on.

The contents of the sampsio.dmagecr data set are described at http://support.sas.com/documentation/cdl/en/emgs/59885/HTML/default/a001026918.htm.

You can load the sampsio.dmagecr data set into your CAS session by specifying your CAS engine libref in the second statement in the following DATA step:

```
data mycas.dmagecr;
   set sampsio.dmagecr;
run;
```

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

The following statements run the SVM algorithm on the mycas.dmagecr data table by using the **svmTrain** action:

```
proc cas;
    action svm.svmtrain /
        table="dmagecr"
        nominals={"checking" "history" "purpose" "savings" "employed"
                  "marital" "coapp" "property" "other" "job" "housing"
                  "telephon" "foreign" "good_bad"}
        inputs={"duration" "amount" "installp" "resident" "existcr"
                "depends" "age" "checking" "history" "purpose" "savings"
                "employed" "marital" "coapp" "property" "other" "job"
                "housing" "telephon" "foreign"}
        target="good_bad";
run;
```

The **table** parameter names the input data table to be analyzed. The **nominals** parameter lists the nominal input variables and target variable to be used in the training. The **inputs** parameter lists all the input variables to be used in the training. The **target** parameter specifies the variable to be predicted.

The "Training Results" table in Output 21.1.1 shows that the inner product of weights is 11.6121718, the bias is 2.12967726, and the number of support vectors is 531, of which 481 are on the margin.

Output 21.1.1 German Credit Data Training Results

Results from svm.svmTrain

Training Results	
Inner Product of Weights	11.6121718
Bias	2.12967726
Total Slack (Constraint Violations)	492.87883
Norm of Longest Vector	4.17809329
Number of Support Vectors	531
Number of Support Vectors on Margin	481
Maximum F	4.65134808
Minimum F	-2.5713179
Number of Effects	20
Columns in Data Matrix	61

The "Misclassification Matrix" table in Output 21.1.2 shows that among the total 1,000 observations, 700 observations are classified as good and 300 observations are classified as bad. The number of correctly predicted good observations is 626, and the number of correctly predicted bad observations is 158. Thus the accuracy is 78.4%, which is indicated in the "Fit Statistics" table in Output 21.1.3.

Output 21.1.2 German Credit Misclassification Matrix

Misclassification Matrix

	Training Prediction		
Observed	**bad**	**good**	**Total**
bad	158	142	300
good	74	626	700
Total	232	768	1000

Output 21.1.3 German Credit Accuracy

Fit Statistics

Statistic	**Training**
Accuracy	0.7840
Error	0.2160
Sensitivity	0.5267
Specificity	0.8943

Chapter 22
Text Mining Action Set

Contents

Syntax **943**
The tmMine Action 944
The tmScore Action 984
The tmSvd Action 1020
Details **1041**
Working with the TextMining Action Set 1041
Examples **1043**
Example 22.1: Derive Topics from a Document Collection 1043
Example 22.2: Assign Discovered Topics to New Documents 1045
Example 22.3: Apply the SVD and Topic Discovery after Parsing 1047

Text Mining Action Set: Syntax

tmMine Action

CASL Syntax

textMining.tmMine <**result**=*results*> <**status**=*rc*> /

binaryLocation="*string*"

cellWeight="LOG" | "NONE"

child={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

complexTag=TRUE | FALSE

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

defaultEntitiesPriority=*integer*

***required parameter** **docId**="*variable-name*"

docPro={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

docStdMultiple=*double*

***required parameter documents**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

***required parameter name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

***required parameter name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-*

```
        parameters},
    *required parameter name="table-name",
      onDemand=TRUE | FALSE,
      orderBy={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      singlePass=TRUE | FALSE,
      vars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
      }, {...}},
      where="where-expression"
}
  entities="NONE" | "STD"
  exactDocPro=TRUE | FALSE
  exactWeight=TRUE | FALSE
  includeEmptyDocument=TRUE | FALSE
  k=integer
  language="ARABIC" | "CHINESE" | "CROATIAN" | "CZECH" | "DANISH" | "DUTCH" |
    "ENGLISH" | "FARSI" | "FINNISH" | "FRENCH" | "GERMAN" | "GREEK" | "HEBREW" |
    "HINDI" | "HUNGARIAN" | "INDONESIAN" | "ITALIAN" | "JAPANESE" | "KOREAN" |
    "NORWEGIAN" | "POLISH" | "PORTUGUESE" | "ROMANIAN" | "RUSSIAN" | "SLOVAK" |
    "SLOVENE" | "SPANISH" | "SWEDISH" | "TAGALOG" | "THAI" | "TURKISH" | "VIETNAMESE"
  legacyNames=TRUE | FALSE
  liti={
```

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

```
    }, {...}},
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
maxK=integer
multiterm={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
```

```
        nfl=integer
    }, {...}},
    groupByMode="NOSORT" | "REDISTRIBUTE",
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    orderBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
nThreads=integer
norm="ALL" | "DOC" | "NONE" | "WORD"
nounGroups=TRUE | FALSE
numLabels=integer
offset={
    caslib="string",
```

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

parent={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

parseConfig={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

reduce=*integer*

resolution="HIGH" | "LOW" | "MED"

rotate="PROMAX" | "VARIMAX"

rowPivot=*double*

s={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

saveState={

caslib="*string*",

name="*table-name*",

promote=TRUE | FALSE,

replace=TRUE | FALSE

}

```
selectAttribute={
    opType="IGNORE" | "KEEP",
   *required parameter tagList={"string-1" <, "string-2", ...>}
}
selectEntity={
    opType="IGNORE" | "KEEP",
   *required parameter tagList={"string-1" <, "string-2", ...>}
}
selectPos={
    opType="IGNORE" | "KEEP",
   *required parameter tagList={"string-1" <, "string-2", ...>}
}
showDroppedTerms=TRUE | FALSE
startList={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
       *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
       *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
```

```
    }, {...}},
    groupByMode="NOSORT" | "REDISTRIBUTE",
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    orderBy={{
        format="string",
        formattedLength=integer,
        label="string",
       *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
       *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
stemming=TRUE | FALSE
stopList={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
```

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

singlePass=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

```
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
synonyms={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    groupByMode="NOSORT" | "REDISTRIBUTE",
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
```

```
    onDemand=TRUE | FALSE,
    orderBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
tagging=TRUE | FALSE
target="variable-name"
termStdMultiple=double
termTopics={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
```

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

termWeight="ENTROPY" | "MI" | "NONE"

terms={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

***required parameter** **text**="*variable-name*"

tolerance=*double*

topicDecision=TRUE | FALSE

topics={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

u={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

v={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

```
    }
    wordPro={
        caslib="string",
        compress=TRUE | FALSE,
        indexVars={"variable-name-1" <, "variable-name-2", ...>},
        label="string",
        maxMemSize=64-bit-integer,
        name="table-name",
        onDemand=TRUE | FALSE,
        promote=TRUE | FALSE,
        replace=TRUE | FALSE,
        replication=integer,
        timeStamp="string",
        where={"string-1" <, "string-2", ...>}
    }
;
```

Parameter Descriptions

binaryLocation="*string*"

specifies the location of the language binary files. The location must be valid. If it is not specified, a default installation location is used.

cellWeight="LOG" | "NONE"

specifies how the elements in the term-by-document matrix (the parent output table) are weighted.

child={*casouttable*}

specifies the name of the output CAS table to contain a compressed representation of the sparse term-by-document matrix with raw counts.

For more information about specifying the child parameter, see the common casouttable parameter (Appendix A: Common Parameters).

complexTag=TRUE | FALSE

Specifies if the complex tagging process should be used for tokenizing and accumulating. This parameter requires a SAS Visual Text Analytics license.

Default FALSE

copyVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies a list of variables from the documents table that are to be retained on the output docPro table. This parameter requires a SAS Visual Text Analytics license or a SAS Visual Data Mining and Machine Learning license.

Alias keepVars, keepVar

defaultEntitiesPriority=*integer*

specifies the priority of the default standard LITI file when both custom and default standard LITI files are used. The default standard LITI file has the lowest priority compared to other LITI files. This parameter requires a SAS Visual Text Analytics license.

Default 1

Range 0–32

* **docId**="*variable-name*"

specifies the character or numeric variable on the documents table that contains the ID of each document.

Default "DOC_ID"

docPro={*casouttable*}

specifies the name of the table to contain the SVD projections of the documents.

For more information about specifying the docPro parameter, see the common casouttable parameter (Appendix A: Common Parameters).

docStdMultiple=*double*

Specifies how many standard deviations above the mean to set the document cutoff. This parameter requires a SAS Visual Text Analytics license.

Default 1

Range 0–10

* **documents**={*castable*}

names the input CAS table of documents to be parsed. You must include a text variable specified with textVar and a document ID variable specified with docIdVar.

Long form documents={name="*table-name*"}

Shortcut form documents="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table

from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

entities="NONE" | "STD"

specifies whether to extract entities in parsing. If set to None, no entities are output. If set to STD, the standard entities are output.

Default NONE

exactDocPro=TRUE | FALSE

Specifies if the exact document projection values should be output. This parameter requires a SAS Visual Text Analytics license.

Default TRUE

exactWeight=TRUE | FALSE

specifies if the exact entries on the u table are to be used in the topic computation, otherwise the values are rounded to three decimal places.

Alias exactWeights

Default FALSE

includeEmptyDocument=TRUE | FALSE

Indicates if empty document indicators are included on parent table. This parameter requires a SAS Visual Text Analytics license.

Default FALSE

k=*integer*

specifies the number of dimensions to be extracted (also the number of derived topics). If the input data is too small for the requested number of dimensions, this value is adjusted to complete the calculation.

Alias numTopics

Range 1–1000

language="ARABIC" | "CHINESE" | "CROATIAN" | "CZECH" | "DANISH" | "DUTCH" | "ENGLISH" | "FARSI" | "FINNISH" | "FRENCH" | "GERMAN" | "GREEK" | "HEBREW" | "HINDI" | "HUNGARIAN" | "INDONESIAN" | "ITALIAN" | "JAPANESE" | "KOREAN" | "NORWEGIAN" | "POLISH" | "PORTUGUESE" | "ROMANIAN" | "RUSSIAN" | "SLOVAK" | "SLOVENE" | "SPANISH" | "SWEDISH" | "TAGALOG" | "THAI" | "TURKISH" | "VIETNAMESE"

specifies the language used in the text variable of the input document table.

Default ENGLISH

legacyNames=TRUE | FALSE

specifies whether to use the legacy variable names on tables. This parameter requires a SAS Visual Text Analytics license or a SAS Visual Data Mining and Machine Learning license.

Default FALSE

liti={*castable*}

Specifies the input table that contains the custom entity extraction object. This parameter requires a SAS Visual Text Analytics license.

Long form liti={name="*table-name*"}

Shortcut form liti="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

maxK=*integer*

specifies the maximum number of dimensions to be extracted. The maxK option can be used in conjunction with the resolution option to dynamically select the recommended number of dimensions. If you wish to use a specific number of dimensions use maxK and set the resolution to high, or use the k parameter.

Default 10

Range 1–1000

multiterm={*castable*}

specifies the name of the CAS table that contains a list of multi-word terms and their part-of-speech types. Each multi-word term is parsed as a single token.

Long form multiterm={name="*table-name*"}

Shortcut form multiterm="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

nThreads=*integer*

specifies the number of threads to be used per node. The value must be an integer. When the value is 0, the number of threads equals the number of CPUs.

Default 8

Minimum value 0

norm="ALL" | "DOC" | "NONE" | "WORD"

indicates whether the document projections, term projections, or both are normalized. The normalization converts the representation from depending on angles between vectors to one based on Euclidean distances between vectors.

Default ALL

nounGroups=TRUE | FALSE

specifies that noun group extraction is to occur in parsing. When set to TRUE, noun groups become additional rows on the position table and this is reflected in the terms and parent tables. This parameter requires a SAS Visual Text Analytics license or a SAS Visual Data Mining and Machine Learning license.

Default TRUE

numLabels=*integer*

specifies the number of terms to use in the descriptive label for each topic.

Default 5

Range 1–500

offset={*casouttable*}

specifies the name of the output CAS table to contain the position information about the occurrences of child terms in the document collection.

For more information about specifying the offset parameter, see the common casouttable parameter (Appendix A: Common Parameters).

parent={*casouttable*}

specifies the name of the output CAS table to contain a compressed representation of the sparse term-by-document matrix.

For more information about specifying the parent parameter, see the common casouttable parameter (Appendix A: Common Parameters).

parseConfig={*casouttable*}

specifies the name of the config CAS table to contain parsing configuration information.

For more information about specifying the parseConfig parameter, see the common casouttable parameter (Appendix A: Common Parameters).

reduce=*integer*

specifies the minimum number of documents a term should be in to be kept. The value must be an integer.

