## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>What’s New in SAS Visual Data Mining and Machine Learning 8.5 Procedures</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Introduction</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Shared Concepts</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>The ASTORE Procedure</td>
<td>27</td>
</tr>
<tr>
<td>5</td>
<td>The BNET Procedure</td>
<td>49</td>
</tr>
<tr>
<td>6</td>
<td>The BOOLRULE Procedure</td>
<td>89</td>
</tr>
<tr>
<td>7</td>
<td>The FACTMAC Procedure</td>
<td>121</td>
</tr>
<tr>
<td>8</td>
<td>The FASTKNN Procedure</td>
<td>141</td>
</tr>
<tr>
<td>9</td>
<td>The FISM Procedure</td>
<td>153</td>
</tr>
<tr>
<td>10</td>
<td>The FOREST Procedure</td>
<td>163</td>
</tr>
<tr>
<td>11</td>
<td>The GMM Procedure</td>
<td>207</td>
</tr>
<tr>
<td>12</td>
<td>The GRADBOOST Procedure</td>
<td>233</td>
</tr>
<tr>
<td>13</td>
<td>The GVARCLUS Procedure</td>
<td>281</td>
</tr>
<tr>
<td>14</td>
<td>The KPCA Procedure</td>
<td>303</td>
</tr>
<tr>
<td>15</td>
<td>The MBANALYSIS Procedure</td>
<td>325</td>
</tr>
<tr>
<td>16</td>
<td>The MTLearn Procedure</td>
<td>339</td>
</tr>
<tr>
<td>17</td>
<td>The MWPCA Procedure</td>
<td>363</td>
</tr>
<tr>
<td>18</td>
<td>The NNET Procedure</td>
<td>383</td>
</tr>
<tr>
<td>19</td>
<td>The RPCA Procedure</td>
<td>419</td>
</tr>
<tr>
<td>20</td>
<td>The SEMISUPLEARN Procedure</td>
<td>445</td>
</tr>
<tr>
<td>21</td>
<td>The SPARSEML Procedure</td>
<td>461</td>
</tr>
<tr>
<td>22</td>
<td>The SVDD Procedure</td>
<td>477</td>
</tr>
<tr>
<td>23</td>
<td>The SVMACHINE Procedure</td>
<td>545</td>
</tr>
<tr>
<td>24</td>
<td>The TEXTMINE Procedure</td>
<td>569</td>
</tr>
<tr>
<td>25</td>
<td>The TMSCORE Procedure</td>
<td>615</td>
</tr>
<tr>
<td>26</td>
<td>The TSNE Procedure</td>
<td>623</td>
</tr>
</tbody>
</table>

**Subject Index**

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>637</td>
</tr>
</tbody>
</table>

**Syntax Index**

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>645</td>
</tr>
</tbody>
</table>
Overview

SAS Visual Data Mining and Machine Learning 8.5 includes two new procedures and a number of enhancements to existing procedures.
New Procedures

The KPCA Procedure

The KPCA procedure performs kernel principal component analysis (KPCA). KPCA is a nonlinear extension to the widely used principal component analysis (PCA). It uses the “kernel trick” to implicitly map the original data to some high-dimensional RKHS (reproducing kernel Hilbert space) and implement PCA in that space. The resulting projections onto the kernel principal components can capture nonlinear patterns in the data. The KPCA procedure features fast training and fast scoring methods, which greatly alleviate the computational and memory burden that is associated with ordinary KPCA. These methods are based on low-rank matrix approximation coupled with k-means clustering as a sampling scheme. Thanks to this approximation algorithm, PROC KPCA is capable of handling large data sets efficiently.

The SPARSEML Procedure

The SPARSEML procedure was designed for machine learning with sparse input data sets. It currently implements the support vector machines (SVM) algorithm in SAS Viya for binary classification. By using the parallel coordinate descent optimization method, the SPARSEML procedure can train large data sets that can be both wide and deep. Like many other predictive modeling tools, PROC SPARSEML uses input data to train a model and generates an analytic store that can be deployed through the ASTORE procedure. It can load data from multiple nodes and perform computations in parallel.

Enhancements

The AUTOTUNE Statement

- This statement now enables you to output a hyperparameter importance ODS table.
- This statement now supports more options, saving the configuration history, and including warm start.
The FACTMAC Procedure

- The procedure now accepts data that contain missing values without issuing an error. Observations that have missing values are excluded from the analysis.

The FOREST Procedure

- The NUMBIN= and MINLEAFSIZE= options can now be set by autotuning.
- The default value of the NUMBIN= option is now 50 instead of 20, and the default value of the BINMETHOD= option is QUANTILE instead of BUCKET.
- For isolation forests, the TARGET statement is no longer required, and fit statistics that include variable importance are no longer computed.

The GRADBOOST Procedure

- The MINLEAFSIZE= option can now be set by autotuning.
- The default values of some options have changed: the NUMBIN= option default is now 50 instead of 20, the MAXDEPTH= option default is now 4 instead of 5, the RIDGE= option default is now 1 instead of 0, and the BINMETHOD= option default is now QUANTILE instead of BUCKET.

The SEMISUPLEARN Procedure

- PROC SEMISUPLEARN now supports the AUTOTUNE statement.
The SVMACHINE Procedure

- The EARLYSTOP option is added. If this option is specified and the PARTITION statement is also specified, then the generated model is based on the validation accuracy.
Chapter 2
Introduction

Overview of SAS Visual Data Mining and Machine Learning Procedures

This book describes the data mining and machine learning procedures that are available in SAS Visual Data Mining and Machine Learning. These procedures provide data mining and machine learning algorithms that have been specially developed to take advantage of the distributed environment that the SAS Viya platform provides. Supervised learning methods that are available include forest and gradient boosting models, neural networks, support vector machines, and factorization machines. Procedures for scoring via an analytic store and for text mining are also included.

In addition to the data mining and machine learning procedures described in this book, SAS Visual Data Mining and Machine Learning provides procedures for sampling, data exploration, clustering, dimension reduction, model assessment, and additional supervised learning, which are described in SAS Visual Statistics: Procedures.
Experimental Software

Experimental software is sometimes included as part of a production-release product. It is provided to (sometimes targeted) customers in order to obtain feedback. All experimental uses are marked Experimental in this document. Whenever an experimental procedure, statement, or option is used, a message is printed to the SAS log to indicate that it is experimental.

The design and syntax of experimental software might change before its status becomes production. Experimental software has been tested prior to release, but it has not necessarily been tested to production-quality standards and so should be used with care.

About This Book

This book assumes that you are familiar with Base SAS software and with the books *SAS Language Reference: Concepts* and *Base SAS Procedures Guide*. It also assumes that you are familiar with basic SAS System concepts, such as using the DATA step to create SAS data sets and using Base SAS procedures (such as the PRINT and SORT procedures) to manipulate SAS data sets.

Chapter Organization

This book is organized as follows:

Chapter 2, this chapter, provides an overview of the data mining and machine learning procedures that are available in SAS Visual Data Mining and Machine Learning, and it summarizes related information, products, and services.

Chapter 3 provides information about topics that are common to multiple procedures. Topics include how to use SAS Cloud Analytic Services (CAS) sessions and how to load a SAS data set onto a CAS server. This chapter also documents the CODE and PARTITION statements, which are used across a number of procedures.

Subsequent chapters describe the data mining and machine learning procedures. These chapters appear in alphabetical order by procedure name and are organized as follows:

- The “Overview” section briefly describes the analysis provided by the procedure.
- The “Getting Started” section provides a quick introduction to the procedure through a simple example.
- The “Syntax” section describes the SAS statements and options that control the procedure.
- The “Details” section discusses methodology and other topics, such as ODS tables.
- The “Examples” section contains examples that use the procedure.
- The “References” section contains references for the methodology.
Typographical Conventions

This book uses several type styles for presenting information. The following list explains the meaning of the typographical conventions used in this book:

- **roman** is the standard type style used for most text.
- **UPPERCASE ROMAN** is used for SAS statements, options, and other SAS language elements when they appear in text. However, you can enter these elements in your own SAS programs in lowercase, uppercase, or a mixture of the two.
- **UPPERCASE BOLD** is used in the “Syntax” sections’ initial lists of SAS statements and options.
- **oblique** is used in the syntax definitions and in text to represent arguments for which you supply a value.
- **VariableName** is used for the names of variables and data sets when they appear in text.
- **bold** is used for matrices and vectors.
- **italic** is used for terms that are defined in text, for emphasis, and for references to publications.
- **monospace bold** is used for example code. In most cases, this book uses lowercase type for SAS code.

Options Used in Examples

The HTMLBLUE style is used to create the graphs and the HTML tables that appear in the online documentation. The PEARLJ style is used to create the PDF tables that appear in the documentation. A style template controls stylistic elements such as colors, fonts, and presentation attributes. You can specify a style template for an HTML ODS destination as follows:

```
ods html style=HTMLBlue;
```

You can also specify a style template for a PDF ODS destination as follows:

```
ods pdf style=PearlJ;
```

Most of the PDF tables are produced by using the following SAS System option:

```
options papersize=(6.5in 9in);
```

If you run the examples, you might get slightly different output. This is a function of the SAS System options that are used and the precision that your computer uses for floating-point calculations.
Where to Turn for More Information

Online Documentation
You can access the documentation by going to http://support.sas.com/documentation.

SAS Technical Support Services
The SAS Technical Support staff is available to respond to problems and answer technical questions regarding the use of procedures in this book. Go to http://support.sas.com/techsup for more information.
Chapter 3
Shared Concepts

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to Shared Concepts</td>
<td>9</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>10</td>
</tr>
<tr>
<td>Loading a SAS Data Set onto a CAS Server</td>
<td>10</td>
</tr>
<tr>
<td>Syntax Common to SAS Visual Data Mining and Machine Learning Procedures</td>
<td>12</td>
</tr>
<tr>
<td>AUTOTUNE Statement</td>
<td>12</td>
</tr>
<tr>
<td>CODE Statement</td>
<td>18</td>
</tr>
<tr>
<td>PARTITION Statement</td>
<td>20</td>
</tr>
<tr>
<td>Details for SAS Visual Data Mining and Machine Learning Procedures</td>
<td>21</td>
</tr>
<tr>
<td>Using Validation and Test Data</td>
<td>21</td>
</tr>
<tr>
<td>Multithreading</td>
<td>22</td>
</tr>
<tr>
<td>Hyperparameter Tuning</td>
<td>23</td>
</tr>
<tr>
<td>Data Partitioning</td>
<td>23</td>
</tr>
<tr>
<td>Hyperparameter Tuning Process</td>
<td>24</td>
</tr>
<tr>
<td>Determining the Number of Parallel Evaluations</td>
<td>25</td>
</tr>
<tr>
<td>Lookup Table and History Table</td>
<td>26</td>
</tr>
<tr>
<td>References</td>
<td>26</td>
</tr>
</tbody>
</table>

Introduction to Shared Concepts

This book describes data mining and machine learning procedures that run in SAS Viya. One component of SAS Viya is SAS Cloud Analytic Services (CAS), which is the analytic server and associated cloud services. The following subsections describe how to set up and use CAS sessions.

The section “Syntax Common to SAS Visual Data Mining and Machine Learning Procedures” on page 12 describes the common syntax elements that are supported by some of the procedures in this book. In some cases, individual procedures implement these common elements in slightly different ways. When this occurs, the differences are described in the respective procedure chapters.

The section “Details for SAS Visual Data Mining and Machine Learning Procedures” on page 21 provides details that are common to some of the procedures in this book.
Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS statement and the LIBNAME statement, see *SAS Cloud Analytic Services: User’s Guide*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Loading a SAS Data Set onto a CAS Server

Procedures in this book require the input data to reside on a CAS server. To work with a SAS data set, you must first load the data set onto the CAS server. Data loaded on the CAS server are called *data tables*. This section lists three methods of loading a SAS data set onto a CAS server. In this section, `mycas` is the name of the caslib that is connected to the `mysess` CAS session.

- You can use a single DATA step to create a data table on the CAS server as follows:

```sas
data mycas.Sample;
  input y x @@;
datalines;
  .46 1 .47 2 .57 3 .61 4 .62 5 .68 6 .69 7
;```
Note that DATA step operations might not work as intended when you perform them on the CAS server instead of the SAS client.

- You can create a SAS data set first, and when it contains exactly what you want, you can use another DATA step to load it onto the CAS server as follows:

```sas
data Sample;
  input y x @@;
  datalines;
 .46 1 .47 2 .57 3 .61 4 .62 5 .68 6 .69 7 .78 8
 ;
 data mycas.Sample;
  set Sample;
 run;
```

- You can use the CASUTIL procedure as follows:

```sas
proc casutil sessref=mysess;
  load data=Sample casout="Sample";
 quit;
```

The CASUTIL procedure can load data onto a CAS server more efficiently than the DATA step. For more information about the CASUTIL procedure, see *SAS Cloud Analytic Services: User’s Guide*.

The mycas caslib stores the Sample data table, which can be distributed across many machine nodes. You must use a caslib reference in procedures in this book to enable the SAS client machine to communicate with the CAS session. For example, the following FACTMAC procedure statements use a data table that resides in the mycas caslib:

```sas
proc factmac data = mycas.Sample;
  ...statements...
 run;
```

You can delete your data table by using the DELETE procedure as follows:

```sas
proc delete data = mycas.Sample;
 run;
```

The Sample data table is accessible only in the mysess session. When you terminate the mysess session, the Sample data table is no longer accessible from the CAS server. If you want your Sample data table to be available to other CAS sessions, then you must promote your data table. For more information about data tables, see *SAS Cloud Analytic Services: User’s Guide*. 
Syntax Common to SAS Visual Data Mining and Machine Learning Procedures

AUTOTUNE Statement

\[ \text{AUTOTUNE} \ < \text{options} > \ ; \]

This section applies to the following procedures: BNET, FACTMAC, FOREST, GRADBOOST, MTLearn, NNET, SEMISUPLEARN, SVMACHINE, and TSNE.

Table 3.1 summarizes the options you can specify in the AUTOTUNE statement. All options except the TUNINGPARAMETERS= option are described in detail in the list that follows Table 3.1. For more information about the TUNINGPARAMETERS= option, see the specific procedure chapter.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDLOOKUP</td>
<td>Specifies that the table specified in the HISTORYTABLE= option contain the rows from the table specified in the LOOKUPTABLE= option</td>
</tr>
<tr>
<td>EVALHISTORY=</td>
<td>Specifies how to report the evaluation history of the tuner</td>
</tr>
<tr>
<td>FRACTION=</td>
<td>Specifies the fraction of observations to use for validation</td>
</tr>
<tr>
<td>HISTORYTABLE=</td>
<td>Specifies the CAS table that contains the evaluation history</td>
</tr>
<tr>
<td>KFOLD=</td>
<td>Specifies the number of folds for k-fold cross validation</td>
</tr>
<tr>
<td>LIVEUPDATE</td>
<td>Specifies that the table specified in the HISTORYTABLE= option be updated at every evaluation</td>
</tr>
<tr>
<td>LOCALSEARCH</td>
<td>Enables local search optimization</td>
</tr>
<tr>
<td>LOOKUPTABLE=</td>
<td>Specifies the CAS table to use for evaluation lookup</td>
</tr>
<tr>
<td>MAXBAYES=</td>
<td>Specifies the maximum number of points in the kriging model</td>
</tr>
<tr>
<td>MAXEVALS=</td>
<td>Specifies the maximum number of evaluations</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>MAXTIME=</td>
<td>Specifies the maximum time for all iterations</td>
</tr>
<tr>
<td>MAXTRAINTIME=</td>
<td>Specifies the maximum time for a model training</td>
</tr>
<tr>
<td>NCONVITER=</td>
<td>Specifies the number of convergence iterations</td>
</tr>
<tr>
<td>NOGRIDSHUFFLE</td>
<td>Requests that the grid points not be shuffled</td>
</tr>
<tr>
<td>NPARALLEL=</td>
<td>Specifies the number of parallel sessions</td>
</tr>
<tr>
<td>NSUBSESSIONWORKERS=</td>
<td>Specifies the number of workers in parallel sessions</td>
</tr>
<tr>
<td>OBJECTIVE=</td>
<td>Specifies the objective function</td>
</tr>
<tr>
<td>POPSIZE=</td>
<td>Specifies the population size when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>SAMPLESIZE=</td>
<td>Specifies the sample size when SEARCHMETHOD=LHS or SEARCHMETHOD=RANDOM</td>
</tr>
</tbody>
</table>
Table 3.1  continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEARCHMETHOD=</td>
<td>Specifies the search method that the optimizer uses</td>
</tr>
<tr>
<td>SECONDOBJECTIVE=</td>
<td>Specifies the second objective to use for tuning</td>
</tr>
<tr>
<td>SELECTINITPOINT</td>
<td>Specifies that the tuner select the best evaluation from the lookup table</td>
</tr>
<tr>
<td>TARGETEVENT=</td>
<td>Specifies the target event for ROC-based calculations</td>
</tr>
<tr>
<td>TRAINFRACTION=</td>
<td>Specifies the fraction of observations to use for training</td>
</tr>
<tr>
<td>TUNINGPARAMETERS=</td>
<td>Specifies the custom tuning parameters</td>
</tr>
<tr>
<td>USEPARAMETERS=</td>
<td>Specifies how to handle the TUNINGPARAMETERS= option</td>
</tr>
</tbody>
</table>

APPENDLOOKUP

specifies that the rows from the table specified in the LOOKUPTABLE= option be appended to the output table that is created using the HISTORYTABLE= option.

This option is ignored if you do not specify both the HISTORYTABLE= and LOOKUPTABLE= options.

EVALHISTORY=ALL | LOG | NONE | TABLE

specifies how to report the evaluation history of the tuner.

You can specify one of the following values:

- **ALL**: reports each evaluation in the log and creates the EvaluationHistory ODS table.
- **LOG**: prints the following information to the log for each evaluation: evaluation number, objective value, best objective value up to that point, evaluation time, and elapsed time since the beginning of the tuning process.
- **NONE**: suppresses reporting of evaluations in the log and does not create the EvaluationHistory ODS table.
- **TABLE**: creates the EvaluationHistory ODS table, which contains all evaluated points. The table contains columns for the evaluation number, all tuning parameters, and the objective function value.

By default, EVALHISTORY=TABLE.

FRACTION=number

specifies the fraction of all data to be used for validation, where number must be between 0.01 and 0.99, inclusive. If you specify this option, the tuner uses a single-partition validation for finding the objective value (validation error estimate). Using this option might not be advisable for small or unbalanced data tables, where the random assignment of the validation subset might not provide a good estimate of error. For large, balanced data tables, a single-partition validation is usually sufficient for estimating error; a single partition is more efficient than cross validation in terms of the total execution time.

If a PARTITION statement is specified, the validation partition defined in that statement is used, and this option is ignored. You cannot specify this option in combination with the KFOLD= option.

If the TRAINFRACTION= option is used, then by default the value of the FRACTION= option is one minus the value of the TRAINFRACTION= option. Otherwise, by default, FRACTION=0.3.
**HISTORYTABLE=** `CAS-libref.data-table`

specifies the data table created by the tuning process that contains the evaluation history. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the section “Using CAS Sessions and CAS Engine Librefs” on page 10.

**KFOLD=** `number`

specifies the number of folds (partitions) in the cross validation process, where `number` must be between 2 and 20, inclusive. If you specify this option, the tuner uses cross validation to find the objective value. In cross validation, each model evaluation requires `number` of training executions (on `number`–1 data folds) and `number` of scoring executions (on one hold-out fold). Thus, the evaluation time is increased by approximately a factor of `number`. For small to medium data tables or for unbalanced data tables, cross validation provides on average a better representation of error across the entire data table (a better generalization error).

If you do not specify either this option or the `FRACTION=` option, then the default of `FRACTION=0.3` is used. If a `PARTITION` statement is specified, the validation partition defined in that statement is used, and this option is ignored. You cannot specify this option in combination with the `FRACTION=` option.

**LIVEUPDATE**

specifies that the table specified in the `HISTORYTABLE=` option be updated at every evaluation. This option forces the table to be a promoted CAS table so that it can be seen by other sessions.

This option is ignored if you do not specify the `HISTORYTABLE=` option.

**LOCALSEARCH**

enables local search optimization for the tuning process.

**LOOKUPTABLE=** `CAS-libref.data-table`

specifies the data table previously created by the tuning process that you want to use as the lookup table. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the input data table. For more information about this two-level name, see the section “Using CAS Sessions and CAS Engine Librefs” on page 10.

**MAXBAYES=** `number`

specifies the maximum number of points in the Kriging model. This option is honored only when `SEARCHMETHOD=BAYESIAN` and when `number` has a minimum value of 10.

By default, `MAXBAYES=100`.

**MAXEVALS=** `number`

specifies the maximum number of configuration evaluations allowed for the tuner, where `number` must be an integer greater than or equal to 3. When the `number` of evaluations is reached, the tuner terminates the search and returns the results. To produce a single objective function value (validation error estimate), each configuration evaluation requires either a single model training and scoring execution on a validation partition, or a number of training and scoring executions equal to the value of the `KFOLD=` option for cross validation. The `MAXEVALS=` option might lead to termination before the value of the `MAXITER=` option or the `MAXTIME=` option is reached.

By default, `MAXEVALS=50`. 
**MAXITER=number**
specifies the maximum number of iterations of the optimization tuner, where *number* must be greater than or equal to 1. Each iteration normally involves a number of objective evaluations up to the value of the POPSIZE= option. The MAXITER= option is used only when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN; it is ignored when SEARCHMETHOD=RANDOM or SEARCHMETHOD=LHS. The MAXITER= option might lead to termination before the value of the MAXEVALS= option or the MAXTIME= option is reached.

By default, MAXITER=5.

**MAXTIME=number**
specifies the maximum time (in seconds) allowed for the tuner, where *number* must be greater than or equal to 1. When this value is reached, the tuner terminates the search and returns results. The actual run time for optimization might be longer because it includes the remaining time needed to finish the current evaluation. For long-running model training (large data tables), the actual run time might significantly exceed *number*. The MAXTIME= option might lead to termination before the value of the MAXEVALS= option or the MAXITER= option is reached.

By default, MAXTIME=36000.

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

By default, there is no maximum time allowed for a single train.

**NCONVITER=number**
specifies the *number* of convergence iterations after which tuning is terminated.

By default, NCONVITER=4.

**NOGRIDSHUFFLE**
requests that the points that are generated by the grid search method not be shuffled before execution.

**NPARALLEL=number**
specifies the number of evaluations to be performed in parallel, where *number* must be greater than or equal to 0.

By default, NPARALLEL=0, which indicates that the value of *number* is determined as follows:

- If SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN, then the number of parallel evaluations is equal to the value of the POPSIZEx option minus 1.
- If SEARCHMETHOD=LHS or SEARCHMETHOD=RANDOM, then the number of parallel evaluations is equal to the value of the SAMPLESIZE= option, with a maximum value of 32.

**NSUBSESSIONWORKERS=number**
specifies the number of workers to use in parallel subsessions. When alternative configurations are evaluated in parallel, a number of subsessions is created by the tuner, with each using multiple workers. By default, the number of workers that are used in a parallel subsession is determined as described in the section “Determining the Number of Parallel Evaluations” on page 25.


**OBJECTIVE=function**

specifies which measure of model performance the tuner uses as the objective function.

You can specify one of the following values for function. Some values can be specified only when the target variable is of a particular type, as shown in parentheses.

- **ASE**
  uses average squared error as the objective function.

- **AUC**
  uses area under the curve as the objective function (nominal type only).

- **F05**
  uses the F0.5 coefficient as the objective function (nominal type only).

- **F1**
  uses the F1 coefficient as the objective function (nominal type only).

- **GAMMA**
  uses the gamma coefficient as the objective function (nominal type only).

- **GINI**
  uses the Gini coefficient as the objective function (nominal type only).

- **KS**
  uses the Kolmogorov-Smirnov coefficient as the objective function (nominal type only).

- **MAE**
  uses the mean absolute error as the objective function (interval type only).

- **MCE**
  uses the misclassification rate as the objective function (nominal type only).

- **MCLL**
  uses the multiclass log loss as the objective function (nominal type only).

- **MISC**
  uses the misclassification error percentage as the objective function (nominal type only).

- **MSE**
  uses the mean squared error as the objective function (interval type only).

- **MSLE**
  uses the mean squared logarithmic error as the objective function (interval type only).

- **RASE**
  uses the root average squared error as the objective function.

- **RMAE**
  uses the root mean absolute error as the objective function (interval type only).

- **RMSLE**
  uses the root mean squared logarithmic error as the objective function (interval type only).

- **TAU**
  uses the tau coefficient as the objective function (nominal type only).

By default, **OBJECTIVE=MISC** for nominal targets, and **OBJECTIVE=MSE** for interval targets.

**POPSIZE=number**

specifies the maximum population size, where population is the number of configuration evaluations in one iteration (population). The number must be greater than or equal to 1. In some cases, the tuner algorithm might generate a number of new configurations that is smaller than number. The **POPSIZE=** option is used only when **SEARCHMETHOD=GA** or **SEARCHMETHOD=BAYESIAN**; it is ignored when **SEARCHMETHOD=RANDOM** or **SEARCHMETHOD=LHS**.

By default, **POPSIZE=10**.
**SAMPLESIZE=number**
specifies the total number of evaluations, where *number* must be greater than or equal to 1. This option is only used when SEARCHMETHOD=RANDOM or SEARCHMETHOD=LHS; it is ignored when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN.

By default, SAMPLESIZE=50.

**SEARCHMETHOD=BAYESIAN | GA | GRID | LHS | RANDOM**
specifies the search method to use for tuning. You can specify the following values:

- **BAYESIAN**: builds a kriging surrogate model to approximate the objective value and uses this surrogate model to generate new alternative configurations at each iteration. The kriging model is continuously updated during the search process.
- **GA**: uses an initial Latin hypercube sample that seeds a genetic algorithm to generate a new population of alternative configurations at each iteration.
- **GRID**: uses all combinations of selected values of the hyperparameters.
- **LHS**: uses a Latin hypercube to generate a single sample of configurations that is uniform in each tuning parameter but random in combinations.
- **RANDOM**: generates a single sample of purely random configurations.

By default, SEARCHMETHOD=GA.

**SECONDOBJECTIVE=NONE | SCORETIME | TRAINTIME**
specifies the type of the second objective to use for tuning. You can specify the following values:

- **NONE**: does not use a second objective for tuning.
- **SCORETIME**: specifies the model scoring time as the second objective for tuning.
- **TRAINTIME**: specifies the model training time as the second objective for tuning.

By default, SECONDOBJECTIVE=NONE.

**SELECTINITPOINT**
specifies that the tuner select the best evaluation from the table specified in the LOOKUPTABLE= option as the initial point for the tuning process.

This option is ignored if you do not specify the LOOKUPTABLE= option.

**TARGETEVENT=string**
specifies the target event to use for calculating the selected objective function. This option is ignored when the value of the OBJECTIVE= option is not AUC, F1, F05, GINI, GAMMA, TAU, or KS.

If you do not specify the TARGETEVENT= option, the tuner selects one of the target levels and uses it for calculating the specified objective function.
Chapter 3: Shared Concepts

TRAINFRACTION=number
specifies the fraction of all data to be used for training, where number must be between 0.01 and 0.99, inclusive. If you specify this option, the tuner uses a single-partition validation to find the objective value (validation error estimate). Using this option might not be advisable for small or unbalanced data tables, where the random assignment of the validation subset might not provide a good estimate of error. For large, balanced data tables, a single-partition validation is usually sufficient for estimating error; a single partition is more efficient than cross validation in terms of the total execution time.

If a PARTITION statement is specified, the validation partition defined in that statement is used, and this option is ignored. You cannot specify this option in combination with the KFOLD= option.

If the FRACTION= option is used, then by default the value of the TRAINFRACTION= option is one minus the value of the FRACTION= option. Otherwise, by default, TRAINFRACTION=0.7.

TUNINGPARAMETERS=(suboption | . . . | < suboption >)
TUNEPARMS=(suboption | . . . | < suboption >)
specifies which parameters to tune and which ranges to tune over. If USEPARAMETERS=STANDARD, this option is ignored.

For more information about which tuning suboptions are available, see the specific procedure chapters.

USEPARAMETERS=COMBINED | CUSTOM | STANDARD
specifies which set of parameters to tune.

You can specify the following values:

COMBINED tunes the parameters that are specified in the TUNINGPARAMETERS= option and uses default bounds and initial values to tune all other parameters.

CUSTOM tunes only the parameters that are specified in the TUNINGPARAMETERS= option.

STANDARD tunes all parameters by using their default bounds and initial values.

By default, USEPARAMETERS=COMBINED.

---

**CODE Statement**

```sas
CODE < options > ;
```

This section applies to the following procedures: FOREST and GRADBOOST.

The CODE statement writes SAS DATA step code for computing predicted values of the fitted model to a file, to a catalog entry, or to a CAS table. To score new data, you can then include the file or the catalog entry in a DATA step, or you can specify the CAS table in the runCodeTable action in the dataStep action set.

Table 3.2 summarizes the options that you can specify in the CODE statement.
Table 3.2  CODE Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENT</td>
<td>Adds comments to the generated code</td>
</tr>
<tr>
<td>FILE=</td>
<td>Names the file in which to save the generated code</td>
</tr>
<tr>
<td>FORMATWIDTH=</td>
<td>Specifies the numeric format width for the regression coefficients</td>
</tr>
<tr>
<td>INDENTSIZE=</td>
<td>Specifies the number of spaces to indent the generated code</td>
</tr>
<tr>
<td>LABELID=</td>
<td>Specifies a number used to construct names and labels</td>
</tr>
<tr>
<td>LINESIZE=</td>
<td>Specifies the line size for the generated code</td>
</tr>
<tr>
<td>NOTRIM</td>
<td>Compares formatted values, including blank padding</td>
</tr>
<tr>
<td>OUT=</td>
<td>Names an output CAS table in which to save the generated code</td>
</tr>
</tbody>
</table>

If you do not specify the `FILE=` option or the `OUT=` option and if your SAS client has a default path, then the SAS scoring code is written to an external file named `_code_`. You can specify the following options in the CODE statement.

**COMMENT**
adds comments to the generated code.

**FILE=** filename
names the external file that saves the generated code. When enclosed in a quoted string (for example, `FILE="c:\mydir\scorecode.sas"`), this option specifies the path and filename for writing the code to an external file. If you do not specify a path but your SAS client has a default path, then the code is written to an external file named `filename` at that location. You can also specify an unquoted `filename` of no more than eight characters. If the `filename` is assigned as a fileref in a Base SAS FILENAME statement, the file specified in the FILENAME statement is opened; otherwise, if your SAS client has a default path, an external file named `filename` is created. This option cannot be specified with the `OUT=` option.

**FORMATWIDTH=** width
specifies the width to use in formatting derived numbers such as parameter estimates. You can specify a value in the range 12 to 32; the default is 20.

**INDENTSIZE=** n
specifies the number of spaces to indent the generated code. You can specify a value in the range 0 to 10; the default is 3.

**LABELID=** value
specifies a number used to construct array names and statement labels in the generated code. You can specify a value in the range 0 to 1024; the default is randomly chosen.

**LINESIZE=** value
**LS=** value
specifies the line size for the generated code. You can specify a value in the range 64 to 254; the default is 120.
NOTRIM
bases comparisons of formatted values on the full format width, including blank padding. By default, blanks at the beginning and end of strings are ignored.

OUT=CAS-libref.data-table
creates a CAS table that contains the score code. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the section “Using CAS Sessions and CAS Engine Librefs” on page 10. You can specify this CAS table in the runCodeTable action in the dataStep action set to score another data table; for more information, see SAS Viya: System Programming Guide. This option cannot be specified with the FILE= option.

---

PARTITION Statement

PARTITION partition-option ;

This section applies to the following procedures: FOREST and GRADBOOST.

The PARTITION statement specifies how observations in the input data set are logically partitioned into disjoint subsets for model training, validation, and testing. For more information, see the section “Using Validation and Test Data” on page 21. Either you can designate a variable in the input data table and a set of formatted values of that variable to determine the role of each observation, or you can specify proportions to use for randomly assigning observations to each role.

You must specify exactly one of the following partition-options:

FRACTION(< TEST=fraction > < VALIDATE=fraction > < SEED=number >)
randomly assigns specified proportions of the observations in the input data table to the roles. You specify the proportions for testing and validation by using the TEST= and VALIDATE= suboptions. If you specify both the TEST= and VALIDATE= suboptions, then the sum of the specified fractions must be less than 1 and the remaining fraction of the observations are assigned to the training role. The SEED= option specifies an integer that is used to start the pseudorandom number generator for random partitioning of data for training, testing, and validation. If you do not specify SEED=number or if number is less than or equal to 0, the seed is generated by reading the time of day from the computer’s clock.

ROLE=variable (< TEST=value ’ > < TRAIN=value ’ > < VALIDATE=value ’ >)
ROLEVAR=variable (< TEST=value ’ > < TRAIN=value ’ > < VALIDATE=value ’ >)
names the variable in the input data table whose values are used to assign roles to each observation. This variable cannot also appear as an analysis variable in other statements or options. The TEST=, TRAIN=, and VALIDATE= suboptions specify the formatted values of this variable that are used to assign observation roles. If you do not specify the TRAIN= suboption, then all observations whose role is not determined by the TEST= or VALIDATE= suboption are assigned to the training role.
Details for SAS Visual Data Mining and Machine Learning Procedures

Using Validation and Test Data

This section applies to the following procedures: FOREST and GRADBOOST.

When you have sufficient data, you can divide your data into three parts called the training, validation, and test data. During the selection process, models are fit on the training data, and the prediction errors for the models so obtained are found by using the validation data. This prediction error on the validation data can be used to decide when to terminate the selection process and to decide which model to select. Finally, after a model has been selected, the test set can be used to assess how the selected model generalizes on data that played no role in selecting the model.

In some cases, you might want to use only training and test data. For example, you might decide to use an information criterion to decide which effects to include and when to terminate the selection process. In this case, no validation data are required, but test data can still be useful in assessing the predictive performance of the selected model. In other cases, you might decide to use validation data during the selection process but forgo assessing the selected model on test data. Hastie, Tibshirani, and Friedman (2001) note that it is difficult to provide a general rule for how many observations you should assign to each role. They note that a typical split might be 50% for training and 25% each for validation and testing.

You use a PARTITION statement to logically subdivide the input data table into separate roles. You can specify the fractions of the data that you want to reserve as test data and validation data. For example, the following statements randomly divide the inData data table, reserving 50% for training and 25% each for validation and testing:

```
proc forest data=mycas.inData;
    partition fraction(test=0.25 validate=0.25);
    ... 
run;
```

You can specify the SEED= option in the PARTITION statement to create the same partition data tables for a particular number of compute nodes. However, changing the number of compute nodes changes the initial distribution of data, resulting in different partition data tables.

In some cases, you might need to exercise more control over the partitioning of the input data table. You can do this by naming both a variable in the input data table and a formatted value of that variable for each role. For example, the following statements assign roles to the observations in the inData data table that are based on the value of the variable Group in that data table. Observations whose value of Group is 'Group 1' are assigned for testing, and those whose value is 'Group 2' are assigned to training. All other observations are ignored.

```
proc forest data=mycas.inData;
    partition fraction(test='Group 1' validate='Group 2');
    ... 
run;
```
multithreading

Threading refers to the organization of computational work into multiple tasks (processing units that can be scheduled by the operating system). A task is associated with a thread. Multithreading refers to the concurrent execution of threads. When multithreading is possible, substantial performance gains can be realized compared to sequential (single-threaded) execution. The number of threads spawned by a procedure in this book is determined by your installation.

The tasks that are multithreaded by procedures in this book are primarily defined by dividing the data that are processed on a single machine among the threads—that is, the procedures implement multithreading through a data-parallel model. For example, if the input data table has 1,000 observations and the procedure is running on four threads, then 250 observations are associated with each thread. All operations that require access to the data are then multithreaded. These operations include the following (not all operations are required for all procedures):

- variable levelization
- effect levelization
- formation of the initial crossproducts matrix
- formation of approximate Hessian matrices for candidate evaluation during model selection
- objective function calculation
- gradient calculation
- Hessian calculation
- scoring of observations

In addition, operations on matrices such as sweeps can be multithreaded provided that the matrices are of sufficient size to realize performance benefits from managing multiple threads for the particular matrix operation.
Hyperparameter Tuning

This section applies to the following procedures: BNET, FACTMAC, FOREST, GRADBOOST, MTLEARN, NNET, SEMISUPLEARN, SVMACHINE, and TSNE.

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 the AUTOTUNE statement to identify the best settings for them.

To tune hyperparameters, the AUTOTUNE statement directs the procedure to 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 PARTITION statement is used (in this case, the tuner does not partition the training data, but instead uses the data roles that are specified in the PARTITION statement for training and 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 AUTOTUNE statement calls the sampling action set, which creates the training and validation partitions. The stratified action in the sampling action set 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 in the sampling action set 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 FRACTION= option in the AUTOTUNE statement. For more information about the sampling action set, see Chapter 20, “Sampling and Partitioning Action Set” (SAS Visual Statistics: Programming Guide).
**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 KFOLD= option in the AUTOTUNE statement.

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 25.

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 25). 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 four alternative search methods: Bayesian, random, LHS, and grid. The preceding steps are performed differently, depending on the search method that you use:

- For the Bayesian search method, the preceding steps are performed in the same sequence, with the following changes. Instead of using genetic operations to generate 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 method, step 3 in the preceding sequence is eliminated; a single sample of candidate configurations is generated and evaluated in step 2. This single sample of configurations is generated in one of the following ways:
  
  - randomly, for the random search method
  - by 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= option 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 those values as the grid levels for each hyperparameter. 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, then the total number of combinations can be many hundreds or thousands. If the MAXEVALS= option is specified, the tuner uses it 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 PARTITION statement 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.

**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 24 is determined by the population size for the GA or Bayesian search method or by the sample size for the random or LHS sampling method. The maximum number of parallel evaluations is limited 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= option in the AUTOTUNE statement, the number of workers is determined using the size of the data table: NSUBSESSIONWORKERS = 1 + nDataRows * nDataColumns * 2E–8. If determined on the basis of 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 either as one less than the population size for the GA or Bayesian search method, or as the sampling size for the random or LHS 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= option in the AUTOTUNE statement, it is limited to 4. You can override this limit up to a value of 32 by specifying the NPARALLEL= option (the value of the NPARALLEL= option 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. You can override this limit by a factor of 2 by specifying the NPARALLEL= option; the new limit is $2W/n$ (the value of the NPARALLEL= option is reduced if it is greater than $2W/n$).

**Lookup Table and History Table**

Tuning a predictive model often requires multiple runs of a tuning action and various adjustments of the hyperparameter bounds or their initial values. This often means training a large number of model configurations for each new tuning process, some of which might be duplicates of configurations from the previous tuning runs. To avoid rerunning the same configurations, you can use a lookup table, which you specify in the LOOKUPTABLE= option in the AUTOTUNE statement. You can create this lookup table manually, such as by using a DATA step, or you can produce it automatically in any procedure call in which you specify the HISTORYTABLE= option in the AUTOTUNE statement. Before evaluating a new predictive model configuration, the tuner first checks the lookup table for a potential match. The lookup table must contain all the hyperparameter columns and the objective column; if any required columns are not found, the tuner rejects the lookup table. If a matching configuration is found, its objective value is read from the lookup table and the configuration is not evaluated. The tuner matches only the hyperparameter values and ignores all other columns in the lookup table. For convenience, when you do multiple runs of a procedure in which you use autotuning, you can collect the history of all runs in one table by using the APPENDLOOKUP option in the AUTOTUNE statement. This causes the tuner to append all unused configurations from the lookup table to the history table, which can then be used again as the lookup table for the next procedure call in which you use autotuning. It is essential that the lookup table contain only the appropriate data. If any nontuned parameters have changed, or if the data that are used for the predictive model have changed, then the objective values inside the lookup table might no longer be valid for the new tuning process. The tuner is oblivious to any changes in the problem setup between any two runs of the tuning process and will use the objective values from the lookup table as long as the hyperparameter values match the configuration to be evaluated.

**References**

Chapter 4
The ASTORE Procedure

Contents

Overview: ASTORE Procedure .......................................................... 28
PROC ASTORE Features ................................................................. 28
Using CAS Sessions and CAS Engine Librefs ................................. 28
Getting Started: ASTORE Procedure .............................................. 29
Home Equity Data Set ................................................................. 29
Creating an Analytic Store ............................................................. 30
Using the Analytic Store ............................................................... 31
Syntax: ASTORE Procedure .......................................................... 32
PROC ASTORE Statement ............................................................. 32
DESCRIBE Statement ................................................................. 32
DOWNLOAD Statement ............................................................... 33
SCORE Statement ...................................................................... 34
SETOPTION Statement ............................................................... 35
UPLOAD Statement ................................................................. 41
Details: ASTORE Procedure ........................................................ 41
ODS Table Names ................................................................. 42
Examples: ASTORE Procedure ....................................................... 43
Example 4.1: Scoring a Single Record ............................................ 43
Example 4.2: Describing the Store ............................................... 44
Example 4.3: Downloading the Store to the Local File System .......... 46
Example 4.4: Uploading the Local Store from the Local File System 47
Example 4.5: Scoring with Multiple Stores ..................................... 47
Overview: ASTORE Procedure

The ASTORE procedure is an interactive procedure in which each statement runs immediately. You can use the ASTORE procedure describe an analytic store, manage it, and score new data with the analytic store. The analytic store is the result of a SAVESTATE statement from another analytic procedure; it is a binary file that contains that procedure’s state after it completes the training phase of data analysis.

PROC ASTORE Features

The ASTORE procedure enables you to do the following:

- describe limited information about the analytic store
- move analytic stores between the client and the server
- produce different types of DS2 scoring code that can run locally using the DS2 procedure
- produce DS2 language scoring code that can run in SAS Viya
- score an input data table and produce an output data table by using a specified analytic store and optional DS2 scoring code that uses the analytic store
- use multiple specified analytic stores and DS2 scoring code to score an input data table and produce an output data table
- consume code that is created by a DESCRIBE statement; you can edit the code and send it again in a SCORE statement (because PROC ASTORE is interactive)

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```sas
   proc options option=(CASHOST CASPORT);
   run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:
cas mysess;
libname mycas cas sessref=mysess;

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

    cas mysess terminate;

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”

---

### Getting Started: ASTORE Procedure

#### Home Equity Data Set

Several examples in this chapter use the home equity data set hmeq, which is available in the Sampsio library that SAS provides. The data set contains observations for 5,960 mortgage applicants. A variable named Bad indicates whether the customer has paid on his or her loan or has defaulted on it. Table 4.1 describes the variables in Hmeq.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Role</th>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad</td>
<td>Response</td>
<td>Binary</td>
<td>1 = customer defaulted on the loan or is seriously delinquent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = customer is current on loan payments</td>
</tr>
<tr>
<td>CLAge</td>
<td>Predictor</td>
<td>Interval</td>
<td>Age of oldest credit line in months</td>
</tr>
<tr>
<td>CLNo</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of credit lines</td>
</tr>
<tr>
<td>DebtInc</td>
<td>Predictor</td>
<td>Interval</td>
<td>Debt-to-income ratio</td>
</tr>
<tr>
<td>Delinq</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of delinquent credit lines</td>
</tr>
<tr>
<td>Derog</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of major derogatory reports</td>
</tr>
<tr>
<td>Job</td>
<td>Predictor</td>
<td>Nominal</td>
<td>Occupational category</td>
</tr>
<tr>
<td>Loan</td>
<td>Predictor</td>
<td>Interval</td>
<td>Requested loan amount</td>
</tr>
<tr>
<td>MortDue</td>
<td>Predictor</td>
<td>Interval</td>
<td>Amount due on existing mortgage</td>
</tr>
<tr>
<td>nlnq</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of recent credit inquiries</td>
</tr>
<tr>
<td>Reason</td>
<td>Predictor</td>
<td>Binary</td>
<td>'DebtCon' = debt consolidation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'HomeImp' = home improvement</td>
</tr>
<tr>
<td>Value</td>
<td>Predictor</td>
<td>Interval</td>
<td>Value of current property</td>
</tr>
<tr>
<td>YoJ</td>
<td>Predictor</td>
<td>Interval</td>
<td>Years at present job</td>
</tr>
</tbody>
</table>
You can load the `sampsio.hmeq` 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` (as in the section “Using CAS Sessions and CAS Engine Librefs” on page 28), but you can substitute any appropriately defined CAS engine libref.

This DATA step includes an `id` variable, which is bound to the observation number `_N_`. The `id` variable is used to join the input records with their corresponding scores.

```sas
data mycas.hmeq;
  set sampsio.hmeq;
  id = _N_; 
run;
```

Because distributed computing orders distributed data differently than traditional SAS procedures orders them, one or more variables that can act as the record identifier must be in the input data table. Here the record identifier is a single variable `id`.

---

**Creating an Analytic Store**

When you score complex analytic models, you must use a different approach from the usual simple scoring code. Instead you can save the state of the model in a binary file called an analytic store. This binary file can be used later to score the model. This example shows how to produce an analytic store that is created by the `SVMACHINE` procedure. For more information about the `SVMACHINE` procedure, see Chapter 23, “The SVMACHINE Procedure.”

The following PROC `SVMACHINE` call specifies two INPUT statements: one specifies the variables `loan`, `mortdue`, `value` as interval variables; the other specifies the variables `reason`, `job`, `delinq`, and `ninq` as classification variables. The TARGET statement indicates that the variable `bad` is chosen as the target. The ID statement indicates that the `id` variable must be present in the output data table in order to join records from the input table to their corresponding record in the output table. The SAVESTATE statement saves the state of the `SVMACHINE` procedure in the analytic store, which is stored in the table `mycas.savehmeq`.

```sas
proc svmachine data=mycas.hmeq;
  input loan mortdue value /level=interval;
  input reason job delinq ninq /level=nominal;
  target bad;
  id id;
  savestate rstore=mycas.savehmeq;
run;
```

**NOTE:** PROC ASTORE does not have to run immediately after PROC `SVMACHINE`.

Two different procedure runs at different times will produce two analytic stores that have different keys, even though the two runs might appear to be identical. Running the preceding code twice will produce two stores with different key identifiers.
Using the Analytic Store

The most important task of the ASTORE procedure is to score an input table by using the information in the analytic store, which is stored in a data table in CAS.

In this example, the input data table is mycas.hmeq, the output data table is mycas.scoreout1, and the analytic store is in the data table mycas.savehmeq. All the input tables must be loaded in your CAS session. The resulting output table is created in the same CAS session.

```
proc astore;
  score data=mycas.hmeq
    out=mycas.scoreout1
    rstore=mycas.savehmeq;
quit;
data scoreout1;
  set mycas.scoreout1;
run;
proc sort data=scoreout1;
  by id;
run;
```

The following statements print the observations, as shown in Output 4.1.

```
proc print data=scoreout1(obs=5);
run;
```

**Figure 4.1** Scoring with PROC ASTORE

<table>
<thead>
<tr>
<th>Obs</th>
<th>id</th>
<th><em>P</em></th>
<th>P_BAD1</th>
<th>P_BAD0</th>
<th>l_BAD</th>
<th><em>WARN</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.00000</td>
<td>0.00000</td>
<td>1.00000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1.00000</td>
<td>0.00000</td>
<td>1.00000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1.00000</td>
<td>0.00000</td>
<td>1.00000</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>-0.84616</td>
<td>0.64103</td>
<td>0.35897</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>5</td>
<td>1.00000</td>
<td>0.00000</td>
<td>1.00000</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 4: The ASTORE Procedure

Syntax: ASTORE Procedure

The following statements are available in the ASTORE procedure:

```plaintext
PROC ASTORE ;
  SCORE score-options ;
  DESCRIBE describe-options ;
  DOWNLOAD download-options ;
  UPLOAD upload-options ;
  SETOPTION name value ;
```

PROC ASTORE is interactive: each statement is executed immediately.

The following sections describe the PROC ASTORE statement and then describe the other statements in alphabetical order.

PROC ASTORE Statement

```plaintext
PROC ASTORE ;
```

The PROC ASTORE statement invokes the procedure and does not require any options.

DESCRIBE Statement

```plaintext
DESCRIBE STORE=local-file-name < describe-options > ;
DESCRIBE RSTORE=CAS-libref.data-table < describe-options > ;
```

The DESCRIBE statement specifies the name or identifier of an analytic store either in the local file system by using the STORE= option or in a data table stored in CAS by using the RSTORE= option. The DESCRIBE statement can also produce DS2 basic scoring code. You can edit the basic scoring code to add transformations to the input variables, flag or override the decision made for the record, work with ensembles, and so on. Because PROC ASTORE is interactive, you can edit the result from the DESCRIBE statement and send it in a subsequent SCORE statement. The edited file must comply with the DS2 language syntax. For more information about the DS2 language, see SAS DS2 Language Reference.

You must specify exactly one of the following options:

- **STORE=local-file-name**
  - specifies either the file reference or the full path of a valid store file that was created earlier by another procedure that processed a SAVESTATE statement.

- **RSTORE=CAS-libref.data-table**
  - specifies the CAS table that contains the analytic store. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 28.

You can also specify the following describe-option:
EPCODE< =code-file-name>
sends DS2 language code either to the SAS log (if you do not specify a code-file-name) or to an external code file that can run in CAS and that is identified by code-file-name, which is either the file reference or the full path and member name of the external code file.

NOTE: The DS2 code enables you to score concurrently with multiple analytic stores as long as they share the same input and output variables. The store key identifier plays an important role in managing multiple analytic stores in a single run.

You can also specify the following describe-options in the order shown, if you want to copy some variables from the input data table to the output data table while scoring with the EPCODE= option:

DATA= CAS-libref.data-table
names the input data table for PROC ASTORE to use. CAS-libref.data-table is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about CAS-libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 28.

data-table specifies the name of the input data table.

COPYVARS= variable | (variables)

COPYVAR= variable | (variables)
copies one or more variables from the input data table that you specify in the DATA= option to the output data table.

NOTE: The COPYVARS= option must follow the DATA= option.

DOWNLOAD Statement

DOWNLOAD RSTORE= CAS-libref.data-table STORE= store-file-name ;
The DOWNLOAD statement retrieves an external binary analytic store that was produced by another procedure from the CAS session and stores it in the local file system.

You must specify the following options in any order:

RSTORE= CAS-libref.data-table
specifies a data table that contains the state to be downloaded. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 28.
**STORE=local-file-name**
specifies either the file reference or the full path of the local analytic store file to which the contents of the data table are to be downloaded.

---

**SCORE Statement**

```plaintext
SCORE DATA=CAS-libref.data-table OUT=CAS-libref.data-table <...> <OUT=CAS-libref.data-table>
RSTORE=CAS-libref.data-table <...> <RSTORE=CAS-libref.data-table>
<COPYVARS=(variables)> <EPICODE<=code-file-name> ;
```

The SCORE statement enables you to score both simple and complex models.

You must specify the following `score-options`:

**DATA=** `CAS-libref.data-table`

names the input data table for PROC ASTORE to use. `CAS-libref.data-table` is a two-level name, where

- `CAS-libref` refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 28.

- `data-table` specifies the name of the input data table.

**OUT=** `CAS-libref.data-table`

names the output data table for PROC ASTORE to use. `CAS-libref.data-table` is a two-level name, where

- `CAS-libref` refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 28.

- `data-table` specifies the name of the output data table.

You must specify one `OUT=` option for each model except the text analytical models. You can specify additional `OUT=` options, one for each table that the model produces. You can use the DESCRIBE statement to see how many output tables a model produces. If a model produces more than one output table, you must specify the `OUT=` options in the order in which they are listed in the DESCRIBE statement.

**RSTORE=** `CAS-libref.model-file-name`

specifies the data table in CAS to contain the analytic store. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 28.
You must specify at least one RSTORE= option. You can specify as many additional RSTORE= options as you need. When you specify more than one RSTORE= option, you must specify EPCODE= option.

You can also specify the following options:

**COPYVARS=** `variable | (variables)`

**COPYVAR=** `variable | (variables)`

transfers one or more `variables` from the input data table that you specify in the DATA= option to the output data table that you specify in the OUT= option.

**NOTE:** If you specify the COPYVARS= option, it must follow the DATA= option.

**EPCODE=** `code-file-name`

names the location of the optional scoring code file (which was created by the DESCRIBE statement) and loads that file into the CAS session for scoring. You can use this option when you have changed the contents of the scoring code. You must specify this option when you specify more than one RSTORE= option.

---

### SETOPTION Statement

**SETOPTION name value ;**

The SETOPTION statement associates a numeric `value` to a specified `name`. During scoring, PROC ASTORE uses the `value` for the option that corresponds to `name`.

**name**

specifies an identifier for the option, where `name` must begin with an alphabetic character or underscore, can contain only alphanumerics and underscores, and is limited to 200 characters.

**value**

specifies a numeric `value` for the named option.

Table 4.2 lists the actions that can create a store that PROC ASTORE can consume, along with a list of the available SETOPTION `names` for that store. The `name` and `value` descriptions follow the table.

<table>
<thead>
<tr>
<th>Action</th>
<th>name value Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>bnet</td>
<td>None None</td>
</tr>
<tr>
<td>dlExportModel</td>
<td>DEVICE0 (device0)</td>
</tr>
<tr>
<td>dlExportModel</td>
<td>DEVICE1 (device1)</td>
</tr>
<tr>
<td>dlExportModel</td>
<td>DEVICEk (devicek)</td>
</tr>
<tr>
<td>dlExportModel</td>
<td>NDEVICES (ndevices)</td>
</tr>
<tr>
<td>dlExportModel</td>
<td>TENSORRT (tensortt)</td>
</tr>
<tr>
<td>dnnExportModel</td>
<td>USEGPU (usegpu)</td>
</tr>
<tr>
<td>factmac</td>
<td>None None</td>
</tr>
<tr>
<td>exportTextModel</td>
<td>EXTEND_OUT_CHAR VAR_BYTES (bytes)</td>
</tr>
</tbody>
</table>

---
<table>
<thead>
<tr>
<th>Action</th>
<th>name value</th>
</tr>
</thead>
<tbody>
<tr>
<td>filterDesign</td>
<td>USETRANSFERFUNCTION</td>
</tr>
<tr>
<td>forestTrain</td>
<td>None None</td>
</tr>
<tr>
<td>gampl</td>
<td>ALPHA alpha</td>
</tr>
<tr>
<td>gampl</td>
<td>COMPUTE_COMPONENTWISE_STATS compstat</td>
</tr>
<tr>
<td>gampl</td>
<td>COMPUTE_CONFIDENCE_LIMIT conflimit</td>
</tr>
<tr>
<td>gampl</td>
<td>COMPUTE_INTO into</td>
</tr>
<tr>
<td>gampl</td>
<td>COMPUTE_LINEAR_PREDICTOR linpred</td>
</tr>
<tr>
<td>gampl</td>
<td>COMPUTE_STANDARD_ERROR stderr</td>
</tr>
<tr>
<td>gampl</td>
<td>SET INTO CUTPOINT cutpoint</td>
</tr>
<tr>
<td>gamselect</td>
<td>COMPUTE_COMPONENTWISE_STATS compstat</td>
</tr>
<tr>
<td>gamselect</td>
<td>COMPUTE_INTO into</td>
</tr>
<tr>
<td>gamselect</td>
<td>COMPUTE_LINEAR_PREDICTOR linpred</td>
</tr>
<tr>
<td>gamselect</td>
<td>SET INTO CUTPOINT cutpoint</td>
</tr>
<tr>
<td>gbtreeTrain</td>
<td>None None</td>
</tr>
<tr>
<td>genmod</td>
<td>SET INTO CUTPOINT cutpoint</td>
</tr>
<tr>
<td>glm</td>
<td>ALPHA alpha</td>
</tr>
<tr>
<td>glm</td>
<td>COMPUTE_CONFIDENCE_LIMIT conflimit</td>
</tr>
<tr>
<td>gmm</td>
<td>None None</td>
</tr>
<tr>
<td>gprReg</td>
<td>None None</td>
</tr>
<tr>
<td>graphMultiReg</td>
<td>None None</td>
</tr>
<tr>
<td>hmm</td>
<td>DECODE decode</td>
</tr>
<tr>
<td>hmm</td>
<td>DECODE_WINDOW decodewindow</td>
</tr>
<tr>
<td>hmm</td>
<td>EVALUATE evaluate</td>
</tr>
<tr>
<td>hmm</td>
<td>FILTER filter</td>
</tr>
<tr>
<td>hmm</td>
<td>FORECAST forecast</td>
</tr>
<tr>
<td>hmm</td>
<td>FORECAST ALPHA forecastalpha</td>
</tr>
<tr>
<td>hmm</td>
<td>FORECAST_COV forecastcov</td>
</tr>
<tr>
<td>hmm</td>
<td>FORECAST_LEAD forecastlead</td>
</tr>
<tr>
<td>hmm</td>
<td>SMOOTH smooth</td>
</tr>
<tr>
<td>hmm</td>
<td>SMOOTH_WINDOW smoothwindow</td>
</tr>
<tr>
<td>iml</td>
<td>None None</td>
</tr>
<tr>
<td>kpca</td>
<td>KPCA_NPC kpcanpc</td>
</tr>
<tr>
<td>logistic</td>
<td>SET INTO CUTPOINT cutpoint</td>
</tr>
<tr>
<td>mbAnalysis</td>
<td>ARM_SCORING_ALGO algo</td>
</tr>
<tr>
<td>mbAnalysis</td>
<td>NORM norm</td>
</tr>
<tr>
<td>mixed</td>
<td>None None</td>
</tr>
<tr>
<td>rlExportModel</td>
<td>None None</td>
</tr>
<tr>
<td>robustpca</td>
<td>RPCA_ANOMALYDETECTION_METHOD admethod</td>
</tr>
<tr>
<td>robustpca</td>
<td>RPCA_NUMSIGVARS numsigvars</td>
</tr>
<tr>
<td>robustpca</td>
<td>RPCA_SIGMACOEFF sigmacoef</td>
</tr>
<tr>
<td>robustpca</td>
<td>RPCA_PROJECTION_TYPE projtype</td>
</tr>
<tr>
<td>robustpca</td>
<td>RPCA_USEMATRIX usematrix</td>
</tr>
<tr>
<td>smCalib</td>
<td>MODELID modelid</td>
</tr>
<tr>
<td>smCalib</td>
<td>PROJECTID projectid</td>
</tr>
</tbody>
</table>
The following name value pairs are available for the actions that are listed in Table 4.2:

**ALPHA alpha**
sets the significance level to be used for the construction of confidence intervals. You can specify a value between 0 and 1. The default value is 0.05. This parameter influences only the confidence intervals and does not affect the predicted values.

**ARM_SCORING_ALGO algo**
specifies which algorithm to use for scoring. You can specify a value of 1 (use Association Rules) or 2 (use Exclusive Recommendation) or 3 (use Recommendation). The default value is 1.

**BWCOL bwcol**
specifies whether to output the bandwidth columns. You can specify a value of 0 (do not output) or 1 (output). The default value is 0.

**COMPUTE_COMPONENTWISE_STATS compstat**
specifies whether to compute the following componentwise statistics for each spline term, if they have been requested: linear predictor, standard error, and confidence band. You can specify a value of 0 (do not compute) or 1 (compute). The default value is 0.

**COMPUTE_CONFIDENCE_LIMIT conflimit**
specifies whether to produce additional statistics such as confidence intervals, where conflimit is either 0 or 1. If set to 1, additional statistics are produced for each observation, including the confidence intervals for generalized linear models; for generalized additive models, Bayesian confidence bands are computed. The default value is 0.

**COMPUTE_INTO into**
specifies whether to compute the predicted response level. You can specify a value of 0 (do not compute) or 1 (compute). The default value is 0.

**COMPUTE_LINEAR_PREDICTOR linpred**
specifies whether to compute the predicted response level. You can specify a value of 0 (do not compute) or 1 (compute). The default value is 0.
**Chapter 4: The ASTORE Procedure**

**COMPUTE_STANDARD_ERROR stderr**
specifies whether to compute the standard error of the linear predictor. You can specify a value of 0 (do not compute) or 1 (compute). The default value is 0.

**DECODE decode**
specifies whether to output columns that are related to the decoded result. You can specify a value of 0 (do not output) or 1 (output). The default value is 0.

**DECODE_WINDOW decodewindow**
specifies the window length (or the so-called detection delay) for decoding. If you specify this name value pair, then the value of DECODE decode is set to 1. You can specify a nonnegative integer. The default value is 0.

**DEVICE0 device0**
enables the first GPU device in the system. This applies only when the value of USEGPU usegpu is set to 1.

**DEVICE1 device1**
enables the second GPU device in the system. This applies only when the value of USEGPU usegpu is set to 1 and the value of NDEVICES ndevices is set to be greater than 2.

**DEVICEk devicek**
enables the $k + 1$th GPU device in the system. This applies only when the value of USEGPU usegpu is set to 1 and the value of NDEVICES ndevices is set to be greater than $k + 1$.

**EVALUATE evaluate**
specifies whether to include the log-likelihood column in the output. You can specify a value of 0 (do not output) or 1 (output). The default value is 1.

**EXTEND_OUT_CHAR_VAR_BYTES bytes**
specifies how many bytes to increase the length of each output character variable. The default value is 0.

**FILTER filter**
specifies whether to output columns that are related to the filtered result. You can specify a value of 0 (do not output) or 1 (output). The default value is 0.

**FORECAST forecast**
specifies whether to output columns that are related to forecasts. You can specify a value of 0 (do not output) or 1 (output). The default value is 0.

**FORECAST_ALPHA forecastalpha**
specifies the forecast confidence limit size. You can specify a value between 0 and 1. The default value is 0.95.

**FORECAST_COV forecastcov**
specifies whether to output columns that are related to the covariance of forecasts. You can specify a value of 0 (do not output) or 1 (output). The default value is 1.
FORECAST_LEAD forecastlead
specifies the number of multistep forecast values to compute. You can specify a nonnegative integer.
The default value is 1.

KPCA_NPC kpcanpc
specifies the number of principal components to obtain from scoring new data. The valid value is a positive integer no larger than the rank of the kernel matrix (the rank is stored in the state). PROC ASTORE issues an error if kpcanpc is larger than the kernel matrix rank. If the user inputs decimals, the number is cast to an integer. The default value is 4.

MODELID modelid
specifies the ID of the model that is used for scoring, if multiple bandwidths are used to train the models for support vector data description (SVDD). You can specify modelid as a number between 0 and the number of bandwidths that is used to train the models for SVDD. The default value is 0, which means that scoring uses all models for SVDD.

For the smCalib action, this modelid specifies the ID of the model to use for scoring. This must be one of the model IDs that the smCalib action saves for the specified PROJECTID. The default value is 0, which means that scoring uses the internal model for the smCalib action.

NDEVICES ndevices
specifies the number of GPU devices to use. This applies only when the value of USEGPU usegpu is set to 1.

NORM norm
specifies whether to normalize the input while scoring. You can specify a value of 0 (do not normalize) or 1 (normalize). The default value is 0.

PROJECTID projectid
specifies the ID of the project that is used for scoring. You must specify a valid projectid that is saved from the smCalib action. The project IDs are not necessarily contiguous. The default value is 0.

RPCA_ANOMALYDETECTION_METHOD admethod
specifies the method of anomaly detection, where admethod must be either 0 or 1. If admethod is set to 1, a method that is based on the root sum of squares of values in the sparse part of the scoring observation is applied. This method is referred to as R4S. If admethod is set to 0, a method that counts the number of significant values in the sparse part of the scoring observation is applied to the scoring observations. This method is referred to as SIGVARS. The default value is what you specify in the robustpca action.

RPCA_NUMSIGVARS numsigvars
specifies the minimum number of variables that must have significant values in the sparse matrix for the observation to be considered an anomaly. The default value is what you specify in the robustpca action.

RPCA_PROJECTION_TYPE projtype
specifies the type of projection, where projtype must be either 0, 1, or 2. If projtype is set to 1, the scoring observations are projected onto the low-rank subspace. If projtype is set to 2, the scoring observations are projected onto the low-rank space, but the sparse part of the observation is stored in the resulting scoring table. If projtype is set to 0, the scoring observations are projected onto the principal component space. The default value is 0.
Chapter 4: The ASTORE Procedure

**RPCA_SIGMACOEFF sigmacoef**  
specifies how many standard deviations away from the mean of the column an entry in the sparse matrix must be to be considered a significant value. The default value is what you specify in the `robustpca` action.

**RPCA_USEMATRIX usematrix**  
specifies which matrix to use for standardizing the values of the sparse part of the scoring observations for anomaly detection, where `usematrix` must be either 0 or 1. If `usematrix` is set to 1, the sparse part of the scoring observations is standardized on the basis of the standard deviation of the sparse matrix for anomaly detection. If `usematrix` is set 0, the sparse part of the scoring observations is standardized on the basis of the standard deviation of the input data for anomaly detection. The default value is what you specify in the `robustpca` action.

**SET_INTO_CUTPOINT cutpoint**  
specifies the cutpoint value that is used in binary and binomial response models or generalized additive models to classify observations, where `cutpoint` is a number between 0 and 1. If the predicted probability of an observation equals or exceeds `cutpoint`, the observation is classified as an event; otherwise, it is classified as a nonevent. The default value is 0.5. If you specify a value of 0, the default value is used.

**SMOOTH smooth**  
specifies whether to output columns that are related to the smoothed result. You can specify a value of 0 (do not output) or 1 (output). The default value is 0.

**SMOOTH_WINDOW smoothwindow**  
specifies the window length (or the so-called detection delay) for smoothing. If you specify this `name value` pair, then the value of SMOOTH `smooth` is set to 1. You can specify a nonnegative integer. The default value is 1.

**TENSORRT tensorrt**  
specifies whether to use NVIDIA’s TensorRT to optimize the scoring on the basis of your specific NVIDIA GPU architecture. The default value is 0 (not used). When `tensorrt` is set to 1 (enabled), the first specified GPU device or default device (0) is used to optimize scoring. When `tensorrt` is enabled, the network optimization for each GPU architecture can require significant time. Set `tensorrt` to 1 when the number of observations is considerably large, because the optimization is performed during model setup. This approach enables subsequent model-processing inferences to use the optimized deployed model.

**USEGPU usegpu**  
specifies whether to use GPUs to perform the calculations for scoring. You can specify a value of 0 (do not use) or 1 (use). The default value is 0. If `usegpu` is set to 1, the parameter `nThreads` must be set to 1. For options specific to SAS Event Stream Processing for deep learning, see SAS Event Stream Processing: Using Streaming Analytics.

**USETRANSFERFUNCTION func**  
specifies the type of filter parameters to use for scoring. If `func` is set to 1, the numerator and denominator coefficients of the filter transfer function are used for scoring. Otherwise, the zeros, poles, and gain of the filter transfer function are used for scoring. The default value is 0.
UPLOAD Statement

```
UPLOAD upload-options;
```

The UPLOAD statement moves an analytic store from the local file system into a data table in CAS.

You must specify the following `upload-options` in any order:

- **RSTORE=CAS-libref.data-table**
  
  specifies the CAS table to which the store is sent. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the `caslib` and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the `DATA=` option and the section “Using CAS Sessions and CAS Engine Librefs” on page 28.

- **STORE=local-file-name**
  
  specifies either the file reference or the full path of the valid store file that was created earlier by some analytic engine and exists in the local file system.

Details: ASTORE Procedure

The DESCRIBE statement displays the following basic information about the store:

- store key
- some basic information to describe the model
- input variables
- output variables

If you specify the EPCODE= option in the DESCRIBE statement, PROC ASTORE produces basic DS2 language statements, which it sends to the SAS log unless you specify an optional `code-file-name` to send them to a specified file. The DS2 code contains empty method blocks like the following:

```
method preScoreRecord();
end;
method postScoreRecord();
end;
method run();
  set sasep.in; /* read in the record */
  preScoreRecord(); /* Optional: process the input variables as needed */
  sc.scoreRecord(); /* score one record */
  postScoreRecord(); /* Optional: process the output variables as needed */
end;
```

You can use the `preScoreRecord` method block to transform the input variables, and you can change or flag the scores in the `postScoreRecord` method block.
The UPLOAD statement produces output that includes the store key for future reference in the DS2 language code. The code that is produced by the EPCODE= option in the DESCRIBE statement includes the same key. If you upload the store to CAS and you specify the EPCODE= option in the DESCRIBE statement again, you will observe the same key. The key of the store is dependent on the store and not on whether the store is located in the local file system or in CAS.

**ODS Table Names**

Each table that the ASTORE procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. These names are listed in Table 4.3.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Name of the component that saved the state and the time it was saved</td>
<td>DESCRIBE</td>
</tr>
<tr>
<td>InputVariables</td>
<td>List of input variables from the procedure that saved the state</td>
<td>DESCRIBE</td>
</tr>
<tr>
<td>OutputVariables</td>
<td>List of output variables from the procedure that saved the state</td>
<td>DESCRIBE</td>
</tr>
<tr>
<td>StoreKey</td>
<td>Key information from the UPLOAD statement</td>
<td>UPLOAD</td>
</tr>
<tr>
<td></td>
<td>Key information from the DOWNLOAD statement</td>
<td>DOWNLOAD</td>
</tr>
<tr>
<td></td>
<td>Key information from the DESCRIBE statement</td>
<td>DESCRIBE</td>
</tr>
<tr>
<td>Timing</td>
<td>Timing details</td>
<td>SCORE</td>
</tr>
</tbody>
</table>

**NOTE:** Analytic stores can become very large; breaking down the different stages of scoring a data table in CAS is important and informative.
### Examples: ASTORE Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

**Example 4.1: Scoring a Single Record**

This is a simple example that creates a table that contains one record whose ID is 100 and uses an analytic store that was produced by the SVMACHINE procedure in the section “Creating an Analytic Store” on page 30.

The following DATA step creates the mycas.hmeq1 table in your CAS session by loading the hmeq data table that was created in section “Home Equity Data Set” on page 29 and extracting the record whose ID is 100:

```sas
data mycas.hmeq1;
  set mycas.hmeq;
  if (id = 100);
run;
```

The following statements score the mycas.hmeq1 table with the analytic store in the table mycas.savehmeq to produce the output table mycas.hmeq1out, which is shown in Output 4.1.1.

```sas
proc astore;
  score data=mycas.hmeq1
    rstore=mycas.savehmeq
    out=mycas.hmeq1out;
quit;
proc print data= mycas.hmeq1out ;
run;
```

**Output 4.1.1** Scoring a Single Record

<table>
<thead>
<tr>
<th>Obs</th>
<th>id</th>
<th>P</th>
<th>P_BAD1</th>
<th>P_BAD0</th>
<th>I_BAD</th>
<th>WARN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>1.00000</td>
<td>6.2878E-8</td>
<td>1.00000</td>
<td>0</td>
<td>M</td>
</tr>
</tbody>
</table>
Example 4.2: Describing the Store

In this simple example, no DS2 language code is requested. The DESCRIBE statement produces tables that describe some of the contents store as seen in Output 4.2.1 and sends the basic code to the file svmepcode.sas.

```
proc astore;
    describe rstore=mycas.savehmeq
        epcode="svmepcode.sas";
quit;
```

Output 4.2.1 contains the following ODS tables:

- The “Key Information” table displays the string identifier of the store. This is the same string that is contained in the code that the EPCODE= option in DESCRIBE statement produces.
- The “Basic Information” table displays the analytic engine that produced the store and the time when the store was created by processing a SAVESTATE statement.
- The “Input Variables” table displays the input variables.
- The “Output Variables” table displays the output variables.

Output 4.2.1

Output Tables from the DESCRIBE statement

The ASTORE Procedure

<table>
<thead>
<tr>
<th>Key Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>776BBE3C9FD31B52722B8AA88FC5463AE1A730D2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Basic Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytic Engine</td>
</tr>
<tr>
<td>Time Created</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>LOAN</td>
</tr>
<tr>
<td>MORTDUE</td>
</tr>
<tr>
<td>VALUE</td>
</tr>
<tr>
<td>REASON</td>
</tr>
<tr>
<td>JOB</td>
</tr>
<tr>
<td>DELINQ</td>
</tr>
<tr>
<td>NINQ</td>
</tr>
<tr>
<td>BAD</td>
</tr>
<tr>
<td>id</td>
</tr>
</tbody>
</table>
Output 4.2.1 continued

<table>
<thead>
<tr>
<th>Name</th>
<th>Length</th>
<th>Type</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>8</td>
<td>Num</td>
<td></td>
</tr>
<tr>
<td><em>P</em></td>
<td>8</td>
<td>Num</td>
<td>Decision Function</td>
</tr>
<tr>
<td>P_BAD1</td>
<td>8</td>
<td>Num</td>
<td>Predicted: BAD=1</td>
</tr>
<tr>
<td>P_BAD0</td>
<td>8</td>
<td>Num</td>
<td>Predicted: BAD=0</td>
</tr>
<tr>
<td>I_BAD</td>
<td>32</td>
<td>Char</td>
<td>Into: BAD</td>
</tr>
<tr>
<td><em>WARN</em></td>
<td>4</td>
<td>Char</td>
<td>Warnings</td>
</tr>
</tbody>
</table>

The following statements show the contents of the svmepcode.sas file; this is DS2 language code:

```sas
data sasep.out;
dcl package score sc();
dcl double "LOAN";
dcl double "MORTDUE";
dcl double "VALUE";
dcl nchar(7) "REASON";
dcl nchar(7) "JOB";
dcl double "DELINQ";
dcl double "NINQ";
dcl double "BAD";
dcl double "id";
dcl double "_P_" having label n'Decision Function';
dcl double "P_BAD1" having label n'Predicted: BAD=1';
dcl double "P_BAD0" having label n'Predicted: BAD=0';
dcl nchar(32) "I_BAD" having label n'Into: BAD';
dcl nchar(4) "_WARN_" having label n'Warnings';
Keep
  "id"
  "_P_"
  "P_BAD1"
  "P_BAD0"
  "I_BAD"
  "_WARN_
;
varlist allvars[all_];
method init();
  sc.setvars(allvars);
  sc.setKey(n'14F2E0A7B9A73581F79904C6E900D60ABCC51EDB');
end;
method preScoreRecord();
end;
method postScoreRecord();
end;
method term();
end;
method run();
  set sasep.in;
  preScoreRecord();
  sc.scoreRecord();
  postScoreRecord();
```

NOTE: The `sc.setKey` in the `method init` method block contains a string that identifies an analytic store. Every time you produce a new store, the key changes, even if you think the runs are identical.

You can view or edit the `svmepcode.sas` file that resides in the local file system.

```sas
method preScoreRecord();
   /* insert input variable transformations here */
end;
method postScoreRecord();
   /* change or flag the decisions here */
end;
method run();
   set sasep.in;  /* read in the record */
   preScoreRecord();  /* Optional: process the input variables as needed */
   sc.scoreRecord();  /* score one record */
   postScoreRecord();  /* Optional: process the output variables as needed */
end;
```

Transformations of an input variable should be in the `preScoreRecord` method block. You can alter the decisions made from scoring one record in the `postScoreRecord` method block. If you do not intend to alter the contents of the DS2 code in either of these method blocks, then you do not need to specify the `EPCODE=` option in the `SCORE` statement.

NOTE: The store key will be different every time you use the SAVESTATE statement in a procedure that supports it.

---

**Example 4.3: Downloading the Store to the Local File System**

The following statements extract the analytic store that is saved in the data table `mycas.savehmeq` from the CAS session to the local file `svmlocalcopy` in the local file system.

```sas
proc astore;
   download rstore=mycas.savehmeq
      store="svmlocalcopy";
quit;
```

In addition to downloading the actual file, PROC ASTORE writes a note in the log that shows how many bytes were downloaded.
Example 4.4: Uploading the Local Store from the Local File System

The following statements send the analytic store `svmlocalcopy` from the local file system to the data table `mycas.savehmeq` in the CAS session.

```sas
proc astore;
  upload rstore=mycas.savehmeqnew
    store="svmlocalcopy";
quit;
```

The UPLOAD statement produces the store key in the listing. You can use this key to construct the embedded processing code on your own, but it is simpler for you to use the EPCODE= option in the DESCRIBE statement to produce the resulting minimal code and then edit the contents.

Example 4.5: Scoring with Multiple Stores

This example first shows how to produce another analytic store that is created by the FOREST procedure. For more information about the FOREST procedure, see Chapter 10, “The FOREST Procedure.”

This example uses the same home equity data table (`mycas.hmeq`) that was loaded in the session as described in section “Home Equity Data Set” on page 29. The SAVESTATE statement in the following FOREST procedure code saves the state in an analytic store, which is stored in the table `mycas.savehmeq2`:

```sas
proc forest data=mycas.hmeq seed=17;
  input loan mortdue value /level=interval;
  input reason job delinq ninq /level=nominal;
  target bad;
  id id;
  savestate rstore=mycas.savehmeq2;
run;
```

The following statements score the `mycas.hmeq` table with two analytic stores, which are stored in the tables `mycas.savehmeq` and `mycas.savehmeq2`, to produce the output table `mycas.hmeq2out`:

```sas
proc astore;
  score data=mycas.hmeq rstore=mycas.savehmeq rstore=mycas.savehmeq2
    epcode="epcode2.sas" out=mycas.hmeq2out;
quit;
```

The contents of the `epcode2.sas` file are as follows.

```sas
data sasep.out;
  dcl package score sc();
  dcl package score sc2();
  dcl double "LOAN";
  dcl double "MORTDUE";
  dcl double "VALUE";
  dcl nchar(7) "REASON";
  dcl nchar(7) "JOB";
  dcl double "DELINQ";
  dcl double "NINQ";
```
Chapter 4: The ASTORE Procedure

dcl double "BAD";
dcl double "id";
dcl double "P_" having label n'Decision Function';
dcl double "P_BAD1" having label n'Predicted: BAD=1';
dcl double "P_BAD0" having label n'Predicted: BAD=0';
dcl nchar(32) "I_BAD" having label n'Into: BAD';
dcl nchar(4) "_WARN_" having label n'Warnings';
dcl double "tempP1";
dcl double "tempP2";
Keep
    "id"
    "P_"
    "P_BAD1"
    "P_BAD0"
    "I_BAD"
    "_WARN_"
;
varlist allvars[_all_];
method init();
    sc.setvars(allvars);
    sc.setKey(n'14F2E0A7B9A73581F79904C6E900D60ABCC51EDB');
    sc2.setvars(allvars);
    sc2.setKey(n'FC7E82345DF91C75364F7BFD177BF93AA9129A26');
end;
method preScoreRecord();
end;
method postScoreRecord();
    P_BAD0 = 0.5*( P_BAD0 + tempP1);
    P_BAD1 = 0.5*( P_BAD1 + tempP2);
end;
method term();
end;
method run();
    set sasep.in;
    preScoreRecord();
    sc.scoreRecord();
    tempP1 = P_BAD0;
    tempP2 = P_BAD1;
    sc2.scoreRecord();
    postScoreRecord();
end;
enddata;
Overview: BNET Procedure

The BNET procedure learns a Bayesian network from an input data table in SAS Viya. A Bayesian network is a directed acyclic graphical model in which nodes represent random variables and the links between nodes represent direct dependency among random variables. Variables that are connected through intermediary variables in the network are conditionally independent given the intermediary variables. Because the Bayesian network provides conditional independence structure and a conditional probability table at each node, the model has been used successfully as a predictive model in supervised data mining. For more information about Bayesian networks, see Pearl (1988).

The BNET procedure can learn different types of Bayesian network structures, including naive, tree-augmented naive (TAN), Bayesian network-augmented naive (BAN), parent-child Bayesian network, general Bayesian network, and Markov blanket. PROC BNET performs efficient variable selection through independence tests, and it selects the best model automatically from the specified parameters. It also generates SAS DATA step code or an analytic store to score data. It can load data from multiple nodes and perform computations in parallel.

PROC BNET Features

The BNET procedure has the following features:

- structure learning through efficient local learning algorithms
- efficient variable selection through independence tests
- automatic selection of the best parameters by using a validation data subset
- learning of different types of Bayesian network structures
- capability to specify links to include in or exclude from a network
- handling of both nominal and interval input variables
- binning of the interval input variables
- handling of missing values
- multithreading during the training and scoring phases
Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

Consider a study of the analgesic effects of treatments on elderly patients who have neuralgia. Two test treatments and a placebo are compared. The response variable is whether the patient reported pain or not. Researchers recorded the age and gender of 60 patients and the duration of complaint before the treatment began. The following DATA step creates the data set Neuralgia:

```
Data Neuralgia;
  input Treatment $ Sex $ Age Duration Pain $ @@;
  datalines;
  P F  68  1  No  B M  74  16  No  P F  67  30  No
  P M  66  26  Yes B F  67  28  No  B F  77  16  No
  A F  71  12  No  B F  72  50  No  B F  76  9  Yes
  A M  71  17  Yes A F  63  27  No  A F  69  18  Yes
  B F  66  12  No A M  62  42  No  P F  64  1  Yes
  A F  64  17  No  P M  74  4  No  A F  72  25  No
  P M  70  1  Yes B M  66  19  No  B M  59  29  No
  A F  64  30  No A M  70  28  No  A M  69  1  No
  B F  78  1  No  P M  83  1  Yes B F  69  42  No
  B M  75  30  Yes P M  77  29  Yes P F  79  20  Yes
  A M  70  12  No A F  69  12  No  B F  65  14  No
  B M  70  1  No  B M  67  23  No  A M  76  25  Yes
  P M  78  12  Yes B M  77  1  Yes B F  69  24  No
  P M  66  4  Yes P F  65  29  No  P M  60  26  Yes
  A M  78  15  Yes B M  75  21  Yes A F  67  11  No
  P F  72  27  No  P F  70  13  Yes A M  75  6  Yes
  B F  65  7  No  P F  68  27  Yes P M  68  11  Yes
  P M  67  17  Yes B M  70  22  No  A M  65  15  No
  P F  67  1  Yes A M  67  10  No  P F  72  11  Yes
  A F  74  1  No  B M  80  21  Yes A F  69  3  No
;```

The Neuralgia data set contains five variables: Treatment, Sex, Age, Duration, and Pain. The last variable, Pain, is the target variable. Pain=Yes indicates that the patient felt pain, and Pain=No indicates no pain. The variable Treatment is a nominal variable that has three levels: A and B represent the two test treatments, and P represents the placebo treatment. The gender of the patients is indicated by the nominal variable Sex. The variable Age is the age of the patients, in years, when treatment began. The duration of complaint, in months, before the treatment began is indicated by the variable Duration.