Default 10

Range 1–32767

resolution="HIGH" | "LOW" | "MED"

specifies the desired resolution level for the recommended number of dimensions to be extracted by the SVD.

Default HIGH

rotate="PROMAX" | "VARIMAX"

specifies the type of rotation used to maximize the explanatory power of each topic. A VARIMAX rotation produces uncorrelated topics and a PROMAX rotation produces correlated topics.

Default VARIMAX

rowPivot=*double*

Specifies the row pivot weight for document normalization of the parent table before the SVD. A negative value turns off the rowPivot process. When topics are requested a rowPivot=1 value is used by default. This parameter requires a SAS Visual Text Analytics license.

Default -1

Range -1–1

s={*casouttable*}

specifies the S matrix, which is a diagonal matrix that is output in compressed form, with two variables and k rows. The variable _ID_ indicates the row and column of the entry and the variable S contains the singular values.

For more information about specifying the s parameter, see the common casouttable parameter (Appendix A: Common Parameters).

saveState={*casouttable*}

specifies the name of the table for storing the TEXTMINE model. This parameter requires a SAS Visual Text Analytics license or a SAS Visual Data Mining and Machine Learning

license.

Long form saveState={name="*table-name*"}

Shortcut form saveState="*table-name*"

caslib="*string*"

specifies the name of the caslib to use.

name="*table-name*"

specifies the name to associate with the table.

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

selectAttribute={*seltag*}

specifies a list of attribute types to be kept or ignored.

opType="IGNORE" | "KEEP"

Specifies what to do with terms with selected tags. KEEP: terms without selected tags will be ignored. IGNORE: terms with selected tags will be ignored.

Default KEEP

* **tagList**={*"string-1"* <, *"string-2"*, ...>}

Specifies a list of tags. Unsupported tags triggers a warning message.

selectEntity={*seltag*}

specifies a list of entity types to be kept or ignored. If this parameter is specified, entities must be set to STD.

opType="IGNORE" | "KEEP"

Specifies what to do with terms with selected tags. KEEP: terms without selected tags will be ignored. IGNORE: terms with selected tags will be ignored.

Default KEEP

* **tagList**={*"string-1"* <, *"string-2"*, ...>}

Specifies a list of tags. Unsupported tags triggers a warning message.

selectPos={*seltag*}

specifies a list of part-of-speech types to be kept or ignored.

opType="IGNORE" | "KEEP"

Specifies what to do with terms with selected tags. KEEP: terms without selected tags will be ignored. IGNORE: terms with selected tags will be ignored.

Default KEEP

* **tagList**={*"string-1"* <, *"string-2"*, ...>}

Specifies a list of tags. Unsupported tags triggers a warning message.

showDroppedTerms=TRUE | FALSE

specifies whether to include terms that have a keep status of N in the OUTTERMS output table.

Default FALSE

startList={*castable*}

specifies the input CAS table that contains the terms that are to be kept for the analysis. If specified, the table must have the Term (varchar) variable. A Role (varchar) variable is optional.

Long form startList={name="*table-name*"}

Shortcut form startList="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

stemming=TRUE | FALSE

specifies whether stemming is to occur in parsing. When set to True, terms are evaluated to see if they belong to common parent form and the information is added to the position table.

Default TRUE

stopList={*castable*}

specifies the input CAS table that contains the terms to exclude from the analysis. If specified, the table must have the Term (varchar) variable. A Role (varchar) variable is optional.

Long form stopList={name="*table-name*"}

Shortcut form stopList="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1*=*any-list-or-data-type-1* <, *key-2*=*any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

synonyms={*castable*}

specifies the input CAS table that contains user-defined synonyms to be used in the analysis. If specified, the table must have the following variables (all varchar): Term, Parent. Termrole and parentrole variables are optional.

Long form synonyms={name="*table-name*"}

Shortcut form synonyms="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in

the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

tagging=TRUE | FALSE

specifies whether part-of-speech tagging is used in parsing.

Default TRUE

target="*variable-name*"

specifies the numeric or character variable that contains a category level on the documents table. This parameter is optional unless you plan to use MI as term weight in accumulation.

termStdMultiple=*double*

Specifies how many standard deviations above the mean to set the term cutoff. This parameter requires a SAS Visual Text Analytics license.

Default 1

Range 0–10

termTopics={*casouttable*}

specifies the name of the output CAS table to contain the term-by-topic sparse matrix information.

For more information about specifying the termTopics parameter, see the common casouttable parameter (Appendix A: Common Parameters).

termWeight="ENTROPY" | "MI" | "NONE"

specifies how terms are weighted. Valid values are Entropy, None and MI (Mutual Information). MI requires target variable in the position table, which is generated by parsing

action.

terms={*casouttable*}

specifies the output CAS table to contain the summary information about the terms in the document collection. This parameter requires a SAS Visual Text Analytics license or a SAS Visual Data Mining and Machine Learning license.

For more information about specifying the terms parameter, see the common casouttable parameter (Appendix A: Common Parameters).

* **text**="*variable-name*"

specifies the character variable in the documents table that contains the text to be processed.

Default "text"

tolerance=*double*

specifies the stopping threshold for the iterative factorization algorithm. If 0 is specified the default value is used.

Default 1e-05

Range 0–1

topicDecision=TRUE | FALSE

Specifies to include topic membership decisions and document cutoffs in the output tables. This parameter requires a SAS Visual Text Analytics license or a SAS Visual Data Mining and Machine Learning license.

Default FALSE

topics={*casouttable*}

specifies the output CAS table to contain the topics that are discovered.

For more information about specifying the topics parameter, see the common casouttable parameter (Appendix A: Common Parameters).

u={*casouttable*}

specifies the U matrix, which contains the left singular vectors. The matrix U is number of terms by k+1.

For more information about specifying the u parameter, see the common casouttable parameter (Appendix A: Common Parameters).

v={*casouttable*}

specifies the transpose of the matrix containing the right singular vectors. The matrix V is number of documents by k+1.

For more information about specifying the v parameter, see the common casouttable parameter (Appendix A: Common Parameters).

wordPro={*casouttable*}

specifies the table to contain the projections of the terms. If k dimensions of the SVD are found and the input data set contains n terms, this table will have n rows and k+1 columns.

For more information about specifying the wordPro parameter, see the common casouttable parameter (Appendix A: Common Parameters).

Text Mining Action Set: Syntax

tmScore Action

CASL Syntax

textMining.tmScore <**result**=*results*> <**status**=*rc*> /

binaryLocation="*string*"

cellWeight="LOG" | "NONE"

***required parameter** **docId**="*variable-name*"

docPro={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

***required parameter** **documents**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*
}, {...}},
computedVarsProgram="*string*",
dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},
groupBy={{
format="*string*",
formattedLength=*integer*,
label="*string*",
*required parameter **name**="*variable-name*",
nfd=*integer*,
nfl=*integer*
}, {...}},
groupByMode="NOSORT" | "REDISTRIBUTE",
importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},
*required parameter **name**="*table-name*",
onDemand=TRUE | FALSE,
orderBy={{
format="*string*",
formattedLength=*integer*,
label="*string*",
*required parameter **name**="*variable-name*",
nfd=*integer*,
nfl=*integer*
}, {...}},
singlePass=TRUE | FALSE,
vars={{
format="*string*",
formattedLength=*integer*,
label="*string*",
*required parameter **name**="*variable-name*",
nfd=*integer*,

```
        nfl=integer
      }, {...}},
      where="where-expression"
}
 legacyNames=TRUE | FALSE
 liti={
      caslib="string",
      computedOnDemand=TRUE | FALSE,
      computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
       *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
      }, {...}},
      computedVarsProgram="string",
      dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
      groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
       *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
      }, {...}},
      groupByMode="NOSORT" | "REDISTRIBUTE",
      importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
        "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
        parameters},
     *required parameter name="table-name",
      onDemand=TRUE | FALSE,
      orderBy={{
```

```
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
multiterm={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
```

```
        formattedLength=integer,
        label="string",
       *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    groupByMode="NOSORT" | "REDISTRIBUTE",
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
  *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    orderBy={{
        format="string",
        formattedLength=integer,
        label="string",
       *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
       *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
nThreads=integer
parent={
```

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

***required parameter** **parseConfig**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

***required parameter** **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

***required parameter** **name**="*variable-name*",

nfd=*integer*,

```
        nfl=integer
      }, {...}},
    groupByMode="NOSORT" | "REDISTRIBUTE",
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    orderBy={{
        format="string",
        formattedLength=integer,
        label="string",
       *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
      }, {...}},
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
       *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
      }, {...}},
    where="where-expression"
}
  termTopics={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
```

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

singlePass=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

```
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
    }
*required parameter terms={
        caslib="string",
        computedOnDemand=TRUE | FALSE,
        computedVars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        computedVarsProgram="string",
        dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
        groupBy={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        groupByMode="NOSORT" | "REDISTRIBUTE",
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
      *required parameter name="table-name",
```

```
        onDemand=TRUE | FALSE,
        orderBy={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        singlePass=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
}
*required parameter text="variable-name"
  topics={
        caslib="string",
        computedOnDemand=TRUE | FALSE,
        computedVars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        computedVarsProgram="string",
```

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

singlePass=TRUE | FALSE,

vars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

where="*where-expression*"

```
}
u={
    caslib="string",
    computedOnDemand=TRUE | FALSE,
    computedVars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    computedVarsProgram="string",
    dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2, ...>},
    groupBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    groupByMode="NOSORT" | "REDISTRIBUTE",
    importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
      parameters},
   *required parameter name="table-name",
    onDemand=TRUE | FALSE,
    orderBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
```

```
        nfd=integer,
        nfl=integer
    }, {...}},
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
  }
  uIndex="variable-name"
;
```

Parameter Descriptions

binaryLocation="*string*"

specifies the location of the language binary files. The location must be valid. If it is not specified, a default installation location is used.

cellWeight="LOG" | "NONE"

specifies how the elements in the term-by-document matrix (the parent output table) are weighted.

* **docId**="*variable-name*"

specifies the character or numeric variable on the documents table that contains the ID of each document.

Default "DOC_ID"

docPro={*casouttable*}

specifies the table to contain the SVD projections of the documents. The matrix used to make the projections is the input table U.

For more information about specifying the docPro parameter, see the common casouttable parameter (Appendix A: Common Parameters).

* **documents**={*castable*}

names the input CAS table of documents to be parsed. You must include a text variable specified with textVar and a document ID variable specified with docIdVar.

Long form documents={name="*table-name*"}

Shortcut form documents="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the

computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

legacyNames=TRUE | FALSE

specifies whether to use the legacy variable names on tables. This parameter requires a SAS Visual Text Analytics license or a SAS Visual Data Mining and Machine Learning license.

Default FALSE

liti={*castable*}

Specifies the input table that contains the custom entity extraction object. This parameter requires a SAS Visual Text Analytics license.

Long form liti={name="*table-name*"}

Shortcut form liti="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

multiterm={*castable*}

Specifies the name of the CAS table that contains a list of multi-word terms and their part-of-speech types. Each multi-word term is parsed as a single token. This parameter requires a SAS Visual Text Analytics license.

Long form multiterm={name="*table-name*"}

Shortcut form multiterm="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

nThreads=*integer*

specifies the number of threads to be used per node. The value must be an integer. When the value is 0, the number of threads equals the number of CPUs.

Default 8

Range 0–64

parent={*casouttable*}

specifies the name of the output CAS table to contain a compressed representation of the sparse term-by-document matrix.

For more information about specifying the parent parameter, see the common casouttable parameter (Appendix A: Common Parameters).

* **parseConfig**={*castable*}

specifies the name of the input CAS table that contains parsing configuration information

Alias scoreConfig

Long form parseConfig={name="*table-name*"}

Shortcut form parseConfig="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that

are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

termTopics={*castable*}

Specifies the input textTopic table that is used for assigning learned topics to new data.

Long form termTopics={name="*table-name*"}

Shortcut form termTopics="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

* **terms**={*castable*}

specifies the name of the input table that contains information about the terms in the document collection. The table is used to determine which terms to use in the topic calculation.

Long form terms={name="*table-name*"}

Shortcut form terms="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-*

parameters}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

* **text**="*variable-name*"

specifies the character variable in the documents table that contains the text to be processed.

Default "text"

topics={*castable*}

Specifies the input topic table that is used for assigning learned topics to new data.

Long form topics={name="*table-name*"}

Shortcut form topics="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

u={*castable*}

specifies the input U matrix, which contains the left singular vectors. The matrix is used to form the document projections.

Long form u={name="*table-name*"}

Shortcut form u="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

uIndex="*variable-name*"

specifies the name of the index variable on the input U matrix.