You can load the Neuralgia data set into your CAS session by using your CAS engine libref in the first statement of the following DATA step:

```
data mycas.neuralgia;
  set neuralgia;
run;
```
Probability Table

The following statements produce Figure 5.2, which shows the conditional probability table for each node in the network. You can use these probability tables for scoring or inferences or both. The conditional probability tables together with the network structure determine the Bayesian network.

```
proc print data=mycas.network noobs label;
  var _parentnode_ _parentcond_ _childnode_ _childcond_ _value_;  
  where _type_="PROBABILITY";  
run;
```
**Variable Selection**

The network shown in Figure 5.1 does not include the variable Sex or Duration, because PROC BNET automatically removes these variables from the network by using independence tests. PROC BNET produces the table in Figure 5.3, which shows the results of independence tests that are conducted to select variables. Variables that are found to be marginally independent or conditionally independent of the target variable are removed from the network. PROC BNET removes Duration from the network because the $p$-value of the chi-square and G-square statistics of Duration are greater than 0.05 (the default value of the ALPHA= option). Sex is conditionally independent of Pain given Treatment; therefore, PROC BNET removes Sex from the network.

**Figure 5.3 Variable Selection**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Selected</th>
<th>Chi-Square</th>
<th>Pr &gt; ChiSq</th>
<th>G-Square</th>
<th>Pr &gt; GSq</th>
<th>Mutual Information</th>
<th>DF</th>
<th>Conditional Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sex</td>
<td>No</td>
<td>7.20000</td>
<td>0.0658</td>
<td>7.59454</td>
<td>0.0552</td>
<td>0.34481</td>
<td>3</td>
<td>Treatment</td>
</tr>
<tr>
<td>Treatment</td>
<td>Yes</td>
<td>13.71429</td>
<td>0.0011</td>
<td>14.02297</td>
<td>0.0009</td>
<td>0.45652</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Age</td>
<td>Yes</td>
<td>14.60003</td>
<td>0.0007</td>
<td>15.27118</td>
<td>0.0005</td>
<td>0.47404</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Duration</td>
<td>No</td>
<td>2.25795</td>
<td>0.3234</td>
<td>3.34851</td>
<td>0.1874</td>
<td>0.23298</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
As you review the network, you notice that Pain is the parent of Age. Given the data, there is a strong relationship between Age and Pain that explains this connection, but the direction is not as desired. You can use the INNETWORK= option as in the following statements to guide the network so that both Treatment and Age become parents of Pain:

```
Data Innetwork;
   length parent $ 10;
   input Parent $ Child $ Flag;
datalines;
Treatment Pain 1
Age   Pain 1

data mycas.innetwork;
   set innetwork;
run;

proc bnet data=mycas.neuralgia numbin=3 outnetwork=mycas.network
   innetwork=mycas.innetwork;
   target Pain;
   input Treatment Sex/level=nominal;
   input Age Duration/level=interval;
   ods output varselect=varselect;
run;
```

The revised network is as desired, with Treatment and Age as parents of Pain, as shown in Figure 5.4.

```
proc print data=mycas.network noobs label;
   var _parentnode_ _parentcond_ _childnode_ _childcond_ _value_;
   where _type_="STRUCTURE``; 
run;
```

You can get the revised conditional probabilities associated with this network by running the following code:

```
proc print data=mycas.network noobs label;
   var _parentnode_ _parentcond_ _childnode_ _childcond_ _value_; 
   where _type_="PROBABILITY``; 
run;
```

Figure 5.5 shows the revised conditional probabilities, given this network.
**Figure 5.5** Probability Table with INNETWORK= Option

<table>
<thead>
<tr>
<th>Parent Node</th>
<th>Parent Condition</th>
<th>Child Node</th>
<th>Child Condition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Treatment A</td>
<td></td>
<td></td>
<td></td>
<td>0.33333</td>
</tr>
<tr>
<td>Treatment B</td>
<td></td>
<td></td>
<td></td>
<td>0.33333</td>
</tr>
<tr>
<td>Treatment P</td>
<td></td>
<td></td>
<td></td>
<td>0.33333</td>
</tr>
<tr>
<td>Age &lt;67</td>
<td></td>
<td></td>
<td></td>
<td>0.25397</td>
</tr>
<tr>
<td>Age &lt;75</td>
<td></td>
<td></td>
<td></td>
<td>0.50794</td>
</tr>
<tr>
<td>Age &gt;=75</td>
<td></td>
<td></td>
<td></td>
<td>0.23810</td>
</tr>
<tr>
<td>Treatment A</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.14286</td>
</tr>
<tr>
<td>Age &lt;67</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.14286</td>
</tr>
<tr>
<td>Treatment A</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.85714</td>
</tr>
<tr>
<td>Age &lt;67</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.85714</td>
</tr>
<tr>
<td>Treatment A</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.21429</td>
</tr>
<tr>
<td>Age &lt;75</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.21429</td>
</tr>
<tr>
<td>Treatment A</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.78571</td>
</tr>
<tr>
<td>Age &lt;75</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.78571</td>
</tr>
<tr>
<td>Treatment A</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.80000</td>
</tr>
<tr>
<td>Age &gt;=75</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.80000</td>
</tr>
<tr>
<td>Treatment A</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.20000</td>
</tr>
<tr>
<td>Age &gt;=75</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.20000</td>
</tr>
<tr>
<td>Treatment B</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.14286</td>
</tr>
<tr>
<td>Age &lt;67</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.14286</td>
</tr>
<tr>
<td>Treatment B</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.85714</td>
</tr>
<tr>
<td>Age &lt;67</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.85714</td>
</tr>
<tr>
<td>Treatment B</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.10000</td>
</tr>
<tr>
<td>Age &lt;75</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.10000</td>
</tr>
<tr>
<td>Treatment B</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.90000</td>
</tr>
<tr>
<td>Age &lt;75</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.90000</td>
</tr>
<tr>
<td>Treatment B</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.66667</td>
</tr>
<tr>
<td>Age &gt;=75</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.66667</td>
</tr>
<tr>
<td>Treatment B</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.33333</td>
</tr>
<tr>
<td>Age &gt;=75</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.33333</td>
</tr>
<tr>
<td>Treatment P</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.71429</td>
</tr>
<tr>
<td>Age &lt;67</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.71429</td>
</tr>
<tr>
<td>Treatment P</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.28571</td>
</tr>
<tr>
<td>Age &lt;67</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.28571</td>
</tr>
<tr>
<td>Treatment P</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.61538</td>
</tr>
<tr>
<td>Age &lt;75</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.61538</td>
</tr>
<tr>
<td>Treatment P</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.38462</td>
</tr>
<tr>
<td>Age &lt;75</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.38462</td>
</tr>
<tr>
<td>Treatment P</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.83333</td>
</tr>
<tr>
<td>Age &gt;=75</td>
<td>Pain Yes</td>
<td></td>
<td></td>
<td>0.83333</td>
</tr>
<tr>
<td>Treatment P</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.16667</td>
</tr>
<tr>
<td>Age &gt;=75</td>
<td>Pain No</td>
<td></td>
<td></td>
<td>0.16667</td>
</tr>
</tbody>
</table>
Syntax: BNET Procedure

The following statements are available in the BNET procedure:

```
PROC BNET < options > ;
  AUTOTUNE < options > ;
  CODE FILE = filename ;
  FREQ variable ;
  ID variables ;
  INPUT variables / < LEVEL=INTERVAL | NOMINAL > ;
  OUTPUT OUT = CAS-libref.data-table < option > ;
  PARTITION partition-option ;
  SAVESTATE RSTORE = CAS-libref.data-table ;
  TARGET variable ;
```

The PROC BNET statement, the TARGET statement, and the INPUT statement are required. You can specify only one TARGET statement, but you can specify multiple INPUT statements. The following sections describe the PROC BNET statement and then describe the other statements in alphabetical order.

PROC BNET Statement

```
PROC BNET < options > ;
```

The PROC BNET statement invokes the procedure. Table 5.1 summarizes important options in the PROC BNET statement by function.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Options</strong></td>
<td>For detailed information about each setting, see the detailed documentation for the DATA step.</td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the input data set</td>
</tr>
<tr>
<td>MISSINGINT=</td>
<td>Specifies how to handle missing values for interval variables</td>
</tr>
<tr>
<td>MISSINGNOM=</td>
<td>Specifies how to handle missing values for nominal variables</td>
</tr>
<tr>
<td>NUMBIN=</td>
<td>Specifies the number of binning levels for interval variables</td>
</tr>
<tr>
<td>PRESCREENING=</td>
<td>Specifies the initial screening for the input variables</td>
</tr>
<tr>
<td>VARSELECT=</td>
<td>Specifies the selection for the input variables</td>
</tr>
<tr>
<td><strong>Independence Test Options</strong></td>
<td>Specifies the significance level for independence tests by using chi-square or G-square statistics</td>
</tr>
<tr>
<td>ALPHA=</td>
<td>Specifies the method to use for independence tests</td>
</tr>
<tr>
<td>INDEPTTEST=</td>
<td>Specifies the significance level for independence tests by using mutual information</td>
</tr>
<tr>
<td>MIALPHA=</td>
<td>Specifies the CAS table that contains the included and excluded arcs that are defined using pairs of parent and child variables</td>
</tr>
</tbody>
</table>

Table 5.1 PROC BNET Statement Options
Table 5.1  continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXPARENTS=</td>
<td>Specifies the maximum number of parents for each node in the network</td>
</tr>
<tr>
<td>PARENTING=</td>
<td>Specifies the structure learning method</td>
</tr>
<tr>
<td>STRUCTURE=</td>
<td>Specifies the network structure type</td>
</tr>
</tbody>
</table>

Model Selection Options
BESTMODEL  Requests that the best model be selected

Network Output Options
OUTNETWORK=  Specifies the output CAS table that contains the final network and the associated probabilities

You can specify the following options:

ALPHA=number
specifies the significance level for independence tests by using chi-square or G-square statistics. The valid range is 0 to 1, inclusive. 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 BESTMODEL option, PROC BNET uses the first number and ignores the remaining numbers.

By default, ALPHA=0.05.

BESTMODEL
selects the best model by using a validation data subset. You can specify the validation data subset by using the PARTITION statement. If you specify this option, you can specify multiple values for the ALPHA=, PRESCREENING=, VARSELECT=, STRUCTURE=, and PARENTING= options. PROC BNET uses the misclassification errors that arise from the validation data to automatically decide the best set of parameter values among these options. If you supply multiple values of these options in the absence of the BESTMODEL option, then only the first value among the multiple values is used.

By default, the final network is produced for only one set of parameters. You must specify the BESTMODEL option to invoke a search for the best network by using all possible combinations of the specified parameters.

NOTE: If you specify the BESTMODEL option in the PROC BNET statement, then the AUTOTUNE statement is ignored.

DATA=CAS-libref.data-table
names the input data table for PROC BNET to use. CAS-libref.data-table is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about CAS-libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 51.

data-table specifies the name of the input data table.
INDEPTEST=ALL | CHIGSQUARE | CHISQUARE | GSQUARE | MI

specifies the method to use for independence tests. You can specify the following values:

ALL uses the chi-square, the G-square statistics, and the normalized mutual information for independence tests. A variable is independent of the target if both the $p$-values of the chi-square and the G-square statistics are greater than the specified ALPHA= value and the normalized mutual information is less than the value that is specified in MIALPHA= option.

CHIGSQUARE uses both the chi-square and the G-square statistics for independence tests. A variable is independent of the target if both the $p$-values of the chi-square and the G-square statistics are greater than the specified ALPHA= value.

CHISQUARE uses the chi-square statistics for independence tests. A variable is independent of the target if the $p$-value of the statistics is greater than the specified ALPHA= value.

GSQUARE uses the G-square statistics for independence tests. A variable is independent of the target if the $p$-value of the statistics is greater than the specified ALPHA= value.

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 that is specified in the MIALPHA= option.

By default, INDEPTEST=CHISQUARE.

INNETWORK=CAS-libref.data-table

INNET=names the input CAS data table that contains the arcs defined as parent-child variable pairs to be included in or excluded from the final network. The flag column in this table indicates that the arc is an included arc if the value of flag is 1, and the arc is an excluded arc if the value of flag is 0. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 51.

MAXPARENTS=integer

specifies the maximum number of parents that are allowed for each node in the network structure. The valid range is 1 to 16, inclusive. If you specify the BESTMODEL option, PROC BNET calculates from 1 to integer and decides the best number of parents.

By default, MAXPARENTS=5.

MIALPHA=number

specifies the threshold for independence tests by using mutual information. The valid range is 0 to 1, inclusive.

By default, MIALPHA=0.05.
MISSINGINT=IGNORE | IMPUTE
specifies how to handle missing values for all interval input variables. This option applies to training
data, validation data, testing data, and any data that are used for scoring. You can specify the following
values:

- **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.

By default, MISSINGINT=IGNORE.

MISSINGNOM=IGNORE | IMPUTE | LEVEL
specifies how to handle the missing values for all nominal input variables. You can specify the
following values:

- **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.

By default, MISSINGNOM=IGNORE.

NTHREADS=number-of-threads
specifies the number of threads to use. The default is the minimum CPU count of all the nodes.

NUMBIN=integer
NBIN=integer
specifies the number of binning levels for all interval variables. PROC BNET bins each interval
variable into \( \text{integer} \) equal-width levels. The valid range of \( \text{integer} \) is 2 to 1,024, inclusive.

By default, NUMBIN=5.

OUTNETWORK=CAS-libref.data-table
OUTNET=CAS-libref.data-table
names the output CAS data table to contain the network structure and the probability distributions.
\( \text{CAS-libref.data-table} \) is a two-level name, where \( \text{CAS-libref} \) refers to the \text{caslib} and session identifier,
and \( \text{data-table} \) specifies the name of the output data table. For more information about this two-level
name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on
page 51.

PARENTING=BESTONE | BESTSET
specifies the algorithm for orienting the network structure. You can specify the following values:

- **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, PROC BNET
tests multiple sets of variables together and chooses the best set as the parents of
the node.
If you want to choose between the two methods, you can specify both of them and also specify the
BESTMODEL option. If you specify both methods but you do not specify the BESTMODEL option,
PROC BNET uses the first specified method, and ignores the other.

By default, PARENTING=BESTSET.

**PRESCREENING=0 | 1**

specifies the initial screening for the input variables. You can specify the following values:

- 0 uses all the input variables.
- 1 uses only the input variables that are dependent on the target.

If you want to choose the best model with or without prescreening, you can specify PRESCREEN-
ING=0 1 or PRESCREENING= 1 0 and also specify the BESTMODEL option. If you specify both but
you do not specify the BESTMODEL option, PROC BNET uses the first specified value, and ignores
the other.

If you specify STRUCTURE=GENERAL, then any setting of the PRESCREENING= option other
than 1 is ignored and PRESCREENING=1 is used.

By default, PRESCREENING=1.

**PRINTTARGET**

generates the “Predicted Probability Variables” table, which displays the target variable and the
predicted probability variables, and the “Predicted Target Variable” table, which displays the predicted
target variable.

By default, these two tables are not generated.

**STRUCTURE=values**

specifies the network structure. Together with the MAXPARENTS= option, this option determines
which network structure the procedure learns from the training data. You can specify one or more of
the following values, separated by spaces:

- **GENERAL | GN** learns a general Bayesian network over the target and input variables. PROC
  BNET learns a general Bayesian network by using a search algorithm that grows
  partial networks to completion by exploring among possible completions. For the
  GENERAL network structure, any specification of the PRESCREENING= option
  is ignored and PRESCREENING=1 is used. In addition, any specification of the
  V ARISELECT= option is ignored and V ARISELECT=0 is used. The meaning of
  V ARISELECT= option is explained in the following discussion of this option.
- **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, PROC BNET further determines the parents of the target, the
  links from the parents to the children, and the links among the children. When you
  specify STRUCTURE=MB, the procedure learns the Markov blanket regardless of
  the values of PRESCREENING= and V ARISELECT= options.
- **NAIVE** assumes a naive Bayesian network structure (that is, the target has a direct link
to each input variable). If MAXPARENTS=1, the structure is a naive Bayesian
network (NB). If the value of the MAXPARENTS= option is greater than 1, the
structure is a Bayesian network-augmented naive Bayesian network (BAN).
PC learns the parent-child Bayesian network structure (PC). STRUCTURE=PC differs from STRUCTURE=NAIVE 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 the PC structure.

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.

If you want to choose the best structure among several structures, you can specify multiple values in any combination, separated by spaces, and also specify the BESTMODEL option. If you specify multiple structures but you do not specify the BESTMODEL option, PROC BNET uses the first value that you specify and ignores the rest.

By default, STRUCTURE=PC.

VARSELECT=0 | 1 | 2 | 3
specifies how input variables are selected beyond the prescreening. You can specify the following values:

0 uses all input variables that remain after the initial screening is performed as specified in the PRESCREENING= option.

1 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 not rendered conditionally independent of the target given any other input variable.

2 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 not rendered conditionally independent of the target given any subset of other input variables.

3 determines the Markov blanket of the target variable and uses only the variables in the Markov blanket.

By default, VARSELECT=1.

If you specify VARSELECT=1, 2, or 3, PROC BNET automatically tests each input variable for unconditional independence of the target regardless of the value of the PRESCREENING= option. If no variables are left at a particular variable selection level, PROC BNET reverts to the previous level. For example, if you specify VARSELECT=3 and there are no variables in the Markov blanket of the target, PROC BNET uses the variables from the previous level, VARSELECT=2.

If you want to choose the best model among different levels of variable selections, you can specify any combination of values for the VARSELECT= option and also specify the BESTMODEL option. If you specify multiple values for the VARSELECT= option but you do not specify the BESTMODEL option, PROC BNET uses the first specified value and ignores the remaining values.

If you specify STRUCTURE=GENERAL, then any setting of the VARSELECT= option other than 0 is ignored and VARSELECT=0 is used.

The INNETWORK= option enables you to override the selections that are made by the PRESCREENING= and VARSELECT= options. Thus a variable that is selected by independence tests might not be
in the network if you have excluded it by using the INNETWORK= option. Similarly, a variable that is deemed independent of the target might get into the network if you have included it by using the INNETWORK= option. Hence the selections that are made using independence tests are not final, and specifications of the INNETWORK= option can override these selections.

**AUTOTUNE Statement**

```
AUTOTUNE < options > ;
```

The AUTOTUNE statement searches for the best combination of values for the ALPHA, INDEPTEST, MAXPARENTS, MIALPHA, MISSINGINT, MISSINGNOM, NUMBIN, PARENTING, PRESCREENING, STRUCTURE, and VARSELECT options in the PROC BNET statement.

Table 5.2 summarizes the *options* that you can specify in the AUTOTUNE statement. For more information about all options except the TUNINGPARAMETERS= option, see the option’s description in the section “AUTOTUNE Statement” on page 12 in Chapter 3, “Shared Concepts.” The TUNINGPARAMETERS= option is described following Table 5.2.

**NOTE:** Processing the AUTOTUNE statement is computationally expensive and requires a significant amount of time.

**NOTE:** If you specify both the AUTOTUNE statement and the BESTMODEL option in the PROC BNET statement, the AUTOTUNE statement is ignored.

### Table 5.2 AUTOTUNE Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDLOOKUP</td>
<td>Specifies that the table specified in the HISTORYTABLE= option contain the rows from the table specified in the LOOKUPTABLE= option</td>
</tr>
<tr>
<td>EVALHISTORY=</td>
<td>Specifies how to report the evaluation history of the tuner</td>
</tr>
<tr>
<td>FRACTION=</td>
<td>Specifies the fraction of observations to use for validation</td>
</tr>
<tr>
<td>HISTORYTABLE=</td>
<td>Specifies the CAS table that contains the evaluation history</td>
</tr>
<tr>
<td>KFOLD=</td>
<td>Specifies the number of folds for ( k )-fold cross validation</td>
</tr>
<tr>
<td>LIVEUPDATE</td>
<td>Specifies that the table specified in the HISTORYTABLE= option be updated at every evaluation</td>
</tr>
<tr>
<td>LOCALSEARCH</td>
<td>Enables local search optimization</td>
</tr>
<tr>
<td>LOOKUPTABLE=</td>
<td>Specifies the CAS table to use for evaluation lookup</td>
</tr>
<tr>
<td>MAXBAYES=</td>
<td>Specifies the maximum number of points in the kriging model</td>
</tr>
<tr>
<td>MAXEVALS=</td>
<td>Specifies the maximum number of evaluations</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>MAXTIME=</td>
<td>Specifies the maximum time for all iterations</td>
</tr>
<tr>
<td>MAXTRAINTIME=</td>
<td>Specifies the maximum time for a model training</td>
</tr>
<tr>
<td>NCONVITER=</td>
<td>Specifies the number of convergence iterations</td>
</tr>
<tr>
<td>NOGRIDSHUFFLE</td>
<td>Requests that the grid points not be shuffled</td>
</tr>
<tr>
<td>NPARALLEL=</td>
<td>Specifies the number of parallel sessions</td>
</tr>
<tr>
<td>NSUBSESSIONWORKERS=</td>
<td>Specifies the number of workers in parallel sessions</td>
</tr>
</tbody>
</table>
### Table 5.2 continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECTIVE=</td>
<td>Specifies the objective function</td>
</tr>
<tr>
<td>POPSIZE=</td>
<td>Specifies the population size when SEARCHMETHOD=GA or</td>
</tr>
<tr>
<td></td>
<td>SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>SAMPLESIZE=</td>
<td>Specifies the sample size when SEARCHMETHOD=LHS or</td>
</tr>
<tr>
<td></td>
<td>SEARCHMETHOD=RANDOM</td>
</tr>
<tr>
<td>SEARCHMETHOD=</td>
<td>Specifies the search method that the optimizer uses</td>
</tr>
<tr>
<td>SECONDOBJECTIVE=</td>
<td>Specifies the second objective to use for tuning</td>
</tr>
<tr>
<td>SELECTINITPOINT</td>
<td>Specifies that the tuner select the best evaluation from the lookup table</td>
</tr>
<tr>
<td>TARGETEVENT=</td>
<td>Specifies the target event for ROC-based calculations</td>
</tr>
<tr>
<td>TRAINFRACTION=</td>
<td>Specifies the fraction of observations to use for training</td>
</tr>
<tr>
<td>TUNINGPARAMETERS=</td>
<td>Specifies the custom tuning parameters</td>
</tr>
<tr>
<td>USEPARAMETERS=</td>
<td>Specifies how to handle the TUNINGPARAMETERS= option</td>
</tr>
</tbody>
</table>

TUNINGPARAMETERS=(suboption | . . . | < suboption>)
TUNEPARMS=(suboption | . . . | < suboption>)

specifies which parameters to tune and which ranges to tune over. If you specify USEPARAMETERS=STANDARD, this option is ignored.

You can specify one or more of the following suboptions:

#### ALPHA (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)

specifies information about the significance level for independence tests by using chi-square or G-square statistics, where number or any value in the value-list must be a real number in the range 0 to 1, inclusive. For more information, see the ALPHA= option in the PROC BNET statement.

You can specify the following additional suboptions:

**LB=number**

specifies the minimum significance level to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=0.01.

**UB=number**

specifies the maximum significance level to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=0.99.

**VALUES=value-list**

specifies a list of significance levels to consider during tuning, where value-list is a space-separated list of numbers in the range 0 to 1. If you specify this suboption, you cannot specify either the LB= or UB= suboption.
INIT=number
specifies the initial significance level for the tuner to use.
By default, INIT=0.05.

EXCLUDE
excludes the significance level from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

INDEPT (VALUES=value-list INIT=value EXCLUDE)
specifies information about the method to use for independence tests. For more information, see the INDEPT= option in the PROC BNET statement.
You can specify the following additional suboptions:

VALUES=value-list
specifies a list of methods to use for independence tests, where value-list is a space-separated list of CHIGSQUARE, CHISQUARE, MI, and GSQUARE.

INIT=CHIGSQUARE | CHISQUARE | MI | GSQUARE
specifies the initial method to use for independence tests.
By default, INIT=CHIGSQUARE.

EXCLUDE
excludes the INDEPT suboption from the tuning process. If you specify EXCLUDE, any specified VALUES= and INIT= suboptions are ignored.

MAXPARENTS (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies information about the maximum number of parents that are allowed for each node in the network structure. The valid range is 1 to 16, inclusive. For more information, see the MAXPARENTS= option in the PROC BNET statement.
You can specify the following additional suboptions:

LB=number
specifies a lower bound on the maximum number of parents to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.
By default, LB=1.

UB=number
specifies an upper bound on the maximum number of parents to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.
By default, UB=16.

VALUES=value-list
specifies a list of values to consider for the maximum number of parents in the network structure, where value-list is a space-separated list of numbers. If you specify this suboption, you cannot specify the LB= and UB= suboptions.
**INIT=number**
specifies the initial maximum number of parents in the network structure.

By default, INIT=5.

**EXCLUDE**
excludes the maximum number of parents from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**MIALPHA (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**
specifies information about the threshold for independence tests by using mutual information. For more information, see the MIALPHA= option in the PROC BNET statement.

You can specify the following additional suboptions:

**LB=number**
specifies the minimum threshold value to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=0.

**UB=number**
specifies the maximum threshold value to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=1.

**VALUES=value-list**
specifies a list of threshold values to consider during tuning, where value-list is a space-separated list of numbers in the range 0 to 1. If you specify this suboption, you cannot specify the LB= and UB= suboptions.

**INIT=number**
specifies the initial threshold value for the tuner to use.

By default, INIT=0.05.

**EXCLUDE**
excludes the threshold value from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**MISSINGINT (VALUES=value-list INIT=value EXCLUDE)**
specifies information about how to handle missing values for interval variables during the tuning process. For more information, see the MISSINGINT= option in the PROC BNET statement.

You can specify the following additional suboptions:

**VALUES=value-list**
specifies a list of values for the tuner to try for the MISSINGINT suboption, where you can specify IMPUTE or IGNORE (or both) in a space-separated value-list.
**INIT=** IMPUTE | IGNORE
specifies whether to start tuning by imputing or ignoring missing values of interval variables.

By default, INIT=IGNORE.

**EXCLUDE**
excludes the MISSINGINT suboption from the tuning process. If you specify EXCLUDE, any specified VALUES= and INIT= suboptions are ignored.

**MISSINGNOM (VALUES=** value-list **INIT=** value **EXCLUDE)**
specifies information about how to handle missing values for nominal variables during the tuning process. For more information, see the MISSINGNOM= option in the PROC BNET statement.

You can specify the following additional suboptions:

VALUES=** value-list
specifies a list of values for the tuner to handle missing values for nominal variables, where value-list is a combination of one or more of the following values in a space-separated list: IMPUTE, IGNORE, and LEVEL.

INIT=** IMPUTE | IGNORE | LEVEL
specifies the initial value to use in tuning the MISSINGNOM suboption.

By default, INIT=IGNORE.

**EXCLUDE**
excludes the MISSINGNOM suboption from the tuning process. If you specify EXCLUDE, any specified VALUES= and INIT= suboptions are ignored.

**NUMBIN (LB=** number **UB=** number **VALUES=** value-list **INIT=** number **EXCLUDE)**
specifies information about the number of binning levels for all interval variables. The valid range is 2 to 20. For more information, see the NUMBIN= option in the PROC BNET statement.

You can specify the following additional suboptions:

**LB=** number
specifies a lower bound of binning levels to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=2.

**UB=** number
specifies an upper bound of binning levels to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=20.

VALUES=** value-list
specifies a list of values of binning levels to consider, where value-list is a space-separated list of numbers. If you specify this suboption, you cannot specify the LB= and UB= suboptions.
INIT=number
specifies the initial number of binning levels.

By default, INIT=5.

EXCLUDE
excludes the number of binning levels from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

PARENTING (VALUES=value-list INIT=value EXCLUDE)
specifies information about the algorithm for orienting the network structure. For more information, see the PARENTING= option in the PROC BNET statement.

You can specify the following additional suboptions:

VALUES=value-list
specifies a list of algorithms for orienting the network structure during tuning, where value-list is a space-separated list of BESTONE and BESTSET.

INIT=BESTONE | BESTSET
specifies the initial algorithm for orienting the network structure for the tuner to use.

By default, INIT=BESTSET.

EXCLUDE
excludes the algorithm selection for orienting the network structure from the tuning process. If you specify this suboption, any specified VALUES= and INIT= suboptions are ignored.

PRESCREENING (VALUES=value-list INIT=value EXCLUDE)
specifies information about the initial screening for the input variables. For more information, see the PRESCREENING= option in the PROC BNET statement.

You can specify the following additional suboptions:

VALUES=value-list
specifies a list of initial screening values for the input variables, where value-list is a space-separated list of 0 and 1.

INIT=number
specifies the initial screening for the input variables.

By default, INIT=1.

EXCLUDE
excludes the initial screening for the input variables from the tuning process. If you specify this suboption, any specified VALUES= and INIT= suboptions are ignored.

STRUCTURE (VALUES=value-list INIT=value EXCLUDE)
specifies information about the network structure. For more information, see the STRUCTURE= option in the PROC BNET statement.

You can specify the following additional suboptions:
VALUES=value-list
   specifies a list of structures, where value-list is a space-separated list of MB, NAIVE, PC, and TAN. You cannot specify the value GENERAL (or GN) for the VALUES= or INIT= suboption in the AUTOTUNE statement.

INIT=MB | NAIVE | PC | TAN
   specifies the initial network structure.
   By default, INIT=PC.

EXCLUDE
   excludes the network structure from the tuning process. If you specify this suboption, any specified VALUES= and INIT= suboptions are ignored.

VARSELECT (VALUES=value-list INIT=value EXCLUDE)
   specifies information about how to select input variables during the tuning process after the prescreening. For more information, see the VARSELECT= option in the PROC BNET statement.
   You can specify the following additional suboptions:

VALUES=value-list
   specifies a list of input variables to consider for the tuning process after the prescreening, where value-list is a space-separated list of 0, 1, 2, and 3.

INIT=number
   specifies the initial value to consider for the tuning process after the prescreening.
   By default, INIT=1.

EXCLUDE
   excludes the VARSELECT suboption from the tuning process. If you specify EXCLUDE, any specified VALUES= and INIT= suboptions are ignored.

---

**CODE Statement**

```sas
CODE FILE=filename ;
```

The CODE statement is optional in PROC BNET. If you use a CODE statement, SAS DATA step code is generated and stored in a file that can be used for scoring purposes.
FREQ Statement

FREQ variable ;

The variable in the FREQ statement identifies a numeric variable in the data set that contains the frequency of occurrence for each observation. The BNET procedure treats each observation as if it appeared \( n \) times, where \( n \) is the value of the variable for the observation. If \( n \) is not an integer, it is truncated to an integer. If \( n \) is less than 1 or is missing, the observation is ignored. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

ID Statement

ID variables ;

The optional ID statement lists one or more variables from the input data set to be copied to the prediction output data set. The ID statement accepts both numeric and character variables. The variables in an ID statement can also appear in any other statements.

INPUT Statement

INPUT variables </LEVEL=INTERVAL | NOMINAL > ;

The INPUT statement specifies one or more variables as input variables. You can specify multiple INPUT statements. If the INPUT statement contains a duplicate variable, PROC BNET considers the variable as input variable and ignores the repetitions. You can specify the following option in each INPUT statement:

LEVEL=INTERVAL | NOMINAL

specifies the type of all the variables in the current INPUT statement. You can specify the following values:

NOMINAL treats all the variables in the current INPUT statement as nominal variables.

INTERVAL treats all the variables in the current INPUT statement as interval variables.

By default, LEVEL=INTERVAL for numeric variables and LEVEL=NOMINAL for categorical variables.
**OUTPUT Statement**

```output
OUTPUT OUT=CAS-libref.data-table < options > ;
```

The OUTPUT statement creates a data table to contain the predicted target values of the input table.

You must specify the following option:

**OUT=** `CAS-libref.data-table`

names the output data table for PROC BNET to use. You must specify this option before any other options. `CAS-libref.data-table` is a two-level name, where

- **CAS-libref** refers to a collection of information that is defined in the LIBNAME statement and includes the `caslib`, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 51.

- **data-table** specifies the name of the output data table.

This table includes variables that are specified either in the `COPYVARS=` option or in the ID statement. If you specify `PARTITION` statement, the output includes one more column, _ROLE_. _ROLE_ is a reserved name. If it exist in the input data table and you specify it in the `COPYVARS=` option or in the ID statement, you need to use the `ROLE=` option to change the generated column’s name.

You can also specify the following `options`:

- `COPYVAR=variable`  
  `COPYVARS=(variables)`  
  lists one or more variables from the input data table to be copied to the output data table.

- `ROLE=rolename`  
  renames the generated column _ROLE_ in the output data table to the specified role.

---

**PARTITION Statement**

```partition
PARTITION partition-option ;
```

The PARTITION statement specifies how observations in the input data set are logically partitioned into disjoint subsets for model training, validation, and testing. For more information, see the section “Using Validation and Test Data” on page 21 in Chapter 3, “Shared Concepts.” Either you can designate a variable in the input data table and a set of formatted values of that variable to determine the role of each observation, or you can specify proportions to use for randomly assigning observations to each role.

You must specify exactly one of the following `partition-options`:
Chapter 5: The BNET Procedure

FRACTION(< TEST=fraction > < VALIDATE=fraction > < SEED=number >)
randomly assigns specified proportions of the observations in the input data table to the roles. You specify the proportions for testing and validation by using the TEST= and VALIDATE= suboptions. If you specify both the TEST= and VALIDATE= suboptions, then the sum of the specified fractions must be less than 1 and the remaining fraction of the observations are assigned to the training role. The SEED= option specifies an integer that is used to start the pseudorandom number generator for random partitioning of data for training, testing, and validation. If you do not specify SEED=number or if number is less than or equal to 0, the seed is generated by reading the time of day from the computer’s clock.

ROLE=variable (< TEST=value > < TRAIN=value > < VALIDATE=value >)
ROLEVAR=variable (< TEST=value > < TRAIN=value > < VALIDATE=value >)

names the variable in the input data table whose values are used to assign roles to each observation. This variable cannot also appear as an analysis variable in other statements or options. The TEST=, TRAIN=, and VALIDATE= suboptions specify the formatted values of this variable that are used to assign observation roles. If you do not specify the TRAIN= suboption, then all observations whose role is not determined by the TEST= or VALIDATE= suboption are assigned to the training role.

SAVESTATE Statement

SAVESTATE RSTORE=CAS-libref.data-table ;

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 4, “The ASTORE Procedure.”

You must specify the following option:

RSTORE=CAS-libref.data-table
specifies a data table in which to save the analytic store for the model. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 51.

TARGET Statement

TARGET variable ;

The TARGET statement names the variable that PROC BNET predicts. PROC BNET treats the TARGET variable as nominal.

The target values are levelized in descending order. If multiple target statements are supplied then the first statement applies.
Details: BNET Procedure

Independence Tests

Both variable selection and structure learning require either independence tests between two variables or conditional independence tests given some other variables. PROC BNET 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

$$G^2 = 2 \sum_{i=1}^{r} \sum_{j=1}^{c} O_{ij} \ln \left( \frac{O_{ij}}{E_{ij}} \right)$$

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 considered to be independent.

The mutual information between X and Y is defined as

$$I(X, Y) = \sum_{x=1}^{r} \sum_{y=1}^{c} p(x, y) \ln \left( \frac{p(x, y)}{p(x)p(y)} \right)$$

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 PROC BNET options are related to independence tests:

- The **INDEPTEST=** option specifies which test statistic or combination of them to use.
- The **ALPHA=** option specifies the significance level for the chi-square and the G-square statistics.
- The **MIALPHA=** option 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 PROC BNET 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 to the product of the local probability distributions; that is, $p(U) = \prod_{X \in U} p(X | \pi(X))$, 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.

PROC BNET 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**). When a **STRUCTURE** option is set to **GENERAL** then **VARSELECT=0** is used and any other setting of **VARSELECT** is ignored. Similarly, when a **STRUCTURE** option is set to **GENERAL** then **PRESCREENING=1** is used and any other setting of **PRESCREENING** is ignored.

Specifying the **INNETWORK=** option in the PROC BNET statement enables you to override the variable selections that are made by the **PRESCREENING=** and **VARSELECT=** options. Thus, a variable that is selected by independence tests might not be in the network if you have excluded it by using the **INNETWORK=** option. Similarly, a variable that is deemed independent of the target might get in the network if you have included it by using the **INNETWORK=** option. Hence, the selections that are made using independence tests are not final, and the **INNETWORK=** option specifications can override these selections.

PROC BNET uses specialized data structures to efficiently compute the contingency tables for any variable combination, and it uses dynamic candidate generation to reduce the false candidates (variable combinations).
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.

PROC BNET 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

\[
\text{BIC}(G, D) = N \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} p(\pi_{ij}) p(X_i = v_{ik} | \pi_{ij}) \ln p(X_i = v_{ik} | \pi_{ij}) - \frac{M}{2} \ln N
\]

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 \( k \)th value of \( X_i \), \( q_i \) is the number of value combinations of \( X_i \)'s parents, \( \pi_{ij} \) is the \( j \)th 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.

PROC BNET 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 \( S = \{Z\} \cup S' \), \( S' \subseteq U - \{X, Y, Z\} \), 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, \( X \leftarrow Y \leftarrow Z \), \( X \leftarrow Y \rightarrow Z \), and \( X \leftarrow Y \rightarrow Z \) belong to the same equivalence class. In these cases, PROC BNET uses the BIC score to determine the directions of the edges.

PROC BNET learns different types of network structures: naive Bayesian (NB), tree-augmented naive (TAN), Bayesian network-augmented naive (BAN), parent-child Bayesian network (PC), Markov blanket (MB), and general Bayesian network (GENERAL). Based on the network structure that is specified, it uses different algorithms. For example, if you specify STRUCTURE=TAN, the procedure uses the maximum spanning tree to learn the tree structure, where the weight for an edge is the mutual information between the two nodes. PROC BNET uses either the BESTONE or BESTSET value of the PARENTING= option to learn the other network structures (BAN, PC, MB).

PROC BNET 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

\[
\text{BIC}(X, T) = \max(\text{BIC}(X \rightarrow T), \text{BIC}(T \rightarrow X))
\]

where \( \text{BIC}(X \rightarrow T) \) is the BIC score when \( X \) is the parent of \( T \) (ignoring all the remaining variables) and \( \text{BIC}(T \rightarrow X) \) is the BIC score when \( X \) is the child of \( T \) (ignoring all the remaining variables).

PROC BNET 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. PROC BNET 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 value 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.

The general Bayesian network relaxes the assumption of a naive network that all input variables are directly connected to the target variable. It thus requires more computing but is capable of producing networks that you can interpret by using intuition and prior knowledge of network connections. Combined with the INNETWORK= option, you can use the knowledge that you glean from the data together with your subject expertise to arrive at a satisfactory network.

**Parameter Learning**

Parameter learning determines the probability distribution for each node in a network structure. In PROC BNET, 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

\[
\arg\max_c p(T = c | x_1, x_2, \ldots, x_{n-1}) = p(x_1, x_2, \ldots, x_{n-1} | T = c) \times K = \prod_i p(x_i | \pi(X_i)) \times K
\]

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 \(p(x_i | \pi(X_i))\), PROC BNET uses an additive smoothing technique:

\[
\hat{p}(x_i | \pi(X_i)) = \frac{\text{counts_of}(x_i, \pi(X_i)) + \alpha}{\text{counts_of}(\pi(X_i)) + \alpha \times n_i}
\]

where \(\alpha\) is the smoothing parameter (\(\alpha = 0\) corresponds to no smoothing, and PROC BNET 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. PROC BNET still smooths the probabilities when every class is observed.
Displayed Output

The following sections describe the output that PROC BNET 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 total 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.

Variable Selection Report

The “Variable Selection Report” table contains the variable selection results as determined using independence tests. These results can be overridden by the INNETWORK= option in the PROC BNET statement. If a roll-back occurs because the desired list is empty, this report shows the test results for desired list. This will help you in determining the reason for roll-back.

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.
ODS Table Names

Each table that the BNET procedure creates has a name associated with it. You must use this name to refer to the table when you use the ODS statements. These names are listed in Table 5.3.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FitStatistics</td>
<td>Contains the fit statistics of the network</td>
<td>PROC BNET</td>
<td>Default</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Contains the initial training settings, such as significance level, structure, and number of bins</td>
<td>PROC BNET</td>
<td>Default</td>
</tr>
<tr>
<td>NObs</td>
<td>Contains the number of observations for training, validation and testing, and so on</td>
<td>PROC BNET</td>
<td>Default</td>
</tr>
<tr>
<td>PredIntoName</td>
<td>Predicted target variable</td>
<td>PROC BNET</td>
<td>PRINTTARGET</td>
</tr>
<tr>
<td>PredProbName</td>
<td>Predicted probability variables</td>
<td>PROC BNET</td>
<td>PRINTTARGET</td>
</tr>
<tr>
<td>ValidInfo</td>
<td>Contains the validation results</td>
<td>PROC BNET</td>
<td>BESTMODEL</td>
</tr>
<tr>
<td>VarInfo</td>
<td>Contains the variable information such as number of levels, number of missing values, and so on</td>
<td>PROC BNET</td>
<td>Default</td>
</tr>
<tr>
<td>VarLevel</td>
<td>Contains the details of each level of the variables</td>
<td>PROC BNET</td>
<td>Default</td>
</tr>
<tr>
<td>VarOrder</td>
<td>Contains the order of the input variables</td>
<td>PROC BNET</td>
<td>Default</td>
</tr>
<tr>
<td>VarSelect</td>
<td>Contains the variable selection results using independence tests</td>
<td>PROC BNET</td>
<td>Default</td>
</tr>
</tbody>
</table>
Examples: BNET Procedure

Example 5.1: Naive Bayesian Network

This example shows how you can use PROC BNET to learn a naive Bayesian network for the Iris data that is available in the Sashelp library. The following DATA step loads the Iris data into your CAS session. This DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```sas
data mycas.iris;
  set sashelp.iris;
run;
```

The following statements specify MAXPARENTS=1, PRESCREENING=0, and VARSELECT=0 to request that PROC BNET use only one parent for each node and use all the input variables:

```sas
proc bnet data=mycas.iris numbin=3 structure=Naive maxparents=1
  prescreening=0 varselect=0
  outnetwork=mycas.network;
  target Species;
  input PetalWidth PetalLength SepalLength SepalWidth/level=interval;
run;
```

The following statements produce Output 5.1.1, which shows the network structure. In the structure, Species is the parent of PetalWidth, PetalLength, SepalLength, and SepalWidth.