Default "_TermNum_"

Text Mining Action Set: Syntax

tmSvd Action

CASL Syntax

textMining.tmSvd <**result**=*results*> <**status**=*rc*> /

config={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

```
    orderBy={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    singlePass=TRUE | FALSE,
    vars={{
        format="string",
        formattedLength=integer,
        label="string",
      *required parameter name="variable-name",
        nfd=integer,
        nfl=integer
    }, {...}},
    where="where-expression"
}
count="variable-name"
docId="variable-name"
docPro={
    caslib="string",
    compress=TRUE | FALSE,
    indexVars={"variable-name-1" <, "variable-name-2", ...>},
    label="string",
    maxMemSize=64-bit-integer,
    name="table-name",
    onDemand=TRUE | FALSE,
    promote=TRUE | FALSE,
    replace=TRUE | FALSE,
    replication=integer,
```

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

docStdMultiple=*double*

exactDocPro=TRUE | FALSE

exactWeight=TRUE | FALSE

k=*integer*

legacyNames=TRUE | FALSE

maxK=*integer*

nThreads=*integer*

norm="ALL" | "DOC" | "NONE" | "WORD"

numLabels=*integer*

***required parameter** **parent**={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

```
        }, {...}},
        groupByMode="NOSORT" | "REDISTRIBUTE",
        importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" |
          "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-specific-
          parameters},
      *required parameter name="table-name",
        onDemand=TRUE | FALSE,
        orderBy={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        singlePass=TRUE | FALSE,
        vars={{
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
}
  resolution="HIGH" | "LOW" | "MED"
  rotate="PROMAX" | "VARIMAX"
  rowPivot=double
  s={
        caslib="string",
        compress=TRUE | FALSE,
        indexVars={"variable-name-1" <, "variable-name-2", ...>},
```

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

scoreConfig={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

termId="*variable-name*"

termStdMultiple=*double*

termTopics={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

terms={

caslib="*string*",

computedOnDemand=TRUE | FALSE,

computedVars={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

```
    *required parameter name="table-name",
     onDemand=TRUE | FALSE,
     orderBy={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
     }, {...}},
     singlePass=TRUE | FALSE,
     vars={{
          format="string",
          formattedLength=integer,
          label="string",
        *required parameter name="variable-name",
          nfd=integer,
          nfl=integer
     }, {...}},
     where="where-expression"
}
tolerance=double
topicDecision=TRUE | FALSE
topics={
     caslib="string",
     compress=TRUE | FALSE,
     indexVars={"variable-name-1" <, "variable-name-2", ...>},
     label="string",
     maxMemSize=64-bit-integer,
     name="table-name",
     onDemand=TRUE | FALSE,
     promote=TRUE | FALSE,
```

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

u={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

}

v={

caslib="*string*",

compress=TRUE | FALSE,

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

label="*string*",

maxMemSize=*64-bit-integer*,

name="*table-name*",

onDemand=TRUE | FALSE,

promote=TRUE | FALSE,

replace=TRUE | FALSE,

replication=*integer*,

timeStamp="*string*",

where={*"string-1"* <, *"string-2"*, ...>}

```
    }
    wordPro={
        caslib="string",
        compress=TRUE | FALSE,
        indexVars={"variable-name-1" <, "variable-name-2", ...>},
        label="string",
        maxMemSize=64-bit-integer,
        name="table-name",
        onDemand=TRUE | FALSE,
        promote=TRUE | FALSE,
        replace=TRUE | FALSE,
        replication=integer,
        timeStamp="string",
        where={"string-1" <, "string-2", ...>}
    }
;
```

Parameter Descriptions

config={*castable*}

specifies the name of the input CAS table that contains parsing configuration information

Alias parseConfig

Long form config={name="*table-name*"}

Shortcut form config="*table-name*"

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

count="*variable-name*"

specifies the variable that contains the, possibly weighted, term count. The values in this variable must be numeric. There can be no missing values in this variable.

Default "_COUNT_"

docId="*variable-name*"

specifies the variable that contains the document ID. The type of this variable can either be numeric or a string. There can be no missing values in this variable.

Default "_DOCUMENT_"

docPro={*casouttable*}

specifies the name of the table to contain the SVD projections of the documents.

For more information about specifying the docPro parameter, see the common casouttable parameter (Appendix A: Common Parameters).

docStdMultiple=*double*

Specifies how many standard deviations above the mean to set the document cutoff. This parameter requires a SAS Visual Text Analytics license.

Default 1

Range 0–10

exactDocPro=TRUE | FALSE

Specifies if the exact document projection values should be output. This parameter requires a SAS Visual Text Analytics license.

Default TRUE

exactWeight=TRUE | FALSE

Alias exactWeights

Default FALSE

k=*integer*

specifies the number of dimensions to be extracted (also the number of derived topics). If the input data is too small for the requested number of dimensions, this value is adjusted to

complete the calculation.

Alias numTopics

Range 1–1000

legacyNames=TRUE | FALSE

specifies whether to use the legacy variable names on tables. This parameter requires a SAS Visual Text Analytics license or a SAS Visual Data Mining and Machine Learning license.

Default FALSE

maxK=*integer*

specifies the maximum number of dimensions to be extracted. The maxK option can be used in conjunction with the resolution option to dynamically select the recommended number of dimensions. If you wish to use a specific number of dimensions use maxK and set the resolution to high, or use the k parameter.

Default 10

Range 1–1000

nThreads=*integer*

specifies number of threads to be used per node. If not set, or if a value of 0 is specified, all available threads will be used.

Default 8

Range 0–64

norm="ALL" | "DOC" | "NONE" | "WORD"

indicates whether the document projections, term projections, or both are normalized. The normalization converts the representation from depending on angles between vectors to one based on Euclidean distances between vectors.

Default ALL

numLabels=*integer*

specifies the number of terms to use in the descriptive label for each topic.

Default 5

Range 1–500

* **parent**={*castable*}

specifies the input CAS table that contains the term-by-document matrix in transaction form. The table must have at last three variables, one containing the document id, a second containing the term id, and the third containing the value in the cell corresponding to that particular term and document.

Long form parent={name="*table-name*"}

Shortcut form parent="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix

A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

resolution="HIGH" | "LOW" | "MED"

specifies the desired resolution level for the recommended number of dimensions to be extracted by the SVD.

Default HIGH

rotate="PROMAX" | "VARIMAX"

specifies the type of rotation used to maximize the explanatory power of each topic. A VARIMAX rotation produces uncorrelated topics and a PROMAX rotation produces correlated topics.

Default VARIMAX

rowPivot=*double*

Specifies the row pivot weight for document normalization of the parent table before the SVD. A negative value turns off the rowPivot process. When topics are requested a rowPivot=1 value is used by default. This parameter requires a SAS Visual Text Analytics license.

Default -1

Range -1–1

s={*casouttable*}

specifies the S matrix, which is a diagonal matrix that is output in compressed form, with two variables and k rows. The variable _ID_ indicates the row and column of the entry and the variable S contains the singular values.

For more information about specifying the s parameter, see the common casouttable parameter (Appendix A: Common Parameters).

scoreConfig={*casouttable*}

Specifies the output scoring config file.

For more information about specifying the scoreConfig parameter, see the common casouttable parameter (Appendix A: Common Parameters).

termId="*variable-name*"

specifies the variable that contains the term ID. The contents of this variable must be an integer greater than or equal to 1. There can be no missing values in this variable.

Default "_TERMNUM_"

termStdMultiple=*double*

Specifies how many standard deviations above the mean to set the term cutoff. This parameter requires a SAS Visual Text Analytics license.

Default 1

Range 0–10

termTopics={*casouttable*}

specifies the name of the output CAS table to contain the term-by-topic sparse matrix information.

For more information about specifying the termTopics parameter, see the common casouttable parameter (Appendix A: Common Parameters).

terms={*castable*}

specifies the name of the input table that contains information about the terms in the document collection. The table is used to determine which terms to use in the topic

calculation.

Long form terms={name="*table-name*"}

Shortcut form terms="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

tolerance=*double*

specifies the stopping threshold for the iterative factorization algorithm. If 0 is specified the default value is used.

Default 1e-06

Range 0–1

topicDecision=TRUE | FALSE

Specifies to include topic membership decisions and document cutoffs in the output tables. This parameter requires a SAS Visual Text Analytics license or a SAS Visual Data Mining and Machine Learning license.

Default FALSE

topics={*casouttable*}

specifies the output CAS table to contain the topics that are discovered.

For more information about specifying the topics parameter, see the common casouttable parameter (Appendix A: Common Parameters).

u={*casouttable*}

specifies the U matrix, which contains the left singular vectors. The matrix U is number of terms by k+1.

For more information about specifying the u parameter, see the common casouttable parameter (Appendix A: Common Parameters).

v={*casouttable*}

specifies the transpose of the matrix containing the right singular vectors. The matrix V is number of documents by k+1.

For more information about specifying the v parameter, see the common casouttable parameter (Appendix A: Common Parameters).

wordPro={*casouttable*}

specifies the table to contain the projections of the terms. If k dimensions of the SVD are found and the input data set contains n terms, this table will have n rows and k+1 columns.

For more information about specifying the wordPro parameter, see the common casouttable parameter (Appendix A: Common Parameters).

Details

Working with the TextMining Action Set

About Text Mining

Text mining uses a statistical approach to enable you to learn from text and analyzes an entire document collection as a whole. Most text mining applications fall under one of two categories: exploratory text mining and predictive modeling with text. Exploratory text mining uncovers patterns found in a document collection that can be useful for summarizing or describing the collection. Predictive modeling with text, by contrast, uses a target variable to find the patterns that emerge when the values of the target variable are analyzed against the text.

What Are Some Applications of the TextMining Action Set?

The following examples show how you can apply the **`TextMining`** action set:

- You have a large collection of documents and you are interested in discovering the main themes and concepts in the documents. The output topics table provides a summary of themes and concepts.
- You want to build a predictive model to predict a target variable and you have text associated with most records in your table. The output docPro table contains a numerical representation whose variables can be used as input to a predictive modeling action. The inputs can include other quantitative and qualitative variables.
- You are working on an end-to-end natural language application and need the terms of each text entry parsed, tagged, and returned in the order in which they occur. The position table provides this information.
- You need a technique to compare documents or terms for similarity. The docPro and wordPro tables provide representations of documents or terms that can be interpreted as points in a K-dimensional space. (K is a value that you specify before computing the singular value decomposition, or SVD.) When documents or terms are in this space, those that are near one another are similar and those that are far apart are dissimilar.

What Does the tmMine Action Do?

The **`tmMine`** action is designed to process text and create quantitative representations of the text for use in text mining and other kinds of text analytics. One variable must contain the text and is specified in the **`tmMine`** action during input. Even though the variable might contain text such as a tweet, a few terms, a sentence, or a multipage document, the content in a single variable of a single row is referred to as a document and the entire table as the collection. The **`tmMine`** action combines into a single action functions from the following actions:

- **`tpParse`** provides natural language processing and tokenization of the text in your documents.
- **`tpAccumulate`** provides term counts and weighting summaries for the terms in the collection.

- **tmSvd** finds a representation of your documents and terms and reveals relationships between them by using a matrix factorization called the singular value decomposition (SVD).

The **tmMine** action uses these actions to provide the topic results for your collection and prepare your text data for input into a predictive model.

What Does the tmScore Action Do?

The **tmScore** action enables models that are built using the **tmMine** action to be applied to new data. Several of the output tables of the **tmMine** action become inputs to the **tmScore** action. Then, with new text data, the **tmScore** action assigns each document a representation that is derived from the training data that were used in the corresponding **tmMine** action call.

What Does the tmSvd Action Do?

The **tmSvd** action applies a matrix factorization to the output parent table of the accumulation action. It uses the occurrence data from the entire collection to produce a best-fit, low-dimensional representation that can be used to represent documents and terms as a vector of numbers. Because its functionality is also embedded in the **tmMine** action, the **tmSvd** action is necessary only when you use the **tpParse** and **tpAccumulate** actions from the **textParse** action set or when you output a parent table from the **tmMine** action and later would like to apply the SVD to it.

How Are Topics Derived?

The topics are defined by the weights found on the U matrix of the SVD factorization. Each row of U corresponds to a term and each column to a topic; a high weight in a cell of U means that the term contributes significantly to the corresponding topic. The highest-weighted terms for each topic are listed in the output topics table and, based on the weighting of those terms, the documents are also given a topic membership value for each topic. This value is found in the output document projection table.

How Are Document Normalizations Used?

Two types of document normalizations can be used in the **tmMine** and **tmSVD** actions:

- Standard document unit normalization of the document projection vectors is controlled by the **norm** parameter and defaults to the value "ALL" when document projections are requested but topic results are not requested. This normalization works particularly well when the document vectors are to be used in a predictive model or clustering because documents can be compared by Euclidean distance rather than by angle measurements.
- Row-pivot normalization is the default when discovering topics because it dampens the tendency of long documents to belong to too many topics. This approach differs from the **norm** unit normalization technique because it normalizes the contents of the parent table and is applied before the SVD factorization rather than after.

How Are Term and Document Cutoffs Calculated?

Term and document cutoffs are used to determine when a term or document belongs to a particular topic. A term cutoff and a document cutoff are calculated for each topic that has been discovered. For a particular topic, if the term's weight is greater than or equal to the cutoff, the term is included in the topic. Similarly, if a document's projection value for a particular topic is greater than or equal to the document cutoff, the document is assigned to the topic.

A term cutoff, c_i, for topic i is derived as follows from the set of weights w_i that is found in the termTopics or U table,

$$c_i = \bar{w}_i + \text{termStdMultiple} \times sd_{w_i}$$

where termStdMultiple defaults to 1. The mean, $\bar{w}_i$, and standard deviation, sd_{w_i}, are taken of the absolute value of the term weights for topic i.

A corresponding calculation is done for the document cutoffs.

Examples

Example 22.1: Derive Topics from a Document Collection

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

This example demonstrates how to derive topics from a collection by using the **tmMine** action. The following DATA step creates an input data table named mycas.reviews. This step assumes that the CAS engine libref is named mycas, but you can substitute any appropriately named CAS engine libref.

```
data mycas.reviews;
infile datalines delimiter='|' missover;
length text $300 category $20;
input text$ positive category$ did;
datalines;
 This is the greatest phone ever! love it! It can replace my tv! |1|electronics|1
 The phone's battery life is too short and screen resolution is low.|0|electronics|2
 The screen resolution is low, but I love this tv. Good viewing.|1|electronics|3
 The movie itself is great and I liked watching it. Good acting! |1|movies|4
 The movie's story is boring and the acting is poor.|0|movies|5
 I watched this movie but it was boring. |0|movies|6
 The book has a terrific plot!|1|books |7
 The book's plot was suspenseful. Good read.|1|books|8
```

```
    I love the author, but this book is a waste of time to read.|0|books|9
;
run;
```

The following PROC CAS step loads the included stop list into the active caslib for use:

```
  proc cas;
      loadtable caslib="ReferenceData" path="en_stoplist.sashdat";
  run;
quit;
```

The following PROC CAS step calls the **tmMine** action, which parses the text, applies the stop list, calculates the SVD factorization, and discovers topics all in a single action. When the action completes, Output 22.1.1 through Output 22.1.3 are printed.

```
proc cas;
    loadactionset "textMining";
    action tmMine;
    param
    docId="did"
    documents={ name="reviews"}
    text="text"
    nounGroups= False
    tagging = True
    stemming= True
    stopList ={ name="en_stoplist"}
    parseConfig={name="config", replace=TRUE}
    parent ={ name="parent",replace=TRUE}
    offset ={name="offset",replace=TRUE}
    terms ={ name="terms", replace=TRUE}
    reduce=2
    k=3
    docPro ={ name="docpro", replace=TRUE}
    topics ={ name="topics", replace=TRUE}
    u ={ name="svdu", replace=TRUE}
    numLabels=3
topicDecision=True
;
action table.fetch /table="topics", orderBy="_TopicID_"; run;
action table.fetch /table="docpro", orderBy="did"; run;
action table.fetch /table="svdu", orderBy="_TermNum_"; run;
run;
quit;
```

Output 22.1.1 displays the contents of the mycas.topics table, which contains the discovered topics..

Output 22.1.1 Discovered Topics

Results from table.fetch

Selected Rows from Table TOPICS

Index	Topic ID	Topic	Category	Number of Terms	Number of Docs	Term Cutoff	Document Cutoff
1	1	book, plot, read	Mult	3	2	0.381	0.65
2	2	movie, +bore, +watch	Mult	4	3	0.385	0.632
3	3	tv, phone, resolution	Mult	4	3	0.385	0.524

Output 22.1.2 displays the contents of the mycas.docpro table, which contains the relationship of each document to each topic.

Output 22.1.2 Document Projections

Results from table.fetch

Selected Rows from Table DOCPRO

Index	did	Score for book, plot, read	Score for movie, +bore, +watch	Score for tv, phone, resolution	book, plot, read	movie, +bore, +watch	tv, phone, resolution
1	1	0	0	0.5798821773	0	0	1
2	2	0	0	0.6905117599	0	0	1
3	3	0	0	0.6110259518	0	0	1
4	4	0	0.6825195	0	0	1	0
5	5	0	0.7457203512	0	0	1	0
6	6	0	0.8399661992	0	0	1	0
7	7	0.8185079558	0	0	1	0	0
8	8	0.8608494806	0	0	1	0	0
9	9	0.6458854573	0	0	0	0	0

Output 22.1.3 displays the contents of the mycas.svdu table, which contains the topic score of each term to each topic in the training data.

Output 22.1.3 Relationship between Topics and Terms

Results from table.fetch

Selected Rows from Table SVDU

Index	_TermNum_	_Col1_	_Col2_	_Col3_
1	1	-0.030316754	0.2151383406	0.2264079152
2	2	-0.023548807	0.0108277192	0.4187120658
3	3	0.0005739315	0.5119842534	-0.023157874
4	4	-0.023118229	-0.024552378	0.4631866963
5	5	0.4942887978	-0.017434609	0.0361238772
6	6	0.006516217	0.4386106117	0.0034250391
7	7	0.1556906713	-0.036380438	0.378769758
8	8	-0.012714918	-0.02769837	0.4603405044
9	9	-0.019071159	0.4704832817	-0.019501009
10	10	0.5951267514	-0.021917194	-0.019394378
11	11	0.0141264472	0.4927909226	-0.047333542
12	12	0.5785006335	-0.004751182	-0.081927976
13	13	-0.023548807	0.0108277192	0.4187120658
14	14	0.1981660672	0.1768502227	0.1289030697

Example 22.2: Assign Discovered Topics to New Documents

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level

name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

```
    proc cas;
        loadtable caslib="ReferenceData" path="en_stoplist.sashdat";
    run;
quit;
```

The **tmScore** action enables you to apply a topic model to new data. The following DATA step creates some text content to be scored:

```
data mycas.scoreReviews;
infile datalines delimiter='|' missover;
   length text $300;
   input text$ did;
   datalines;
Loved the movie! I watched it twice.|1
I liked the book. Great read! Good Plot!|2
;
run;
```

The **tmScore** action enables you to assign topic scores to the new documents. The following PROC CAS step calls the **tmScore** action and uses the config table, U table, and Terms table from the previous example as input:

```
proc cas;
loadactionset "textMining";
action tmScore;
param
   docId="did"
   documents={name="scoreReviews"}
   text="text"
   terms={name="terms"}
   parseConfig={name="config"}
   u={name="svdu"}
   docPro ={ name="scoreDocpro", replace=TRUE}
   topics={name="topics"}
   topicDecision=True
;
action table.fetch /table="scoreDocpro"; run;
run;
quit;
```

Output 22.2.1 displays the contents of the mycas.scoreDocpro table, which contains the results. Each document is assigned a score that indicates how strongly it encapsulates each of the three topics that were discovered in the first example.

Output 22.2.1 Scored Documents

Results from table.fetch

Selected Rows from Table SCOREDOCPRO							
Index	did	_Col1_	_Col2_	_Col3_	_TextTopic_1	_TextTopic_2	_TextTopic_3
1	1	0	0.6815315604	0	0	1	0
2	2	0.8020158503	0	0	1	0	0

Example 22.3: Apply the SVD and Topic Discovery after Parsing

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*, and *SAS Viya: System Programming Guide*.

Finally, the **tmSvd** action can be called separately if you decide to derive topics after you have initially parsed them. The following PROC CAS step calls the **tmSvd** action and uses the Parent and Terms table output from the **tmMine** action to discover the topics.

```
    proc cas;
        loadtable caslib="ReferenceData" path="en_stoplist.sashdat";
    run;
quit;

 proc cas;
    loadactionset "textMining";
    action tmSvd;
    param
      parent={ name="parent"}
      terms={name="terms"}
      k=3
      u ={ name="svdu", replace=TRUE}
      numLabels=3
      topics={name="topicsSVD",replace=TRUE}
;
action table.fetch /table="topicsSVD"; run;
run;
quit;
```

The singular value decomposition and topic discovery can be computed alone after parsing has been done. Output 22.3.1 displays the contents of the mycas.topicsvd table, which contain the results of the discovered topics.

Output 22.3.1 Discovered Topics

Results from table.fetch

Selected Rows from Table TOPICSSVD

Index	Topic ID	Topic	Term Cutoff
1	1	book, plot, read	0.381
2	2	movie, +bore, +watch	0.385
3	3	tv, phone, resolution	0.385

Chapter 23
t-Distributed Stochastic Neighbor Embedding Action Set

Contents

Syntax **1049**
The tSne Action 1050
Details **1058**
Displayed Output 1059
Example **1059**
Example 23.1: Computing a Two-Dimensional Embedding of Input Data 1059
References **1060**

t-Distributed Stochastic Neighbor Embedding Action Set: Syntax

tSne Action

CASL Syntax

tSne.tSne <**result**=*results*> <**status**=*rc*> /

attributes={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

display={

caseSensitive=TRUE | FALSE,

exclude=TRUE | FALSE,

excludeAll=TRUE | FALSE,

keyIsPath=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>},

pathType="LABEL" | "NAME",

traceNames=TRUE | FALSE

}

inputs={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

learningRate=*double*

maxIters=*integer*

```
 nDimensions=integer
 output={
    *required parameter casOut={
          caslib="string"
          compress=TRUE | FALSE
          indexVars={"variable-name-1" <, "variable-name-2", ...>}
          label="string"
          maxMemSize=64-bit-integer
          name="table-name"
          onDemand=TRUE | FALSE
          promote=TRUE | FALSE
          replace=TRUE | FALSE
          replication=integer
          timeStamp="string"
          where={"string-1" <, "string-2", ...>}
    },
      copyVars="ALL" | "ALL_MODEL" | "ALL_NUMERIC" | {"variable-name-1" <, "variable-
        name-2", ...>}
 }
 outputTables={
      groupByVarsRaw=TRUE | FALSE,
      includeAll=TRUE | FALSE,
      names={"string-1" <, "string-2", ...>} | {key-1={casouttable-1} <, key-2={casouttable-2},
        ...>},
      repeated=TRUE | FALSE,
      replace=TRUE | FALSE
 }
 perplexity=double
 seed=double
*required parameter table={
      caslib="string",
      computedOnDemand=TRUE | FALSE,
      computedVars={{
```

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

computedVarsProgram="*string*",

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>},

groupBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

groupByMode="NOSORT" | "REDISTRIBUTE",

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*},

*required parameter **name**="*table-name*",

onDemand=TRUE | FALSE,

orderBy={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}},

singlePass=TRUE | FALSE,

vars={{

```
            format="string",
            formattedLength=integer,
            label="string",
          *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}},
        where="where-expression"
    }
;
```

Parameter Descriptions

attributes={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variable attributes.

For more information about specifying the attributes parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

display={*displayTables*}

specifies a list of results tables to send to the client for display.

For more information about specifying the display parameter, see the common displayTables parameter (Appendix A: Common Parameters).

inputs={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the analysis.

For more information about specifying the inputs parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

learningRate=*double*

specifies the learning rate to use for the optimization.

maxIters=*integer*

specifies the maximum number of iterations.

nDimensions=*integer*

specifies the number of embedding dimensions to estimate.

Alias nDims

Default 2

Range 2–3

output={*outputStatement*}

specifies the output data table in which to save the computed embedding.

For more information about specifying the output parameter, see the common outputStatement parameter (Appendix A: Common Parameters).

outputTables={*outputTables*}

lists the names of results tables to save as CAS tables on the server.

For more information about specifying the outputTables parameter, see the common outputTables parameter (Appendix A: Common Parameters).

perplexity=*double*

specifies the perplexity to use for the embedding computation.

Default 30

seed=*double*

specifies the seed value for random number generation.

Default 0

* **table**={*castable*}

specifies the settings for an input table.

Long form table={name="*table-name*"}

Shortcut form table="*table-name*"

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Alias compOnDemand

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

groupBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the variables to use for grouping results.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

groupByMode="NOSORT" | "REDISTRIBUTE"

specifies how to create groups.

Default NOSORT

NOSORT

group the data without sorting on each machine, and then group the data again on the controller.

REDISTRIBUTE

transfer rows between nodes to guarantee ordering within groups. This method is slower.

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

orderBy={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

For more information about specifying the orderBy parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

For more information about specifying the vars parameter, see the common casinvardesc parameter (Appendix A: Common Parameters).

where="*where-expression*"

specifies an expression for subsetting the input data.