```sas
proc print data=mycas.network noobs label;
  var _parentnode_ _childnode_;
  where _type_="STRUCTURE";
run;
```

Output 5.1.1  Naive Bayesian Network Structure

<table>
<thead>
<tr>
<th>Parent Node</th>
<th>Child Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Species</td>
<td>PetalLength</td>
</tr>
<tr>
<td>Species</td>
<td>PetalWidth</td>
</tr>
<tr>
<td>Species</td>
<td>SepalLength</td>
</tr>
<tr>
<td>Species</td>
<td>SepalWidth</td>
</tr>
</tbody>
</table>
Example 5.2: Tree-Augmented Naive Bayesian Network

This example also uses the Iris data set that is available in the Sashelp library. In the following statements, STRUCTURE=TAN results in a tree-augmented Bayesian network:

```sas
data mycas.iris;
set sashelp.iris;
run;

proc bnet data=mycas.iris numbin=3 structure=TAN
  prescreening=0 varselect=0
  outnetwork=mycas.network;
  target Species;
  input PetalWidth PetalLength SepalLength SepalWidth/level=interval;
run;
```

The following statements produce Output 5.2.1, which shows the network structure. In the structure, Species is a parent of PetalWidth, PetalLength, SepalLength, and SepalWidth. In addition, PetalWidth is a parent of PetalLength, SepalWidth is a parent of SepalLength, and PetalWidth is a parent of SepalWidth.

```sas
proc print data=mycas.network noobs label;
  var _parentnode_ _childnode_;
  where _type_="STRUCTURE";
run;
```

**Output 5.2.1** TAN Network Structure

<table>
<thead>
<tr>
<th>Parent Node</th>
<th>Child Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Species</td>
<td>PetalLength</td>
</tr>
<tr>
<td>PetalWidth</td>
<td>PetalLength</td>
</tr>
<tr>
<td>Species</td>
<td>PetalWidth</td>
</tr>
<tr>
<td>Species</td>
<td>SepalLength</td>
</tr>
<tr>
<td>PetalLength</td>
<td>SepalLength</td>
</tr>
<tr>
<td>Species</td>
<td>SepalWidth</td>
</tr>
<tr>
<td>SepalLength</td>
<td>SepalWidth</td>
</tr>
</tbody>
</table>

Example 5.3: Parent-Child Bayesian Network

This example also uses the Iris data set that is available in the Sashelp library. In the following statements, STRUCTURE=PC results in a parent-child Bayesian network:

```sas
data mycas.iris;
  set sashelp.iris;
run;

proc bnet data=mycas.iris numbin=3 structure=PC
  prescreening=0 varselect=0
  outnetwork=mycas.network;
  target Species;
```
Example 5.4: Markov Blanket

This example uses the HMEQ sample data set that is available in the Sampsio library to learn a Markov blanket Bayesian network, which is specified by STRUCTURE=MB:

```plaintext
data mycas.hmeq;
  set sampsio.hmeq;
run;

proc bnet data=mycas.hmeq indeptest=MI mialpha=0.2 structure=MB nbin=5
  missingint=IMPUTE missingnom=LEVEL
  outnetwork=mycas.network;
  target Bad;
  input Reason Job Delinq Derog Ninq/level=nominal;
  input Loan Mortdue Value Yoj Clage Clno Debtinc/level=interval;
run;
```

The following statements produce Output 5.4.1, which shows the network structure. In the structure, Bad is a parent of Delinq and Derog, Delinq and Ninq are the other parents of Derog, and Ninq is the other parent of Delinq.

```plaintext
proc print data=mycas.network noobs label;
  var _parentnode_ _childnode_;
  where _type_="STRUCTURE";
run;
```

### Output 5.3.1 Parent-Child Network Structure

<table>
<thead>
<tr>
<th>Parent Node</th>
<th>Child Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>PetalLength</td>
<td>Species</td>
</tr>
<tr>
<td>Species</td>
<td>PetalWidth</td>
</tr>
<tr>
<td>Species</td>
<td>SepalLength</td>
</tr>
<tr>
<td>Species</td>
<td>SepalWidth</td>
</tr>
</tbody>
</table>

---

Example 5.4: Markov Blanket

The following statements produce Output 5.3.1, which shows the network structure. In the structure, PetalLength is the parent of Species, and Species is the parent of PetalWidth, SepalLength, and SepalWidth.

```plaintext
input PetalWidth PetalLength SepalLength SepalWidth/level=interval;
run;
```

The following statements produce Output 5.3.1, which shows the network structure. In the structure, PetalLength is the parent of Species, and Species is the parent of PetalWidth, SepalLength, and SepalWidth.

```plaintext
proc print data=mycas.network noobs label;
  var _parentnode_ _childnode_;
  where _type_="STRUCTURE";
run;
```
Example 5.5: Bayesian Network—Augmented Naive Bayesian Network

This example also uses the HMEQ sample data set that is available in the Sampsio library to learn a BAN structure, which is specified by STRUCTURE=NAIVE:

```plaintext
data mycas.hmeq;
  set sampsio.hmeq;
run;

proc bnet data=mycas.hmeq numbin=10 alpha=0.1 structure=Naive
  missingint=IMPUTE missingnom=LEVEL
  outnetwork=mycas.network;
  target Bad;
  input Reason Job Delinq Derog Ninq/level=nominal;
  input Loan Mortdue Value Yoj Clage Clno Debtinc/level=interval;
run;
```

The following statements produce Output 5.5.1, which shows the network structure. In the structure, Bad is a parent of Delinq, Derog, Job, Ninq, Clage, Clno, Loan, and Mortdue. In addition, Delinq is the other parent of Derog, and Job is a parent of both Mortdue and Clno.

```plaintext
proc print data=mycas.network noobs label;
  var _parentnode_ _childnode_
  where _type_="STRUCTURE";
run;
```

Output 5.5.1 BAN Network Structure

<table>
<thead>
<tr>
<th>Parent Node</th>
<th>Child Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD</td>
<td>DELINQ</td>
</tr>
<tr>
<td>BAD</td>
<td>DEROG</td>
</tr>
<tr>
<td>DELINQ</td>
<td>DEROG</td>
</tr>
<tr>
<td>BAD</td>
<td>JOB</td>
</tr>
<tr>
<td>BAD</td>
<td>NINQ</td>
</tr>
<tr>
<td>BAD</td>
<td>CLAGE</td>
</tr>
<tr>
<td>BAD</td>
<td>CLNO</td>
</tr>
<tr>
<td>JOB</td>
<td>CLNO</td>
</tr>
<tr>
<td>BAD</td>
<td>LOAN</td>
</tr>
<tr>
<td>BAD</td>
<td>MORTDUE</td>
</tr>
<tr>
<td>JOB</td>
<td>MORTDUE</td>
</tr>
</tbody>
</table>
Example 5.6: Bayesian Network—General Network

This example also uses the HMEQ sample data set that is available in the Sampsio library to learn a general structure, which is specified by STRUCTURE=GN:

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

proc bnet data=mycas.hmeq numbin=10 alpha=0.1 structure=GN
  missingint=IMPUTE missingnom=LEVEL
  outnetwork=mycas.network;
  target Bad;
  input Reason Job Delinq Derog Ninq/level=nominal;
  input Loan Mortdue Value Yoj Clage Clno Debtinc/level=interval;
run;
```

The following statements produce the table in Output 5.6.1, which shows the network structure. In the structure, Bad is embedded in a general network, showing its relationship to other variables.

```
proc print data=mycas.network noobs label;
  var _parentnode_ _childnode_;
  where _type_="STRUCTURE";
run;
```

Output 5.6.1  General Network Structure

<table>
<thead>
<tr>
<th>Parent Node</th>
<th>Child Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELINQ</td>
<td>BAD</td>
</tr>
<tr>
<td>DEROG</td>
<td>DELINQ</td>
</tr>
<tr>
<td>CLNO</td>
<td>JOB</td>
</tr>
<tr>
<td>DEROG</td>
<td>NINQ</td>
</tr>
<tr>
<td>LOAN</td>
<td>REASON</td>
</tr>
<tr>
<td>BAD</td>
<td>CLAGE</td>
</tr>
<tr>
<td>CLAGE</td>
<td>CLNO</td>
</tr>
<tr>
<td>BAD</td>
<td>DEBTINC</td>
</tr>
<tr>
<td>VALUE</td>
<td>LOAN</td>
</tr>
<tr>
<td>VALUE</td>
<td>MORTDUE</td>
</tr>
<tr>
<td>JOB</td>
<td>VALUE</td>
</tr>
<tr>
<td>REASON</td>
<td>YOJ</td>
</tr>
</tbody>
</table>
Example 5.7: Model Selection

This example uses the German Credit sample data that is available in the Sampsio library to learn the best Bayesian network model among all network structures: naive, TAN, PC, and MB, with or without variable selection. PROC BNET also tries to choose the best value for the MAXPARENTS= option. About 30% of the input data is used for validation.

```sas
data dmagecr_part;
  set sampsio.dmagecr;
  seed=12345;
  if ranuni(seed) < 0.7 then partind=1;
  else partind=0;
  id=_N_;
run;

data mycas.dmagecr;
  set dmagecr_part;
run;

proc bnet data=mycas.dmagecr numbin=10 alpha=0.05
  structure=Naive TAN PC MB varselect=0 1 bestmodel
  outnetwork=mycas.network;
  target Good_Bad;
  input Checking History Purpose Savings Employed Installp Marital Coapp
    Resident Property Other Housing Existcr Job Depends Telephon
    Foreign/level=nominal;
  input Age Amount Duration/level=interval;
  partition rolevar= PartInd (TRAIN='1' VALIDATE='0' TEST='2');
  ods output nobs=nobs fitstatistics=fit validinfo=validinfo;
run;
```

Output 5.7.1 shows the information of number of observations: 695 observations are used for training and 305 are used for validation.

**Output 5.7.1  Model Selection: Number of Observations**

<table>
<thead>
<tr>
<th>The BNET Procedure</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
<td>1000</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>1000</td>
</tr>
<tr>
<td>Number of Observations Used for Training</td>
<td>695</td>
</tr>
<tr>
<td>Number of Observations Used for Validation</td>
<td>305</td>
</tr>
<tr>
<td>Number of Observations Used for Testing</td>
<td>0</td>
</tr>
</tbody>
</table>

Output 5.7.2 shows the fit statistics. In the resulting network, there are 13 nodes and 13 links between the nodes, and the number of parameters is 129.
Output 5.7.2  Model Selection: Fit Statistics

<table>
<thead>
<tr>
<th>Fit Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Nodes</td>
</tr>
<tr>
<td>Number of Links</td>
</tr>
<tr>
<td>Average Degree</td>
</tr>
<tr>
<td>Maximum Number of Parents in Network</td>
</tr>
<tr>
<td>Number of Parameters</td>
</tr>
<tr>
<td>Score</td>
</tr>
<tr>
<td>Validation Misclassification Rate</td>
</tr>
<tr>
<td>Test Misclassification Rate</td>
</tr>
</tbody>
</table>

Output 5.7.3 shows the validation results for each parameter combination. The PC Bayesian network structure has misclassified 73 observations out of 305 validation observations when VARSELECT=0 and the value of the MAXPARENTS= option is greater than or equal to 2. The TAN structure has 81 misclassification errors when VARSELECT=0 and MAXPARENTS=2. The naive Bayesian network has 81 misclassification errors when VARSELECT=0 and the value of the MAXPARENTS= option is greater than or equal to 2. The MB Bayesian network structure has 113 misclassification errors.
Output 5.7.3  Model Selection: Validation Information

<table>
<thead>
<tr>
<th>Best Model</th>
<th>Misclassification Rate</th>
<th>Misclassification Errors</th>
<th>N Observations</th>
<th>Significance Threshold</th>
<th>Prescreening</th>
<th>Variable Selection</th>
<th>Structure</th>
<th>Parenting Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>0.23934</td>
<td>73</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>0 PC</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.23934</td>
<td>73</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>0 PC</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.23934</td>
<td>73</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>0 PC</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.26557</td>
<td>81</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>0 TAN</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.26557</td>
<td>81</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>0 Naive</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.26557</td>
<td>81</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>0 Naive</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.26557</td>
<td>81</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>0 Naive</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.26885</td>
<td>82</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>1 PC</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.26885</td>
<td>82</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>1 PC</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.28197</td>
<td>86</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>0 PC</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.28197</td>
<td>86</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>0 Naive</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.34754</td>
<td>106</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>1 PC</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.34754</td>
<td>106</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>1 Naive</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.34754</td>
<td>106</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>1 Naive</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.34754</td>
<td>106</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>1 Naive</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.35738</td>
<td>109</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>1 TAN</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.37049</td>
<td>113</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>3 MB</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.37049</td>
<td>113</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>3 MB</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.37049</td>
<td>113</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>3 MB</td>
<td>BestSet</td>
<td></td>
</tr>
<tr>
<td>0.37049</td>
<td>113</td>
<td>305</td>
<td></td>
<td>0.05</td>
<td>1</td>
<td>3 MB</td>
<td>BestSet</td>
<td></td>
</tr>
</tbody>
</table>
Output 5.7.3  continued

<table>
<thead>
<tr>
<th>Validation Information</th>
<th>Max N Parents</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

The following statements produce Output 5.7.4, which shows that PROC BNET has learned a PC Bayesian network structure. In the structure, Checking is the parent of Good_Bad, Housing is the parent of Property, and Good_Bad is the parent of all the selected input variables except its parent Checking.

    proc print data=mycas.network noobs label;
    var _parentnode_ _childnode_;
    where _type_="STRUCTURE";
    run;
### Output 5.7.4 Model Selection: Best Structure

<table>
<thead>
<tr>
<th>Parent Node</th>
<th>Child Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>checking</td>
<td>good_bad</td>
</tr>
<tr>
<td>good_bad</td>
<td>employed</td>
</tr>
<tr>
<td>good_bad</td>
<td>foreign</td>
</tr>
<tr>
<td>good_bad</td>
<td>history</td>
</tr>
<tr>
<td>good_bad</td>
<td>housing</td>
</tr>
<tr>
<td>good_bad</td>
<td>other</td>
</tr>
<tr>
<td>good_bad</td>
<td>property</td>
</tr>
<tr>
<td>housing</td>
<td>property</td>
</tr>
<tr>
<td>good_bad</td>
<td>purpose</td>
</tr>
<tr>
<td>good_bad</td>
<td>savings</td>
</tr>
<tr>
<td>good_bad</td>
<td>age</td>
</tr>
<tr>
<td>good_bad</td>
<td>amount</td>
</tr>
<tr>
<td>good_bad</td>
<td>duration</td>
</tr>
</tbody>
</table>

---

**References**

Chapter 6
The BOOLRULE Procedure

Contents

Overview: BOOLRULE Procedure .................................................. 90
PROC BOOLRULE Features ......................................................... 90
Using CAS Sessions and CAS Engine Librefs ................................. 91
Getting Started: BOOLRULE Procedure ...................................... 92
Syntax: BOOLRULE Procedure .................................................... 95
PROC BOOLRULE Statement ..................................................... 96
DOCINFO Statement ............................................................. 98
OUTPUT Statement ............................................................. 99
SCORE Statement ............................................................. 100
TERMINFO Statement .......................................................... 101
Details: BOOLRULE Procedure .................................................. 101
BOOLLEAR for Boolean Rule Extraction ....................................... 101
   Term Ensemble Process .................................................... 102
   Rule Ensemble Process .................................................... 104
Measurements Used in BOOLLEAR ............................................. 105
   Precision, Recall, and the F1 Score ..................................... 105
   g-Score ........................................................................ 105
   Estimated Precision ........................................................ 106
   Improvability Test .......................................................... 106
Shrinking the Search Space ....................................................... 107
   Feature Selection ........................................................ 107
   Significance Testing ....................................................... 107
   k-Best Search ................................................................ 108
   Improvability Test ........................................................ 108
   Early Stop Based on the F1 Score ....................................... 108
Output Data Sets ................................................................. 108
   CANDIDATETERMS= Data Table ...................................... 108
   RULES= Data Table ......................................................... 109
   RULETERMS= Data Table ................................................ 110
Scoring Data Set .................................................................. 110
   OUTMATCH= Data Table ................................................... 110
Examples: BOOLRULE Procedure .............................................. 111
   Example 6.1: Rule Extraction for Binary Targets ..................... 111
   Example 6.2: Rule Extraction for a Multiclass Target .............. 113
   Example 6.3: Using Events in Rule Extraction ......................... 114
   Example 6.4: Scoring ....................................................... 116
References ............................................................................. 119
Overview: BOOLRULE Procedure

The BOOLRULE procedure is a SAS Viya procedure that enables you to extract Boolean rules from large-scale transactional data.

The BOOLRULE procedure can automatically generate a set of Boolean rules by analyzing a text corpus that has been processed by the TEXTMINE procedure and is represented in a transactional format. For example, the following rule set is generated for documents that are related to bank interest:

\[
\begin{align*}
& (\text{cut} \ ^\land \ \text{rate} \ ^\land \ \text{bank} \ ^\land \ \text{percent} \ ^\land \sim \text{sell}) \quad \text{or} \\
& (\text{market} \ ^\land \ \text{money} \ ^\land \sim \text{year} \ ^\land \ \text{percent} \ ^\land \sim \text{sale}) \quad \text{or} \\
& (\text{repurchase} \ ^\land \ \text{fee}) \quad \text{or} \\
& (\text{rate} \ ^\land \ \text{prime} \ \text{rate}) \quad \text{or} \\
& (\text{federal} \ ^\land \ \text{rate} \ ^\land \ \text{maturity})
\end{align*}
\]

In this example, \(^\land\) indicates a logical “and,” and \(\sim\) indicates a logical negation. The first line of the rule set says that 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 procedure has three advantages when you use a supervised rule-based model to analyze your large-scale transactional data. First, it focuses on modeling the positive documents in a category. Therefore, it is more robust when the data are imbalanced.\(^1\) Second, the rules can be easily interpreted and modified by a human expert, enabling better human-machine interaction. Third, the procedure adopts a set of effective heuristics to significantly shrink the search space for search rules, and its basic operations are set operations, which can be implemented very efficiently. Therefore, the procedure is highly efficient and can handle very large-scale problems.

PROC BOOLRULE Features

The BOOLRULE procedure processes large-scale transactional data in parallel to achieve efficiency and scalability. The following list summarizes the basic features of PROC BOOLRULE:

- Boolean rules are automatically extracted from large-scale transactional data.
- The extracted rules can be easily understood and tuned by humans.
- Important features are identified for each category.
- Imbalanced data are handled robustly.
- Binary-class and multiclass categorization are supported.
- Events for defining labels for documents are supported.
- All processing phases use a high degree of multithreading.

---

\(^1\) A data table is imbalanced if it contains many more negative samples than positive samples, or vice versa.
Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```plaintext
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```plaintext
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```plaintext
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
### Getting Started: BOOLRULE Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

The following DATA step creates a data table that contains 20 observations that have three variables. The `Text` variable contains the input documents. The `apple_fruit` variable contains the label of documents: a value of 1 indicates that the document is related to the apple as the fruit or to the apple tree. The `DID` variable contains the ID of the documents. Each row in the data table represents a document for analysis.

```sas
data mycas.getstart;
  infile datalines delimiter='|' missover;
  length text $150;
  input text$ apple_fruit did$;
  datalines;
  Delicious and crunchy apple is one of the popular fruits | 1 |d01
  Apple was the king of all fruits. | 1 |d02
  Custard apple or Sitaphal is a sweet pulpy fruit | 1 |d03
  apples are a common tree throughout the tropics | 1 |d04
  apple is round in shape, and tastes sweet | 1 |d05
  Tropical apple trees produce sweet apple| 1| d06
  Fans of sweet apple adore Fuji because it is the sweetest of| 1 |d07
  this apple tree is small | 1 |d08
  Apple Store shop iPhone x and iPhone x Plus.| 0 |d09
  See a list of Apple phone numbers around the world.| 0 |d10
  Find links to user guides and contact Apple Support, | 0 |d11
  Apple counters Samsung Galaxy launch with iPhone gallery | 0 |d12
  Apple Smartphones - Verizon Wireless.| 0 |d13
  Apple mercurial chief executive, was furious.| 0 |d14
  Apple has upgraded the phone.| 0 |d15
  the great features of the new Apple iPhone x.| 0 |d16
  Apple sweet apple iphone.| 0 |d17
  Apple apple will make cars | 0 |d18
  Apple apple also makes watches| 0 |d19
  Apple apple makes computers too| 0 |d20
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 TEXTMINE procedure to parse the input text data. The generated term-by-document matrix is stored in a data table named `mycas.bow`. The summary information about the terms in the document collection is stored in a data table named `mycas.terms`.

```sas
proc textmine data=mycas.getstart language="english";
  doc_id
did;
  var
text;
```
parse
   nonoungroups
   entities = none
   outparent = mycas.bow
   outterms = mycas.terms
   reducef = 1;
run;

The following statements use the BOOLRULE procedure to extract rules:

   proc boolrule
       data = mycas.bow
       docid = _document_
       termid = _termnum_
       docinfo = mycas.getstart
       terminfo = mycas.terms
       minsupports = 1
       mpos = 1
       gpos = 1;
       docinfo
           id = did
           targets = (apple_fruit);
       terminfo
           id = key
           label = term;
       output
           rules = mycas.rules
           ruleterms = mycas.ruleterms;
run;

The mycas.bow and mycas.terms data sets are specified as input in the DATA= and TERMINFO= options, respectively, in the PROC BOOLRULE statement. In addition, the DOCID= and TERMID= options in the PROC BOOLRULE statement specify the columns of the mycas.bow data table that contain the document ID and term ID, respectively.

The DOCINFO statement specifies the following information about the mycas.GetStart data table:

- The ID= option specifies the column that contains the document ID. The variables in this column are matched to the document ID variable that is specified in the DOCID= option in the PROC BOOLRULE statement in order to fetch target information about documents for rule extraction.
- The TARGETS= option specifies the target variables.

The TERMININFO statement specifies the following information about the mycas.terms data table:

- The ID= option specifies the column that contains the term ID. The variables in this column are matched to the term ID variable that is specified in the TERMID= option in the PROC BOOLRULE statement in order to fetch information about terms for rule extraction.
- The LABEL= option specifies the column that contains the text of the terms.

The OUTPUT statement requests that the extracted rules be stored in the data table mycas.Rules.
Figure 6.1 shows the SAS log that PROC BOOLRULE generates; the log provides information about the default configurations used by the procedure, about where the procedure runs, and about the input and output files. The log shows that the mycas.rules data table contains two observations, indicating that the BOOLRULE procedure identified two rules for the apple_fruit category.

![Figure 6.1 SAS Log](image)

The following statements PROC PRINT to show the contents of the mycas.rules data table that the BOOLRULE procedure generates:

```plaintext
cproc print data = mycas.rules;
  var target ruleid rule F1 precision recall;
run;
```

Figure 6.2 shows the output of PROC PRINT, which contains two rules. For information about the output of the RULES= option, see the section “RULES= Data Table” on page 109.

![Figure 6.2 The mycas.rules Data Table](image)

The following statements run the BOOLRULE procedure to match rules in documents and run PROC PRINT to show the results:

```plaintext
cproc boolrule
data = mycas.bow
docid = _document_
termid = _termnum_;  
score
  ruleterms = mycas.ruleterms
outmatch = mycas.matches;
run;
cproc print data=mycas.matches;
run;
```

Figure 6.3 shows the output of PROC PRINT, the mycas.matches data table. For information about the output of the OUTMATCH= option, see the section “OUTMATCH= Data Table” on page 110.
Syntax: BOOLRULE Procedure

The following statements are available in the BOOLRULE procedure:

```plaintext
PROC BOOLRULE <options> ;
   DOCINFO <options> ;
   TERMINFO <options> ;
   OUTPUT <options> ;
   SCORE <options> ;
```

The following sections describe the PROC BOOLRULE statement and then describe the other statements in alphabetical order.
Chapter 6: The BOOLRULE Procedure

PROC BOOLRULE Statement

PROC BOOLRULE < options > ;

The PROC BOOLRULE statement invokes the procedure. Table 6.1 summarizes the options in the statement by function. The options are then described fully in alphabetical order.

<table>
<thead>
<tr>
<th>option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic Options</strong></td>
<td></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the input data table (which must be in transactional format) for rule extraction</td>
</tr>
<tr>
<td>DOCID=</td>
<td>Specifies the variable in the DATA= data table that contains the document ID</td>
</tr>
<tr>
<td>DOCINFO=</td>
<td>Specifies the input data table that contains information about documents</td>
</tr>
<tr>
<td>GNEG=</td>
<td>Specifies the minimum g-score needed for a negative term to be considered for rule extraction</td>
</tr>
<tr>
<td>GPOS=</td>
<td>Specifies the minimum g-score needed for a positive term or a rule to be considered for rule extraction</td>
</tr>
<tr>
<td>MAXCANDIDATES=</td>
<td>Specifies the number of term candidates to be selected for each category</td>
</tr>
<tr>
<td>MAXTRIESIN=</td>
<td>Specifies the $k_{in}$ value for $k$-best search in the term ensemble process for creating a rule</td>
</tr>
<tr>
<td>MAXTRIESOUT=</td>
<td>Specifies the $k_{out}$ value for $k$-best search in the rule ensemble process for creating a rule set</td>
</tr>
<tr>
<td>MINSUPPORTS=</td>
<td>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</td>
</tr>
<tr>
<td>MNEG=</td>
<td>Specifies the $m$ value for computing estimated precision for negative terms</td>
</tr>
<tr>
<td>MPOS=</td>
<td>Specifies the $m$ value for computing estimated precision for positive terms</td>
</tr>
<tr>
<td>TERMID=</td>
<td>Specifies the variable in the DATA= data table that contains the term ID</td>
</tr>
<tr>
<td>TERMINFO=</td>
<td>Specifies the input data table that contains information about terms</td>
</tr>
</tbody>
</table>

You must specify the following option:

**DATA=CAS-libref.data-table**

**DOC=CAS-libref.data-table**

names the input data table for PROC BOOLRULE to use. *CAS-libref.data-table* is a two-level name, where
CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about CAS-libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 91.

data-table specifies the name of the input data table.

Each row of the input data table must contain one variable for the document ID and one variable for the term ID. Both the document ID variable and the term ID variable can be either a numeric or character variable. The BOOLRULE procedure does not assume that the data table is sorted by either document ID or term ID.

You can also specify the following options:

**DOCID=** variable
specifies the variable that contains the ID of each document. The document ID can be either a number or a string of characters.

**DOCINFO=** CAS-libref.data-table
names the input data table that contains information about documents. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 91.

Each row of the input data table must contain one variable for the document ID. The BOOLRULE procedure uses the document ID in the DATA= data table to search for the document ID variable in this data table to obtain information about documents (for example, the categories of each document).

**GNEG=** g-value
specifies the minimum g-score needed for a negative term to be considered for rule extraction in the term ensemble. If you do not specify this option, the value that is specified for the GPOS= option (or its default value) is used. For more information about g-score, see the section “g-Score” on page 105.

**GPOS=** g-value
specifies the minimum g-score needed for a positive term to be considered for rule extraction in the term ensemble. A rule also needs to have a g-score that is higher than g-value to be considered in the rule ensemble. The g-value is also used in the improvability test. A rule is improvable if the g-score that is computed according to the improvability test is larger than g-value. By default, GPOS=8.

**MAXCANDIDATES=** n
**MAXCANDS=** n
specifies the number of term candidates to be selected for each category. Rules are built by using only these term candidates. By default, MAXCANDS=500.

**MAXTRIESIN=** n
specifies the $k_{in}$ value for the $k$-best search in the term ensemble process for creating rules. For more information, see the section “$k$-Best Search” on page 108. By default, MAXTRIESIN=150.
Chapter 6: The BOOLRULE Procedure

MAXTRIESOUT=n
specifies the $k_{out}$ value for the $k$-best search in the rule ensemble process for creating a rule set. For more information, see the section “$k$-Best Search” on page 108. By default, MAXTRIESOUT=50.

MINSUPPORTS=n
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. By default, MINSUPPORTS=3.

MNEG=m
specifies the $m$ value for computing estimated precision for negative terms. If you do not specify this option, the value specified for the MPOS= option (or its default value) is used.

MPOS=m
specifies the $m$ value for computing estimated precision for positive terms. By default, MPOS=8.

TERMID=variable
specifies the variable that contains the ID of each term. The variable can be either a number or a string of characters. If the TERMINFO= option is not specified, variable is also used as the label of terms.

TERMINFO=CAS-libref.data-table
names the input data table that contains information about terms. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 91.

Each row of the input data table must contain one variable for the term ID. If you specify this option, you must use the TERMINFO statement to specify which variables in the data table contain the term ID and the term label, respectively. The BOOLRULE procedure uses the term ID in the DATA= data table to search for the term ID variable in this data table to obtain information about the terms. If you do not specify this option, the content of the TERMID= variable is also used as the label of terms.

DOCINFO Statement

DOCINFO < options > ;

The DOCINFO statement specifies information about the data table that is specified in the DOCINFO= option in the PROC BOOLRULE statement.

You can specify the following options:

EVENTS=(value1, value2, ...)
specifies the values of target variables that are considered as positive events or categories of interest as follows:

- When TARGETTYPE=BINARY, the values of each target variable that is specified in the TARGET= option correspond to positive events. All other values correspond to negative events.
- When TARGETTYPE=BINARY, for any variable specified in the TARGET= option that is a numeric variable, “1” is considered to be a positive event by default.
- When TARGETTYPE=BINARY, for any variable specified in the TARGET= option that is a character variable, “Y” is considered to be a positive event by default.
• You cannot specify this option when TARGETTYPE=MULTICLASS.

ID=variable

specifies the variable that contains the document ID. To fetch the target information about documents, the values in the variable are matched to the document ID variable that is specified in the DOCID= option in the PROC BOOLRULE statement. The variable can be either a numeric variable or a character variable. Its type must match the type of the variable that is specified in the DOCID= option in the PROC BOOLRULE statement.

TARGET=(variable, variable, . . .)

specifies the target variables. A target variable can be either a numeric variable or a character variable.

• When TARGETTYPE=BINARY, you can specify multiple target variables, and each target variable corresponds to a category.

• When TARGETTYPE=MULTICLASS, you can specify only one target variable, and each of its levels corresponds to a category.

TARGETTYPE=BINARY | MULTICLASS

specifies the type of the target variables. You can specify the following values:

BINARY indicates that multiple target variables can be specified and each target variable corresponds to a category.

MULTICLASS indicates that only one target variable can be specified and each level of the target variable corresponds to a category.

By default, TARGETTYPE=BINARY.

OUTPUT Statement

OUTPUT < options > ;

The OUTPUT statement specifies the data tables that contain the results that the BOOLRULE procedure generates.

You can specify the following options:

CANDIDATETERMS=CAS-libref.data-table

specifies a data table to contain the terms that have been selected by the BOOLRULE procedure for rule creation. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 91.

If MAXCANDIDATES=p in the BOOLRULE statement, the procedure selects at most p terms for each category to be considered for rule extraction. For more information about this data table, see the section “Output Data Sets” on page 108.
**RULES=** `CAS-libref.data-table`

specifies a data table to contain the rules that have been generated by the BOOLRULE procedure for each category. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 91.

For more information about this data table, see the section “Output Data Sets” on page 108.

**RULETERMS=** `CAS-libref.data-table`

specifies a data table to contain the terms in each rule that is generated by the BOOLRULE procedure. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 91.

For more information about this data table, see the section “Output Data Sets” on page 108.

---

**SCORE Statement**

```
SCORE <options>; 
```

The SCORE statement specifies the input data table that contains the terms in rules and the output data table to contain the scoring results.

You can specify the following **options**:

**OUTMATCH=** `CAS-libref.data-table`

specifies a data table to contain the rule-matching results (that is, whether a document satisfies a rule). `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 91.

For more information about this data table, see the section “Scoring Data Set” on page 110.

**RULETERMS=** `CAS-libref.data-table`

specifies a data table that contains the terms in each rule that the BOOLRULE procedure generates. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 91.

For more information about this data table, see the section “RULETERMS= Data Table” on page 110.
**TERMINFOS Statement**