Details

The **tSne** action implements the *t*-distributed stochastic neighbor embedding (*t*-SNE) method. The *t*-SNE method is useful for dimensionality reduction; visualization of high-dimensional data; feature engineering; and preprocessing for subsequent clustering or supervised modeling. Its applications include text mining, image processing, and bioinformatics. The **tSne** action computes a low-dimensional representation, also called an embedding, of high-dimensional data into two or three dimensions. Unlike other dimension reduction methods, such as principal component analysis, the *t*-SNE method is appropriate for nonlinear data and emphasizes existing groupings in the data.

The method is named *t*-SNE because it models the pairwise distances in low dimensions according to Student's *t*-distribution. The *t*-distribution with one degree of freedom has heavier tails than the Gaussian distribution, which means that it assigns higher probability values to large distances. This allows *t*-SNE to relax pairwise distances for non-neighboring observations, whereas distances between closely neighboring observations are more exactly preserved. This behavior is desirable because it mitigates the crowding problem in high-dimensional data representation and makes existing groups in the data visually evident.

The **tSne** action operates on an input data table. For each observation in the input data table, the action returns either two or three computed columns that contain the embedding coordinates. It computes the embedding by minimizing the Kullback-Leibler divergence between the joint probabilities in high dimensions and the joint probabilities in low dimensions, and it stores the embedding in the output data table that is specified by the **output** parameter.

The method estimates the embeddings y_i in two or three dimensions by minimizing the Kullback-Leibler loss, which is defined as

$$C = \mathrm{KL}(P \| Q) = \sum_{i=1}^{N} \sum_{j=1}^{N} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

where N is the number of observations and p_{ij} is the similarity between observations i and j in the data table. It is defined as

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N}$$

where $p_{i|j}$ and $p_{j|i}$ are conditional probabilities and q_{ij} is the estimated similarity in low-dimensional space, which follows Student's *t*-distribution with one degree of freedom and is given by

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}$$

The conditional probability $p_{i|j}$ follows the expression

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2)}{2\sigma_i^2}$$

where σ_i is a bandwidth parameter that varies for each observation x_i. The **tSne** action sets σ_i in such a way that the perplexity

$$\mathrm{Perp}_i = 2^{-\sum_j p_{j|i} \log_2 p_{j|i}}$$

matches the fixed **perplexity** parameter value that the user specifies. In general, lower perplexity yields embeddings that have smaller, more separated groupings.

The **tSne** action estimates the values of y_i by minimizing C via gradient descent. This is an iterative optimization process that the user controls by specifying the **maxIters** and **learningRate** parameters.

Displayed Output

The **tSne** action displays various tables for analysis. The following sections describe the output tables in the order of their appearance when the related options are specified.

Number of Observations

The "Number of Observations" table displays the number of observations that are read from the input data table and the number that are used.

Model Information

The "Model Information" table displays basic information about the parameters that are used in the analysis. This information includes the learning rate, maximum number of iterations, perplexity, and seed value for random number generation.

Iteration History

The "Iteration History" table displays the iteration number and the loss.

Output CAS Tables

The **tSne** action always produces a table that lists the CAS library, the data table name, and the number of rows and columns in each output data table. When you specify the **output** parameter, the **tSne** action also produces the output data table that displays the embedding dimensions of each observation from the input data table.

Example

Example 23.1: Computing a Two-Dimensional Embedding of Input Data

This section contains PROC CAS code.

NOTE: Input data must be in a CAS table that is accessible in your CAS session. This table has a two-level name; the first level is your CAS engine libref and the second level is the table name. You refer to this table in the CAS procedure by specifying only the second level. For more information about two-level names, see Chapter 3, "Shared Concepts" (*SAS Visual Data Mining and Machine Learning: Procedures Guide*). For more information about PROC CAS and programming in CASL, see *Getting Started with CASL*, *SAS Cloud*

Analytic Services: CAS Procedure Programming Guide and Reference, and *SAS Viya: System Programming Guide*.

This example shows how to use the **tSne** action to compute a two-dimensional embedding of observations in an input data table. The example uses the Iris data from Fisher (1936), which contain morphological measurements of 50 specimens from each of three different species of iris flowers: *Iris setosa*, *I. versicolor*, and *I. virginica*. Mezzich and Solomon (1980) discuss a variety of cluster analyses that use the Iris data. The **tSne** action returns a two-dimensional representation of each observation. The analysis uses four variables: SepalLength, SepalWidth, PetalLength, and PetalWidth. The remaining variables in the data table are not used.

You can load the Iris data into your CAS session by using the following DATA step. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.iris;
set sashelp.iris;
id=_n_;
run;
```

The following statements run the **tSne** action and create a two-dimensional embedding of the Iris data:

```
proc cas;
  loadactionset "tSne";
  action tSne result=R / table={name="iris"},
      inputs        = {"SepalLength", "SepalWidth", "PetalLength", "PetalWidth"},
      nDimensions   = 2,
      perplexity    = 5,
      learningRate  = 100,
      maxIters      = 500,
      output        = {casOut={name="tsne_out", replace="TRUE"},
                       copyvars={"id", "species"}};
run;
```

The **table** parameter names the input data table. The **inputs** parameter specifies the input variables to be used. The **nDimensions** parameter requests that the model return two embedding dimensions. The **perplexity** parameter specifies 5 as the value of the perplexity. The **learningRate** parameter specifies 100 as the learning rate for the optimization. The **maxIters** parameter specifies 500 as the maximum number of iterations. The **output** parameter outputs the embedding to the tsne_out data table and copies the ID and Species variables to the output.

References

Fisher, R. A. (1936). "The Use of Multiple Measurements in Taxonomic Problems." *Annals of Eugenics* 7:179–188.

Mezzich, J. E., and Solomon, H. (1980). *Taxonomy and Behavioral Science*. New York: Academic Press.

Appendix: Common Parameters

Common Parameters 1
aircodegen 2
biomedimageTableIn 6
casinvardesc 18
casOptml 19
casouttable 27
codegen 29
displayTables 32
importOptions 34
linksVarT 40
multimember 41
outputStatement 43
outputTables 46
polynomial 49

Common Parameter: aircodegen

CASL Syntax

aircodegen={
casOut={
caslib="*string*"
compress=TRUE | FALSE
indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}
label="*string*"
maxMemSize=*64-bit-integer*
name="*table-name*"
onDemand=TRUE | FALSE
promote=TRUE | FALSE
replace=TRUE | FALSE
replication=*integer*
timeStamp="*string*"
where={*"string-1"* <, *"string-2"*, ...>}
},
comment=TRUE | FALSE,
fmtWdth=*integer*,
iProb=TRUE | FALSE,
indentSize=*integer*,
intoCutPt=*double*,
labelId=*integer*,
lineSize=*integer*,
noTrim=TRUE | FALSE,
pCatAll=TRUE | FALSE,
tabForm=TRUE | FALSE
}

Parameter Descriptions

casOut={*casouttable*}

specifies the settings for an output table.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

comment=TRUE | FALSE

when set to True, adds comments to the DATA step code.

Default FALSE

fmtWdth=*integer*

specifies the width to use for formatting derived numbers such as parameter estimates in the DATA step code.

Default 20

Range 0–32

iProb=TRUE | FALSE

specifies whether individual probabilities should be generated.

Default FALSE

indentSize=*integer*

specifies the number of spaces to indent the DATA step code for each indent level.

Default 3

Range 0–10

intoCutPt=*double*

specifies the cutoff point for the INTO column.

Default 0.5

Range 0–1

labelId=*integer*

specifies the label ID to use in array names and statement labels in the DATA step code. By default, a random positive integer is used.

lineSize=*integer*

specifies the line size for the generated code.

Default 120

Range 64–254

noTrim=TRUE | FALSE

requests that the comparison of variables with formatted values be based on the full format width, with padding. By default, leading and trailing blanks are removed from the formatted values.

Default FALSE

pCatAll=TRUE | FALSE

when set to True, generates the probabilities for all levels of the response.

Default FALSE

tabForm=TRUE | FALSE

when set to True, generates the code in way that is appropriate for storing in a table.

Default FALSE

Common Parameter: biomedimageTableIn

CASL Syntax

```
*required parameter biomedimageTableIn={
    bioMedDimension="variable-name",
    bioMedId="variable-name",
    dimension="variable-name",
    id="variable-name",
    image="variable-name",
    imageFormat="variable-name",
    label="variable-name",
    orientation="variable-name",
    path="variable-name",
    position="variable-name",
    resolution="variable-name",
    size="variable-name",
    sliceIndex="variable-name",
    spacing="variable-name",
   *required parameter table={
        caslib="string"
        computedOnDemand=TRUE | FALSE
        computedVars={{
            format="string",
            formattedLength=integer,
            label="string",
           *required parameter name="variable-name",
            nfd=integer,
            nfl=integer
        }, {...}}
        computedVarsProgram="string"
        dataSourceOptions={key-1=any-list-or-data-type-1 <, key-2=any-list-or-data-type-2,
```

```
            ...>}
          importOptions={fileType="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" |
            "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", fileType-
            specific-parameters}
        *required parameter name="table-name"
          onDemand=TRUE | FALSE
          singlePass=TRUE | FALSE
          vars={{
              format="string",
              formattedLength=integer,
              label="string",
            *required parameter name="variable-name",
              nfd=integer,
              nfl=integer
          }, {...}}
          where="where-expression"
      },
       type="variable-name"
  }
```

Parameter Descriptions

bioMedDimension="*variable-name*"

specifies the name of the image table column containing biomedical image dimensions.

Default "IMCORE_BIOMEDDIM_COL_NAME"

bioMedId="*variable-name*"

specifies the name of the image table column containing biomedical image identifiers.

Default "IMCORE_BIOMEDID_COL_NAME"

dimension="*variable-name*"

specifies the name of the image table column containing dimensions of images.

Default "_dimension_"

id="*variable-name*"

specifies the name of the variable in the input table that identifies each image.

Default "_id_"

image="*variable-name*"

specifies the name of the image table column containing image binaries.

Default "_image_"

imageFormat="*variable-name*"

specifies the name of the image table column containing formats of image binaries.

Default "_imageFormat_"

label="*variable-name*"

specifies the name of the column that contains labels of images.

Default "_label_"

orientation="*variable-name*"

specifies the name of the image table column containing image orientations.

Default "_orientation_"

path="*variable-name*"

specifies the name of the image table column containing file paths.

Default "_path_"

position="*variable-name*"

specifies the name of the image table column containing image positions.

Default "_position_"

resolution="*variable-name*"

specifies the name of the image table column containing resolutions of images.

Default "_resolution_"

size="*variable-name*"

specifies the name of the image table column containing byte lengths of image binaries.

Default "_size_"

sliceIndex="*variable-name*"

specifies the name of the image table column containing slice indices.

Default "IMCORE_SLICEINDEX_COL_NAME"

spacing="*variable-name*"

specifies the name of the image table column containing image spacings.

Default "_spacing_"

* **table**={*castable*}

The castable value can be one or more of the following:

caslib="*string*"

specifies the caslib that contains the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.

computedOnDemand=TRUE | FALSE

when set to True, creates the computed variables when the table is loaded instead of when the action begins.

Default FALSE

computedVars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

Alias compVars

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

computedVarsProgram="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

Alias compPgm

dataSourceOptions={*key-1=any-list-or-data-type-1* <, *key-2=any-list-or-data-type-2*, ...>}

specifies data source options.

Alias options, dataSource

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

specifies the settings for reading a table from a data source.

Alias import

The value that you specify for fileType determines the other parameters that apply. For more information about this common parameter, see importOptions (Appendix A: Common Parameters).

* **name**="*table-name*"

specifies the name of the table to use.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

singlePass=TRUE | FALSE

when set to True, does not create a transient table on the server. Setting this parameter to True can be efficient, but the data might not have stable ordering upon repeated runs.

Default FALSE

vars={{*casinvardesc-1*} <, {*casinvardesc-2*}, ...>}

specifies the variables to use in the action.

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

where="*where-expression*"

specifies an expression for subsetting the input data.

type="*variable-name*"

specifies the name of the image table column containing types (jpg, bmp, etc.) of images.

Default "_type_"

Parameters for fileType="AUTO"

No parameters apply when you specify AUTO

Parameters for fileType="BASESAS"

charMultiplier=*double*

specifies a multiplier value to expand fixed-width character variables that might require transcoding. The lengths are increased to avoid character data truncation. The lengths are increased by multiplying the length by the specified value.

Default 1

Range 1–5

dataTransferMode="AUTO" | "PARALLEL" | "SERIAL"

specifies how data is transferred between the data source and SAS Cloud Analytic Services.

Default AUTO

encryptionPassword="*string*" | *binary-large-object*

specifies a password for encrypting or decrypting stored data.

password="*string*"

specifies the password for a password-protected data set. Use this parameter if the data set is password-protected or uses SAS proprietary encryption.

read="*string*"

specifies the Read password for the SAS data set.

varcharConversion=*double*

specifies the number of bytes to use as a threshold. When a column with the CHAR data type has a length that is equal to or exceeds the specified number of bytes, the VARCHAR data type is used.

Default 0

Range 1–32767

Parameters for fileType="CSV"

allowTruncation=TRUE | FALSE

specifies that truncation of character data that is too long for a given field is allowed.

Default TRUE

delimiter="*string*"

specifies the character to use as the field delimiter.

Default ","

encoding="*string*"

specifies the text encoding of the file. If you do not specify this parameter, then the session encoding is used.

Default "utf-8"

getNames=TRUE | FALSE

when set to True, the values in the first line of the file are used as variable names.

Default TRUE

guessRows=*64-bit-integer*

specifies the number of rows to scan in order to determine data types for variables.

Default 20

Minimum value 1

locale="*string*"

specifies the locale for interpreting data in the file

nThreads=*integer*

specifies the number of threads to use on each machine in the server. By default, the server uses one thread for each CPU that is licensed to use SAS software.

Default 0

Minimum value 0

stripBlanks=TRUE | FALSE

removes leading and trailing blanks from character variables.

Default FALSE

varChars=TRUE | FALSE

when set to True, variable-length strings are used for character variables.

Default TRUE

vars={{*casvarcsvdesc-1*} <, {*casvarcsvdesc-2*}, ...>}

specifies the names, types, formats, and other metadata for variables.

The casvarcsvdesc value can be one or more of the following:

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

informat="*string*"

specifies the informat to apply to the variable.

label="*string*"

specifies the descriptive label for the variable.

length=*integer*

specifies the unformatted length of the variable. This parameter applies to fixed-length character variables (type="CHAR") only.

Default 8

Minimum value 1

name="*string*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

type="BINARY" | "CHAR" | "DATE" | "DATETIME" | "DECQUAD" | "DECSEXT" | "DOUBLE" | "INT32" | "INT64" | "TIME" | "VARBINARY" | "VARCHAR"

specifies the data type for the variable.

Default DOUBLE

Parameters for fileType="DOCUMENT"

fileExtList={"CSV", "DOC", "DOCX", "EPUB", "HTML", "KEY", "MBOX", "MSG", "NUMBERS", "ODP", "ODS", "ODT", "PAGES", "PDF", "PPT", "PPTX", "PST", "RTF", "TNEF", "TXT", "WPD", "XLS", "XLSX", "XML"}

specifies the file extensions for subsetting document selection.

recurse=TRUE | FALSE

recursively import documents when the caslib definition permits access to subdirectories.

Default TRUE

tikaConv=TRUE | FALSE

specifies to convert the documents with the Apache Tika document conversion library.

Default FALSE

Parameters for fileType="DTA"

No parameters apply when you specify DTA

Parameters for fileType="ESP"

maxTime=*integer*

specifies the number of seconds to receive data from SAS Event Stream Processing before declaring an end of file on the stream. The data is read until an end of file is found, and then the action stops running.

Default 5

sasTypes=TRUE | FALSE

when set to True, the table is created with character and float data types only. When set to False, the table uses the same data types that are used in the ESP event.

Default FALSE

Parameters for fileType="EXCEL"

getNames=TRUE | FALSE

when set to True, the values in the first line of the file are used as variable names.

Default TRUE

range="*string*"

specifies a subset of the cells to import. For example, the range B2..E8 is the range address for a rectangular block of 12 cells, where the top left cell is B2 and the bottom right cell is E8.

sheet="*string*"

specifies the name of the worksheet to import.

Parameters for fileType="FMT"

name="*string*"

specifies a file system path and filename. The file must be a SAS item store that includes the formats to import.

Parameters for fileType="HDAT"

encryptionPassword="*string*" | *binary-large-object*

specifies a password for encrypting or decrypting stored data.

Parameters for fileType="JMP"

No parameters apply when you specify JMP

Parameters for fileType="LASR"

compPgm="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

compVars={*"string-1"* <, *"string-2"*, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

nThreads=*integer*

specifies the number of threads to use on each machine in the server. By default, the server uses one thread for each CPU that is licensed to use SAS software.

Default 0

Minimum value 0

parallelMode="FALLBACK" | "FORCE" | "NONE"

specifies how the table is transferred from SAS LASR Analytic Server to SAS Cloud Analytic Services.

Default FALLBACK

FALLBACK

specifies to attempt a parallel load. If the parallel load does not succeed, then a serial load is performed.

FORCE

specifies to perform a parallel load. If the parallel load cannot be performed, such as for network connectivity issues, then the load fails.

NONE

specifies to perform a serial load.

preserveOrder=TRUE | FALSE

when set to True, the rows are inserted into the new table in the same order as they are received from the SAS LASR Analytic Server. Creating the table is less efficient when this parameter is used.

Default FALSE

varchars=TRUE | FALSE

when set to True, variable-length strings are used for character variables.

Default FALSE

vars={*"string-1"* <, *"string-2"*, ...>}

specifies the variables to use in the action.

where="*where-expression*"

specifies an expression for subsetting the input data.

Parameters for fileType="SPSS"

No parameters apply when you specify SPSS

Parameters for fileType="XLS"

getNames=TRUE | FALSE

when set to True, the values in the first line of the file are used as variable names.

Default TRUE

range="*string*"

specifies a subset of the cells to import. For example, the range B2..E8 is the range address for a rectangular block of 12 cells, where the top left cell is B2 and the bottom right cell is E8.

sheet="*string*"

specifies the name of the worksheet to import.

Common Parameter: casinvardesc

CASL Syntax

casinvardesc={{

format="*string*",

formattedLength=*integer*,

label="*string*",

*required parameter **name**="*variable-name*",

nfd=*integer*,

nfl=*integer*

}, {...}}

Parameter Descriptions

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

label="*string*"

specifies the descriptive label for the variable.

* **name**="*variable-name*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

Common Parameter: casOptml

CASL Syntax

casOptml={

algorithm="HF" | "LBFGS" | "SGD",

lbfgsOpt={

lineSearchMethod="ARMIJO" | "HYBRID" | "MORETHUENTE" | "STRONGWOLFE" | "WOLFE"

numCorrections=*64-bit-integer*

},

optmlOpt={

clipWeightMaxNorm=*double*

fConv=*double*

fConvWindow=*64-bit-integer*

gTol=*double*

maxEvals=*64-bit-integer*

maxIters=*64-bit-integer*

maxTime=*double*

regL1=*double*

regL2=*double*

},

printOpt={

logLevel=*64-bit-integer*

printFreq=*64-bit-integer*

printLevel="PRINTBASIC" | "PRINTDETAIL" | "PRINTNONE"

},

sgdOpt={

adaptiveRate=TRUE | FALSE

annealingRate=*double*

commFreq=*64-bit-integer*

learningRate=*double*

miniBatchSize=*64-bit-integer*