```
TERMINFO < options > ;
```

The TERMINFO statement specifies information about the data table that is specified in the TERMINFO= option in the PROC BOOLRULE statement. If you specify the TERMINFO= data table in the PROC BOOLRULE statement, you must also include this statement to specify which variables in the data table contain the term ID and the term label, respectively.

You can specify the following *options*:

- **ID=variable**
  specifies the variable that contains the term ID. To fetch the text of terms, the values in `variable` are matched to the term ID variable that is specified in the TERMD= option in the PROC BOOLRULE statement. The `variable` can be either a numeric variable or a character variable. Its type must match the type of the variable that is specified in the TERMD= option in the PROC BOOLRULE statement.

- **LABEL=variable**
  specifies the variable that contains the text of the terms, where `variable` must be a character variable.

**Details: BOOLRULE Procedure**

PROC BOOLRULE implements the BOOLLEAR technique for rule extraction. This section provides details about various aspects of the BOOLRULE procedure.

**BOOLLEAR for Boolean Rule Extraction**

Rule-based text categorization algorithms uses text rules to classify documents. Text rules are interpretable and can be effectively learned even when the number of positive documents is very limited. BOOLLEAR (Cox and Zhao 2014) is a novel technique for Boolean rule extraction. When you supply a text corpus that contains multiple categories, BOOLLEAR extracts a set of binary rules from each category and represents each rule in the form of a conjunction, where each item in the conjunction denotes the presence or absence of a particular term. The BOOLLEAR process is as follows (criteria and measurements that are used in this process are described in the next section):

1. Use an information gain criterion to form an ordered term candidate list. The term that best predicts the category is first on the list, and so on. Terms that do not have a significant relationship to the category are removed from this 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 with the category in out-of-sample data, using additive smoothing. Create a rule that consists of that term.

3. If the “estimated precision” of the current rule could not possibly be improved by adding more terms as qualifiers, then go to step 6.
4. Starting with the next term on the list, determine whether the conjunction of the current rule with that
term (via either term presence or term absence) significantly improves the information gain and also
improves estimated precision.

5. If there is at least one combination that 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 obtained in step 3 has a higher estimated precision than the current “highest precision”
rule, replace the current rule with the new rule.

7. Increment the current term to the next term in the ordered candidate term list and go to step 2. Continue
repeating 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 steps 1 to 7. If it is not, then exit.

9. If so, remove from the document set all documents that match the new rule, add this rule to the rule set,
and go to step 1 to start creating the next rule in the rule set.

BOOLLEAR contains two essential processes for rule extraction: a term ensemble process (steps 4–5), which
creates rules by adding terms; and a rule ensemble process (steps 2–9), which creates a rule set. The rule set
can then be used for either content exploration or text categorization. Both the term ensemble process and the
rule ensemble process are iterative processes. The term ensemble process forms an inner loop of the rule
ensemble process. Efficient heuristic search strategies and sophisticated evaluation criteria are designed to
ensure state-of-the-art performance of BOOLLEAR.

Term Ensemble Process

The term ensemble process iteratively adds terms to a rule. When the process finishes, it returns a rule that
can be used as a candidate rule for the rule ensemble process. Figure 6.4 shows the flowchart of the term
ensemble process.
Before adding terms to a rule, BOOLLEAR first sorts the candidate terms in descending order according to their \( g \)-score with respect to the target category. It then starts to add terms to the rule iteratively. In each iteration of the term ensemble process, BOOLLEAR takes a term \( t \) from the ordered candidate term list and determines whether adding the term to the current rule \( r \) can improve the rule’s estimated precision. To ensure that the term is good enough, BOOLLEAR tries \( k_{in} - 1 \) additional terms in the term list, where \( k_{in} \) is the maximum number of terms to examine for improvement. If none of these terms is better (results in a lower \( g \)-score of the current rule \( r \)) than term \( t \), the term is considered to be \( k \)-best, where \( k = k_{in} \), and BOOLLEAR updates the current rule \( r \) by adding term \( t \) to it. If one of the \( k_{in} - 1 \) additional terms is better than term \( t \), BOOLLEAR sets that term as \( t \) and tries \( k_{in} - 1 \) additional terms to determine whether this new \( t \) is better than all of those additional terms. BOOLLEAR repeats until the current term \( t \) is \( k \)-best or until it reaches the end of the term list. After a term is added to the rule, BOOLLEAR marks the term as used and continues to identify the next \( k \)-best term from the unused terms in the sorted candidate term list. When a \( k \)-best term is identified, BOOLLEAR adds it to the rule. BOOLLEAR keeps adding \( k \)-best terms until the rule cannot be further improved. By trying to identify a \( k \)-best term instead of the global best, BOOLLEAR shrinks its search space to improve its efficiency.
Rule Ensemble Process

The rule ensemble process iteratively creates and adds new rules to a rule set. When the process finishes, it returns the rule set, which can then be used for text categorization. Figure 6.5 shows the flowchart of the rule ensemble process.

Figure 6.5 Rule Ensemble for Creating a Rule Set

In each iteration of the rule ensemble process, BOOLLEAR tries to find a rule $r$ that has the highest precision in classifying the previously unclassified positive samples. For the first iteration, all samples are unclassified. To ensure that the precision of rule $r$ is good enough, BOOLLEAR generates $k_{\text{out}} - 1$ additional rules, where $k_{\text{out}}$ is an input parameter that you specify in the MAXTRIESOUT= option in the PROC BOOLRULE statement. If one of these rules has a higher precision than rule $r$, BOOLLEAR sets that rule as the new rule $r$ and generates another $k_{\text{out}} - 1$ rules to determine whether this new rule is the best among them. BOOLLEAR repeats this process until the current rule $r$ is better than any of the $k_{\text{out}} - 1$ rules that are generated after it. The obtained rule $r$ is called a $k$-best rule, where $k = k_{\text{out}}$. When BOOLLEAR obtains a $k$-best rule, it adds that rule to the rule set and removes from the corpus all documents that satisfy the rule. In order to reduce the possibility of generating redundant rules, BOOLLEAR then determines whether the F1 score of the rule set is improved. If the F1 score is improved, BOOLLEAR goes to the next iteration and uses the updated corpus to generate another rule. Otherwise, it treats the current rule set as unimprovable, stops the search, and outputs the currently obtained rule set. Note that to identify a “good” rule, BOOLLEAR does not go through all the potential rules to find the global “best,” because doing so can be computationally intractable when the number of candidate terms is large. Also, before BOOLLEAR generates a rule, it orders the terms in the candidate term set by their correlation to the target. So it is reasonable to expect that the obtained $k$-best rule is close to a globally best rule in terms of its capability for improving the F1 score of the rule set. For information about the F1 score, see the section “Precision, Recall, and the F1 Score” on page 105.
Measurements Used in BOOLLEAR

This section provides detailed information about the measurements that are used in BOOLLEAR to evaluate terms and rules.

Precision, Recall, and the F1 Score

Precision measures the probability that the observation is actually positive when a classifier predicts it to be positive; recall measures the probability that a positive observation will be recognized; and the F1 score is the harmonic mean of precision and recall. A good classifier should be able to achieve both high precision and high recall. The precision, recall, and F1 score are defined as

\[
\text{precision} = \frac{TP}{TP + FP} \\
\text{recall} = \frac{TP}{TP + FN} \\
F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\]

where TP is the true-positive (the number of documents that are predicted to be positive and are actually positive), FP is the false-positive (the number of documents that are predicted to be positive but are actually negative), TN is the true-negative (the number of documents that are predicted to be negative and are actually negative), and FN is the false-negative (the number of documents that are predicted to be negative but are actually positive). A classifier thus obtains a high F1 score if and only if it can achieve both high precision and high recall. The F1 score is a better measurement than accuracy when the data are imbalanced, because a classifier can obtain very high accuracy by predicting that all samples belong to the majority category.

\(g\)-Score

BOOLLEAR uses the \(g\)-test (which is also known as the likelihood-ratio or maximum likelihood statistical significance test) as an information gain criterion to evaluate the correlation between terms and the target. The \(g\)-test generates a \(g\)-score, which has two beneficial properties: as a form of mutual information, it is approximately equivalent to information gain in the binary case; and because it is distributed as a chi-square, it can also be used for statistical significance testing. The \(g\)-test is designed to compare the independence of two categorical variables. Its null hypothesis is that the proportions at one variable are the same for different values of the second variable. Given the TP, FP, FN, and TN of a term, the term’s \(g\)-score can be computed as

\[
g = 2 \times \sum_{i=\{TP, TN, FP, FN\}} O(i) \log \left( \frac{O(i)}{E(i)} \right)
\]

Accuracy is defined as \(\frac{TP+TN}{TP+FP+TN+FN}\).
where P is the number of positive documents; N is the number of negative documents; O(TP), O(FP), O(TN), and O(FN) refer to the observed TP, FP, TN, and FN of a term; and E(TP), E(FP), E(TN), and E(FN) refer to the expected TP, FP, TN, and FN of a term. A term has a high g-score if it appears often in positive documents but rarely in negative documents, or vice versa.

**Estimated Precision**

Estimated precision helps BOOLLEAR shorten its search path and avoid generating overly specific rules.

The precision is estimated by a form of additive smoothing with additional correction \((err_i)\) to favor shorter rules over longer rules:

\[
\text{precision}^m_i(t) = \frac{TP_{i,t} + \frac{P}{N+P} \times m}{TP_{i,t} + FP_{i,t} + m + err_i} - err_{i-1}
\]

\[
err_i = \frac{TP_{i,t}}{TP_{i,t} + FP_{i,t}} - \frac{TP_{i,t} + \frac{P}{N+P} \times m}{TP_{i,t} + FP_{i,t} + m + err_{i-1}}
\]

In the preceding equations, \(m(\leq 1)\) is a parameter that you specify for bias correction. A large \(m\) is called for when a very large number of rules are evaluated, in order to minimize selection bias. \(TP_{i,t}\) and \(FP_{i,t}\) are the true-positive and false-positive of rule \(t\) when the length of the rule is \(i\).

**Improvability Test**

BOOLLEAR tests for improvability in the term ensemble step for “in-process” model pruning. To determine whether a rule is improvable, BOOLLEAR applies the g-test to a perfect confusion table that is defined as

\[
\begin{array}{c|c|c}
TP & 0 \\
0 & FP
\end{array}
\]

In this table, TP is the true-positive of the rule and FP is the false-positive of the rule. The g-score that is computed by using this table reflects the maximum g-score that a rule could possibly obtain if a perfectly
discriminating term were added to the rule. If the \( g \)-score is smaller than a number that you specify to indicate a maximum \( p \)-value for significance in the GPOS= and GNEG= options, BOOLLEAR considers the rule to be unimprovable.

---

**Shrinking the Search Space**

Exhaustively searching the space of possible rules is impractical because of the exponential number of rules that would have to be searched (\( 2^m \) rules, where \( m \) is the number of candidate terms). In addition, an exhaustive search usually leads to overfitting by generating many overly specific rules. Therefore, BOOLLEAR implements the strategies described in the following sections to dramatically shrink the search space to improve its efficiency and help it avoid overfitting.

**Feature Selection**

BOOLLEAR uses the \( g \)-test to evaluate terms. Assume that MAXCANDIDATES=\( p \) and MINSUPPORTS=\( c \) in the PROC BOOLRULE statement. A term is added to the ordered candidate term list if and only if the following two conditions hold:

1. The term is a top \( p \) term according to its \( g \)-score.
2. The term appears in more than \( c \) documents.

The size of the candidate term list controls the size of the search space. The smaller the size, the fewer terms are used for rule extraction, and therefore the smaller the search space is.

**Significance Testing**

In many rule extraction algorithms, rules are built until they perform perfectly on a training set, and pruning is applied afterwards. In contrast, BOOLLEAR prunes “in-process.” The following three checks are a form of in-process pruning; rules are not expanded when their expansion does not meet these basic requirements. These requirements help BOOLLEAR truncate its search path and avoid generating overly specific rules.

- **Minimum positive document coverage**: BOOLLEAR requires that a rule be satisfied by at least \( s \) positive documents, where \( s \) is the value of the MINSUPPORTS= option in the PROC BOOLRULE statement.
- **Early stop based on \( g \)-test**: BOOLLEAR stops searching when the \( g \)-score that is calculated for improving (or starting) a rule does not meet required statistical significance levels.
- **Early stop based on estimated precision**: BOOLLEAR stops building a rule when the estimated precision of the rule does not improve when the current best term is added to the rule. This strategy helps BOOLLEAR shorten its search path.
$k$-Best Search

In the worst case, BOOLLEAR could still examine an exponential number of rules, although the heuristics described here minimize that chance. But because the terms are ordered by predictiveness of the category beforehand, a $k$-best search is used to further improve the efficiency of BOOLLEAR: If BOOLLEAR tries unsuccessfully to expand (or start) a rule numerous times with the a priori “best” candidates, then the search can be prematurely ended. Two optional parameters, $k_{in}$ and $k_{out}$, determine the maximum number of terms and rules to examine for improvement. The $k_{in}$ parameter (which is specified in the MAXTRIESIN= option) is used in the term ensemble process: if $k_{in}$ consecutive terms have been checked for building possible rules and none of them are superior to the best current rule, the search is terminated. The $k_{out}$ parameter (which is specified in the MAXTRIESOUT= option) is used in the rule ensemble process: if $k_{out}$ consecutive terms have been checked to add to a rule and they do not generate a better rule, then the search for expanding that rule is terminated. This helps BOOLLEAR shorten its search path, even with a very large number of candidate terms, with very little sacrifice in accuracy.

Improvability Test

BOOLLEAR tests whether adding a theoretical perfectly discriminating term to a particular rule could possibly have both a statistically significant result and a higher estimated precision than the current rule. If it cannot, then the current rule is recognized without additional testing as the best possible rule, and no further expansion is needed.

Early Stop Based on the F1 Score

BOOLLEAR stops building the rule set if adding the current best rule does not improve the rule set’s F1 score. Thus the F1 score is treated as the objective to maximize.

Output Data Sets

This section describes the output data sets that PROC BOOLRULE produces when you specify the corresponding option in the OUTPUT statement.

CANDIDATETERMS= Data Table

The CANDIDATETERMS= option in the OUTPUT statement specifies a data table to contain the terms that have been selected by the procedure for rule creation. If MAXCANDIDATES=$p$ in the PROC BOOLRULE statement, the procedure selects a maximum of $p$ terms for each category.

Table 6.2 shows the fields in this data table.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>The category that the term is selected for (this field corresponds to the Target field in the RULES= data table)</td>
</tr>
<tr>
<td>Rank</td>
<td>The rank of the term in the ordered term list for the category (term rank starts from 1)</td>
</tr>
<tr>
<td>Term</td>
<td>A lowercase version of the term</td>
</tr>
</tbody>
</table>
Table 6.2  continued

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>The term identifier of the term</td>
</tr>
<tr>
<td>GScore</td>
<td>The g-score of the term that is obtained for the target category</td>
</tr>
<tr>
<td>Support</td>
<td>The number of documents in which the term appears</td>
</tr>
<tr>
<td>TP</td>
<td>The number of positive documents in which the term appears</td>
</tr>
<tr>
<td>FP</td>
<td>The number of negative documents in which the term appears</td>
</tr>
</tbody>
</table>

RULES= Data Table

The RULES= option in the OUTPUT statement specifies the output data table to contain the rules that have been generated for each category.

Table 6.3 shows the fields in this data table.

Table 6.3  Fields in the RULES= Data Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>The target category that the term is selected to model</td>
</tr>
<tr>
<td>Target_var</td>
<td>The variable that contains the target</td>
</tr>
<tr>
<td>Target_val</td>
<td>The value of the target variable</td>
</tr>
<tr>
<td>Ruleid</td>
<td>The ID of a rule (Ruleid starts from 1)</td>
</tr>
<tr>
<td>Ruleid_loc</td>
<td>The ID of a rule in a rule set (in each rule set, Ruleid_loc starts from 1)</td>
</tr>
<tr>
<td>Rule</td>
<td>The text content of the rule</td>
</tr>
<tr>
<td>TP</td>
<td>The number of positive documents that are satisfied by the rule set when the rule is added to the rule set</td>
</tr>
<tr>
<td>FP</td>
<td>The number of negative documents that are satisfied by the rule set when the rule is added to the rule set</td>
</tr>
<tr>
<td>Support</td>
<td>The number of documents that are satisfied by the rule set when the rule is added to the rule set</td>
</tr>
<tr>
<td>rTP</td>
<td>The number of positive documents that are satisfied by the rule when the rule is added to the rule set</td>
</tr>
<tr>
<td>rFP</td>
<td>The number of negative documents that are satisfied by the rule when the rule is added to the rule set</td>
</tr>
<tr>
<td>rSupport</td>
<td>The number of documents that are satisfied by the rule when the rule is added to the rule set</td>
</tr>
<tr>
<td>F1</td>
<td>The F1 score of the rule set when the rule is added to the rule set</td>
</tr>
<tr>
<td>Precision</td>
<td>The precision of the rule set when the rule is added to the rule set</td>
</tr>
<tr>
<td>Recall</td>
<td>The recall of the rule set when the rule is added to the rule set</td>
</tr>
</tbody>
</table>

This data table contains the discovered rule sets for predicting the target levels of the target variable. In each rule set, the order of the rules is important and helps you interpret the results. The first rule is trained using all the data; the second rule is trained on the data that did not satisfy the first rule; and subsequent rules are built only after the removal of observations that satisfy previous rules. The fit statistics (TP, FP, Support, F1,
Precision, and Recall) of each rule are cumulative and represent totals that include using that particular rule along with all the previous rules in the rule set.

When you specify TARGETTYPE=MULTICLASS in the DOCINFO statement, each target level of the target variable defines a category and the target field contains the same content as the Target_val field. When TARGETTYPE=BINARY in the DOCINFO statement, each target variable defines a category and the target field contains the same content as the Target_var field.

**RULETERMS= Data Table**

The RULETERMS= option in the OUTPUT statement specifies a data table to contain the terms in the rules. The information in this data table is used in the scoring phase for scoring documents.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>The target category that the term is selected to model</td>
</tr>
<tr>
<td>Target_var</td>
<td>The variable that contains the target</td>
</tr>
<tr>
<td>Target_val</td>
<td>The value of the target variable</td>
</tr>
<tr>
<td>Ruleid</td>
<td>The ID of a rule (Ruleid starts from 1)</td>
</tr>
<tr>
<td>Ruleid_loc</td>
<td>The ID of a rule in a rule set (in each rule set, Ruleid_loc starts from 1)</td>
</tr>
<tr>
<td>Rule</td>
<td>The text content of the rule</td>
</tr>
<tr>
<td><em>termnum</em></td>
<td>The ID of a term that is used in the rule</td>
</tr>
<tr>
<td>Direction</td>
<td>Specifies whether the term is positive or negative (if Direction=1, the term is positive; if Direction=–1, the term is negative)</td>
</tr>
<tr>
<td>Weight</td>
<td>The weight of a term</td>
</tr>
</tbody>
</table>

Term weights are used for scoring documents. The weight of a negative term is always –1. If a positive term is in rule r and there are k positive terms in the rule, the weight of this positive term is \(1/k + 0.000001\). If a document contains all the positive terms in the rule but none of the negative terms, the score of the document is \(k \times (1/k + 0.000001) > 1\), indicating that the document satisfies the rule. Otherwise, the document’s score is less than 1, indicating that the document does not satisfy the rule.

**Scoring Data Set**

This section describes the output data set that PROC BOOLRULE produces when you specify the corresponding option in the SCORE statement.

**OUTMATCH= Data Table**

The OUTMATCH= option in the SCORE statement specifies the output data table to contain the rule-matching results (that is, whether a document satisfies a 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 are present in the document. PROC BOOLRULE also outputs a special rule for which ID=0. If a document satisfies the rule for which ID=0, then the document does not satisfy any rule in the RULETERMS= table. For this special rule, the target has a missing value.
Table 6.5 shows the fields in this data table.

Table 6.5  Fields in the OUTMATCH= Data Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Document</em></td>
<td>ID of the document that satisfies the rule</td>
</tr>
<tr>
<td><em>Target</em></td>
<td>ID of the target that the rule is generated for</td>
</tr>
<tr>
<td><em>Rule_ID</em></td>
<td>ID of the rule that the document satisfies</td>
</tr>
</tbody>
</table>

Examples: BOOLRULE Procedure

Example 6.1: Rule Extraction for Binary Targets

This example generates rules for a data table that contains various types of customer reviews. The following DATA step creates the mycas.reviews data table, which contains nine observations that have four variables. The text variable contains the input reviews. The positive variable contains the sentiment of the reviews: a value of 1 indicates that the review is positive, and a value of 0 indicates that the review is negative. The category variable contains the category of the reviews. The did variable contains the ID of the documents. Each row in the data table represents a document for analysis.

```plaintext
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
    ;
run;
```

The following TEXTMINE procedure call parses the mycas.reviews data table, stores the term-by-document matrix in the mycas.reviews_bow data table in transactional format, and stores terms that appeared in the mycas.reviews data table in the mycas.reviews_terms data table:

```plaintext
proc textmine data=mycas.reviews;
    doc_id
        did;
    var
```
The following statements run PROC BOOLRULE to extract rules from the mycas.reviews_bow data table and run PROC PRINT to show the results. By default, TARGETTYPE=BINARY. One target variable, positive, is specified; this variable indicates whether the reviews are positive or negative.

```sas
proc boolrule
  data = mycas.reviews_bow
  docid = _document_
  termid = _termnum_
  docinfo = mycas.reviews
  terminfo = mycas.reviews_terms
  minsupports = 1
  mpos = 1
  gpos = 1;
  docinfo
    id = did
    targets = (positive);
  terminfo
    id = key
    label = term;
output
  ruleterms = mycas.ruleterms
  rules = mycas.rules;
run;
data rules;
set mycas.rules;
proc print data=rules;
  var target ruleid rule F1 precision recall;
run;
```

Output 6.1.1 shows that the mycas.rules data table contains rules that are generated for the “positive” categories.

### Output 6.1.1 The mycas.rules Data Table

<table>
<thead>
<tr>
<th>Obs</th>
<th>TARGET</th>
<th>RULEID</th>
<th>RULE</th>
<th>F1</th>
<th>PRECISION</th>
<th>RECALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>positive</td>
<td>1</td>
<td>the &amp; ~'s &amp; ~not</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example 6.2: Rule Extraction for a Multiclass Target

This example uses the same input table and the same TEXTMINE procedure call that are used in Example 6.1 to illustrate how you can extract rules for a multiclass target. The DATA step and procedure call are repeated here for convenience.

The following DATA step creates the mycas.reviews data table, which contains nine observations that have four variables. The text variable contains the input reviews. The positive variable contains the sentiment of the reviews: a value of 1 indicates that the review is positive, and a value of 0 indicates that the review is negative. The category variable contains the category of the reviews. The did variable contains the ID of the documents. Each row in the data table represents a document for analysis.

```
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 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
;run;
```

The following TEXTMINE procedure call parses the mycas.reviews data table, stores the term-by-document matrix in the mycas.reviews_bow data table in transactional format, and stores terms that appeared in the mycas.reviews data table in the mycas.reviews_terms data table:

```
proc textmine data=mycas.reviews;
  doc_id
    did;
  var
    text;
  parse
    nonoungroups
    notagging
    entities = none
    outparent = mycas.reviews_bow
    outterms = mycas.reviews_terms
    reducef = 1;
run;
```

The following statements run PROC BOOLRULE to extract rules from the mycas.reviews_bow data table and run PROC PRINT to show the results. TARGETTYPE=MULTICLASS is specified, and category is specified as the target variable, which contains three levels: “electronics,” “movies,” and “books.” Each level defines a category for which the BOOLRULE procedure extracts rules.
proc boolrule
   data  =  mycas.reviews_bow
   docid =  _document_
   termid =  _termnum_
   docinfo =  mycas.reviews
   terminfo =  mycas.reviews_terms
   minsupports =  1
   mpos =  1
   gpos =  1;
   docinfo
      id =  did
      targettype =  multiclass
      targets =  (category);
   terminfo
      id =  key
      label =  term;
   output
      ruleterms =  mycas.ruleterms
      rules =  mycas.rules;
run;

data rules;
set mycas.rules;
proc print data=rules;
   var target ruleid rule F1 precision recall;
run;

Output 6.2.1 shows that the mycas.rules data table contains rules that are generated for the “electronics,” “movies,” and “books” categories.

Output 6.2.1  The mycas.rules Data Table

<table>
<thead>
<tr>
<th>Obs</th>
<th>TARGET</th>
<th>RULEID</th>
<th>RULE</th>
<th>F1</th>
<th>PRECISION</th>
<th>RECALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>electronics</td>
<td>3</td>
<td>phone</td>
<td>0.8</td>
<td>1</td>
<td>0.66667</td>
</tr>
<tr>
<td>2</td>
<td>electronics</td>
<td>4</td>
<td>resolution &amp; ~it</td>
<td>1.0</td>
<td>1</td>
<td>1.00000</td>
</tr>
<tr>
<td>3</td>
<td>books</td>
<td>2</td>
<td>book</td>
<td>1.0</td>
<td>1</td>
<td>1.00000</td>
</tr>
<tr>
<td>4</td>
<td>movies</td>
<td>1</td>
<td>movie &amp; ~love</td>
<td>1.0</td>
<td>1</td>
<td>1.00000</td>
</tr>
</tbody>
</table>

Example 6.3: Using Events in Rule Extraction

This example uses the same input table and the same TEXTMINE procedure call that are used in Example 6.1 to illustrate how you can use events in rule extraction. The DATA step and procedure call are repeated here for convenience.

When TARGETTYPE=MULTICLASS, each level of the target variable defines a category for rule extraction. If you want to extract rules for only a subset of the levels of the target variable, you can use the EVENTS= option to specify the categories for which you want to extract rules.

The following DATA step creates the mycas.reviews data table, which contains nine observations that have four variables. The text variable contains the input reviews. The positive variable contains the sentiment of
Example 6.3: Using Events in Rule Extraction

the reviews: a value of 1 indicates that the review is positive and a value of 0 indicates that the review is negative. The category variable contains the category of the reviews. The did variable contains the ID of the documents. Each row in the data table represents a document for analysis.

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
;
run;

The following TEXTMINE procedure call parses the mycas.reviews data table, stores the term-by-document matrix in the mycas.reviews_bow data table in transactional format, and stores terms that appeared in the mycas.reviews data table in the mycas.reviews_terms data table:

proc textmine data=mycas.reviews;
  doc_id did;
  var text;
  parse
    nonoungroups
    notagging
    entities = none
    outparent = mycas.reviews_bow
    outterms = mycas.reviews_terms
    reducef = 1;
run;

The following statements run PROC BOOLRULE to extract rules from the mycas.reviews_bow data table and run PROC PRINT to show the results. TARGETTYPE=BINARY is specified, and category is specified as the target variable, which contains three levels: “electronics,” “movies,” and “books.” Because the “movies” and “books” levels are specified in the EVENTS= option, PROC BOOLRULE procedure extracts rules for “movies” and “books,” but not “electronics.”

proc boolrule
  data = mycas.reviews_bow
  docid = _document_
  termid = _termnum_
  docinfo = mycas.reviews
  terminfo = mycas.reviews_terms
  minsupports = 1
mpos = 1
gpos = 1;
docinfo
id = did
targettype = binary
targets = (category)
events = ("movies" "books");
terminfo
id = key
label = term;
output
ruleterms = mycas.ruleterms
rules = mycas.rules;
run;
data rules;
set mycas.rules;
proc print data=rules;
  var target ruleid rule F1 precision recall;
run;

Output 6.3.1 shows that the mycas.rules data table contains rules that are generated for the “movies” and “books” categories.

Output 6.3.1 The mycas.rules Data Table

<table>
<thead>
<tr>
<th>Obs</th>
<th>TARGET</th>
<th>RULEID</th>
<th>RULE</th>
<th>F1</th>
<th>PRECISION</th>
<th>RECALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>category</td>
<td>1</td>
<td>movie</td>
<td>0.8</td>
<td>1</td>
<td>0.66667</td>
</tr>
<tr>
<td>2</td>
<td>category</td>
<td>2</td>
<td>book</td>
<td>1.0</td>
<td>1</td>
<td>1.00000</td>
</tr>
</tbody>
</table>

Example 6.4: Scoring

This example uses the same input table and the same TEXTMINE procedure call that are used in Example 6.1 to illustrate how you can match extracted rules in documents. Then it adds the DATA step to generate testing data. The DATA step and procedure call are repeated here for convenience.

The following DATA step creates the mycas.reviews data table, which contains nine observations that have four variables. The text variable contains the input reviews. The positive variable contains the sentiment of the reviews: a value of 1 indicates that the review is positive, and a value of 0 indicates that the review is negative. The category variable contains the category of the reviews. The did variable contains the ID of the documents. Each row in the data table represents a document for analysis.

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

Example 6.4: Scoring

This example uses the same input table and the same TEXTMINE procedure call that are used in Example 6.1 to illustrate how you can match extracted rules in documents. Then it adds the DATA step to generate testing data. The DATA step and procedure call are repeated here for convenience.

The following DATA step creates the mycas.reviews data table, which contains nine observations that have four variables. The text variable contains the input reviews. The positive variable contains the sentiment of the reviews: a value of 1 indicates that the review is positive, and a value of 0 indicates that the review is negative. The category variable contains the category of the reviews. The did variable contains the ID of the documents. Each row in the data table represents a document for analysis.

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.
I watched this movie on TV, it's not good on a small screen.
watched the movie first and loved it, the book is even better!
I like the story in this book, they should put it on screen.
I love the author, but this book is a waste of time, don't buy it.

The following DATA step generates the testing data, which contain two observations that have two variables. The text variable contains the input reviews. The did variable contains the ID of the documents. Each row in the data table represents a document for analysis.

data mycas.reviews_test;
  infile datalines delimiter='|' missover;
  length text $300;
  input text$ did;
  datalines;
  love it! a great phone, even better than advertised
  I like the book, GREATEST in this genre
run;

The following TEXTMINE procedure call parses the mycas.reviews data table, stores the term-by-document matrix in the mycas.reviews_bow data table in transactional format, and stores terms that appeared in the mycas.reviews data table in the mycas.reviews_terms data table:

proc textmine data=mycas.reviews;
  doc_id
    did;
  var
    text;
  parse
    nonoungroups
    notagging
    entities = none
    outparent = mycas.reviews_bow
    outterms = mycas.reviews_terms
    outconfig = mycas.parseconfig
    reducef = 1;
run;

The following statements run PROC BOOLRULE to extract rules from the mycas.reviews_bow data table. TARGETTYPE=BINARY is specified. One target variable, positive, is specified; this variable indicates whether the reviews are positive or negative.

proc boolrule
  data = mycas.reviews_bow
  docid = _document_
  termid = _termnum_
  docinfo = mycas.reviews
  terminfo = mycas.reviews_terms
  minsupports = 1
The TMSCORE procedure uses the parsing configuration that is stored in the mycas.parseconfig data table to parse the mycas.reviews_test data table. The term-by-document matrix is stored in the mycas.reviews_test_bow data table.

```
proc tmscore
  data = mycas.reviews_test
terms = mycas.reviews_terms
config = mycas.parseconfig
outparent = mycas.reviews_test_bow;
  doc_id did;
  var text;
run;
```

The following statements run PROC BOOLRULE to match rules in the testing data and run PROC PRINT to show the matching results:

```
proc boolrule
  data = mycas.reviews_test_bow
docid = _document_
termid = _termnum_
  score
    ruleterms = mycas.ruleterms
    outmatch = mycas.match;
run;
```

```
proc print data=mycas.match; run;
```

The mycas.match data table in **Output 6.4.1** shows which documents satisfy which rules.

**Output 6.4.1** The mycas.match Data Table

<table>
<thead>
<tr>
<th>Obs</th>
<th>DOCUMENT</th>
<th>TARGET</th>
<th>RULE_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Chapter 7
The FACTMAC Procedure

Contents

Overview: FACTMAC Procedure ....................................................... 122
PROC FACTMAC Features .......................................................... 122
Using CAS Sessions and CAS Engine Librefs ................................. 123
Getting Started: FACTMAC Procedure ........................................... 124
Syntax: FACTMAC Procedure ....................................................... 126
PROC FACTMAC Statement ........................................................ 126
AUTOTUNE Statement ............................................................... 128
CODE Statement ................................................................. 131
DISPLAY Statement .............................................................. 131
DISPLAYOUT Statement ........................................................ 132
ID Statement ................................................................. 133
INPUT Statement ............................................................. 133
OUTPUT Statement ........................................................... 134
SAVESTATE Statement ......................................................... 134
TARGET Statement ............................................................. 135
Details: FACTMAC Procedure .................................................... 135
Displayed Output ................................................................. 136
ODS Table Names ................................................................. 137
Output Data Tables ............................................................ 137
Examples: FACTMAC Procedure ............................................... 138
Example 7.1: Running PROC FACTMAC with the MovieLens Data Set .... 138
References ................................................................. 140
Overview: FACTMAC Procedure

The FACTMAC procedure implements the factorization machine model in SAS Viya. The flexible factorization machine model has applications in predictive modeling and recommendation (Rendle 2012). Factorization machines generalize matrix factorization, among other techniques. You can use the FACTMAC procedure to read and write data in distributed form, and to perform factorization in parallel by making full use of multicore computers or distributed computing environments.

The FACTMAC procedure estimates factors for each of the nominal input variables you specify, in addition to estimating a global bias and a bias for each level of those nominal input variables. You also specify an interval target variable. The procedure computes the biases and factors by using the stochastic gradient descent (SGD) algorithm, which minimizes the root mean square error (RMSE) criterion on the input data table that you provide. In this method, each iteration attempts to reduce the RMSE. The SGD algorithm proceeds until the maximum number of iterations is reached.

PROC FACTMAC stores the results of the factorization an output data table, which is produced by the OUTMODEL= option. This data table contains the factors in addition to the global bias and the biases for all the levels of the input variables, in addition to the factors. The corresponding level names are listed for ease of reference. The biases and factors are used for scoring.

PROC FACTMAC Features

PROC FACTMAC enables you to use parallel execution for factorization in a distributed computing environment or on a single-machine. The following list summarizes the basic features of PROC FACTMAC:

- is highly distributed and multithreaded
- learns a factorization machine model based on a parallel implementation of the SGD optimization algorithm
Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Chapter 7: The FACTMAC Procedure

Getting Started: FACTMAC Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example shows how to use the FACTMAC procedure to learn a factorization machine model from observations in a SAS data table. This example uses the `cars` data set in the Sashelp library and illustrates the prediction of gas mileage of cars based on make and model. The analysis uses four variables: car make, car model, car type, and a variable named `mpg_city`, which measures the car’s fuel usage (in miles per gallon) for city driving. The remaining variables in the data table are not used.

You can load the `cars` data set into your CAS session by naming your CAS engine libref in the first statement of the following DATA step:

```sas
data mycas.cars;
  set sashelp.cars;
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 PROC FACTMAC and output the results to ODS tables:

```sas
proc factmac data=mycas.cars outmodel=mycas.factors maxiter=50
   nfacators=5  learnstep=0.002;
   input make model type /level=nominal;
   target mpg_city /level=interval;
   output out=mycas.score_out copyvars=(make model type mpg_city);
run;
```

```sas
proc print data=mycas.factors(obs=48);
run;
```

The NFACTORS= option requests that the model estimate five factors, the LEARNSTEP= option sets the optimization learning step to 0.002, the MAXITER= option requests that the optimization stop after 50 iterations, and the OUTMODEL= option requests that the model parameters be written to the mycas.factors data table. The INPUT statement specifies that the make, model, and type variables are to be used as nominal inputs. The TARGET statement specifies that mpg_city is the target variable to be predicted. The OUTPUT statement requests that the predictions be written to the data table mycas.score_out and that the make, model, type and mpg_city variables be copied from the mycas.cars data table to the mycas.score_out data table.

**Figure 7.1** shows the global bias, the bias values for each level, and the list of the factors for the first 48 observations.
Getting Started: FACTMAC Procedure F 125

Figure 7.1 Bias Values and Factors
O b s

V a r ia b le

L e v e l

B ia s

F a c to r 1

F a c to r 2

F a c to r 3

F a c to r 4

F a c to r 5
0 .0 0 0 0 0

1

_ G L O B A L _

2 0 .0 6 0 7

0 .0 0 0 0 0

0 .0 0 0 0 0

0 .0 0 0 0 0

0 .0 0 0 0 0

2

M a k e

A c u ra

-0 .6 3 2 2

-1 .6 5 1 8 9

-2 .3 4 2 1 3

0 .0 9 5 5 3

-4 .0 6 5 5 1

0 .7 2 5 6 3

3

M a k e

A u d i

-1 .5 8 7 1

-0 .2 3 1 3 1

0 .0 7 8 2 8

-0 .3 5 2 9 8

-0 .4 5 8 2 7

-0 .2 0 1 6 6

4

M a k e

B M W

-1 .3 6 0 7

0 .0 0 7 7 7

0 .0 5 4 5 5

-0 .1 9 4 6 6

-0 .0 7 2 7 2

0 .2 2 2 7 5

5

M a k e

B u ic k

-1 .1 7 1 9

2 .9 6 4 9 3

0 .1 0 8 1 5

3 .1 4 0 4 8

-4 .7 4 2 9 3

-0 .2 8 6 9 2

6

M a k e

C a d i lla c

-3 .5 6 0 7

-1 .0 7 0 3 3

-0 .9 9 1 8 7

1 .4 9 6 9 9

0 .5 8 2 4 5

0 .9 5 6 6 9

7

M a k e

C h e v r o le t

-0 .3 9 4 1

0 .0 1 2 1 6

0 .0 0 6 8 8

0 .0 5 3 2 1

0 .0 0 9 2 3

-0 .0 3 5 0 7

8

M a k e

C h r y s le r

-0 .1 9 4 1

0 .5 6 1 6 8

-0 .9 0 6 8 1

-0 .7 7 8 9 9

-1 .3 8 4 9 8

1 .4 8 2 2 0

9

M a k e

D o d g e

-0 .6 7 6 1

0 .5 3 6 0 3

0 .0 5 0 4 4

0 .0 3 6 1 4

0 .7 9 8 6 7

0 .4 6 5 2 2

1 0

M a k e

F o rd

-0 .7 9 9 9

-0 .2 9 5 3 2

-0 .1 6 4 5 4

0 .0 5 8 4 5

-0 .2 7 5 5 0

0 .1 0 5 2 8

1 1

M a k e

G M C

-4 .6 8 5 7

1 .0 3 1 7 1

-1 .9 7 3 1 0

-0 .9 1 0 6 9

-4 .5 4 4 1 8

2 .0 0 9 5 2

1 2

M a k e

H o n d a

7 .7 6 2 8

0 .8 4 9 0 2

-0 .5 4 0 7 3

-0 .3 7 1 4 9

-0 .8 7 8 9 8

-0 .3 1 3 3 4

1 3

M a k e

H u m m e r

-1 0 .0 6 0 7

-3 .2 6 6 5 2

2 .0 6 0 4 4

-3 .3 1 8 1 1

-1 .3 2 7 0 9

-6 .0 3 0 6 8

1 4

M a k e

H y u n d a i

2 .9 3 9 3

0 .3 0 0 7 5

0 .7 2 7 5 7

-0 .7 8 0 5 7

0 .0 1 3 7 1

-0 .1 3 1 1 3

1 5

M a k e

In fin iti

-2 .8 1 0 7

-1 .2 0 2 7 0

2 .2 3 7 6 2

6 .2 7 4 5 9

-3 .9 7 1 3 1

0 .5 1 6 0 8

1 6

M a k e

Is u z u

-4 .0 6 0 7

6 .3 1 9 6 6

-4 .6 1 0 5 2

2 .3 2 4 6 5

-5 .9 8 8 8 2

-2 .3 3 4 9 5

1 7

M a k e

J a g u a r

-2 .5 6 0 7

-2 .1 8 1 7 5

-1 .3 6 3 9 5

-1 .7 2 0 3 4

2 .8 8 5 7 3

1 .2 2 2 0 2

1 8

M a k e

J e e p

-2 .7 2 7 4

1 .3 4 1 3 6

-4 .7 1 5 4 3

5 .6 8 7 4 0

0 .2 5 7 4 8

-5 .3 0 4 7 4

1 9

M a k e

K ia

1 .8 4 8 3

-5 .3 0 7 5 5

3 .5 3 8 6 4

-4 .7 5 6 1 9

5 .2 4 4 4 6

2 .5 5 0 9 8

2 0

M a k e

L a n d R o v e r

-6 .0 6 0 7

-5 .9 8 6 5 7

1 .7 3 5 0 1

3 .1 6 0 5 0

0 .1 6 4 6 9

-3 .8 9 0 3 1

2 1

M a k e

L e x u s

-2 .6 0 6 2

-0 .9 7 0 6 8

1 .8 3 4 4 0

1 .6 1 6 6 1

0 .7 7 8 1 5

0 .4 4 3 8 3

2 2

M a k e

L i n c o ln

-3 .2 8 3 0

1 .3 2 6 8 4

0 .1 6 1 8 5

-0 .6 8 3 7 5

-0 .6 6 1 9 6

1 .1 7 0 9 4

2 3

M a k e

M IN I

6 .4 3 9 3

-0 .1 8 3 1 6

1 .8 8 5 1 4

-0 .7 9 7 1 6

-2 .6 2 5 5 7

5 .7 8 7 8 3

2 4

M a k e

M a z d a

1 .3 9 3 8

-0 .9 1 1 7 1

1 .9 5 7 1 8

-1 .1 3 2 4 1

0 .7 0 1 2 4

0 .0 6 4 8 3

2 5

M a k e

M e rc e d e s -B e n z

-2 .7 1 4 6

-0 .1 6 7 3 8

0 .0 7 7 9 9

-0 .2 0 5 1 7

-0 .2 5 9 4 5

-0 .0 8 4 4 3

2 6

M a k e

M e rc u ry

-2 .5 0 5 2

0 .9 3 3 9 3

2 .2 5 6 0 7

0 .1 7 3 5 7

0 .2 5 7 8 6

1 .2 7 9 7 3

2 7

M a k e

M its u b is h i

0 .8 6 2 3

-0 .2 4 4 3 2

0 .3 5 6 3 5

0 .1 6 7 0 3

0 .3 2 4 7 9

-0 .0 2 8 1 0

2 8

M a k e

N is s a n

-0 .3 5 4 9

-0 .0 0 1 3 2

-0 .0 2 5 6 1

0 .0 0 7 5 1

0 .0 0 2 3 0

-0 .0 2 6 5 2

2 9

M a k e

O ld s m o b i le

0 .9 3 9 3

0 .2 1 0 7 3

3 .4 9 8 5 6

5 .9 2 7 9 0

-6 .2 8 9 7 9

2 .9 2 3 9 4

3 0

M a k e

P o n tia c

0 .4 8 4 7

-0 .1 3 0 6 1

-1 .7 0 5 0 2

-1 .3 5 0 8 6

1 .1 4 5 9 9

1 .4 1 5 3 5

3 1

M a k e

P o rs c h e

-2 .6 3 2 2

-4 .9 0 2 5 9

4 .5 6 2 6 9

4 .1 8 5 5 5

0 .6 6 5 4 3

-4 .0 9 5 0 5

3 2

M a k e

S a a b

0 .3 6 7 8

-0 .1 8 6 6 6

0 .0 0 7 1 4

0 .1 6 1 8 1

0 .8 0 4 3 9

0 .6 1 8 8 9

3 3

M a k e

S a tu r n

4 .3 1 4 3

1 .9 0 3 4 0

-3 .3 8 0 2 0

3 .7 3 0 0 3

0 .3 1 7 8 5

-0 .3 4 1 0 6

3 4

M a k e

S c io n

1 1 .4 3 9 3

6 .3 1 0 0 9

-6 .2 2 9 9 4

6 .2 6 1 7 0

-1 .7 4 4 8 1

5 .7 7 0 1 2

3 5

M a k e

S u b a ru

0 .2 1 2 0

0 .4 8 4 9 4

-0 .3 6 5 5 0

-0 .4 1 6 9 8

-0 .4 9 6 9 6

1 .0 3 9 0 8

3 6

M a k e

S u z u k i

2 .0 6 4 3

-1 .7 2 7 7 3

1 .9 3 5 7 9

-0 .7 7 0 9 9

-0 .0 1 7 6 6

-1 .5 0 6 9 5

3 7

M a k e

T o y o ta

4 .3 6 7 8

0 .2 9 6 5 0

-0 .0 8 5 7 4

-0 .1 3 6 8 1

0 .2 0 4 7 7

-0 .5 4 6 2 3

3 8

M a k e

V o lk s w a g e n

1 .3 3 9 3

-0 .0 5 2 3 6

0 .7 0 8 7 9

-1 .4 9 1 9 1

-1 .3 3 9 2 4

0 .1 9 6 1 4

3 9

M a k e

V o lv o

-0 .3 1 0 7

0 .9 1 5 8 5

-0 .2 0 4 7 9

-0 .6 4 4 1 6

-0 .2 3 1 6 6

0 .7 4 6 5 0

4 0

M o d e l

3 .5 R L 4 d r

-2 .0 6 0 7

4 .2 9 0 0 8

6 .3 2 1 2 7

6 .3 2 6 0 9

-6 .3 2 2 0 3

-4 .1 3 0 5 2

4 1

M o d e l

3 .5 R L w /N a v ig a tio n 4 d r

-2 .0 6 0 7

-2 .5 3 3 5 8

1 .0 7 0 3 3

6 .3 2 2 4 4

4 .6 4 1 8 5

0 .8 8 8 4 1

4 2

M o d e l

3 0 0 M

4 d r

-2 .0 6 0 7

6 .3 3 2 5 8

6 .3 2 1 0 2

-1 .4 6 1 0 5

6 .3 2 5 4 4

6 .3 2 4 0 5

4 3

M o d e l

3 0 0 M

S p e c ia l E d itio n 4 d r

-2 .0 6 0 7

-6 .3 3 3 4 7

4 .2 0 1 5 8

-6 .3 2 7 3 0

-0 .6 8 9 4 6

4 .7 7 9 1 7

4 4

M o d e l

3 2 5 C i 2 d r

-0 .0 6 0 7

0 .7 5 6 8 7

-6 .3 2 1 7 1

6 .3 2 2 5 7

-6 .3 2 4 5 9

6 .3 2 3 0 9

4 5

M o d e l

3 2 5 C i c o n v e r t i b le 2 d r

-1 .0 6 0 7

6 .3 3 2 1 6

-4 .5 4 5 9 8

-0 .4 9 2 6 6

6 .3 2 4 9 6

6 .3 2 2 7 0

4 6

M o d e l

3 2 5 i 4 d r

-0 .0 6 0 7

-0 .1 1 3 5 9

4 .9 8 9 7 9

2 .2 1 1 3 9

5 .5 0 1 7 7

6 .3 2 4 1 6

4 7

M o d e l

3 2 5 x i 4 d r

-1 .0 6 0 7

-6 .3 2 5 3 9

1 .4 9 3 0 4

5 .5 2 1 6 9

-6 .1 3 3 2 5

6 .3 2 5 6 8

4 8

M o d e l

3 2 5 x i S p o rt

-1 .0 6 0 7

6 .3 2 0 9 5

-5 .1 8 6 7 6

-0 .4 5 8 2 7

-4 .1 1 6 2 2

-3 .1 2 2 5 4


Syntax: FACTMAC Procedure

The following statements are available in the FACTMAC procedure:

```
PROC FACTMAC < options > ;
  CODE FILE=filename ;
  DISPLAY < table-list > </ options > ;
  DISPLAYOUT table-spec-list </ options > ;
  ID variables ;
  INPUT variables < LEVEL=Nominal > ;
  OUTPUT OUT=CAS-libref.data-table < options > ;
  SAVESTATE RSTORE=CAS-libref.data-table ;
  TARGET variable < LEVEL=INTERVAL > ;
  AUTOTUNE < options > ;
```

The PROC FACTMAC statement, an INPUT statement, and the TARGET statement are required. You can specify multiple INPUT statements.

The following sections describe the PROC FACTMAC statement and then describe the other statements in alphabetical order.

PROC FACTMAC Statement

```
PROC FACTMAC < options > ;
```

The PROC FACTMAC statement invokes the procedure. Table 7.1 summarizes the options available in the PROC FACTMAC statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Data Table Options</strong></td>
<td></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the input data table</td>
</tr>
<tr>
<td><strong>Factorization Options</strong></td>
<td></td>
</tr>
<tr>
<td>LEARNSTEP=</td>
<td>Specifies the learning step size for the SGD algorithm</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations</td>
</tr>
<tr>
<td>NFACTORS=</td>
<td>Specifies the number of factors to estimate for the model</td>
</tr>
<tr>
<td>NONNEGATIVE</td>
<td>Requests nonnegative factorization</td>
</tr>
<tr>
<td>NTHREADS=</td>
<td>Specifies the number of threads to use on each computation node</td>
</tr>
<tr>
<td>OUTMODEL=</td>
<td>Specifies the output model data table to contain the computed factor parameters</td>
</tr>
<tr>
<td>SEED=</td>
<td>Specifies the seed to be used for pseudorandom number generation</td>
</tr>
<tr>
<td><strong>Output Options</strong></td>
<td></td>
</tr>
<tr>
<td>NOPRINT=</td>
<td>Suppresses ODS output</td>
</tr>
</tbody>
</table>

You can specify the following options:
DATA=\texttt{CAS-libref.data-table}

names the input data table for PROC FACTMAC to use. The default is the most recently created data table. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to a collection of information that is defined in the LIBNAME statement and includes the \texttt{caslib}, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about \texttt{CAS-libref}, see the section “Using CAS Sessions and CAS Engine Librefs” on page 123.