```
        momentum=double
        seed=64-bit-integer
        useLocking=TRUE | FALSE
    },
    state={
        checkpointFreq=64-bit-integer
        saveBest=TRUE | FALSE
        table={
            caslib="string"
            compress=TRUE | FALSE
            indexVars={"variable-name-1" <, "variable-name-2", ...>}
            label="string"
            maxMemSize=64-bit-integer
            name="table-name"
            onDemand=TRUE | FALSE
            promote=TRUE | FALSE
            replace=TRUE | FALSE
            replication=integer
            timeStamp="string"
            where={"string-1" <, "string-2", ...>}
        }
    },
    validate={
        frequency=64-bit-integer
        goal=double
        stagnation=64-bit-integer
    }
}
```

Parameter Descriptions

algorithm="HF" | "LBFGS" | "SGD"

Specifies the optimization solver to use, the default solver is LBFGS.

Default LBFGS

HF

HF solver. (The HF option is experimental in this release.)

LBFGS

Limited memory LBFGS solver.

SGD

SGD solver.

lbfgsOpt={*lbfgsOptions*}

LBFGS solver option set.

The lbfgsOptions value can be one or more of the following:

lineSearchMethod="ARMIJO" | "HYBRID" | "MORETHUENTE" | "STRONGWOLFE" | "WOLFE"

Specifies Line-search algorithms used in LBFGS solver.

Default MORETHUENTE

ARMIJO

Armijo backtracking line search method.

HYBRID

Hybrid line search method.

MORETHUENTE

More and Thuente line search method.

STRONGWOLFE

Line search method with strong wolfe condition.

WOLFE

Line search method with wolfe condition.

numCorrections=*64-bit-integer*

Specifies the number of corrections used in the LBFGS update.

Default 20

Minimum value 1

optmlOpt={*optmlOptions*}

Common option set.

The optmlOptions value can be one or more of the following:

clipWeightMaxNorm=*double*

Specifies the value to rescale the weights when the L2-norm of the weights exceeds the value.

Default 0

Minimum value 0

fConv=*double*

Specifies a stopping value when the objective fails to change more than that value in at least fConvWindow iterations for LBFGS solver.

Default 1e-05

Minimum value 0

fConvWindow=*64-bit-integer*

Specifies an iteration window for application of fConv convergence criteria used by LBFGS solver.

Default 1

Minimum value 1

gTol=*double*

Specifies the stopping tolerance for the first-order optimality error.

Default 1e-05

Minimum value 0

maxEvals=*64-bit-integer*

Specifies the maximum number of function evaluations allowed for a single optimization/training.

Default 0

Minimum value 0

maxIters=*64-bit-integer*

Specifies the maximum iterations allowed for a single optimization/training.

Default 10

Minimum value 0

maxTime=*double*

Specifies the maximum time (in seconds) allowed for a single optimization/training.

Default 0

Minimum value 0

regL1=*double*

Specifies the L1 regularization parameter, the value must be nonnegative.

Default 0

Minimum value 0

regL2=*double*

Specifies the L2 regularization parameter, the value must be nonnegative.

Default 0

Minimum value 0

printOpt={*optmlPrintOptions*}

Print log and display output option set.

The optmlPrintOptions value can be one or more of the following:

logLevel=*64-bit-integer*

Specifies output display level.

Default 0

Minimum value 0

printFreq=*64-bit-integer*

Specifies the frequency the iteration log is printed.

Default 1

Minimum value 1

printLevel="PRINTBASIC" | "PRINTDETAIL" | "PRINTNONE"

Specifies the level of verbosity of messages printed by the optimization algorithm.

Default PRINTBASIC

PRINTBASIC

Print basic message to the log.

PRINTDETAIL

Print detailed message to the log.

PRINTNONE

Disable printing.

sgdOpt={*sgdOptions*}

SGD solver option set.

The sgdOptions value can be one or more of the following:

adaptiveRate=TRUE | FALSE

Specifies the adaptive rate parameter for SGD.

Default FALSE

annealingRate=*double*

Specifies the annealing parameter.

Default 1e-06

Minimum value 0

commFreq=*64-bit-integer*

Specifies the number of minibatches that each computational thread processes before weights are synchronized across all threads and nodes.

Minimum value 0

learningRate=*double*

Specifies the learning rate parameter for SGD.

Default 0.001

Minimum value 0

miniBatchSize=*64-bit-integer*

Specifies the size of the minibatches used in SGD.

Default 1

Minimum value 1

momentum=*double*

Specifies the value for momentum for SGD.

Default 0

Range 0–1

seed=*64-bit-integer*

Specifies the seed for random access of observations on each thread for the SGD algorithm.

useLocking=TRUE | FALSE

If true, thread aggregation is performed using locks, otherwise, atomic (non-deterministic) operation.

Default FALSE

state={*optmlState*}

Save state option set.

The optmlState value can be one or more of the following:

checkpointFreq=*64-bit-integer*

Specifies the frequency of saving iteration state.

Default 0

Minimum value 0

saveBest=TRUE | FALSE

Specifies whether to save best solutions.

Default TRUE

table={*casouttable*}

Specifies save state table name.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger

values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

validate={*optmlValidate*}

Validation option set.

The optmlValidate value can be one or more of the following:

frequency=*64-bit-integer*

Specifies how frequently (in epochs) that validation occurs.

Default 0

Minimum value 0

goal=*double*

Specifies a goal for the validation misclassification rate. When misclassification gets below this goal, the optimization will stop.

Default 0

stagnation=*64-bit-integer*

Specifies a number of consecutive validations with increasing misclassification rates that are allowed before early termination of the optimization.

Default 0

Minimum value 0

Common Parameter: casouttable

CASL Syntax

casouttable={
  caslib="*string*",
  compress=TRUE | FALSE,
  indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>},
  label="*string*",
  maxMemSize=*64-bit-integer*,
  name="*table-name*",
  onDemand=TRUE | FALSE,
  promote=TRUE | FALSE,
  replace=TRUE | FALSE,
  replication=*integer*,
  timeStamp="*string*",
  where={*"string-1"* <, *"string-2"*, ...>}
}

Parameter Descriptions

caslib="*string*"
  specifies the name of the caslib to use.

compress=TRUE | FALSE
  when set to True, data compression is applied to the table.
  Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}
  specifies the list of variables to create indexes for in the output data

label="*string*"
  specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*
  specifies the maximum amount of memory, in bytes, that each thread should allocate for

in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

Common Parameter: codegen

CASL Syntax

codegen={
  casOut={
    caslib="*string*"
    compress=TRUE | FALSE
    indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}
    label="*string*"
    maxMemSize=*64-bit-integer*
    name="*table-name*"
    onDemand=TRUE | FALSE
    promote=TRUE | FALSE
    replace=TRUE | FALSE
    replication=*integer*
    timeStamp="*string*"
    where={*"string-1"* <, *"string-2"*, ...>}
  },
  comment=TRUE | FALSE,
  fmtWdth=*integer*,
  indentSize=*integer*,
  labelId=*integer*,
  lineSize=*integer*,
  noTrim=TRUE | FALSE,
  tabForm=TRUE | FALSE
}

Parameter Descriptions

casOut={*casouttable*}
  specifies the settings for an output table.
  caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is

appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

comment=TRUE | FALSE

when set to True, adds comments to the DATA step code.

Default FALSE

fmtWdth=*integer*

specifies the width to use for formatting derived numbers such as parameter estimates in the DATA step code.

Default 20

Range 0–32

indentSize=*integer*

specifies the number of spaces to indent the DATA step code for each indent level.

Default 3

Range 0–10

labelId=*integer*

specifies the label ID to use in array names and statement labels in the DATA step code. By default, a random positive integer is used.

lineSize=*integer*

specifies the line size for the generated code.

Default 120

Range 64–254

noTrim=TRUE | FALSE

requests that the comparison of variables with formatted values be based on the full format width, with padding. By default, leading and trailing blanks are removed from the formatted values.

Default FALSE

tabForm=TRUE | FALSE

when set to True, generates the code in way that is appropriate for storing in a table.

Default FALSE

Common Parameter: displayTables

CASL Syntax

displayTables={
  caseSensitive=TRUE | FALSE,
  exclude=TRUE | FALSE,
  excludeAll=TRUE | FALSE,
  keyIsPath=TRUE | FALSE,
  names={*"string-1"* <, *"string-2"*, ...>},
  pathType="LABEL" | "NAME",
  traceNames=TRUE | FALSE
}

Parameter Descriptions

caseSensitive=TRUE | FALSE

when set to True, filtering of the results table names for display is case-sensitive.

Default FALSE

exclude=TRUE | FALSE

when set to True, suppresses display of the results tables that are specified in the names parameter.

Default FALSE

excludeAll=TRUE | FALSE

when set to True, suppresses display of all results tables.

Default FALSE

keyIsPath=TRUE | FALSE

when set to False, the results table key is just the table name. By default, the key is the entire path.

Default TRUE

names={*"string-1"* <, *"string-2"*, ...>}

specifies a list of quoted results table names to display on the client.

Requirement The specified values must be unique.

pathType="LABEL" | "NAME"

specifies whether the path for results table keys consists of group names or group labels.

Default NAME

traceNames=TRUE | FALSE

when set to True, logs the results table names in the client journal.

Default FALSE

Common Parameter: importOptions

CASL Syntax

importOptions={**fileType**="AUTO" | "BASESAS" | "CSV" | "DOCUMENT" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLS", *fileType-specific-parameters*}

Parameters for fileType="AUTO"

No parameters apply when you specify AUTO

Parameters for fileType="BASESAS"

charMultiplier=*double*

specifies a multiplier value to expand fixed-width character variables that might require transcoding. The lengths are increased to avoid character data truncation. The lengths are increased by multiplying the length by the specified value.

Default 1

Range 1–5

dataTransferMode="AUTO" | "PARALLEL" | "SERIAL"

specifies how data is transferred between the data source and SAS Cloud Analytic Services.

Default AUTO

encryptionPassword="*string*" | *binary-large-object*

specifies a password for encrypting or decrypting stored data.

password="*string*"

specifies the password for a password-protected data set. Use this parameter if the data set is password-protected or uses SAS proprietary encryption.

read="*string*"

specifies the Read password for the SAS data set.

varcharConversion=*double*

specifies the number of bytes to use as a threshold. When a column with the CHAR data type has a length that is equal to or exceeds the specified number of bytes, the VARCHAR data type is used.

Default 0

Range 1–32767

Parameters for fileType="CSV"

allowTruncation=TRUE | FALSE

specifies that truncation of character data that is too long for a given field is allowed.

Default TRUE

delimiter="*string*"

specifies the character to use as the field delimiter.

Default ","

encoding="*string*"

specifies the text encoding of the file. If you do not specify this parameter, then the session encoding is used.

Default "utf-8"

getNames=TRUE | FALSE

when set to True, the values in the first line of the file are used as variable names.

Default TRUE

guessRows=*64-bit-integer*

specifies the number of rows to scan in order to determine data types for variables.

Default 20

Minimum value 1

locale="*string*"

specifies the locale for interpreting data in the file

nThreads=*integer*

specifies the number of threads to use on each machine in the server. By default, the server uses one thread for each CPU that is licensed to use SAS software.

Default 0

Minimum value 0

stripBlanks=TRUE | FALSE

removes leading and trailing blanks from character variables.

Default FALSE

varChars=TRUE | FALSE

when set to True, variable-length strings are used for character variables.

Default TRUE

vars={{*casvarcsvdesc-1*} <, {*casvarcsvdesc-2*}, ...>}

specifies the names, types, formats, and other metadata for variables.

The casvarcsvdesc value can be one or more of the following:

format="*string*"

specifies the format to apply to the variable.

formattedLength=*integer*

specifies the length of format field plus the format precision.

informat="*string*"

specifies the informat to apply to the variable.

label="*string*"

specifies the descriptive label for the variable.

length=*integer*

specifies the unformatted length of the variable. This parameter applies to fixed-length character variables (type="CHAR") only.

Default 8

Minimum value 1

name="*string*"

specifies the name for the variable.

nfd=*integer*

specifies the length of the format precision.

nfl=*integer*

specifies the length of the format field.

type="BINARY" | "CHAR" | "DATE" | "DATETIME" | "DECQUAD" | "DECSEXT" | "DOUBLE" | "INT32" | "INT64" | "TIME" | "VARBINARY" | "VARCHAR"

specifies the data type for the variable.

Default DOUBLE

Parameters for fileType="DOCUMENT"

fileExtList={"CSV", "DOC", "DOCX", "EPUB", "HTML", "KEY", "MBOX", "MSG", "NUMBERS", "ODP", "ODS", "ODT", "PAGES", "PDF", "PPT", "PPTX", "PST", "RTF", "TNEF", "TXT", "WPD", "XLS", "XLSX", "XML"}

specifies the file extensions for subsetting document selection.

recurse=TRUE | FALSE

recursively import documents when the caslib definition permits access to subdirectories.

Default TRUE

tikaConv=TRUE | FALSE

specifies to convert the documents with the Apache Tika document conversion library.

Default FALSE

Parameters for fileType="DTA"

No parameters apply when you specify DTA

Parameters for fileType="ESP"

maxTime=*integer*

specifies the number of seconds to receive data from SAS Event Stream Processing before declaring an end of file on the stream. The data is read until an end of file is found, and then the action stops running.

Default 5

sasTypes=TRUE | FALSE

when set to True, the table is created with character and float data types only. When set to False, the table uses the same data types that are used in the ESP event.

Default FALSE

Parameters for fileType="EXCEL"

getNames=TRUE | FALSE

when set to True, the values in the first line of the file are used as variable names.

Default TRUE

range="*string*"

specifies a subset of the cells to import. For example, the range B2..E8 is the range address for a rectangular block of 12 cells, where the top left cell is B2 and the bottom right cell is E8.

sheet="*string*"

specifies the name of the worksheet to import.

Parameters for fileType="FMT"

name="*string*"

specifies a file system path and filename. The file must be a SAS item store that includes the formats to import.

Parameters for fileType="HDAT"

encryptionPassword="*string*" | *binary-large-object*

specifies a password for encrypting or decrypting stored data.

Parameters for fileType="JMP"

No parameters apply when you specify JMP

Parameters for fileType="LASR"

compPgm="*string*"

specifies an expression for each computed variable that you include in the computedVars parameter.

compVars={*"string-1"* <, *"string-2"*, ...>}

specifies the names of the computed variables to create. Specify an expression for each variable in the computedVarsProgram parameter.

nThreads=*integer*

specifies the number of threads to use on each machine in the server. By default, the server uses one thread for each CPU that is licensed to use SAS software.

Default 0

Minimum value 0

parallelMode="FALLBACK" | "FORCE" | "NONE"

specifies how the table is transferred from SAS LASR Analytic Server to SAS Cloud Analytic Services.

Default FALLBACK

FALLBACK

specifies to attempt a parallel load. If the parallel load does not succeed, then a serial

load is performed.

FORCE

specifies to perform a parallel load. If the parallel load cannot be performed, such as for network connectivity issues, then the load fails.

NONE

specifies to perform a serial load.

preserveOrder=TRUE | FALSE

when set to True, the rows are inserted into the new table in the same order as they are received from the SAS LASR Analytic Server. Creating the table is less efficient when this parameter is used.

Default FALSE

varchars=TRUE | FALSE

when set to True, variable-length strings are used for character variables.

Default FALSE

vars={*"string-1"* <, *"string-2"*, ...>}

specifies the variables to use in the action.

where="*where-expression*"

specifies an expression for subsetting the input data.

Parameters for fileType="SPSS"

No parameters apply when you specify SPSS

Parameters for fileType="XLS"

getNames=TRUE | FALSE

when set to True, the values in the first line of the file are used as variable names.

Default TRUE

range="*string*"

specifies a subset of the cells to import. For example, the range B2..E8 is the range address for a rectangular block of 12 cells, where the top left cell is B2 and the bottom right cell is E8.

sheet="*string*"

specifies the name of the worksheet to import.

Common Parameter: linksVarT

CASL Syntax

linksVarT={

auxWeight="*variable-name*",

from="*variable-name*",

id="*variable-name*",

to="*variable-name*",

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>},

weight="*variable-name*"

}

Parameter Descriptions

auxWeight="*variable-name*"

specifies the data variable name for the auxiliary link weights.

from="*variable-name*"

specifies the data variable name for the 'from' nodes.

Default "from"

id="*variable-name*"

to="*variable-name*"

specifies the data variable name for the 'to' nodes.

Default "to"

vars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the additional data variable names for link attributes.

weight="*variable-name*"

specifies the data variable name for the link weights.

Default "weight"

Common Parameter: multimember

CASL Syntax

multimember={{

details=TRUE | FALSE,

*required parameter **name**="*string*",

noEffect=TRUE | FALSE,

stdize=TRUE | FALSE,

*required parameter **vars**={*"variable-name-1"* <, *"variable-name-2"*, ...>},

weight={*"variable-name-1"* <, *"variable-name-2"*, ...>}
}, {...}}

Parameter Descriptions

details=TRUE | FALSE

when set to True, requests a table that shows additional details that are related to this effect.

Default FALSE

* **name**="*string*"

specifies the name of the effect.

noEffect=TRUE | FALSE

when set to True, sets to zero the values in the design matrix for columns that correspond to observations whose levels of the multimember variables are all missing.

Default FALSE

stdize=TRUE | FALSE

when set to True, scales the entries in the design matrix that correspond to the multimember effect to have a sum of one for each observation.

Default FALSE

* **vars**={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies classification variables for the multimember effect. The levels of a multimember effect consist of the union of formatted values of the variables that define this effect. Each such level contributes one column to the design matrix. For each observation, the value that corresponds to each level of the multimember effect in the design matrix is the

number of times that this level occurs for the observation.

weight={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies numeric variables by which to weigh the contributions of the classification variables that define the multimember effect. The number of weight variables must match the number of classification variables that define the effect.

Common Parameter: outputStatement

CASL Syntax

outputStatement={

*required parameter **casOut**={

caslib="*string*"

compress=TRUE | FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

label="*string*"

maxMemSize=*64-bit-integer*

name="*table-name*"

onDemand=TRUE | FALSE

promote=TRUE | FALSE

replace=TRUE | FALSE

replication=*integer*

timeStamp="*string*"

where={*"string-1"* <, *"string-2"*, ...>}

},

copyVars="ALL" | "ALL_MODEL" | "ALL_NUMERIC" | {*"variable-name-1"* <, *"variable-name-2"*, ...>}

}

Parameter Descriptions

* **casOut**={*casouttable*}

specifies the settings for an output table.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

copyVars="ALL" | "ALL_MODEL" | "ALL_NUMERIC" | {*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies a list of one or more variables to be copied from the input table to the output table. You can alternatively specify the value ALL, ALL_MODEL, or ALL_NUMERIC, which respectively copies all variables, all variables used in the modeling, or all numeric variables from the input table to the output table.

Common Parameter: outputTables

CASL Syntax

outputTables={

groupByVarsRaw=TRUE | FALSE,

includeAll=TRUE | FALSE,

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>},

repeated=TRUE | FALSE,

replace=TRUE | FALSE

}

Parameter Descriptions

groupByVarsRaw=TRUE | FALSE

when set to True, uses raw instead of formatted group-by values in the CAS table.

Default TRUE

includeAll=TRUE | FALSE

when set to True, automatically creates CAS tables for all results tables.

Default FALSE

names={*"string-1"* <, *"string-2"*, ...>} | {*key-1*={*casouttable-1*} <, *key-2*={*casouttable-2*}, ...>}

specifies the results tables to save as CAS tables on the server. By default, each created CAS table uses the same name as the results table. To use a different name, specify a casouttable value for the corresponding results table name.

caslib="*string*"

specifies the name of the caslib to use.

compress=TRUE | FALSE

when set to True, data compression is applied to the table.

Default FALSE

indexVars={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies the list of variables to create indexes for in the output data

label="*string*"

specifies the descriptive label to associate with the table.

maxMemSize=*64-bit-integer*

specifies the maximum amount of memory, in bytes, that each thread should allocate for in-memory blocks before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable.

TIP You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

name="*table-name*"

specifies the name to associate with the table.

onDemand=TRUE | FALSE

This parameter is deprecated.

Default TRUE

promote=TRUE | FALSE

when set to True, adds the output table with a global scope. This enables other sessions to access the table, subject to access controls. The target caslib must also have a global scope.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing table with the same name.

Default FALSE

replication=*integer*

specifies the number of copies of the table to make for fault tolerance. Larger values result in slower performance and use more memory, but provide high availability for data in the event of a node failure.

Default 1

Minimum value 0

timeStamp="*string*"

specifies the timestamp to apply to the table. Specify the value in the form that is appropriate for your session locale.

where={*"string-1"* <, *"string-2"*, ...>}

specifies an expression for subsetting the output data.

repeated=TRUE | FALSE

when set to True, replicates CAS tables on all nodes. This parameter applies to all tables.

Default FALSE

replace=TRUE | FALSE

when set to True, overwrites an existing CAS table of the same name. This parameter

applies to all tables specified in the names parameter unless overridden by the replace parameter in individual casouttable specifications.

Default TRUE

Common Parameter: polynomial

CASL Syntax

polynomial={{
degree=*integer*,
details=TRUE | FALSE,
labelStyle={
expand=TRUE | FALSE
exponent="*string*"
includeName=TRUE | FALSE
productSymbol="NONE" | "*string*"
},
mDegree=*integer*,
*required parameter **name**="*string*",
noSeparate=TRUE | FALSE,
standardize={
method="MOMENTS" | "MRANGE" | "WMOMENTS"
options="CENTER" | "CENTERSCALE" | "NONE" | "SCALE"
prefix="NONE" | "*string*"
},
*required parameter **vars**={*"variable-name-1"* <, *"variable-name-2"*, ...>}
}, {...}}

Parameter Descriptions

degree=*integer*

specifies the degree of the polynomial. The degree must be a positive integer.

Default 1

Minimum value 1

details=TRUE | FALSE

when set to True, requests a table that shows additional details that are related to this effect.

Default FALSE

labelStyle={*labelstylelist*}

specifies a list of parameters that control how the terms in the polynomial are labeled.

The labelstylelist value can be one or more of the following:

expand=TRUE | FALSE

when set to True, writes each variable that has an exponent greater than 1 as a product of that variable.

Default FALSE

exponent="*string*"

uses exponential notation with the specified exponentiation string to write each variable that has an exponent greater than 1. By default, the symbol ^ is used as the exponentiation operator.

includeName=TRUE | FALSE

when set to True, uses the name of the effect followed by an underscore as a prefix for term labels.

Default FALSE

productSymbol="NONE" | "*string*"

uses the specified string as the product symbol.

mDegree=*integer*

specifies the maximum degree of any variable in a term of the polynomial.

Default 1

Minimum value 1

* **name**="*string*"

specifies the name of the effect.

noSeparate=TRUE | FALSE

when set to True, treats the polynomial as a single effect that has multiple degrees of freedom. The specified effect name is used as the constructed effect name, and the labels of the terms are used as labels of the corresponding parameters.

Default FALSE

standardize={*standardizelist*}

specifies a list of parameters that control how the variables that form the polynomial are standardized.

The standardizelist value can be one or more of the following:

method="MOMENTS" | "MRANGE" | "WMOMENTS"

specifies the method by which to standardize the variables that define the polynomial.

Default MRANGE

options="CENTER" | "CENTERSCALE" | "NONE" | "SCALE"

specifies whether the standardization method is to center, scale, or both center and scale.

Default CENTERSCALE

prefix="NONE" | "*string*"

specifies the prefix to append to standardized variables when forming the term labels.

* **vars**={*"variable-name-1"* <, *"variable-name-2"*, ...>}

specifies numeric variables for the multivariate polynomial effect.

Subject Index

analytic store table
    gmm action, 843

bnet action
    displayed output, 207
    fit statistics table, 207
    iteration report table, 208
    model information table, 207
    number of observations table, 208
    results, 208
    validation information table, 208
    variable information table, 208
    variable level table, 208
    variable order table, 208

cluster covariance table
    gmm action, 843
cluster score table
    gmm action, 842
cluster summary table
    gmm action, 842

describe action
    results table names, 12
descriptive statistics
    gmm action, 841
displayed output
    bnet action, 207
    factmac action, 288
    fastknn action, 305
    fism action, 42
    gmm action, 841
    graphMultiReg action, 329
    labelSpread action, 343
    mbAnalysis action, 40
    svmTrain action, 938
    tSne action, 1059

factmac action, 288
    displayed output, 288
    final exact loss, 289
    interval variables, 289
    iteration history, 289
    model information, 288
    nominal variables, 289
    number of observations, 288
    output data tables, 290
    OutputCasTables table, 289
    results, 289

fastknn action, 305
    displayed output, 305
    model information, 305
    number of observations, 305
    output CAS tables, 305
final exact loss
    factmac action, 289
fism action, 41
    displayed output, 42
    FP-growth algorithm, 41
    limitations, 43
fit statistics table
    bnet action, 207
FP-growth algorithm
    fism action, 41

gmm action
    analytic store table, 843
    cluster covariance table, 843
    cluster score table, 842
    cluster summary table, 842
    descriptive statistics, 841
    displayed output, 841
    model information, 842
    number of observations, 841
    results, 845
graphical lasso
    gVarCluster action, 363
graphical representation of precision matrix
    gVarCluster action, 364
graphMultiReg action, 324
    displayed output, 329
    graphType parameter examples, 324
    interval variables, 329
    iteration history, 329
    missing values, 329
    model information, 329
    number of observations, 329
    output data tables, 330
    OutputCasTables table, 329
    results tables, 330
graphType parameter examples
    graphMultiReg action, 324
gVarCluster action
    graphical lasso, 363
    graphical representation of precision matrix, 364
    missing values, 363
    preprocessing, 364

results, 365

independence tests, 204
interval variables
factmac action, 289
graphMultiReg action, 329
iteration history
factmac action, 289
graphMultiReg action, 329
tSne action, 1059
iteration report table
bnet action, 208

labelSpread action, 343
displayed output, 343
model information, 343
output CAS tables, 344
limitations
fism action, 43
mbAnalysis action, 41

mbAnalysis action, 39
displayed output, 40
limitations, 41
scoring with the score action, 40
missing values
graphMultiReg action, 329
gVarCluster action, 363
model information
factmac action, 288
fastknn action, 305
gmm action, 842
graphMultiReg action, 329
labelSpread action, 343
tSne action, 1059
model information table
bnet action, 207

nominal variables
factmac action, 289
number of observations
factmac action, 288
fastknn action, 305
gmm action, 841
graphMultiReg action, 329
tSne action, 1059
number of observations table
bnet action, 208

output CAS tables
fastknn action, 305
labelSpread action, 344
tSne action, 1059
output data tables
factmac action, 290
graphMultiReg action, 330
OutputCasTables table
factmac action, 289
graphMultiReg action, 329

parameter learning, 207
preprocessing
gVarCluster action, 364

results tables
graphMultiReg action, 330

scoring with the score action
mbAnalysis action, 40
structure learning, 205
svmTrain action
displayed output, 938

tSne action, 1058
displayed output, 1059
iteration history, 1059
model information, 1059
number of observations, 1059
output CAS tables, 1059

validation information table
bnet action, 208
variable information table
bnet action, 208
variable level table
bnet action, 208
variable order table
bnet action, 208
variable selection, 205

Syntax Index

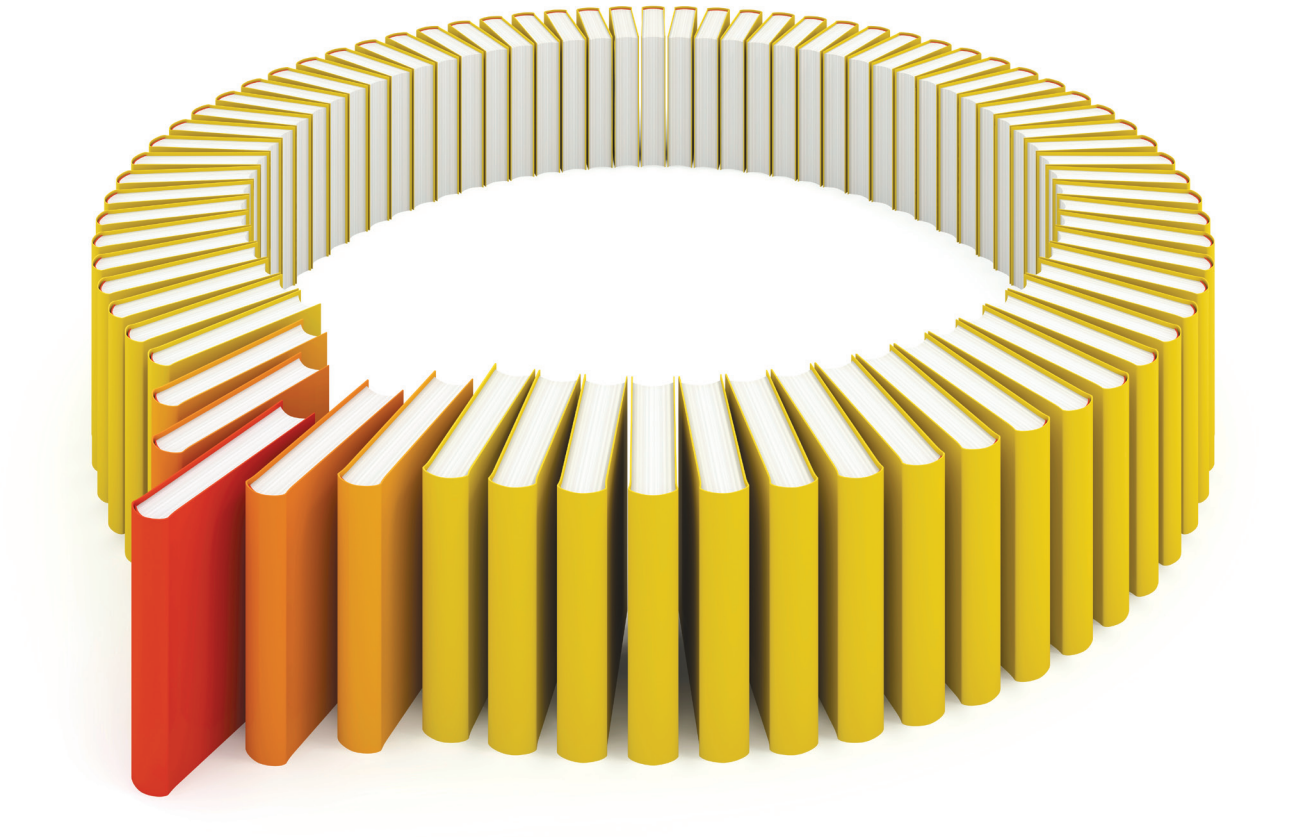
factmac syntax, 288
fastknn syntax, 305
fism syntax, 41

graphMultiReg syntax, 324

labelSpread action syntax, 343

mbAnalysis syntax, 39

tSne syntax, 1058



Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore
for additional books and resources.

sas
THE POWER TO KNOW®

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. © 2013 SAS Institute Inc. All rights reserved. S107969US.0613