\texttt{data-table} specifies the name of the input data table.

LEARNSTEP= \texttt{number}

specifies the learning step size for the stochastic gradient descent (SGD) algorithm, where \texttt{number} is a positive real number. The learning step size controls the amount by which the factors are updated at each iteration.

By default, \texttt{LEARNSTEP}=0.001. This value can be tuned with the AUTOTUNE statement.

MAXITER= \texttt{number}

specifies the maximum number of iterations for the algorithm to perform, where \texttt{number} is an integer greater than or equal to 1. In each iteration of the SGD method, the factors are recomputed.

By default, \texttt{MAXITER}=1. This value can be tuned with the AUTOTUNE statement.

NFACTORS= \texttt{number}

specifies the number of factors to estimate for the model, where \texttt{number} is an integer greater than or equal to 1.

By default, \texttt{NFACTORS}=1. This value can be tuned with the AUTOTUNE statement.

NONNEGATIVE

performs nonnegative factorization, in which the estimated factors are greater than or equal to 0 and the estimated biases are 0.

By default, nonnegative factorization is not performed.

NOPRINT

suppresses ODS output.

NTHREADS= \texttt{number-of-threads}

specifies the number of threads to use for the computation, where \texttt{number-of-threads} is an integer from 1 to 64, inclusive. The default value is the maximum number of available threads per computer.

OUTMODEL= \texttt{CAS-libref.data-table}

specifies the output model data table to contain the computed factor parameters. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the \texttt{caslib} and session identifier, and \texttt{data-table} specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 123.
**SEED=** *random_seed*

specifies an integer that is used to start the pseudorandom number generator. This option enables you to reproduce the same sample output, but only when **NTHREADS=** 1. If you do not specify a seed or you specify a value less than or equal to 0, the seed is generated from reading the time of day from the computer’s clock.

By default, **SEED=** 0.

---

**AUTOTUNE Statement**

**AUTOTUNE** < *options*> ;

The **AUTOTUNE** statement searches for the best combination of values of the **NFACTORS=**, **LEARNSTEP=**, and **MAXITER=** options in the **PROC FACTMAC** statement. You cannot specify both the **OUTPUT** and **AUTOTUNE** statements in the same run of **PROC FACTMAC**.

Table 7.2 summarizes the *options* that you can specify in the **AUTOTUNE** statement. For more information about all options except the **TUNINGPARAMETERS=** option, see the option’s description in the section “**AUTOTUNE Statement**” on page 12 in Chapter 3, “Shared Concepts.” The **TUNINGPARAMETERS=** option is described following Table 7.2.

**Table 7.2**  AUTOTUNE Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>APPENDLOOKUP</strong></td>
<td>Specifies that the table specified in the <strong>HISTORYTABLE=</strong> option contain the rows from the table specified in the <strong>LOOKUPTABLE=</strong> option</td>
</tr>
<tr>
<td><strong>EVALHISTORY=</strong></td>
<td>Specifies how to report the evaluation history of the tuner</td>
</tr>
<tr>
<td><strong>FRACTION=</strong></td>
<td>Specifies the fraction of observations to use for validation</td>
</tr>
<tr>
<td><strong>HISTORYTABLE=</strong></td>
<td>Specifies the CAS table that contains the evaluation history</td>
</tr>
<tr>
<td><strong>KFOLD=</strong></td>
<td>Specifies the number of folds for <em>k</em>-fold cross validation</td>
</tr>
<tr>
<td><strong>LIVEUPDATE</strong></td>
<td>Specifies that the table specified in the <strong>HISTORYTABLE=</strong> option be updated at every evaluation</td>
</tr>
<tr>
<td><strong>LOCALSEARCH</strong></td>
<td>Enables local search optimization</td>
</tr>
<tr>
<td><strong>LOOKUPTABLE=</strong></td>
<td>Specifies the CAS table to use for evaluation lookup</td>
</tr>
<tr>
<td><strong>MAXBAYES=</strong></td>
<td>Specifies the maximum number of points in the kriging model</td>
</tr>
<tr>
<td><strong>MAXEVALS=</strong></td>
<td>Specifies the maximum number of evaluations</td>
</tr>
<tr>
<td><strong>MAXITER=</strong></td>
<td>Specifies the maximum number of iterations when <strong>SEARCHMETHOD=GA</strong> or <strong>SEARCHMETHOD=</strong> BAYESIAN</td>
</tr>
<tr>
<td><strong>MAXTIME=</strong></td>
<td>Specifies the maximum time for all iterations</td>
</tr>
<tr>
<td><strong>MAXTRAINITER=</strong></td>
<td>Specifies the maximum time for a model training</td>
</tr>
<tr>
<td><strong>NCONVITER=</strong></td>
<td>Specifies the number of convergence iterations</td>
</tr>
<tr>
<td><strong>NOGRIDSHUFFLE</strong></td>
<td>Requests that the grid points not be shuffled</td>
</tr>
<tr>
<td><strong>NPARALLEL=</strong></td>
<td>Specifies the number of parallel sessions</td>
</tr>
<tr>
<td><strong>NSUBSESSIONWORKERS=</strong></td>
<td>Specifies the number of workers in parallel sessions</td>
</tr>
<tr>
<td><strong>OBJECTIVE=</strong></td>
<td>Specifies the objective function</td>
</tr>
<tr>
<td><strong>POPSIZE=</strong></td>
<td>Specifies the population size when <strong>SEARCHMETHOD=GA</strong> or <strong>SEARCHMETHOD=</strong> BAYESIAN</td>
</tr>
</tbody>
</table>
Table 7.2 continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMPLESIZE=</td>
<td>Specifies the sample size when SEARCHMETHOD=LHS or SEARCHMETHOD=RANDOM</td>
</tr>
<tr>
<td>SEARCHMETHOD=</td>
<td>Specifies the search method that the optimizer uses</td>
</tr>
<tr>
<td>SECONDOBJECTIVE=</td>
<td>Specifies the second objective to use for tuning</td>
</tr>
<tr>
<td>SELECTINITPOINT</td>
<td>Specifies that the tuner select the best evaluation from the lookup table</td>
</tr>
<tr>
<td>TARGETEVENT=</td>
<td>Specifies the target event for ROC-based calculations</td>
</tr>
<tr>
<td>TRAINFRACTION=</td>
<td>Specifies the fraction of observations to use for training</td>
</tr>
<tr>
<td>TUNINGPARAMETERS=</td>
<td>Specifies the custom tuning parameters</td>
</tr>
<tr>
<td>USEPARAMETERS=</td>
<td>Specifies how to handle the TUNINGPARAMETERS= option</td>
</tr>
</tbody>
</table>

TUNINGPARAMETERS=(suboption | ... | < suboption>)
TUNEPARMS=(suboption | ... | < suboption>)

specifies which parameters to tune and which ranges to tune over. If USEPARAMETERS=STANDARD, this option is ignored.

You can specify one or more of the following suboptions:

NFACTORS (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)

specifies information about the number of factors to use for tuning the factorization machine model. For more information, see the NFACTORS= option in the PROC FACTMAC statement.

You can specify the following additional suboptions:

LB=number
specifies the minimum number of factors to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=5.

UB=number
specifies the maximum number of factors to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=30.

VALUES=value-list
specifies a list of values to consider for the number of factors during tuning, where value-list is a space-separated list of integer numbers greater than or equal to 1. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

INIT=number
specifies the initial number of factors for the tuner to use.

By default, INIT=5.
EXCLUDE
   excludes the number of factors from the tuning process. If you specify this suboption, any
   specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

LEARNSTEP (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
   specifies information about the learning step to use for tuning the factorization machine model.
   For more information, see the LEARNSTEP= option in the PROC FACTMAC statement.
   You can specify the following additional suboptions:

   LB=number
   specifies the minimum learning step to consider during tuning. If you specify this suboption,
   you cannot specify the VALUES= suboption.
   By default, LB=0.001.

   UB=number
   specifies the maximum learning step to consider during tuning. If you specify this suboption,
   you cannot specify the VALUES= suboption.
   By default, UB=1.

   VALUES=value-list
   specifies a list of learning steps to consider during tuning, where value-list is a space-
   separated list of numbers greater than 0. If you specify this suboption, you cannot specify
   either the LB= or UB= suboption.

   INIT=number
   specifies the initial learning step for the tuner to use.
   By default, INIT=0.001.

EXCLUDE
   excludes the learning step from the tuning process. If you specify this suboption, any
   specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

MAXITER (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
   specifies information about the maximum number of iterations to use for tuning the factorization
   machine model. For more information, see the MAXITER= option in the PROC FACTMAC
   statement.
   You can specify the following additional suboptions:

   LB=number
   specifies the minimum number of iterations to consider during tuning. If you specify this
   suboption, you cannot specify the VALUES= suboption.
   By default, LB=10.
**UB=** *number*

specifies the maximum number of iterations to consider during tuning. If you specify this suboption, you cannot specify the **VALUES=** suboption.

By default, UB=200.

**VALUES=** *value-list*

specifies a list of numbers of trees to consider during tuning, where *value-list* is a space-separated list of positive integers. If you specify this suboption, you cannot specify either the **LB=** or **UB=** suboption.

**INIT=** *number*

specifies the initial number of iterations for the tuner to use.

By default, INIT=30.

**EXCLUDE**

excludes the number of iterations from the tuning process. If you specify this suboption, any specified **LB=**, **UB=**, **VALUES=**, and **INIT=** suboptions are ignored.

---

**CODE Statement**

```sas
CODE FILE=filename ;
```

The CODE statement generates SAS DATA step code that mimics the computations that are performed. The generated SAS DATA step code can be used for scoring new observations. Only one CODE statement is processed. If you specify multiple CODE statements, only the first one is used.

You must specify the following option:

**FILE=** *filename*

specifies the name of the file to write the SAS score code to.

The CODE statement is optional. If you do not include a CODE statement, no score code is generated.

---

**DISPLAY Statement**

```sas
DISPLAY < table-list > < / options > ;
```

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE
Chapter 7: The FACTMAC Procedure

The DISPLAYOUT statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “ODS Table Names” on page 137. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

**CASESENSITIVE**

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetted for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

**EXCLUDE**

displays all display tables except those that you specify in the *table-list*.

**EXCLUDEALL**

suppresses display of all tables. This option takes precedence over the other options.

**TRACE**

displays the display table names, labels, and paths.

---

**DISPLAYOUT Statement**

**DISPLAYOUT** *table-spec-list < / options > ;

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

The *table-spec-list* specifies a list of CAS output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

**key=value**

specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the CAS output table name.

**key**

specifies *key* as the ODS table name and also as the CAS output table name.
The ODS table names that you can specify are listed in the section “ODS Table Names” on page 137. You cannot specify the ODS table named OutputCasTables in the `table-spec-list`.

Table names and partial pathnames are discussed under the `DISPLAY` statement. The `DISPLAYOUT` statement does not support regular expressions.

You can specify the following `options` after a slash (/):

**INCLUDEALL** creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the `table-spec-list` specification is ignored.

**NOREPLACE** does not replace any existing CAS output table of the same name.

**REPEATED** replicates all CAS output tables on all nodes.

---

**ID Statement**

```plaintext
ID variables ;
```

The ID statement lists one or more variables that are to be copied from the input data table to the output data tables that are specified in the `OUT=` option in the `OUTPUT` statement and the `RSTORE=` option in the `SAVestate` statement.

---

**INPUT Statement**

```plaintext
INPUT variables < LEVEL=INTERVAL | NOMINAL > ;
```

The INPUT statement specifies the names of the `variables` to be used in the factorization. It names one or more input variables that use common options. If you want to use different options for different variables, you can specify multiple INPUT statements.

You can include the following option in each INPUT statement:

**LEVEL=INTERVAL | NOMINAL**

specifies the level of measurement of the `variables`. You can specify the following values:

- **INTERVAL** specifies that the level of measurement of the `variables` is interval.
- **NOMINAL** specifies that the level of measurement of the `variables` is nominal.

By default, `LEVEL=INTERVAL` for numeric variables and `LEVEL=NOMINAL` for categorical variables.

You must specify at least two nominal input variables. You can also specify any number of interval input variables.
OUTPUT Statement

```
OUTPUT OUT=CAS-libref.data-table < options > ;
```

The OUTPUT statement creates an output data table to contain the results of the procedure run. You cannot specify both the OUTPUT and AUTOTUNE statements in the same run of PROC FACTMAC.

You must specify the following option:

**OUT=**  

names the output data table for PROC FACTMAC to use. You must specify this option before any other options. *CAS-libref.data-table* is a two-level name, where

- *CAS-libref* refers to a collection of information that is defined in the LIBNAME statement and includes the *caslib*, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 123.
- *data-table* specifies the name of the output data table.

You can also specify the following option:

**COPYVAR=**  

**COPYVARS=(**  

lists one or more *variables* from the input data table to be transferred to the output data table.

SAVESTATE Statement

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 4, “The ASTORE Procedure.”

You must specify the following option:

**RSTORE=**  

specifies a data table in which to save the analytic store for the model. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the *caslib* and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 123.
The TARGET statement names the target variable whose values PROC FACTMAC predicts. The target must be interval and must be different from the variables in the INPUT statement. You can include the following option in the OUTPUT statement:

**LEVEL=INTERVAL**

specifies the level of measurement of the variables.

PROC FACTMAC currently accepts only interval target variables.

---

**Details: FACTMAC Procedure**

The factorization machines model is defined as

\[
\hat{y}(x) = w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j x_{j'} \sum_{f=1}^{k} v_{jf} v_{j'f}
\]

where \(x = (x_1, \ldots, x_p)\) 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 \(v_j \in \mathbb{R}^k\). The overall factor matrix \(V \in \mathbb{R}^{p \times k}\) is the concatenation of the row vectors \(v_j\) for \(j = 1, \ldots, p\). The number of factors is \(k\). PROC FACTMAC estimates the model parameters \(w_0, w_1, \ldots, w_p\) and \(V\). The estimation is done by minimizing the root mean square error (RMSE), which is defined by

\[
RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2}
\]

over the training set, subject to the max-norm regularization constraint

\[
\|v_j\|_\infty < B, \ j = 1, \ldots, p
\]

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 PROC FACTMAC are reproducible only if you specify a value greater than 0 for the SEED= option and specify NTHREADS=1, because PROC FACTMAC 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. PROC FACTMAC can still use multiple machines for the analysis even when NTHREADS=1.
Displayed Output

The FACTMAC procedure displays various tables that are related to the factorization. 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 factorization analysis. This information includes the maximum number of iterations, learning step, number of factors, and seed value.

Number of Observations

The “Number of Observations” table displays the number of observations that are read from the input data table and used.

Iteration History

The “Iteration History” table displays the iteration history and approximate loss when the variables that are specified in the INPUT statement are interval.

The “displayed loss” is an approximation for computational efficiency reasons. The final exact loss is shown in the “Final Exact Loss” table.

Final Exact Loss

The “Final Exact Loss” table displays the actual, exact mean square error (MSE) and the 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 for the interval 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. Because this table is not a typical ODS table that contains analytical results, you cannot include it in the table-spec-list in the DISPLAYOUT statement.
ODS Table Names

Each table created by the FACTMAC procedure has a name associated with it, and you must use this name to refer to the table when you use ODS statements. The names of each table and a short description of the contents are listed in Table 7.3.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>DescStatsInt</td>
<td>Descriptive statistics for interval variables</td>
<td>PROC FACTMAC</td>
<td>Default</td>
</tr>
<tr>
<td>FinalLoss</td>
<td>Final exact loss</td>
<td>PROC FACTMAC</td>
<td>Default</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>PROC FACTMAC</td>
<td>Default</td>
</tr>
<tr>
<td>NObs</td>
<td>Number of observations</td>
<td>PROC FACTMAC</td>
<td>Default</td>
</tr>
<tr>
<td>OptIterHistory</td>
<td>Iteration history</td>
<td>PROC FACTMAC</td>
<td>Default</td>
</tr>
<tr>
<td>OutputCASTables</td>
<td>See the section “OutputCasTables Table” on page 136</td>
<td>PROC FACTMAC</td>
<td>Default</td>
</tr>
</tbody>
</table>

Output Data Tables

The FACTMAC procedure creates a data table to which it writes the global biases and the factors. You specify the name of this data table in the OUTMODEL= option in the PROC FACTMAC statement. Details about the data table are listed in Table 7.4.

<table>
<thead>
<tr>
<th>Data Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FACTORS</td>
<td>Lists the global bias, the name of each input variable, each level and the values of the estimated factors</td>
</tr>
</tbody>
</table>
Examples: FACTMAC Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

---

**Example 7.1: Running PROC FACTMAC with the MovieLens Data Set**

This example draws on data that 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](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), 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](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. The file that contains the ratings is `u.data`. Assuming the directory is `/dept/app/doc/vb023/en/src/casml/factmac`, the following DATA step loads the data table from the directory into your CAS session:

```sas
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 show how to use PROC FACTMAC to predict movie ratings:

```sas
proc factmac data=mycas.movlens nfactors=10 learnstep=0.15
  maxiter=20 outmodel=mycas.factors;
  input userid itemid /level=nominal;
  target rating /level=interval;
  output out=mycas.out1 copyvars=(userid itemid rating);
run;
```

---

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.
Example 7.1: Running PROC FACTMAC with the MovieLens Data Set

The following statements print the first 10 observations in the mycas.factors data table, which is specified in the OUTMODEL= option in the PROC FACTMAC statement. The output is shown in Output 7.1.1.

```sas
proc print data=mycas.factors(obs=10);
run;
```

**Output 7.1.1** Bias Values and Factors

<table>
<thead>
<tr>
<th>Obs</th>
<th>Variable</th>
<th>Level</th>
<th>Bias</th>
<th>Factor1</th>
<th>Factor2</th>
<th>Factor3</th>
<th>Factor4</th>
<th>Factor5</th>
<th>Factor6</th>
<th>Factor7</th>
<th>Factor8</th>
<th>Factor9</th>
<th>Factor10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><em>GLOBAL</em></td>
<td></td>
<td>3.52986</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>userid</td>
<td>1</td>
<td>0.08043</td>
<td>0.14034</td>
<td>0.36798</td>
<td>-0.52760</td>
<td>-0.31465</td>
<td>0.22486</td>
<td>-0.28397</td>
<td>0.87780</td>
<td>0.24885</td>
<td>0.02000</td>
<td>-0.14646</td>
</tr>
<tr>
<td>3</td>
<td>userid</td>
<td>2</td>
<td>0.17982</td>
<td>0.54746</td>
<td>0.00647</td>
<td>-0.47194</td>
<td>-0.39421</td>
<td>-0.19759</td>
<td>0.51829</td>
<td>-0.10209</td>
<td>0.01729</td>
<td>0.26124</td>
<td>-0.12959</td>
</tr>
<tr>
<td>4</td>
<td>userid</td>
<td>3</td>
<td>-0.73356</td>
<td>-0.54006</td>
<td>0.01826</td>
<td>0.31498</td>
<td>0.11264</td>
<td>0.73452</td>
<td>0.31893</td>
<td>-0.06499</td>
<td>-0.63882</td>
<td>-0.45544</td>
<td>0.18285</td>
</tr>
<tr>
<td>5</td>
<td>userid</td>
<td>4</td>
<td>0.80347</td>
<td>0.26031</td>
<td>0.04870</td>
<td>-0.25062</td>
<td>-0.01312</td>
<td>-0.29526</td>
<td>0.53290</td>
<td>-0.58693</td>
<td>0.00283</td>
<td>0.36615</td>
<td>0.40131</td>
</tr>
<tr>
<td>6</td>
<td>userid</td>
<td>5</td>
<td>-0.65557</td>
<td>0.51211</td>
<td>0.07824</td>
<td>-0.08614</td>
<td>-0.01463</td>
<td>0.46066</td>
<td>-0.30982</td>
<td>-0.21790</td>
<td>0.37157</td>
<td>-0.61646</td>
<td>-0.29942</td>
</tr>
<tr>
<td>7</td>
<td>userid</td>
<td>6</td>
<td>0.10521</td>
<td>0.17515</td>
<td>0.18334</td>
<td>-0.51516</td>
<td>0.53364</td>
<td>-0.55709</td>
<td>-0.16770</td>
<td>-0.05254</td>
<td>0.39754</td>
<td>0.29666</td>
<td>0.23568</td>
</tr>
<tr>
<td>8</td>
<td>userid</td>
<td>7</td>
<td>0.43540</td>
<td>0.25194</td>
<td>0.07348</td>
<td>-0.05654</td>
<td>-0.02345</td>
<td>-0.24364</td>
<td>0.14093</td>
<td>-0.03300</td>
<td>-0.46717</td>
<td>0.51100</td>
<td>-0.19197</td>
</tr>
<tr>
<td>9</td>
<td>userid</td>
<td>8</td>
<td>0.26675</td>
<td>0.11208</td>
<td>0.57564</td>
<td>0.02860</td>
<td>-0.77657</td>
<td>-0.16036</td>
<td>-0.41215</td>
<td>-0.06069</td>
<td>0.76389</td>
<td>-0.00608</td>
<td>0.11334</td>
</tr>
<tr>
<td>10</td>
<td>userid</td>
<td>9</td>
<td>0.74287</td>
<td>0.18856</td>
<td>-0.10608</td>
<td>0.22230</td>
<td>-1.01906</td>
<td>-0.18707</td>
<td>-0.08458</td>
<td>-0.02147</td>
<td>-0.82411</td>
<td>-0.21252</td>
<td>-0.09411</td>
</tr>
</tbody>
</table>

The following statements print the predicted movie ratings for the first 20 observations, as shown in Output 7.1.2.

```sas
proc print data=mycas.out1(obs=20);
run;
```

**Output 7.1.2** Predicted Movie Ratings

<table>
<thead>
<tr>
<th>Obs</th>
<th>userid</th>
<th>itemid</th>
<th>rating</th>
<th>P_rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>196</td>
<td>242</td>
<td></td>
<td>4.09834</td>
</tr>
<tr>
<td>2</td>
<td>186</td>
<td>302</td>
<td></td>
<td>3.78284</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>377</td>
<td>1</td>
<td>1.42463</td>
</tr>
<tr>
<td>4</td>
<td>244</td>
<td>51</td>
<td>2</td>
<td>3.20907</td>
</tr>
<tr>
<td>5</td>
<td>166</td>
<td>346</td>
<td>1</td>
<td>2.58391</td>
</tr>
<tr>
<td>6</td>
<td>298</td>
<td>474</td>
<td>4</td>
<td>4.60470</td>
</tr>
<tr>
<td>7</td>
<td>115</td>
<td>265</td>
<td>2</td>
<td>3.43183</td>
</tr>
<tr>
<td>8</td>
<td>253</td>
<td>465</td>
<td>5</td>
<td>4.57718</td>
</tr>
<tr>
<td>9</td>
<td>305</td>
<td>451</td>
<td>3</td>
<td>2.96174</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>86</td>
<td>3</td>
<td>4.44866</td>
</tr>
<tr>
<td>11</td>
<td>62</td>
<td>257</td>
<td>2</td>
<td>3.08217</td>
</tr>
<tr>
<td>12</td>
<td>286</td>
<td>1014</td>
<td>5</td>
<td>3.08536</td>
</tr>
<tr>
<td>13</td>
<td>200</td>
<td>222</td>
<td>5</td>
<td>4.46180</td>
</tr>
<tr>
<td>14</td>
<td>210</td>
<td>40</td>
<td>3</td>
<td>3.30407</td>
</tr>
<tr>
<td>15</td>
<td>224</td>
<td>29</td>
<td>3</td>
<td>3.15151</td>
</tr>
<tr>
<td>16</td>
<td>303</td>
<td>785</td>
<td>3</td>
<td>3.03463</td>
</tr>
<tr>
<td>17</td>
<td>122</td>
<td>387</td>
<td>5</td>
<td>4.47396</td>
</tr>
<tr>
<td>18</td>
<td>194</td>
<td>274</td>
<td>2</td>
<td>2.57941</td>
</tr>
<tr>
<td>19</td>
<td>291</td>
<td>1042</td>
<td>4</td>
<td>3.47518</td>
</tr>
<tr>
<td>20</td>
<td>234</td>
<td>1184</td>
<td>2</td>
<td>1.62934</td>
</tr>
</tbody>
</table>
References

Chapter 8
The FASTKNN Procedure

Overview: FASTKNN Procedure

The FASTKNN procedure implements the $k$-nearest neighbor ($k$-NN) search algorithm in SAS Viya. The $k$-NN algorithm has numerous applications, including recommendation systems, image search, fingerprint recognition, and clustering. You can use the FASTKNN procedure to read and write data in distributed form, and to perform search in parallel by making full use of multicore computers or distributed computing environments.

The FASTKNN procedure operates on a query data table and an input data table. For each observation in the query data table, PROC FASTKNN returns the $k$ observations in the input data table that are nearest. The FASTKNN procedure uses the Euclidean distance metric to compute distances between the input and query observations, and stores the results of the search in the output data table that is specified by the OUTPUT statement. The FASTKNN procedure can use exact search or approximate search. Approximate search is faster, but is not guaranteed to return all $k$ neighbors. In addition, PROC FASTKNN stores the distances between the query and input observations in the output data table that is specified by the OUTDIST= option in the PROC FASTKNN statement. You can request that PROC FASTKNN exclude distances that are greater than a specified threshold value.
PROC FASTKNN Features

PROC FASTKNN enables you to use parallel execution for k-NN search in a distributed computing environment or on a single-machine. The following list summarizes the basic features of PROC FASTKNN:

- is highly distributed and multithreaded
- returns k-nearest neighbors based on a parallel implementation of the k-NN search algorithm

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

    proc options option=(CASHOST CASPORT);
    run;

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

    cas mysess;
    libname mycas cas sessref=mysess;

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

    cas mysess terminate;

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Getting Started: FASTKNN Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example shows how to use the FASTKNN procedure to search for the \(k\)-nearest neighbors, from observations in a SAS data table to a set of query observations. In this case, the data are from the hmeq data set. This data set contains information about mortgage applicants. The example selects 4,000 applicants for the input data table, and 100 applicants for the query data table. The FASTKNN procedure 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 `debtinc`. The remaining variables in the data table are not used.

You can load the hmeq 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.

```sas
data mycas.hmeq;
set sampsio.hmeq(obs=4000);
id=_n_; run;
```

```sas
data mycas.query;
set sampsio.hmeq(firstobs=4001 obs=4100);
id=_n_; run;
```

The following statements run PROC FASTKNN and output the results to ODS tables:

```sas
proc fastknn
data = mycas.hmeq
query = mycas.query
outdist = mycas.dist_out
parallelization = input
K = 4
threshdist = 10000;
id id;
input bad loan mortdue value yoj derog delinq clage ninq clno debtinc;
output out = mycas.knn_out;
run;
```

The PARALLELIZATION=INPUT option requests that the input data table be distributed among worker nodes. The K=4 option requests that the model return four neighbors; the THRESHDIST=10000 option requests that the model exclude distances that are greater than 10,000; the ID statement specifies that the id variable be used as a record identifier; and the INPUT statement specifies that the variables `bad`, `loan`, `mortdue`, `value`, `yoj`, `derog`, `delinq`, `clage`, `ninq`, `clno`, and `debtinc` be used as inputs. The OUTPUT statement requests that the neighbors be written to the data table `mycas.knn_out`. The OUTDIST= option requests that the distances from the query observations to the input observations be written to the data table `mycas.dist_out`. 
Syntax: FASTKNN Procedure

The following statements are available in the FASTKNN procedure:

```
PROC FASTKNN <options> ;
  DISPLAY <table-list> </options> ;
  DISPLAYOUT table-spec-list </options> ;
  ID variables ;
  INPUT variables <LEVEL=INTERVAL> ;
  OUTPUT OUT=CAS-libref.data-table <options> ;
```

The PROC FASTKNN statement, the ID statement, and at least one INPUT statement, are required. You can specify multiple INPUT statements.

The following sections describe the PROC FASTKNN statement and then describe the other statements in alphabetical order.

PROC FASTKNN Statement

```
PROC FASTKNN <options> ;
```

The PROC FASTKNN statement invokes the procedure. Table 8.1 summarizes the options available in the PROC FASTKNN statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Data Table Options</strong></td>
<td></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the input data table</td>
</tr>
<tr>
<td>QUERY=</td>
<td>Specifies the query data table</td>
</tr>
<tr>
<td><strong>Nearest Neighbor Options</strong></td>
<td></td>
</tr>
<tr>
<td>K=</td>
<td>Specifies the number of neighbors to return for each query</td>
</tr>
<tr>
<td>METHOD=</td>
<td>Specifies the method to use to compute nearest neighbors</td>
</tr>
<tr>
<td>NOPRINT</td>
<td>Suppresses ODS output</td>
</tr>
<tr>
<td>NTHREADS=</td>
<td>Specifies the number of threads to use on each computation node</td>
</tr>
<tr>
<td>OUTDIST=</td>
<td>Specifies the output data table in which to store the computed distances</td>
</tr>
<tr>
<td>PARALLELIZATION=</td>
<td>Specifies whether to distribute the query data table or the input data table among worker nodes</td>
</tr>
<tr>
<td>SEED=</td>
<td>Specifies the seed to use for pseudorandom number generation</td>
</tr>
<tr>
<td>THRESHDIST=</td>
<td>Specifies the threshold on the distance</td>
</tr>
</tbody>
</table>
You can specify the following options:

**DATA=CAS-libref.data-table**

names the input data table for PROC FASTKNN to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

*CAS-libref* refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 142.

*data-table* specifies the name of the input data table.

**K=number**

specifies the number of neighbors to return, where *number* is a positive integer.

By default, K=2.

**METHOD=APPROXIMATE | EXACT**

specifies which method for the algorithm to use. You can specify the following values:

- **APPROXIMATE** uses approximate search.
- **EXACT** uses exact search.

By default, METHOD=EXACT.

**NOPRINT**

suppresses ODS output.

**NTHREADS=number-of-threads**

specifies the number of threads to use for the computation, where *number-of-threads* is an integer from 1 to 64, inclusive. The default value is the maximum number of available threads per computer.

**OUTDIST=CAS-libref.data-table**

specifies the output model data table to contain the computed pairwise distances between query and input observations. When METHOD=EXACT, all pairwise distances are computed. When METHOD=APPROXIMATE, not all distances are guaranteed to be computed. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 142.

**PARALLELIZATION=INPUT | QUERY**

specifies which data table to distribute among worker nodes. You can specify the following values:

- **INPUT** distributes the input data table among worker nodes. Specify this value only when the input data table is much larger than the query data table.
- **QUERY** distributes the query data table among worker nodes.

By default, PARALLELIZATION=QUERY.
Chapter 8: The FASTKNN Procedure

**QUERY=** *CAS-libref.data-table*

specifies the input data table that contains the query observations. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 142.

**SEED=** *random-seed*

specifies a real number that is used to start the pseudorandom number generator. This option enables you to reproduce the same sample output. If you do not specify a seed or you specify a value less than or equal to 0, the seed is generated from reading the time of day from the computer’s clock.

By default, **SEED=0**.

**THRESHDIST=** *number*

specifies the distance threshold for the model, where *number* is a real number greater than 0. PROC FASTKNN excludes from the computation distances greater than *number*. When **METHOD=EXACT**, all pairwise distances less than or equal to *number* are computed. When **METHOD=APPROXIMATE**, it is possible that only a subset of those distances less than or equal to *number* are computed.

By default, **THRESHDIST=100**.

**DISPLAY Statement**

**DISPLAY < table-list> < / options> ;**

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the **table-list** as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “ODS Table Names” on page 150. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path **Bygroup1.Summary.SelectionSummary**. A partial pathname does not include all groups; for example, **SelectionSummary** and **Summary.SelectionSummary** are partial pathnames for **Bygroup1.Summary.SelectionSummary**.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both **SelectionSummary** and **Summary.SelectionSummary** select **Bygroup1.Summary.SelectionSummary**.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the **Bygroup1.Summary.SelectionSummary** table is selected.
Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the Bygroup1.Summary.SelectionSummary table is not selected.

You can specify the following options after a slash (/):

**CASESENSITIVE**

performs a case-sensitive comparison of table names in the table-list to display table names when tables are subsetted for display. To preserve case, you must enclose table names in the table-list in quotation marks.

**EXCLUDE**

displays all display tables except those that you specify in the table-list.

**EXCLUDEALL**

suppresses display of all tables. This option takes precedence over the other options.

**TRACE**

displays the display table names, labels, and paths.

---

**DISPLAYOUT Statement**

```sas
DISPLAYOUT table-spec-list < / options > ;
```

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

The `table-spec-list` specifies a list of CAS output tables to create. Each entry in the list has either a `key=value` format or a `key` format:

- `key=value` specifies `key` as the ODS table name, path, or partial pathname, and specifies `value` as the CAS output table name.
- `key` specifies `key` as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “ODS Table Names” on page 150. You cannot specify the ODS table named OutputCasTables in the `table-spec-list`.

Table names and partial pathnames are discussed under the DISPLAY statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following options after a slash (/):

**INCLUDEALL**

creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the `table-spec-list` specification is ignored.
**Chapter 8: The FASTKNN Procedure**

NOREPLACE

does not replace any existing CAS output table of the same name.

REPEATED

replicates all CAS output tables on all nodes.

---

**ID Statement**

```
ID variable ;
```

The ID statement lists one variable to be copied from the input data table to the output data tables that are specified in the OUT= option in the OUTPUT statement and in the OUTDIST= option in the PROC FASTKNN statement. You must specify one `variable`, which must be numeric.

---

**INPUT Statement**

```
INPUT variables ;
```

The INPUT statement specifies the `variables` to be used in the nearest neighbor search. The `variables` must be of interval type. You must specify at least one input `variable`.

---

**OUTPUT Statement**

```
OUTPUT OUT=CAS-libref.data-table < options > ;
```

The OUTPUT statement creates a nearest neighbor data table to contain the results of the procedure run.

You must specify the following `option`:

**OUT=**

`CAS-libref.data-table`

names the output data table for PROC FASTKNN to use. You must specify this option before any other options. `CAS-libref.data-table` is a two-level name, where

`CAS-libref`

refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 142.

`data-table`

specifies the name of the output data table.

You can also specify the following `option`:
COPYVAR=variable
COPYVARS=(variables)

d-lists one or more variables from the input data table to be transferred to the output data table.

Details: FASTKNN Procedure

The distance metric in the FASTKNN procedure 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. When METHOD=EXACT, all input observations are considered. When METHOD=APPROXIMATE, the input observations are stored in multiple randomized binary trees. Only the input observations that are stored in either the same leaf node as a query observation or in an adjacent leaf node are considered as candidate neighbors. You can specify the number of binary trees and the maximum number of observations in each leaf node in the NTREES= and MAXPOINTS= options, respectively. Using more trees and a higher maximum number of points yields more accurate results, but uses more computational resources.

Displayed Output

The FASTKNN procedure displays various tables that are related to the \( k \)-nearest neighbor search algorithm. 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 \( k \)-NN algorithm. This information includes the number of neighbors (\( k \)), the method used for the \( k \)-NN search, and the value of the distance threshold that is specified in the THRESHDIST= option in the PROC FASTKNN statement. In addition, when METHOD=APPROXIMATE, the “Model Information” table includes the following values, which are specified in options in the PROC FASTKNN statement: the seed used for random number generation (specified in the SEED= option), the number of binary trees (specified in the NTREES= option), and the maximum number of points in the leaf nodes (specified in the MAXPOINTS= option).
Chapter 8: The FASTKNN Procedure

Number of Observations

The “Number of Observations” table displays the number of observations that are read from the input data table and used.

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, thecaslib in which the table resides, and the number of columns and rows in the CAS table. Because this table is not a typical ODS table that contains analytical results, you cannot include it in the `table-spec-list` in the `DISPLAYOUT` statement.

ODS Table Names

Each table that the FASTKNN procedure creates has a name associated with it, and you must use this name to refer to the table when you use ODS statements. The names of each table and a short description of its contents are listed in Table 8.2.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>PROC FASTKNN</td>
<td>Default</td>
</tr>
<tr>
<td>NObs</td>
<td>Number of observations</td>
<td>PROC FASTKNN</td>
<td>Default</td>
</tr>
<tr>
<td>OutputCASTables</td>
<td>See the section “OutputCasTables Table” on page 150</td>
<td>PROC FASTKNN</td>
<td>Default</td>
</tr>
</tbody>
</table>
Output Data Tables

The FASTKNN procedure creates data tables to which it writes the \( k \)-nearest neighbors and the pairwise distances between query and input. You specify the names of these data tables in the OUTPUT statement and in the OUTDIST= option of the PROC FASTKNN statement, respectively. Details about the data tables are listed in Table 8.3.

Table 8.3  Output Data Tables Produced by PROC FASTKNN

<table>
<thead>
<tr>
<th>Data Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN_OUT</td>
<td>Lists the nearest neighbors</td>
</tr>
<tr>
<td>DIST_OUT</td>
<td>Lists the distances from query</td>
</tr>
<tr>
<td></td>
<td>observations to the input observations</td>
</tr>
</tbody>
</table>
Overview: FISM Procedure

The FISM procedure performs frequent item set mining, which looks for frequent patterns in a large database. The FISM procedure finds frequent patterns by using the FP-growth (frequent-pattern growth) algorithm of Han, Pei, and Yin (2000). The FP-growth algorithm uses a special data structure called the frequent-pattern tree (FP-tree), which retains the item set association information. For more information about the FP-growth algorithm, see the section “FP-Growth Algorithm” on page 158.

The FISM procedure is used by other procedures such as the MBANALYSIS procedure for association rule mining.
PROC FISM Features

The FISM procedure has the following features:

- reads input data in parallel when the data source is on a distributed system
- supports the FP-growth algorithm, which is currently one of the fastest approaches to frequent item set mining
- supports parallel implementation of the FP-growth algorithm in a distributed environment for faster performance
- stores large databases in compact FP-tree format, which uses memory efficiently.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
  libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example finds frequent item sets in the `sampsio.assocs` data set. This data set contains 1,000 observations, each of which contains products that are purchased by a customer.

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:

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

These statements assume that your CAS engine libref is named `mycas`, as in the section “Using CAS Sessions and CAS Engine Librefs” on page 154, but you can substitute any appropriately defined CAS engine libref.

The following statements execute the FISM procedure on the `mycas.assocs` data table and generate output in `mycas.out` data table, which is specified in the OUTPUT statement:

```sas
proc fism data=mycas.assocs items=3 support=100;
  output out=mycas.out outfreq=mycas.outfreq;
  customer Customer;
  target product;
run;
```

The PROC FISM statement specifies that input data are contained in the `mycas.assocs` table and specifies the rule generation criteria: number of items and level of support. The OUTPUT statement names the output tables for storing results. The CUSTOMER statement identifies `Customer` as the identification variable for transactions. The TARGET statement defines `Product` as the target variable for transaction items.

This procedure produces a list of frequent items, which is stored in the `outfreq` table, and frequent item sets, which are stored in the `out` table.
Syntax: FISM Procedure

The following statements are available in the FISM procedure:

```
PROC FISM options ;
  CUSTOMER variable ;
  OUTPUT options ;
  TARGET variable ;
```

All statements are required.

The following sections describe the PROC FISM statement and then describe the other statements in alphabetical order.

PROC FISM Statement

```
PROC FISM options ;
```

The PROC FISM statement invokes the procedure.

You must specify the following option:

- **DATA=CAS-libref.data-table**
  
  names the input data table for PROC FISM to use. `CAS-libref.data-table` is a two-level name, where

  - `CAS-libref` refers to a collection of information that is defined in the LIBNAME statement and includes the `caslib`, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 154.

  - `data-table` specifies the name of the input data table.

You must specify either of the following options:

- **SUPPORT=number**
  
  specifies the minimum level of support (minimum frequency of an item) for a rule, where `number` must be an integer greater than or equal to 0. This option overrides the specification of the PCTSUP= option.

- **PCTSUP=number**

- **SUPPCT=number**

- **SUP_PCT=number**

- **PCTSUPPORT=number**

  specifies the minimum level of support for a rule as a percentage of the number of baskets in the input data table, where `number` must be a real number between 0 and 100, inclusive. This option is ignored if the SUPPORT= option is specified.

You can also specify either of the following options:
ITEMS=number
specifies the number of items in a frequent item set, where number must be an integer between 1 and 1,000. By default, ITEMS=2 when you specify the OUT= option in the OUTPUT statement; otherwise, ITEMS=1 by default. This option is ignored if you specify the NFIS_RANGE= option.

NFIS_RANGE=(number, number)
specifies the range of number of items in the frequent item sets, where number must be an integer between 1 and 1,000.

By default, NFIS_RANGE=(1,1000).

CUSTOMER Statement

CUSTOMER variable ;
The CUSTOMER statement specifies the variable that is used to group the target variable into a basket. You can also specify this statement as ID variable.

OUTPUT Statement

OUTPUT options ;
The OUTPUT statement specifies output tables that contain result of FISM procedure.

You must specify at least one of the following options:

OUT=CAS-libref.data-table
creates a data table that contains information about frequent item sets with their transaction counts and support. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 154.

OUTFREQ=CAS-libref.data-table
creates a data table that contains information about the unique frequent items with their transaction counts and support. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 154.
TARGET Statement

TARGET variable;

The TARGET statement specifies a single nominal variable to use as the target variable. If this variable is not present in the input data table, then PROC FISM exits with an error.

Details: FISM Procedure

The FISM procedure uses support input to identify the frequent item sets. By default, item sets that contain a maximum of two items are generated. You can specify the item set size in the ITEMS= option.

PROC FISM implements the FP-growth algorithm for finding frequent item sets. PROC FISM discovers the frequent item set by using a support (coverage) measure, which indicates how frequently the items appear in the database. Support of a rule for \( A \) and \( B \) is expressed as

\[
A \Rightarrow B : P(A, B) = \frac{X}{Y}
\]

where \( X \) is the number of transactions that contain both \( A \) and \( B \), and \( Y \) is the 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 in 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 algorithms can reduce the search space by doing a restricted generation-and-test process. The Apriori algorithm is the one of the most efficient association rule mining algorithms.

The Apriori algorithm of Agrawal and Srikant (1994) 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 links 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.

When items are sorted in support-descending order, more frequent items are arranged closer to the top of the FP-tree and thus are more likely to be shared. This arrangement suggests that an FP-tree is usually a highly compact projection of the entire data set. However, if the data set 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 sets could be multiple times the size of the original data set.
- Partition projection is done in sequence: an observation $T$ is projected to the $X$-projected data set 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 sets at each level is smaller than the original data set.

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 the conditional FP-tree (CFPT). The mining function is defined as $\text{mine}(\text{CFPT} | \text{node})$, which is also a recursive function that uses a divide-and-conquer process.

### Displayed Output

The FISM procedure does not produce any displayed output tables.

### Results

The results that the FISM procedure produces are stored in tables. You can access these tables by their names, which are shown in Table 9.1.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>Contains general information about the frequent item sets</td>
<td>PROC FISM</td>
<td>OUT=</td>
</tr>
<tr>
<td>outfreq</td>
<td>Contains information about the items’ frequencies</td>
<td>PROC FISM</td>
<td>UTFREQ=</td>
</tr>
</tbody>
</table>
Limitations

When the number of observations is greater than 10 million and the number of items in a frequent item set (as specified in the ITEMS= option in the PROC FISM statement) is considerably large (more than 10), the amount of memory required to process the analysis might be so great that the procedure stops responding.

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

Examples: FISM Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

Example 9.1: Identifying Frequent Item Sets

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example identifies frequent item sets in the `sampsio.assocs` data set. This data set contains 1,000 transactions of 1,000 customers, each of which contains customer, time, and product information.

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:

```sas
data mycas.assocs;
  set sampsio.assocs;
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 FP-growth algorithm on the `mycas.assocs` data table:

```sas
proc fism data=mycas.assocs items=5 support=200;
  output out=mycas.out outfreq=mycas.outfreq;
  customer Customer;
  target product;
run;
```

The DATA= option names the input data table to be analyzed. The ITEMS= option specifies the number of items in a frequent item set. The SUPPORT= option specifies the minimum level of support for a rule. The OUTPUT statement specifies output tables to contain results of the FISM procedure. The CUSTOMER statement requests that the Customer variable be used to group the target variable into baskets. The TARGET statement requests that the Product variable be used as the target variable.
References


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 10
## The FOREST Procedure

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: FOREST Procedure</td>
<td>164</td>
</tr>
<tr>
<td>PROC FOREST Features</td>
<td>165</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>165</td>
</tr>
<tr>
<td>Getting Started: FOREST Procedure</td>
<td>166</td>
</tr>
<tr>
<td>Syntax: FOREST Procedure</td>
<td>170</td>
</tr>
<tr>
<td>PROC FOREST Statement</td>
<td>171</td>
</tr>
<tr>
<td>AUTOTUNE Statement</td>
<td>176</td>
</tr>
<tr>
<td>CODE Statement</td>
<td>181</td>
</tr>
<tr>
<td>CROSSVALIDATION Statement</td>
<td>181</td>
</tr>
<tr>
<td>GROW Statement</td>
<td>182</td>
</tr>
<tr>
<td>ID Statement</td>
<td>183</td>
</tr>
<tr>
<td>INPUT Statement</td>
<td>183</td>
</tr>
<tr>
<td>OUTPUT Statement</td>
<td>184</td>
</tr>
<tr>
<td>PARTITION Statement</td>
<td>185</td>
</tr>
<tr>
<td>SAVESTATE Statement</td>
<td>185</td>
</tr>
<tr>
<td>TARGET Statement</td>
<td>186</td>
</tr>
<tr>
<td>VIICODE Statement</td>
<td>186</td>
</tr>
<tr>
<td>WEIGHT Statement</td>
<td>187</td>
</tr>
<tr>
<td>Details: FOREST Procedure</td>
<td>187</td>
</tr>
<tr>
<td>Bagging the Data</td>
<td>187</td>
</tr>
<tr>
<td>Training a Decision Tree</td>
<td>187</td>
</tr>
<tr>
<td>Loh Method</td>
<td>188</td>
</tr>
<tr>
<td>Predicting an Observation</td>
<td>189</td>
</tr>
<tr>
<td>Measuring Prediction Error</td>
<td>189</td>
</tr>
<tr>
<td>Handling Missing Values</td>
<td>190</td>
</tr>
<tr>
<td>Strategies</td>
<td>190</td>
</tr>
<tr>
<td>Specifics</td>
<td>190</td>
</tr>
<tr>
<td>Handling Values That Are Absent from Training Data</td>
<td>191</td>
</tr>
<tr>
<td>Measuring Variable Importance</td>
<td>191</td>
</tr>
<tr>
<td>Residual Sum of Squares Importance Method</td>
<td>191</td>
</tr>
<tr>
<td>Random Branch Assignment Importance Method</td>
<td>194</td>
</tr>
<tr>
<td>Isolation Forests</td>
<td>194</td>
</tr>
<tr>
<td>Hyperparameter Tuning</td>
<td>195</td>
</tr>
<tr>
<td>k-fold Cross Validation</td>
<td>195</td>
</tr>
<tr>
<td>Determining the Number of Parallel Evaluations</td>
<td>195</td>
</tr>
<tr>
<td>Displayed Output</td>
<td>196</td>
</tr>
</tbody>
</table>
Overview: FOREST Procedure

The FOREST procedure creates a predictive model called a forest (which consists of several decision trees) in SAS Viya. A predictive model defines a relationship between input variables and a target variable. The purpose of a predictive model is to predict a target value from inputs. The FOREST procedure trains the model; that is, it creates the model by using training data in which the target values are known. The model can then be applied to observations in which the target is unknown. If the predictions fit the new data well, the model is said to generalize well. Good generalization is the primary goal for predictive tasks. A predictive model might fit the training data well but generalize poorly.

A decision tree is a type of predictive model that has been developed independently in the statistics and artificial intelligence communities. The FOREST procedure creates a tree recursively: The procedure chooses an input variable and uses it to create a rule to split the data into two or more subsets. The process is then repeated in each subset, and then again in each new subset, and so on until some constraint is met. In the terminology of the tree metaphor, the subsets are nodes, the original data table is the root node, and the final unpartitioned subsets are leaves or terminal nodes. A node is an internal node if it is not a leaf. The data in a leaf determine the estimates of the value of the target variable. These estimates are subsequently applied to predict the target of a new observation that is assigned to the leaf.

The FOREST procedure creates multiple decision trees that differ from each other in two ways: First, the training data for each tree constitute a different sample; each sample is created by sampling with replacement observations from the original training data of the forest. Second, the input variables that are considered for splitting a node are randomly selected from all available inputs. Among these randomly selected variables, the FOREST procedure chooses a single variable, which is associated the most with the target, when it forms a splitting rule.
PROC FOREST Features

The FOREST procedure creates an ensemble of decision trees to predict a single target of either interval or nominal measurement level. An input variable can have an interval or nominal measurement level.

The FOREST procedure ignores any observation from the training data that has a missing target value.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```sas
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```sas
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```sas
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Chapter 10: The FOREST Procedure

Getting Started: FOREST Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

A common use of forest models is to predict whether a mortgage applicant will default on a loan. The home equity data table Hmeq, which is in the Sampsio library that SAS provides, contains observations for 5,960 mortgage applicants. A variable named Bad indicates whether the applicant, after being approved for a loan, paid off or defaulted on the loan.

This example uses the Hmeq data table to build a forest model that is used to score the data and can be used to score data about new loan applicants. **Table 10.1** describes the variables in Hmeq.

**Table 10.1** Variables in the Home Equity (Hmeq) Data Table

<table>
<thead>
<tr>
<th>Variable</th>
<th>Role</th>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
</table>
| Bad      | Response | Binary    | 1 = applicant defaulted on the loan or is seriously delinquent  
|          |        |           | 0 = applicant paid off the loan                      |
| CLAge    | Predictor | Interval | Age of oldest credit line in months                  |
| CLNo     | Predictor | Interval | Number of credit lines                                |
| DebtInc  | Predictor | Interval | Debt-to-income ratio                                  |
| Delinq   | Predictor | Interval | Number of delinquent credit lines                     |
| Derog    | Predictor | Interval | Number of major derogatory reports                    |
| Job      | Predictor | Nominal   | Occupational category                                 |
| Loan     | Predictor | Interval | Requested loan amount                                 |
| MortDue  | Predictor | Interval | Amount due on mortgage                                |
| nInq     | Predictor | Interval | Number of recent credit inquiries                     |
| Reason   | Predictor | Binary    | 'DebtCon' = debt consolidation                        |
|          |        |           | 'HomeImp' = home improvement                          |
| Value    | Predictor | Interval | Value of property                                     |
| YoJ      | Predictor | Interval | Years at present job                                  |

The following statements load the mycas.hmeq data into your CAS session. For this example, the statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```sas
/* Convert variable names to mixed case */
data mycas.hmeq;
  length Bad Loan MortDue Value 8 Reason Job $7
       YoJ Derog Delinq CLAge nInq CLNo DebtInc 8;
  set sampsio.hmeq;
run;
```

```sas
proc print data=mycas.hmeq(obs=10); run;
```

Output 10.1 shows the first 10 observations of mycas.hmeq.
PROC FOREST treats numeric variables as interval inputs unless you specify otherwise. Character variables are always treated as nominal inputs. The following statements run PROC FOREST and save the model in a table named mycas.savedModel:

```plaintext
proc forest data=mycas.hmeq outmodel=mycas.savedModel seed=12345;
  input Delinq Derog Job nInq Reason / level = nominal;
  input CLAge CLNo DebtInc Loan Mortdue Value YoJ / level = interval;
  target Bad / level = nominal;
  ods output FitStatistics=fitstats;
run;
```

No parameters are specified in the PROC FOREST statement; therefore, the procedure uses all default values. For example, the number of trees in the forest is 100, the number of bins for interval input variables is 20, and the number of variables that are examined at each node for a split is the square root of the number of input variables.

The INPUT and TARGET statements are required in order to run PROC FOREST. The INPUT statement indicates which variables to use to build the model, and the TARGET statement indicates which variable the procedure predicts.

Figure 10.2 displays the “Model Information” table. This table shows the values of the training parameters in the first six rows, in addition to some basic information about the trees in the forest.
**Figure 10.2** Model Information

The FOREST Procedure

<table>
<thead>
<tr>
<th>Model Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Trees</td>
<td>100</td>
</tr>
<tr>
<td>Number of Variables Per Split</td>
<td>4</td>
</tr>
<tr>
<td>Seed</td>
<td>12345</td>
</tr>
<tr>
<td>Bootstrap Percentage</td>
<td>60</td>
</tr>
<tr>
<td>Number of Bins</td>
<td>50</td>
</tr>
<tr>
<td>Number of Input Variables</td>
<td>12</td>
</tr>
<tr>
<td>Maximum Number of Tree Nodes</td>
<td>293</td>
</tr>
<tr>
<td>Minimum Number of Tree Nodes</td>
<td>161</td>
</tr>
<tr>
<td>Maximum Number of Branches</td>
<td>2</td>
</tr>
<tr>
<td>Minimum Number of Branches</td>
<td>2</td>
</tr>
<tr>
<td>Maximum Depth</td>
<td>20</td>
</tr>
<tr>
<td>Minimum Depth</td>
<td>20</td>
</tr>
<tr>
<td>Maximum Number of Leaves</td>
<td>147</td>
</tr>
<tr>
<td>Minimum Number of Leaves</td>
<td>81</td>
</tr>
<tr>
<td>Maximum Leaf Size</td>
<td>2240</td>
</tr>
<tr>
<td>Minimum Leaf Size</td>
<td>5</td>
</tr>
<tr>
<td>OOB Misclassification Rate</td>
<td>0.09580537</td>
</tr>
<tr>
<td>Average Number of Leaves</td>
<td>114.62</td>
</tr>
</tbody>
</table>

**Figure 10.3** displays the “Number of Observations” table, which shows how many observations were read and used. If you specify a PARTITION statement, the “Number of Observations” table also displays the number of observations that were read and used per partition.

| Training                  |
|---------------------------|-------|
| Number of Observations Read | 5960  |
| Number of Observations Used | 5960  |

**Figure 10.4** displays the estimates of variable importance. The rows in this figure are sorted by the importance measure. A conclusion from fitting the forest model to these data is that DebtInc is the most important predictor of loan default.
**Figure 10.4** Variable Importance

<table>
<thead>
<tr>
<th>Variable</th>
<th>Importance</th>
<th>Std Dev</th>
<th>Relative Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>DebtInc</td>
<td>379.42</td>
<td>115.66</td>
<td>1.0000</td>
</tr>
<tr>
<td>Value</td>
<td>69.7278</td>
<td>15.6541</td>
<td>0.1838</td>
</tr>
<tr>
<td>Delinq</td>
<td>69.0809</td>
<td>10.4914</td>
<td>0.1821</td>
</tr>
<tr>
<td>CLAge</td>
<td>47.3622</td>
<td>5.0875</td>
<td>0.1248</td>
</tr>
<tr>
<td>Derog</td>
<td>37.8159</td>
<td>5.8441</td>
<td>0.0997</td>
</tr>
<tr>
<td>nlnq</td>
<td>33.3571</td>
<td>2.9258</td>
<td>0.0879</td>
</tr>
<tr>
<td>Loan</td>
<td>30.0531</td>
<td>4.2978</td>
<td>0.0792</td>
</tr>
<tr>
<td>CLNo</td>
<td>29.6481</td>
<td>2.7506</td>
<td>0.0781</td>
</tr>
<tr>
<td>YoJ</td>
<td>24.6732</td>
<td>2.8585</td>
<td>0.0650</td>
</tr>
<tr>
<td>MortDue</td>
<td>24.3190</td>
<td>2.2212</td>
<td>0.0641</td>
</tr>
<tr>
<td>Job</td>
<td>23.9726</td>
<td>2.4132</td>
<td>0.0632</td>
</tr>
<tr>
<td>Reason</td>
<td>4.5996</td>
<td>1.8362</td>
<td>0.0121</td>
</tr>
</tbody>
</table>

Figure 10.5 shows the first 10 and last 10 observations of the fit statistics. When PROC FOREST runs, it computes fit statistics for a sequence of forests that have an increasing number of trees. As the number of trees increases, the fit statistics usually improve (decrease) at first and then level off and fluctuate within a small range. Forest models provide an alternative estimate of the average square error and misclassification rate. This alternative is called the *out-of-bag* (OOB) estimate. The OOB estimate is a convenient substitute for an estimate that is based on test data and is a less biased estimate of how the model will perform on future data. For more information, see the section “Bagging the Data” on page 187. The listing shows that the out-of-bag error estimate is worse (larger) than the estimate that evaluates all observations on all trees. This is common.
### Figure 10.5 Fit Statistics

<table>
<thead>
<tr>
<th>Number of Trees</th>
<th>OOB Average Square Error</th>
<th>OOB Misclassification Rate</th>
<th>Training Average Square Error</th>
<th>Training Misclassification Rate</th>
<th>OOB Log Loss</th>
<th>Training Log Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1043</td>
<td>0.1310</td>
<td>0.0800</td>
<td>0.1029</td>
<td>1.209</td>
<td>0.739</td>
</tr>
<tr>
<td>2</td>
<td>0.0957</td>
<td>0.1194</td>
<td>0.0660</td>
<td>0.0889</td>
<td>0.848</td>
<td>0.272</td>
</tr>
<tr>
<td>3</td>
<td>0.0897</td>
<td>0.1174</td>
<td>0.0617</td>
<td>0.0837</td>
<td>0.616</td>
<td>0.217</td>
</tr>
<tr>
<td>4</td>
<td>0.0889</td>
<td>0.1181</td>
<td>0.0606</td>
<td>0.0824</td>
<td>0.582</td>
<td>0.209</td>
</tr>
<tr>
<td>5</td>
<td>0.0854</td>
<td>0.1149</td>
<td>0.0594</td>
<td>0.0817</td>
<td>0.494</td>
<td>0.205</td>
</tr>
<tr>
<td>6</td>
<td>0.0831</td>
<td>0.1123</td>
<td>0.0575</td>
<td>0.0795</td>
<td>0.417</td>
<td>0.197</td>
</tr>
<tr>
<td>7</td>
<td>0.0818</td>
<td>0.1096</td>
<td>0.0573</td>
<td>0.0802</td>
<td>0.389</td>
<td>0.196</td>
</tr>
<tr>
<td>8</td>
<td>0.0802</td>
<td>0.1068</td>
<td>0.0569</td>
<td>0.0805</td>
<td>0.355</td>
<td>0.196</td>
</tr>
<tr>
<td>9</td>
<td>0.0800</td>
<td>0.1024</td>
<td>0.0566</td>
<td>0.0784</td>
<td>0.348</td>
<td>0.195</td>
</tr>
<tr>
<td>10</td>
<td>0.0789</td>
<td>0.1015</td>
<td>0.0561</td>
<td>0.0792</td>
<td>0.324</td>
<td>0.194</td>
</tr>
<tr>
<td>91</td>
<td>0.0736</td>
<td>0.0965</td>
<td>0.0555</td>
<td>0.0737</td>
<td>0.249</td>
<td>0.195</td>
</tr>
<tr>
<td>92</td>
<td>0.0736</td>
<td>0.0968</td>
<td>0.0554</td>
<td>0.0733</td>
<td>0.249</td>
<td>0.195</td>
</tr>
<tr>
<td>93</td>
<td>0.0735</td>
<td>0.0971</td>
<td>0.0554</td>
<td>0.0738</td>
<td>0.249</td>
<td>0.195</td>
</tr>
<tr>
<td>94</td>
<td>0.0735</td>
<td>0.0971</td>
<td>0.0554</td>
<td>0.0738</td>
<td>0.249</td>
<td>0.195</td>
</tr>
<tr>
<td>95</td>
<td>0.0735</td>
<td>0.0970</td>
<td>0.0554</td>
<td>0.0737</td>
<td>0.249</td>
<td>0.195</td>
</tr>
<tr>
<td>96</td>
<td>0.0735</td>
<td>0.0965</td>
<td>0.0554</td>
<td>0.0737</td>
<td>0.249</td>
<td>0.195</td>
</tr>
<tr>
<td>97</td>
<td>0.0736</td>
<td>0.0963</td>
<td>0.0554</td>
<td>0.0735</td>
<td>0.249</td>
<td>0.195</td>
</tr>
<tr>
<td>98</td>
<td>0.0735</td>
<td>0.0965</td>
<td>0.0554</td>
<td>0.0738</td>
<td>0.249</td>
<td>0.195</td>
</tr>
<tr>
<td>99</td>
<td>0.0735</td>
<td>0.0963</td>
<td>0.0554</td>
<td>0.0742</td>
<td>0.249</td>
<td>0.195</td>
</tr>
<tr>
<td>100</td>
<td>0.0735</td>
<td>0.0958</td>
<td>0.0554</td>
<td>0.0737</td>
<td>0.249</td>
<td>0.195</td>
</tr>
</tbody>
</table>

### Syntax: FOREST Procedure

The following statements are available in the FOREST procedure:

```
PROC FOREST < options >;
   AUTOTUNE < options >;
   CODE < options >;
   CROSSVALIDATION < options >;
   GROW criterion;
   ID variables;
   INPUT variables < / LEVEL=NOMINAL | INTERVAL >;
   OUTPUT OUT=CAS-libref.data-table < option >;
   PARTITION partition-option;
   SAVESTATE RSTORE=CAS-libref.data-table;
   TARGET variable < / LEVEL=NOMINAL | INTERVAL >;
   VIICODE < options >;
   WEIGHT variable;
```

The PROC FOREST, INPUT, and TARGET statements are required. The INPUT statement can appear multiple times.
PROC FOREST Statement

PROC FOREST <options> ;

The PROC FOREST statement invokes the procedure. Table 10.2 summarizes the options in the PROC FOREST statement.

Table 10.2  PROC FOREST Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic Options</strong></td>
<td></td>
</tr>
<tr>
<td>BINMETHOD=</td>
<td>Specifies how to bin interval inputs prior to training</td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the name of the input table</td>
</tr>
<tr>
<td>INBAGFRACTION=</td>
<td>Specifies the fraction of the training data to use for growing each tree</td>
</tr>
<tr>
<td>INMODEL=</td>
<td>Specifies a saved forest model to use to score a new table</td>
</tr>
<tr>
<td>ISOLATION</td>
<td>Generates an isolation forest for anomaly detection instead of a forest for target prediction</td>
</tr>
<tr>
<td>LOH=</td>
<td>Specifies the number of variables to preselect using the Loh method</td>
</tr>
<tr>
<td>NOPRINT</td>
<td>Suppresses ODS output</td>
</tr>
<tr>
<td>NTHREADS=</td>
<td>Specifies the number of threads to use in the computation</td>
</tr>
<tr>
<td>NTREES=</td>
<td>Specifies the number of trees to grow in the forest model</td>
</tr>
<tr>
<td>NUMBIN=</td>
<td>Specifies the number of bins for continuous variables</td>
</tr>
<tr>
<td>OUTMODEL=</td>
<td>Specifies the data table to score the forest model</td>
</tr>
<tr>
<td>PRINTTARGET</td>
<td>Create tables that contain information about columns in the output</td>
</tr>
<tr>
<td>RBAIMP</td>
<td>Creates a variable importance table by using random branch assignment</td>
</tr>
<tr>
<td>SEED=</td>
<td>Specifies the random number seed to use for model building</td>
</tr>
<tr>
<td>VARS_TO_TRY=</td>
<td>Specifies the number of variables to examine at each node split</td>
</tr>
<tr>
<td>VII=</td>
<td>Calculates the importance of the specified types of variable interactions</td>
</tr>
<tr>
<td>VOTE=</td>
<td>Specifies the method for calculating the predicted probabilities for a nominal target</td>
</tr>
</tbody>
</table>

**Splitting Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGNMISSING=</td>
<td>Specifies how to handle missing values in a predictor variable</td>
</tr>
<tr>
<td>MAXBRANCH=</td>
<td>Specifies the maximum number of splits per node</td>
</tr>
<tr>
<td>MAXDEPTH=</td>
<td>Specifies the maximum tree depth</td>
</tr>
<tr>
<td>MINLEAFSIZE=</td>
<td>Specifies the minimum number of observations per leaf</td>
</tr>
<tr>
<td>MINUSEINSEARCH=</td>
<td>Specifies the minimum number of observations to use with the USEINSEARCH policy for handling missing values</td>
</tr>
</tbody>
</table>

You can specify the following **options**:

**ASSIGNMISSING=NONE** | **MACSMALL** | **USEINSEARCH**

specifies how PROC FOREST creates a default splitting rule that is used to handle missing values and unknown levels. An unknown level is a level of a categorical predictor that does not exist in the training data but is encountered during scoring.

This option controls how missing values are used in model training, and controls the creation of the default splitting rule.
The primary splitting rule for a node is created during model training. During model scoring, observations are assigned to a node in a tree based upon the primary splitting rule if the rule’s variable is not missing. If the variable is missing for the observation, then the default splitting rule is used.

The default splitting rule enables all data to be scored, even if the primary rule cannot be used on a particular observation.

You can specify one of the following values to determine the default splitting rule:

- **NONE** excludes observations that have any missing variables from training the forest model. In the scoring phase, this default rule assigns observations that have missing values of an interval predictor variable to the branch with the smallest predictor values, and assigns observations that have unknown and missing nominal levels to the branch with the most training observations.

- **MACSMALL** treats a missing value as a separate, legitimate value in the search for a split for the primary splitting rule. Missing values in interval inputs are treated as less than any other number. In the scoring phase, this default rule assigns missing interval inputs to the leftmost branch of the split, and unknown nominal levels to the largest branch in the split.

- **USEINSEARCH** treats a missing value as a separate, legitimate value in the search for a split for the primary splitting rule. Missing values in interval inputs are treated as a special level that is used during the split process. In the scoring phase, this default rule assigns missing interval inputs to the branch determined during forest growing, and unknown nominal levels to the largest branch in the split.

By default, ASSIGNMISSING=USEINSEARCH.

**BINMETHOD=BUCKET | QUANTILE** specifies how to bin interval input variables prior to growing the forest model. The number of bins that are created is determined by the NUMBIN= option.

You can specify one of the following values:

- **BUCKET** bins interval input variables into fixed-width bins. The width of each bin for a particular variable is calculated by subtracting the smallest value among all observations from the largest value among all observations, and then dividing that result by the number of bins.

- **QUANTILE** bins interval input variables into bins according to their quantile. The width of the bins for a particular variable is not fixed, but the number of observations in each bin is approximately equal.

By default, BINMETHOD=QUANTILE.

**DATA=** specifies the input data table for PROC FOREST to use. The default is the most recently created data table. **CAS-libref.data-table** is a two-level name, where

- **CAS-libref** refers to a collection of information that is defined in the LIBNAME statement and includes the *caslib*, which includes a path to the data, and a session identifier, which
defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about CAS-libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 165.

data-table specifies the name of the input data table.

INBAGFRACTION=\textit{number}

specifies the fraction of the random bootstrap sample of the training data to be used for growing each tree in the forest, where \textit{number} is a value between 0 and 1. Observations can be duplicated in the sample. The number of distinct observations in the sample will be less than the number of observations available for training even when \textit{number} is 1.

By default, INBAGFRACTION=0.6. This value can be tuned with the AUTOTUNE statement.

INMODEL=\texttt{<CAS-libref.>data-table}

specifies the data table that you have previously saved as a forest model by using the \texttt{OUTMODEL=} option in a previous run of PROC FOREST. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the caslib and session identifier, and \texttt{data-table} specifies the name of the input data table. For more information about this two-level name, see the \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 165.

When you use the INMODEL= option, the OUTPUT statement is required and any other options in the PROC FOREST statement, except for NOPRINT and VOTE=, are ignored.

The data table used in the INMODEL= option must include the attributes that are associated with the table produced by the \texttt{OUTMODEL=} option, or the FOREST procedure will error.

ISOLATION < (SAMPLEN=\textit{number}) >

creates an isolation forest for anomaly detection instead of creating a forest for target prediction. For more information about isolation forests, see section “Isolation Forests” on page 194.

You can specify the following option:

SAMPLEN=\textit{number}

 specifies the \textit{number} of observations, sampled without replacement, to use in each tree of the isolation forest.

By default, SAMPLEN=100.

When the ISOLATION option is specified, the GROW and TARGET statements are ignored and the BINMETHOD, INBAGFRACTION, RBAIMP, VARS_TO_TRY, and VII options in the PROC FOREST statement are ignored. Fit statistics are not computed.

LOH=\textit{L}

specifies a number of variables (\textit{L}) that are preselected to consider for candidate splits for each node. The variables are selected using the Loh method.

If \textit{L} is less than the value of the VARS_TO_TRY= option (\textit{m}), then the variables are selected from among the \textit{m} variables. If \textit{L} is greater than or equal to \textit{m}, or if no \textit{L} is specified, then the Loh method is not used.
**MAXBRANCH=b**
specifies the maximum number of children per node in the tree. PROC FOREST tries to create this number of children unless it is impossible (for example, if a split variable does not have enough levels).

By default, MAXBRANCH=2.

**MAXDEPTH=number**
specifies the maximum depth of the tree to be grown. The number of levels in a tree is equal to the depth plus one.

By default, MAXDEPTH=20.

**MINLEAFSIZE=number**
specifies the minimum number of observations that each child of a split must contain in the training data table in order for the split to be considered. The count of observations includes those that are excluded from training as a consequence of specifying a value less than 1 in the INBAGFRACTION= option. By default, MINLEAFSIZE=5. This value can be tuned with the AUTOTUNE statement.

**MINUSEINSEARCH=number**
specifies a threshold for using missing values in the split search when ASSIGNMISSING=USEINSEARCH. If the number of observations in which the splitting variable has missing values is greater than or equal to number, then PROC FOREST uses the USEINSEARCH policy to handle missing values for that variable.

By default, MINUSERINSEARCH=1.

**NOPRINT**
suppresses ODS output.

**NTHREADS=number-of-threads**
specifies the number of threads to use in the computation. The default value is the number of CPUs available on the machine.

**NTREES=number**
specifies the number of trees to grow in the forest model.

By default, NTREES=100. This value can be tuned with the AUTOTUNE statement.

**NUMBIN=number**
specifies the number of bins in which to bin the interval input variables. PROC FOREST bins continuous predictors to a fixed bin size. This option controls the number of bins and thereby also the size of the bins.

By default, NUMBIN=50. This value can be tuned with the AUTOTUNE statement.

**OUTMODEL= <<CAS-libref.>data-table**
specifies the data table to which to save the forest model. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 165.

The data table that results from this option contains information about each node and each tree in the forest model, including the splitting variables, the child nodes, the number of observations at each node, and the predicted response at each node. This table also has attributes attached that contain information about the input variables, the target variable, and the forest model.
PRINTTARGET outputs tables that indicate generated columns in the OUT= table from the OUTPUT statement. For a continuous response, PROC FOREST generates an output table named PredName, which indicates the name of the predicted value column. For a categorical response, PROC FOREST generates an output table named PredIntoName, which indicates the name of the predicted value column, and also an output table named PredProbName, which indicates the names of the predicted probability columns.

RBAIMP creates a variable importance table by using random branch assignment (RBA). This table is created in addition to the normal variable importance table that is calculated using the residual sum of squares (RSS) error. For more information about RBA and RSS variable importance, see the section “Measuring Variable Importance” on page 191.

SEED=number specifies the initial seed for random number generation for model building. The value of number must be an integer. If you do not specify a seed or you specify a value less than or equal to 0, the seed is generated from reading the time of day from the computer’s clock.

VARS_TO_TRY=m
M=m specifies the number of input variables to consider splitting on in a node, where m ranges from 1 to the number of input variables.

By default, m is the square root of the number of input variables. This value can be tuned with the AUTOTUNE statement.

VII=2 | 3
INTERACTIONIMP=2 | 3 calculates the variable interaction importance, which is described in the section “Variable Interaction Importance” on page 193. You can specify the following values:

2 calculates the importance of all two-way variable interactions.
3 calculates the importance of all three-way and all two-way variable interactions.

If you do not specify this option, then the variable interaction importance is not calculated.

VOTE=MAJORITY | PROBABILITY specifies how to calculate the predicted probability of the target levels for a nominal target. The predicted level is the level that has the highest predicted probability. This option affects the scoring and fit statistics of the forest model. You can specify the following values:

MAJORITY specifies that the predicted probability of each target level is equal to the number of trees in the forest that predicted that level as the target, divided by the total number of trees in the forest.

PROBABILITY specifies that the predicted probability of each target level is equal to the probability of that level averaged over each tree in the forest.

When using the INMODEL= option, you can specify how to calculate the predicted probability of the target levels, even if the model was trained with a different value for the VOTE= option.

By default, VOTE=PROBABILITY.
AUTOTUNE Statement

AUTOTUNE < options > ;

The AUTOTUNE statement searches for the best combination of values of the INBAGFRACTION=, MAXDEPTH=, MINLEAFSIZE=, NUMBIN=, NTREES=, and VARS_TO_TRY= options in the PROC FOREST statement. You cannot specify both the AUTOTUNE statement and the CROSSVALIDATION statement in the same procedure run.

Table 10.3 summarizes the options that you can specify in the AUTOTUNE statement. For more information about all options except the TUNINGPARAMETERS= option, see the option’s description in the section “AUTOTUNE Statement” on page 12 in Chapter 3, “Shared Concepts.” The TUNINGPARAMETERS= option is described following Table 10.3.

Table 10.3 AUTOTUNE Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDLOOKUP</td>
<td>Specifies that the table specified in the HISTORYTABLE= option contain the rows from the table specified in the LOOKUPTABLE= option</td>
</tr>
<tr>
<td>EVALHISTORY=</td>
<td>Specifies how to report the evaluation history of the tuner</td>
</tr>
<tr>
<td>FRACTION=</td>
<td>Specifies the fraction of observations to use for validation</td>
</tr>
<tr>
<td>HISTORYTABLE=</td>
<td>Specifies the CAS table that contains the evaluation history</td>
</tr>
<tr>
<td>KFOLD=</td>
<td>Specifies the number of folds for ( k )-fold cross validation</td>
</tr>
<tr>
<td>LIVEUPDATE</td>
<td>Specifies that the table specified in the HISTORYTABLE= option be updated at every evaluation</td>
</tr>
<tr>
<td>LOCALSEARCH</td>
<td>Enables local search optimization</td>
</tr>
<tr>
<td>LOOKUPTABLE=</td>
<td>Specifies the CAS table to use for evaluation lookup</td>
</tr>
<tr>
<td>MAXBAYES=</td>
<td>Specifies the maximum number of points in the kriging model</td>
</tr>
<tr>
<td>MAXEVALS=</td>
<td>Specifies the maximum number of evaluations</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>MAXTIME=</td>
<td>Specifies the maximum time for all iterations</td>
</tr>
<tr>
<td>MAXTRAINTIME=</td>
<td>Specifies the maximum time for a model training</td>
</tr>
<tr>
<td>NCONVITER=</td>
<td>Specifies the number of convergence iterations</td>
</tr>
<tr>
<td>NOGRIDSHUFFLE</td>
<td>Requests that the grid points not be shuffled</td>
</tr>
<tr>
<td>NPARALLEL=</td>
<td>Specifies the number of parallel sessions</td>
</tr>
<tr>
<td>NSUBSESSIONWORKERS=</td>
<td>Specifies the number of workers in parallel sessions</td>
</tr>
<tr>
<td>OBJECTIVE=</td>
<td>Specifies the objective function</td>
</tr>
<tr>
<td>POPSIZE=</td>
<td>Specifies the population size when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>SAMPLESIZE=</td>
<td>Specifies the sample size when SEARCHMETHOD=LHS or SEARCHMETHOD=RANDOM</td>
</tr>
<tr>
<td>SEARCHMETHOD=</td>
<td>Specifies the search method that the optimizer uses</td>
</tr>
<tr>
<td>SECONDOBJECTIVE=</td>
<td>Specifies the second objective to use for tuning</td>
</tr>
<tr>
<td>SELECTINITPOINT</td>
<td>Specifies that the tuner select the best evaluation from the lookup table</td>
</tr>
<tr>
<td>TARGETEVENT=</td>
<td>Specifies the target event for ROC-based calculations</td>
</tr>
</tbody>
</table>
Table 10.3  continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAINFRACTION=</td>
<td>Specifies the fraction of observations to use for training</td>
</tr>
<tr>
<td>TUNINGPARAMETERS=</td>
<td>Specifies the custom tuning parameters</td>
</tr>
<tr>
<td>USEPARAMETERS=</td>
<td>Specifies how to handle the TUNINGPARAMETERS= option</td>
</tr>
</tbody>
</table>

TUNINGPARAMETERS=(suboption | . . . | < suboption>)
TUNEPARMS=(suboption | . . . | < suboption>)
specifies which parameters to tune and which ranges to tune over. If USEPARAMETERS=STANDARD, this option is ignored.

You can specify one or more of the following suboptions:

INBAGFRACTION (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies information about the fraction of the training data to use for each bagged tree while tuning the forest model. For more information, see the INBAGFRACTION= option in the PROC FOREST statement.

You can specify the following additional suboptions:

LB=number
specifies the minimum fraction of training data to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=0.1.

UB=number
specifies the maximum fraction of training data to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=0.9.

VALUES=value-list
specifies a list of fractions of training data to consider during tuning, where value-list is a space-separated list of numbers greater than 0 and less than or equal to 1. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

INIT=number
specifies the initial fraction of training data for the tuner to use.

By default, INIT=0.6.

EXCLUDE
excludes the fraction of the training data to use for each bagged tree from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.
**Chapter 10: The FOREST Procedure**

**MAXDEPTH (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**

specifies information about the maximum depth to which to grow the trees in the forest. For more information, see the `MAXDEPTH=` option in the `PROC FOREST` statement.

You can specify the following additional suboptions:

- **LB=number**
  - specifies a lower bound on the maximum depth to consider during tuning. If you specify this suboption, you cannot specify the `VALUES=` suboption.
  - By default, LB=1.

- **UB=number**
  - specifies an upper bound on the maximum depth to consider during tuning. If you specify this suboption, you cannot specify the `VALUES=` suboption.
  - By default, UB=29.

- **VALUES=value-list**
  - specifies a list of values to consider for the maximum depth of the trees in the forest, where `value-list` is a space-separated list of numbers. If you specify this suboption, you cannot specify either the `LB=` or `UB=` suboption.

- **INIT=number**
  - specifies the initial maximum depth of trees in the forest.
  - By default, INIT=20.

- **EXCLUDE**
  - excludes maximum depth from the tuning process. If you specify this suboption, any specified `LB=`, `UB=`, `VALUES=`, and `INIT=` suboptions are ignored.

**MINLEAFSIZE (LB=number UB= number VALUES=value-list INIT=number EXCLUDE)**

specifies information about tuning the leaf size option for training the decision tree. This option is not tuned by default. If it is specified without any suboptions, then tuning uses the following values: 1, 5, 10, 20, 40, 80, 160, 320. For more information, see the `MINLEAFSIZE=` option in the `PROC FOREST` statement.

You can specify the following additional suboptions:

- **LB=number**
  - specifies the minimum leaf size value to consider during tuning. If you specify this suboption, you cannot specify the `VALUES=` suboption.
  - By default, LB=1.

- **UB=number**
  - specifies the maximum leaf size value to consider during tuning. If you specify this suboption, you cannot specify the `VALUES=` suboption.
  - By default, UB=320.
VALUES=value-list
specifies a list of leaf size values to consider during tuning, where value-list is a space-separated list of positive integers. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

INIT=number
specifies the initial leaf size for the tuner to use.

By default, INIT=5.

EXCLUDE
excludes the leaf size from the tuning process.

NUMBIN (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies information about the number of bins in which to bin the interval inputs while tuning the decision tree. For more information, see the NUMBIN= option in the PROC FOREST statement.

You can specify the following additional suboptions:

LB=number
specifies the minimum number of bins to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=20.

UB=number
specifies the maximum number of bins to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=100.

VALUES=value-list
specifies a list of numbers of bins to consider during tuning, where value-list is a space-separated list of positive integers. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

INIT=number
specifies the initial number of bins for the tuner to use.

By default, INIT=50.

EXCLUDE
excludes the number of bins from the tuning process.

NTREES (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies information about the number of trees in the forest to use for tuning the forest model. For more information, see the NTREES= option in the PROC FOREST statement.

You can specify the following additional suboptions:
**LB=number**
specifies the minimum number of trees to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=20.

**UB=number**
specifies the maximum number of trees to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=150.

**VALUES=value-list**
specifies a list of numbers of trees to consider during tuning, where value-list is a space-separated list of positive integers. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=number**
specifies the initial number of trees for the tuner to use.

By default, INIT=100.

**EXCLUDE**
excludes the number of trees from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**VARS_TO_TRY (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**
specifies information about the number of variables to consider at each split during tree growth while tuning the forest model. For more information, see the VARS_TO_TRY= option in the PROC FOREST statement.

You can specify the following additional suboptions:

**LB=number**
specifies the minimum number of variables to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=1.

**UB=number**
specifies the maximum number of variables to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=min(n,100), where n is the total number of input variables.

**VALUES=value-list**
specifies a list of numbers of variables to consider during tuning, where value-list is a space-separated list of positive integers. If you specify this suboption, you cannot specify either the LB= or UB= suboption.
**CODE Statement**

```sas
CODE <options> ;
```

The CODE statement writes SAS DATA step code for computing predicted values of the fitted model to a file, to a catalog entry, or to a CAS table. To score new data, you can then include the file or the catalog entry in a DATA step, or you can specify the CAS table in the `runCodeTable` action in the `dataStep` action set (for more information, see *SAS Viya: System Programming Guide*).

Table 10.4 summarizes the options available in the CODE statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENT</td>
<td>Adds comments to the generated code</td>
</tr>
<tr>
<td>FILE=</td>
<td>Names the file in which to save the generated code</td>
</tr>
<tr>
<td>FORMATWIDTH=</td>
<td>Specifies the numeric format width for the regression coefficients</td>
</tr>
<tr>
<td>INDENTSIZE=</td>
<td>Specifies the number of spaces to indent the generated code</td>
</tr>
<tr>
<td>LABELID=</td>
<td>Specifies a number used to construct names and labels</td>
</tr>
<tr>
<td>LINESIZE=</td>
<td>Specifies the line size for the generated code</td>
</tr>
<tr>
<td>NOTRIM</td>
<td>Compares formatted values, including blank padding</td>
</tr>
<tr>
<td>OUT=</td>
<td>Names an output CAS table in which to save the generated code</td>
</tr>
</tbody>
</table>

For more information about the syntax of the CODE statement, see the section “CODE Statement” on page 18 in Chapter 3, “Shared Concepts.”

**CROSSVALIDATION Statement**

```sas
CROSSVALIDATION <options> ;
```

The CROSSVALIDATION statement performs $k$-fold cross validation to find the average estimated validation error. You cannot specify this statement if you specify either the AUTOTUNE statement or the PARTITION statement.

You can specify the following options:
**Chapter 10: The FOREST Procedure**

**Kfold**=<number>

specifies the number of partition folds in the cross validation process, where number must be between 2 and 20, inclusive.

By default, KFOLD=5.

**NOPARALLEL**

requests that k-fold cross validation not be run in parallel. By default, the process runs in parallel.

**Nsubsessionworkers**=number

specifies the number of worker nodes to use in parallel subsessions.

By default, the value of the NSUBSESSIONWORKERS= option is determined automatically.

---

**GROW Statement**

```plaintext
GROW criterion ;
```

The GROW statement specifies the criterion by which to split a parent node into child nodes. As it grows the tree, PROC FOREST calculates the specified criterion for each predictor variable and then splits on the predictor variable that optimizes the specified criterion.

For categorical responses, the available criteria are CHAID, CHISQUARE, ENTROPY, GINI, and IGR; the default is IGR. For continuous responses, the available criteria are CHAID, FTEST, and RSS; the default is RSS.

For either categorical or continuous responses, you can specify the following criterion:

**CHAID**

for categorical predictor variables, CHAID uses the value (as specified in the ALPHA= option) of a chi-square statistic (for a classification tree) or an F statistic (for a regression tree) to merge similar levels of the predictor variable until the number of children in the proposed split reaches the number that you specify in the MAXBRANCH= option in the PROC FOREST statement. The p-values for the final split determine the variable on which to split.

For continuous predictor variables, CHAID chooses the best single split until the number of children in the proposed split reaches the value that you specify in the MAXBRANCH= option in the PROC FOREST statement.

For categorical responses only, you can specify the following criteria:

**CHISQUARE**

uses a chi-square statistic to split each variable and then uses the p-values that correspond to the resulting splits to determine the splitting variable.

**ENTROPY**

**GAIN**

uses the gain in information (decrease in entropy) to split each variable and then to determine the split.
GINI
uses the decrease in the Gini index to split each variable and then to determine the split.

IGR
uses the entropy metric to split each variable and then uses the information gain ratio to determine the split.

The default criterion for categorical responses is IGR.

For continuous responses only, you can specify the following criteria:

FTEST
uses an $F$ statistic to split each variable and then uses the resulting $p$-value to determine the split variable.

RSS VARIANCE
uses the change in response variance to split each variable and then to determine the split.

The default criterion for continuous responses is RSS.

---

**ID Statement**

```
ID variables ;
```

The ID statement lists one or more variables that are to be copied from the input data table to the output data tables that are specified in the OUT= option in the OUTPUT statement and the RSTORE= option in the SAVESTATE statement.

---

**INPUT Statement**

```
INPUT variables </LEVEL=INTERVAL | NOMINAL> ;
```

The INPUT statement names input variables that share a common option. You can specify the INPUT statement multiple times.

You can specify the following option:

**LEVEL=INTERVAL | NOMINAL**

specifies the level of measurement of the variables. You can specify the following values:

- INTERVAL
  - specifies that the level of measurement of the variables is interval.

- NOMINAL
  - specifies that the level of measurement of the variables is nominal.

By default, LEVEL=INTERVAL for numeric variables and LEVEL=NOMINAL for categorical variables.
OUTPUT Statement

```
OUTPUT OUT=CAS-libref.data-table < option > ;
```

The OUTPUT statement creates an output data table that contains the results of PROC FOREST.

You must specify the following option:

```
OUT=CAS-libref.data-table
```

names the output data table for PROC FOREST to use. You must specify this option before any other options. `CAS-libref.data-table` is a two-level name, where

- `CAS-libref` refers to a collection of information that is defined in the LIBNAME statement and includes the `caslib`, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 165.

- `data-table` specifies the name of the output data table.

You can also specify the following options:

```
COPYVAR=variable
COPYVARS=(variables)
ROLE< =name >
```

- `COPYVAR=variable` lists one or more variables from the input data table to be transferred to the output data table.

- `COPYVARS=(variables)` lists one or more variables from the input data table to be transferred to the output data table.

- `ROLE< =name >` generates a numeric variable that indicates the role played by each observation in fitting the model. By default, the variable is named `_ROLE_`. You can add an optional `=name` to change the name of this generated variable. For each observation, the interpretation of this variable is shown in Table 10.5.

<table>
<thead>
<tr>
<th>Value</th>
<th>Observation Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>1</td>
<td>Training</td>
</tr>
<tr>
<td>2</td>
<td>Validation</td>
</tr>
<tr>
<td>3</td>
<td>Testing</td>
</tr>
</tbody>
</table>

If you do not partition the input data by using a PARTITION statement, then the value of the role variable is 1 for all observations.
PARTITION Statement

PARTITION partition-option;

The PARTITION statement specifies how observations in the input data set are logically partitioned into disjoint subsets for model training, validation, and testing. For more information, see the section “Using Validation and Test Data” on page 21 in Chapter 3, “Shared Concepts.” Either you can designate a variable in the input data table and a set of formatted values of that variable to determine the role of each observation, or you can specify proportions to use for randomly assigning observations to each role.

You must specify exactly one of the following partition-options:

FRACTION(< TEST=fraction > < VALIDATE=fraction > < SEED=number >)
randomly assigns specified proportions of the observations in the input data table to the roles. You specify the proportions for testing and validation by using the TEST= and VALIDATE= suboptions. If you specify both the TEST= and VALIDATE= suboptions, then the sum of the specified fractions must be less than 1 and the remaining fraction of the observations are assigned to the training role. The SEED= option specifies an integer that is used to start the pseudorandom number generator for random partitioning of data for training, testing, and validation. If you do not specify SEED=number or if number is less than or equal to 0, the seed is generated by reading the time of day from the computer’s clock.

ROLE=variable (< TEST=value > < TRAIN=value > < VALIDATE=value >)
ROLEVAR=variable (< TEST=value > < TRAIN=value > < VALIDATE=value >)

names the variable in the input data table whose values are used to assign roles to each observation. This variable cannot also appear as an analysis variable in other statements or options. The TEST=, TRAIN=, and VALIDATE= suboptions specify the formatted values of this variable that are used to assign observation roles. If you do not specify the TRAIN= suboption, then all observations whose role is not determined by the TEST= or VALIDATE= suboption are assigned to the training role.

SAVESTATE Statement

SAVESTATE RSTORE=CAS-libref.data-table;

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 4, “The ASTORE Procedure.”

You must specify the following option:

RSTORE=CAS-libref.data-table

specifies a data table in which to save the analytic store for the model. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 165.
TARGET Statement

TARGET variable < / LEVEL=NOMINAL | INTERVAL > ;

The TARGET statement names the variable whose values PROC FOREST tries to predict.

You can specify the following option:

LEVEL=NOMINAL | INTERVAL

specifies the level of measurement. You can specify the following values:

NOMINAL specifies that the level of measurement of the variables is nominal.

INTERVAL specifies that the level of measurement of the variables is interval.

By default, LEVEL=INTERVAL for numeric variables and LEVEL=NOMINAL for categorical variables.

VIICODE Statement

VIICODE < options > ;

The VIICODE statement writes SAS DATA step code to a file or to a catalog entry. The SAS DATA step code creates new variables on the basis of the detected variable interactions.

You can specify the following options in addition to all options in the CODE statement. For more information about the syntax of the CODE statement, see the section “CODE Statement” on page 18 in Chapter 3, “Shared Concepts.”

ADD requests that the newly created variables be of the form $V + W$.

LIMIT=number specifies the maximum number of new variables to create. By default, LIMIT=200.

MISS requests that the generated code handle missing values.

MULTIPLY requests that the newly created variables be of the form $V \times W$.

SUBTRACT requests that the newly created variables be of the form $V - W$.

THRESHOLD=number requests that interactions with an importance less than number times the maximum interaction importance be ignored, where number must be between 0 and 1. By default, THRESHOLD=0.0001.

If the VII= option is not specified in the PROC FOREST statement, then the VIICODE statement is ignored. For more information about variable interaction importance, see the section “Variable Interaction Importance” on page 193.
The variable in the WEIGHT statement is used as a weight to perform a weighted analysis of the data. Observations that have nonpositive or missing weights are not included in the analysis. If a WEIGHT statement is not included, all observations that are used in the analysis are assigned a weight of 1.

Details: FOREST Procedure

Bagging the Data

A decision tree in a forest trains on new training data that are derived from the original training data presented to the FOREST procedure. Using different data to train different trees reduces the correlation of the predictions of the trees, which in turn should improve the predictions of the forest.

The FOREST procedure samples the original data with replacement to create the training data for an individual tree. The convention of sampling with replacement originated with Leo Breiman’s bagging algorithm (Breiman 1996, 2001). The word bagging stems from “bootstrap aggregating,” where “bootstrap” refers to a process that uses sampling with replacement. Breiman refers to the observations that are excluded from the sample as out-of-bag (OOB) observations. Therefore, observations in the training sample are called the bagged observations, and the training data for a specific decision tree are called the bagged data.

The INBAGFRACTION= option in the PROC FOREST statement specifies the number of observations to sample with replacement into the bagged data.

Estimating the goodness of fit of the model by using the training data is usually too optimistic; the fit of the model to new data is usually worse than the fit to the training data. Estimating the goodness of fit by using the out-of-bag data is usually too pessimistic at first. With enough trees, the out-of-bag estimates are an unbiased estimate of the generalization fit.

Training a Decision Tree

The FOREST procedure trains a decision tree by splitting the bagged data, then splitting each of the resulting segments, and so on recursively until some constraint is met.

Splitting involves the following subtasks:

1. selecting candidate inputs
2. computing the association of each input with the target
3. searching for the best split that uses the most highly associated inputs

PROC FOREST randomly selects \( m \) candidate input variables independently in every node, where \( m \) is the value of the VARS_TO_TRY= option in the PROC FOREST statement. If you specify \( L \) as the value of
the **LOH**= option and \( L < m \), then PROC FOREST chooses the best \( L \) input variables from the \( m \) variables according to the criterion described in section “Loh Method” on page 188. A split search is performed on all \( L \) or \( m \) variables, and the best rule is kept to split the node.

The reason for searching fewer input variables for a splitting rule instead of searching all inputs and choosing the best split is to improve prediction on new data. An input that offers more splitting possibilities provides the search routine more chances to find a spurious split. Loh and Shih (1997) demonstrate the bias towards spurious splits that result. They also demonstrate that preselecting the input variable and then searching only on that one input reduces the bias. You can choose to preselect a number of input variables by using the **LOH**= option.

The split search seeks to maximize the reduction in the gain for a nominal target and the reduction in variance of an interval target.

---

**Loh Method**

Specify the **LOH**= option to use ideas developed by Loh in a series of papers (Loh and Shih 1997; Loh 2002, 2009).

This method selects the number of variables that have the smallest \( p \)-value of a chi-square test of association in a contingency table. These variables are selected from the **VARS TO TRY**= \( m \) randomly selected variables that are chosen for a single decision tree in the forest.

Let \( Y \) and \( X \) denote the target variable and input variable, respectively. Let \( Y_i \) and \( X_i \) denote their values in observation \( i \). If \( Y \) is categorical, let \( J \) denote the number of values. Similarly, if \( X \) is categorical, let \( K \) denote the number of values.

If both \( Y \) and \( X \) are categorical, then form the \( J \times K \) contingency table of the frequencies of the observations and compute the \( p \)-value. If \( Y \) has an interval measurement level, then note whether \( Y_i \) is greater than or less than the average of \( Y \), \( \bar{Y} \), in the node, and then form the \( 2 \times K \) table of frequencies and compute the \( p \)-value.

If \( X \) has an interval measurement level, then let

\[
K = \begin{cases} 
3 & \text{if } N < 20J \\
4 & \text{otherwise}
\end{cases}
\]

where \( N \) is the number of observations in the calculations and \( J = 2 \) if \( Y \) has an interval measurement level.

If \( K = 3 \), assign \( X_i \) to a table column that is defined by the following boundary points:

\[
\begin{align*}
\xi_1 &= \bar{X} - \sqrt{3}\hat{\sigma}/3 \\
\xi_2 &= \bar{X} + \sqrt{3}\hat{\sigma}/3
\end{align*}
\]

Otherwise, use the boundary points

\[
\begin{align*}
\xi_1 &= \bar{X} - \sqrt{3}\hat{\sigma}/2 \\
\xi_2 &= \bar{X} \\
\xi_3 &= \bar{X} + \sqrt{3}\hat{\sigma}/2
\end{align*}
\]

where \( \bar{X} \) denotes the average value of \( X \) and

\[
\hat{\sigma}^2 = \frac{\sum_i (X_i - \bar{X})^2}{N}
\]
Predicting an Observation

To predict an observation, the FOREST procedure first assigns the observation to a single leaf in each decision tree in the forest, then uses that leaf to make a prediction based on the tree that contains the leaf, and finally simply averages the predictions over the trees. For an interval target, the prediction in a leaf equals the average of the target values among the bagged training observations in that leaf. For a nominal target, the posterior probability of a target category equals the proportion of that category among the bagged training observations in that leaf. The predicted nominal target category is the category that has the largest posterior probability. In case of a tie, the first category that occurs in the training data is the prediction.

Measuring Prediction Error

The FOREST procedure computes the average square error measure of prediction error. For a nominal target, PROC FOREST also computes the misclassification rate and the log-loss.

The average square error for an interval target, the average square error for a nominal target, the misclassification rate, and the log-loss are defined, respectively, as

$$ASE_{\text{int}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

$$ASE_{\text{cat}} = \frac{1}{JN} \sum_{i=1}^{N} \sum_{j=1}^{J} (\delta_{ij} - \hat{p}_{ij})^2$$

$$Misc = \frac{1}{N} \sum_{i=1}^{N} 1(y_i \neq \hat{y}_i)$$

$$LogLoss = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{J} \delta_{ij} \log(\hat{p}_{ij})$$

where $\hat{y}_i$ is the target prediction of observation $i$, $\delta_{ij}$ equals 1 if the nominal target value $j$ occurs in observation $i$ or 0 if it does not, $\hat{p}_{ij}$ is the predicted probability of nominal target value $j$ for observation $i$, $N$ is the number of observations, $J$ is the number of nominal target values (classes), and $\hat{p}_{ij}$ is $\hat{p}_{ij}$ truncated away from 0 and 1:

$$\hat{p}_{ij} = \max(\min(\hat{p}_{ij}, 1 - 10^{-10}), 10^{-10})$$

The definitions are valid whether $\hat{y}_i$ is the usual model prediction or the out-of-bag prediction. The $ASE_{\text{int}}$ that is based on the usual model predictions of the original training data is usually optimistic and smaller than what its value will be on future data.
Handling Missing Values

Strategies

Tree-based models use observations that have missing input values. The FOREST procedure offers the following strategies for handling missing values:

- The simple strategy is to regard a missing value as a special nonmissing value. For a nominal input, a missing value simply constitutes a new categorical value. For an input whose values are ordered, each missing value constitutes a special value that is assigned a place in the ordering that yields the best split. That place is usually different in different nodes of the tree.

This strategy is beneficial when missing values are predictive of certain target values. For example, people who have large incomes might be more reluctant to disclose their income than people who have ordinary incomes. If income were predictive of a target, then a missing income value would be predictive of the target and the missing values would be regarded as a special large-income value. The strategy seems harmless when the distribution of missing values is uncorrelated with the target because no choice of branch for the missing values would help predict the target.

A linear regression could use the same strategy by adding binary indicator variables to designate whether a value is missing. Alternatively, and much more commonly, a linear regression could simply remove observations in which any input is missing. Let $p$ denote the probability that a variable value is missing, and let $v$ denote the number of input variables. The probability that an observation has one or more missing values is $1 - (1 - p)^v$ (assuming missingness is independent and identically distributed among the inputs). If $p = 0.1$ and $v = 10$, then 65% of the observations would have missing values and would be removed from linear regression.

- The alternative strategy for decision trees is to exclude from the search algorithm any observations that have a missing value in the single input variable that defines the splitting rule. If $p = 0.1$ and $v = 10$, then only 10% instead of 65% of the observations are excluded. Although this compares favorably with common linear regression, using observations that have missing values might still be better.

Specifics

If the value of a target variable is missing, the observation is excluded from training and from evaluating the model. If the value of an input variable is missing, PROC FOREST uses the missing value as a legitimate value when ASSIGNMISSING=USEINSEARCH (the default value) and the number of observations in which the splitting variable has missing values is at least as large as the value of the MINUSEINSEARCH= option. When ASSIGNMISSING=USEINSEARCH and the number of observations in which the splitting value has missing values is less than the value of the MINUSEINSEARCH= option, the splitting rule assigns observations that have missing values to the largest branch.

If you specify ASSIGNMISSING=NONE, then PROC FOREST ignores training observations in which the input variables have missing values. When observations that have missing values are scored, if ASSIGNMISSING=NONE was used during model training, then observations that have missing values are scored using ASSIGNMISSING=MACSMALL as the default rule.
Handling Values That Are Absent from Training Data

A splitting rule that uses a categorical variable might not recognize all possible values of the variable because some categories might not exist in the training data. Splitting rules assign unseen categorical values to the branch that has the most in-bag training observations.

Measuring Variable Importance

The importance of a variable is the contribution it makes to the success of the model. For a predictive model, success means good prediction. Often the prediction relies mainly on a few variables. A good measure of importance reveals those variables. The better the prediction, the more closely the model represents reality and the more plausible it is that the important variables represent the true cause of prediction. Some people prefer a simple model so that they can understand it. However, a simple model usually relinquishes details of reality. Sometimes it is better to first find a good model and then ask which variables are important than to first ask which model is good for variable importance and then train that model.

Van der Laan (2006) asks whether a predictive model is appropriate at all. He believes that if variable importance is your goal, then you should predict importance directly instead of fitting a model. If your goal is to select suspicious genes for further study in a laboratory or to find variables in an industrial process that might influence the quality of the product, then his argument is persuasive. However, the purpose of many predictive models is to make predictions. In these cases, gaining insight into causes can be useful.

Variable importance is also useful for selecting variables for a subsequent model. The comparative importance between the selected variables does not matter. Researchers often seek speed and simplicity from the first model and seek accuracy from the subsequent model. Despite this tendency, a forest is often more useful than a simpler regression as a first model when you want interactions because variables contribute to the forest model through interactions.

Several authors have demonstrated that using a forest to first select variables and then using only those variables in a subsequent forest produces a final forest that predicts the target better than only training a forest without the variable selection.

The FOREST procedure implements two methods for computing variable importance, which are described in the following subsections. By default, the variable importance is calculated by using the change in the residual sum of square errors. You can request that the FOREST procedure to also calculate the variable importance by random branch assignment (RBA) by specifying the RBAIMP option.

Residual Sum of Squares Importance Method

The residual sum of squares (RSS) for regression trees is defined as

\[
RSS = \sum_{\lambda} \sum_{i \in \lambda} \left( y_i - \hat{y}_{i \lambda}^T \right)^2
\]

where

- \( i \) is an observation on leaf \( \lambda \)
- \( y_i \) is the predicted value of the response variable of observation \( i \)
192  •  Chapter 10: The FOREST Procedure

- $\hat{y}_{\lambda}^T$ is the actual value of the response variable on leaf $\lambda$

The residual sum of squares (RSS) for classification trees is defined as

$$RSS = \sum_{\lambda} \sum_{\Phi} \N_{\Phi} \left[ \sum_{\tau \neq \Phi} \left( P_{\tau}^{\lambda} \right)^2 + \left( 1 - P_{\Phi}^{\lambda} \right)^2 \right]$$

where

- $\Phi$ is the actual response level
- $\N_{\Phi}$ is the number of observations on leaf $\lambda$ that have response level $\Phi$
- $P_{\tau}^{\lambda}$ is the posterior probability for the response level $\tau$ on leaf $\lambda$
- $P_{\Phi}^{\lambda}$ is the posterior probability for the actual response level $\Phi$ on leaf $\lambda$

For a single tree in the forest, the RSS-based metric measures variable importance based on the change in RSS when a split is found at a node. The change for variable $v$ is

$$\Delta_{d}^{v} = RSS_{d} - \sum_{i} RSS_{i}^{d}$$

where

- $d$ denotes the node
- $i$ denotes the index of a child that this node includes
- $RSS_{d}$ is the RSS if the node is treated as a leaf
- $RSS_{i}^{d}$ is the RSS of the node after it has been split

If the change in RSS is negative (which is possible when you use the validation set), then the change is set to 0.

The RSS-based importance for a single variable, $v$, in a single tree is then defined as

$$\sum_{d=1}^{D} \Delta_{d}^{v}$$

where $D$ is the total number of nodes in which $v$ was used as the splitting variable.

The RSS variable importance for the forest is the average of the RSS variable importance across all trees in the forest.
Variable Interaction Importance

In some cases, interactions of variables are of more interest than a single variable. When you specify the VII= option in the PROC FOREST statement, the procedure computes variable interaction importance as follows.

The two-way interaction importance for variables \( v \) and \( w \) is

\[
\frac{\Delta^v_d + \Delta^w_e}{\Delta^v_d}
\]

where

- \( d \) and \( e \) denote nodes, where node \( d \) is the parent of node \( e \)
- \( v \) and \( w \) are variables, where \( v \) is the splitting variable for node \( d \) and \( w \) is the splitting variable for node \( e \)
- This summation is across all parent-child node combinations with respect to the variables \( v \) and \( w \).

The three-way interaction importance for variables \( v \), \( w \), and \( x \) is

\[
\frac{\Delta^v_d + \Delta^w_e + \Delta^x_f}{\Delta^v_d + \Delta^w_e}
\]

where

- \( d \), \( e \), and \( f \) denote nodes, where node \( d \) is the parent of node \( e \), which is the parent of node \( f \)
- \( v \), \( w \), and \( x \) are variables, where \( v \) is the splitting variable for node \( d \), \( w \) is the splitting variable for node \( e \), and \( x \) is the splitting variable for node \( f \)
- This summation is across all parent-child-grandchild node combinations with respect to the variables \( v \), \( w \), and \( x \).

When comparing variable importance values, you should compare a two-way interaction only with another two-way interaction, or a three-way interaction only with another three-way interaction. Comparing a two-way interaction to a three-way interaction is not meaningful, because the values for the importance are not on the same scale.

In addition to calculating the variable interaction importance, you can also generate SAS DATA step code to create interaction variables and save the code to a file by using the VIICODE statement. When you run the generated SAS DATA step code on the original data or on new data, interaction variables are created.

Created interaction variables are of the form \( W + V \), \( W - V \), or \( W \times V \), where the values used in the calculations come from the variables whose interactions are determined to be important. The values used are as follows:

- For continuous variables, the raw value of the variable is used.
- For categorical variables, the proportion of observations in the training set that is assigned to a specific branch is used.
Random Branch Assignment Importance Method

The random branch assignment (RBA) method computes the importance of an input variable \( v \) by comparing how well the data fit the predictions before and after they are modified. Neville and Tan (2014) motivate and introduce the RBA method of variable importance. To modify the predictions, the FOREST procedure replaces all splitting rules that use variable \( v \) by a rule that randomly assigns an observation to a branch. The probability of assigning an observation to a branch is proportional to the number of observations that are assigned to the branch in the current data. The current data are the training data when RBA is computed during training. Otherwise, the current data are those being scored on an existing model.

The RBA importance can be expressed mathematically as

\[
I_{RBA}(v) \propto \sum_{i=1}^{n} \text{Loss}(y_i, \hat{y}_i) - \sum_{i=1}^{n} \text{Loss}(\hat{y}_i, \hat{y}_i)
\]

where \( \hat{y}_i \) is the modified prediction for observation \( i \) and \( \hat{y}_i \) is the standard prediction.

For an interval target, PROC FOREST computes the RBA importance of squared error loss. For a nominal target, PROC FOREST uses the misclassification rate as the loss function.

Isolation Forests

An isolation forest (Liu, Ting, and Zhou (2008)) is a specially constructed forest that is used for anomaly detection instead of target prediction. When the FOREST procedure creates an isolation forest, it outputs anomaly scores in the scored data table that is specified in the OUTPUT statement.

For each split in an isolation forest, one input variable is randomly chosen. If the variable is an interval variable, then it is split at a random value between the maximum and minimum values of the observations in that node. If the variable is a nominal variable, then each level of the variable is assigned to a random branch. By constructing the forest this way, anomalous observations are likely to have a shorter path from the root node to the leaf node than nonanomalous observations have.

The anomaly score, \( s(x) \), of observation \( x \) is calculated as:

\[
s(x) = 2^{-h(x)}
\]

where \( h(x) \) is the average, over all trees, of the length of the path from the root node to the leaf node that contains observation \( x \), divided by the average length of all paths across all trees.

The anomaly score is always between 0 and 1, where values closer to 1 indicate a higher chance of the observation being an anomaly.
Hyperparameter Tuning

The options that control the training algorithm are often called hyperparameters to distinguish them from the parameters of the trained model. The main parameters in tree-based models are the splitting rules. Optimal hyperparameter settings depend on the data and are never known for certain. Nevertheless, here are some recommendations:

**BINMETHOD=QUANTILE**  Using quantile bins instead of bins of equal width often results in a better fit, especially when some inputs have outliers.

**NUMBIN=**  Using more bins sometimes improves and rarely degrades accuracy. However, training takes longer with more bins, often much longer with many more bins.

For more information about hyperparameter tuning, see the section “Hyperparameter Tuning” on page 23 in Chapter 3, “Shared Concepts.”

You can use the AUTOTUNE statement to tune the following options in the PROC FOREST statement:

- **MAXDEPTH=** option for the maximum depth of trees to grow in the forest
- **NTREES=** option for the number of trees to grow in the forest
- **INBAGFRACTION=** option for the bootstrap sample size for building each tree in the forest
- **VARS_TO_TRY=** option for the number of variables to randomly select at each node split for each tree in the forest

---

**k-fold Cross Validation**

The CROSSVALIDATION statement performs $k$-fold cross validation to assess the accuracy of a model. During cross validation, all data are divided into $k$ subsets (folds), where $k$ is the value of the KFOLD= option. For each fold, a new model is trained on the $(k-1)$ folds and then validated using the selected (holdout) fold. The assessment metrics are then averaged across all the holdout folds. The CROSSVALIDATION statement creates a table that has $k+1$ rows. The first $k$ rows contain the assessment metrics for each holdout fold, and the last row contains the average across all the holdout folds.

**Determining the Number of Parallel Evaluations**

The number of parallel fold evaluations is determined as follows:

1. The number of worker nodes to use in parallel subsessions is determined:
   a) If you specify a value greater than 0 in the NSUBSESSIONWORKERS= option to indicate the number of workers to use in parallel subsessions, then that value is used for each subsession.
   b) If you do not specify a value in the NSUBSESSIONWORKERS= option, then the number of workers for each subsession is the same as the number or workers used in the parent session.
c) If you specify a value of 0 in the NSUBSESSIONWORKERS= option, or if you specify the default value of 0, then the number of workers to use in each subsession defaults to the number of workers used 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 on the server, then the number of workers for each subsession is reduced to the number of workers on 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+\text{Number of Observations} \times \text{Number of Columns} / 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, then the number of parallel evaluations is limited to 16. Otherwise, the number of parallel evaluations is equal to the number of folds.
   b) In distributed mode, the upper limit for the number of parallel evaluations is calculated as 2*\frac{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.

**Displayed Output**

The FOREST procedure displays the parameters that are used to train the model, the fit statistics of the trained model, and other information. The output is organized into various tables, which are discussed here in order of their appearance.

**Model Information**

The “Model Information” table contains the settings of the training parameters. This table also contains some basic information about the trees in the resulting forest. This table is produced by default.

**Number of Observations**

The “Number of Observations” table contains the number of observations that are read from the input data table and the number of observations that are used in the analysis. When you specify the PARTITION statement, the table also indicates the number of observations that are used in each partition. This table is produced by default.

**Variable Importance**

The “Variable Importance” table displays variable importance based on residual sum of square errors, which is explained in the section “Measuring Variable Importance” on page 191. This table is produced by default.
RBA Variable Importance

The “RBA Variable Importance” table displays variable importance based on the random branch assignment (RBA) method, which is explained in the section “Random Branch Assignment Importance Method” on page 194. This table is produced by the RBAIMP option in the PROC FOREST statement.

Fit Statistics

The “Fit Statistics” table contains statistics that measure the model’s goodness of fit. The fit of the model to the data improves as the number of trees in the forest increases. Successive rows in the table contain fit statistics for a forest that has more trees. Fit statistics are described in the section “Measuring Prediction Error” on page 189. This table is produced by default.

Tuner Information

The “Tuner Information” table displays the setup values that the tuner uses. This table is produced by the AUTOTUNE statement.

Tuner Summary

The “Tuner Summary” table displays statistics about the tuning process. This table is produced by the AUTOTUNE statement.

Tuner Timing

The “Tuner Timing” table displays the total time spent on different tasks while tuning. This table is produced by the AUTOTUNE statement.

Best Configuration

The “Best Configuration” table displays the hyperparameters and objective function values for the best configuration. This table is produced by the AUTOTUNE statement.

Tuner Results

The “Tuner Results” table displays the values of the hyperparameters, the objective function for the default configuration (Iteration 0), and up to 10 best found configurations. This table is produced by the AUTOTUNE statement.

Cross-Validation Fit Statistics

The “Cross-Validation Fit Statistics” table contains the per fold and average assessment metrics of $k$-fold cross validation.
Evaluation History

The “Evaluation History” tables displays the values of the hyperparameters and the objective function for all configurations. This table is produced by the AUTOTUNE statement, either by default or when EVALHISTORY=ALL.

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.

Predicted Probability Names

The “Predicted Probability Names” table indicates the names of the created variables in either the score code or the data table that is specified in the OUT= option in the OUTPUT statement. This table is produced when you specify the PRINTTARGET option in the PROC FOREST statement.

ODS Table Names

Each table created by the FOREST procedure has a name associated with it, and you must use this name to refer to the table when you use ODS statements. The names of each table and a short description of the contents are listed in Table 10.6.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>BestConfiguration</td>
<td>Hyperparameters and objective function values for the best configuration</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>CrossValidateMLFitStat</td>
<td>Per fold and average assessment metrics of k-fold cross validation</td>
<td>CROSSVALIDATION</td>
<td>Default</td>
</tr>
<tr>
<td>EvaluationHistory</td>
<td>Values of the hyperparameters and the objective function for all configurations</td>
<td>AUTOTUNE</td>
<td>Default / EVALHISTORY=ALL</td>
</tr>
<tr>
<td>FitStatistics</td>
<td>Fit statistics from the model</td>
<td>PROC FOREST</td>
<td>Default</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>PROC FOREST</td>
<td>Default</td>
</tr>
<tr>
<td>Nobs</td>
<td>Number of observations</td>
<td>PROC FOREST</td>
<td>Default</td>
</tr>
<tr>
<td>OutputCasTables</td>
<td>See the section “OutputCasTables Table” on page 198</td>
<td>PROC FOREST / OUTPUT</td>
<td>OUTMODEL= / Default</td>
</tr>
<tr>
<td>PredName</td>
<td>Predicted name information for interval targets</td>
<td>PROC FOREST</td>
<td>PRINTTARGET</td>
</tr>
<tr>
<td>PredProbName</td>
<td>Predicted probability name information for nominal targets</td>
<td>PROC FOREST</td>
<td>PRINTTARGET</td>
</tr>
</tbody>
</table>
### Table 10.6  continued

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>PredIntoName</td>
<td>Predicted name information for nominal targets</td>
<td>PROC FOREST</td>
<td>PRINTTARGET</td>
</tr>
<tr>
<td>RBAImportance</td>
<td>Random branch assignment variable importance</td>
<td>PROC FOREST</td>
<td>RBAIMP</td>
</tr>
<tr>
<td>TunerInfo</td>
<td>Setup values used by the tuner</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerResults</td>
<td>Values of the hyperparameters, the objective function for the default configuration (Iteration 0), and up to 10 best found configurations</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerSummary</td>
<td>Statistics about the tuning process</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerTiming</td>
<td>Total time spent on different tasks while tuning</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>VariableImportance</td>
<td>Residual sum of squares variable importance</td>
<td>PROC FOREST</td>
<td>Default</td>
</tr>
</tbody>
</table>

### Examples: FOREST Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

### Example 10.1: Scoring New Data by Using a Previous Forest Model

This example illustrates how you can use the **OUTMODEL=** option to save a model table, and later use the model table to score a data table. It uses the JunkMail data set in the Sashelp library.

The JunkMail data set comes from a study that classifies whether an email is junk email (coded as 1) or not (coded as 0). The data set contains 4,601 observations with 59 variables. The response variable is a binary indicator of whether an email is considered spam or not. There are 57 predictor variables that record the frequencies of some common words and characters and the lengths of uninterrupted sequences of capital letters in emails.

You can load the Sashelp.JunkMail data set into your CAS session by naming your CAS engine libref in the first statement of the following DATA step:
data mycas.junkmail;
   set sashelp.junkmail;
run;

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined libref.

The following statements train a forest model and score the training data table. The OUTPUT statement scores the training data and saves the results to a new table named fit_at_runtime.

    proc forest data=mycas.junkmail outmodel=mycas.forest_model seed=12345;
       input 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 / level = interval;
    target class /level=nominal;
    output out=mycas.score_at_runtime;
    ods output FitStatistics=fit_at_runtime;
run;

The preceding statements produce the table shown in Output 10.1.1. The table shows the training and out-of-bag statistics.
Example 10.1: Scoring New Data by Using a Previous Forest Model

Output 10.1.1  Fit Statistics: Fit at Run Time

<table>
<thead>
<tr>
<th>Number of Trees</th>
<th>OOB Average Square Error</th>
<th>Training Average Square Error</th>
<th>OOB Misclassification Rate</th>
<th>Training Misclassification Rate</th>
<th>OOB Log Loss</th>
<th>Training Log Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0911</td>
<td>0.0708</td>
<td>0.1150</td>
<td>0.0913</td>
<td>0.941</td>
<td>0.580</td>
</tr>
<tr>
<td>2</td>
<td>0.0822</td>
<td>0.0529</td>
<td>0.1055</td>
<td>0.0659</td>
<td>0.716</td>
<td>0.229</td>
</tr>
<tr>
<td>3</td>
<td>0.0723</td>
<td>0.0472</td>
<td>0.0917</td>
<td>0.0593</td>
<td>0.531</td>
<td>0.181</td>
</tr>
<tr>
<td>4</td>
<td>0.0705</td>
<td>0.0471</td>
<td>0.0878</td>
<td>0.0587</td>
<td>0.446</td>
<td>0.172</td>
</tr>
<tr>
<td>5</td>
<td>0.0656</td>
<td>0.0450</td>
<td>0.0824</td>
<td>0.0561</td>
<td>0.360</td>
<td>0.163</td>
</tr>
<tr>
<td>6</td>
<td>0.0629</td>
<td>0.0433</td>
<td>0.0797</td>
<td>0.0530</td>
<td>0.313</td>
<td>0.159</td>
</tr>
<tr>
<td>7</td>
<td>0.0623</td>
<td>0.0432</td>
<td>0.0821</td>
<td>0.0552</td>
<td>0.290</td>
<td>0.160</td>
</tr>
<tr>
<td>8</td>
<td>0.0602</td>
<td>0.0430</td>
<td>0.0801</td>
<td>0.0548</td>
<td>0.259</td>
<td>0.160</td>
</tr>
<tr>
<td>9</td>
<td>0.0588</td>
<td>0.0421</td>
<td>0.0759</td>
<td>0.0537</td>
<td>0.251</td>
<td>0.158</td>
</tr>
<tr>
<td>10</td>
<td>0.0583</td>
<td>0.0417</td>
<td>0.0750</td>
<td>0.0506</td>
<td>0.238</td>
<td>0.158</td>
</tr>
<tr>
<td>91</td>
<td>0.0506</td>
<td>0.0385</td>
<td>0.0606</td>
<td>0.0454</td>
<td>0.189</td>
<td>0.153</td>
</tr>
<tr>
<td>92</td>
<td>0.0506</td>
<td>0.0384</td>
<td>0.0602</td>
<td>0.0454</td>
<td>0.189</td>
<td>0.153</td>
</tr>
<tr>
<td>93</td>
<td>0.0507</td>
<td>0.0385</td>
<td>0.0606</td>
<td>0.0454</td>
<td>0.189</td>
<td>0.153</td>
</tr>
<tr>
<td>94</td>
<td>0.0506</td>
<td>0.0385</td>
<td>0.0604</td>
<td>0.0454</td>
<td>0.189</td>
<td>0.153</td>
</tr>
<tr>
<td>95</td>
<td>0.0506</td>
<td>0.0385</td>
<td>0.0602</td>
<td>0.0459</td>
<td>0.189</td>
<td>0.153</td>
</tr>
<tr>
<td>96</td>
<td>0.0506</td>
<td>0.0385</td>
<td>0.0604</td>
<td>0.0456</td>
<td>0.189</td>
<td>0.153</td>
</tr>
<tr>
<td>97</td>
<td>0.0507</td>
<td>0.0385</td>
<td>0.0606</td>
<td>0.0456</td>
<td>0.189</td>
<td>0.153</td>
</tr>
<tr>
<td>98</td>
<td>0.0506</td>
<td>0.0385</td>
<td>0.0606</td>
<td>0.0456</td>
<td>0.189</td>
<td>0.153</td>
</tr>
<tr>
<td>99</td>
<td>0.0507</td>
<td>0.0385</td>
<td>0.0604</td>
<td>0.0454</td>
<td>0.190</td>
<td>0.153</td>
</tr>
<tr>
<td>100</td>
<td>0.0507</td>
<td>0.0385</td>
<td>0.0602</td>
<td>0.0456</td>
<td>0.190</td>
<td>0.153</td>
</tr>
</tbody>
</table>

The following statements use a previously saved model to score new data:

```plaintext
proc forest data=mycas.junkmail inmodel=mycas.forest_model;
 output out=mycas.score_later;
ods output FitStatistics=fit_later;
run;
```

When you specify the INMODEL= option to use a previously created forest model, the fit statistics table no longer shows the out-of-bag statistics, because you are scoring new data. In this example, the scored data are the same as the training data, so you can see that the statistics in Output 10.1.2 match those previously seen in Output 10.1.1.
Output 10.1.2  Fit Statistics: Fit Later

<table>
<thead>
<tr>
<th>Number of Trees</th>
<th>Average Square Error</th>
<th>Misclassification Rate</th>
<th>Log Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0708</td>
<td>0.0913</td>
<td>0.580</td>
</tr>
<tr>
<td>2</td>
<td>0.0529</td>
<td>0.0659</td>
<td>0.229</td>
</tr>
<tr>
<td>3</td>
<td>0.0472</td>
<td>0.0593</td>
<td>0.181</td>
</tr>
<tr>
<td>4</td>
<td>0.0471</td>
<td>0.0587</td>
<td>0.172</td>
</tr>
<tr>
<td>5</td>
<td>0.0450</td>
<td>0.0561</td>
<td>0.163</td>
</tr>
<tr>
<td>6</td>
<td>0.0433</td>
<td>0.0530</td>
<td>0.159</td>
</tr>
<tr>
<td>7</td>
<td>0.0432</td>
<td>0.0552</td>
<td>0.160</td>
</tr>
<tr>
<td>8</td>
<td>0.0430</td>
<td>0.0548</td>
<td>0.160</td>
</tr>
<tr>
<td>9</td>
<td>0.0421</td>
<td>0.0537</td>
<td>0.158</td>
</tr>
<tr>
<td>10</td>
<td>0.0417</td>
<td>0.0506</td>
<td>0.158</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>91</td>
<td>0.0385</td>
<td>0.0454</td>
<td>0.153</td>
</tr>
<tr>
<td>92</td>
<td>0.0384</td>
<td>0.0454</td>
<td>0.153</td>
</tr>
<tr>
<td>93</td>
<td>0.0385</td>
<td>0.0454</td>
<td>0.153</td>
</tr>
<tr>
<td>94</td>
<td>0.0385</td>
<td>0.0454</td>
<td>0.153</td>
</tr>
<tr>
<td>95</td>
<td>0.0385</td>
<td>0.0459</td>
<td>0.153</td>
</tr>
<tr>
<td>96</td>
<td>0.0385</td>
<td>0.0456</td>
<td>0.153</td>
</tr>
<tr>
<td>97</td>
<td>0.0385</td>
<td>0.0456</td>
<td>0.153</td>
</tr>
<tr>
<td>98</td>
<td>0.0385</td>
<td>0.0456</td>
<td>0.153</td>
</tr>
<tr>
<td>99</td>
<td>0.0385</td>
<td>0.0454</td>
<td>0.153</td>
</tr>
<tr>
<td>100</td>
<td>0.0385</td>
<td>0.0456</td>
<td>0.153</td>
</tr>
</tbody>
</table>

This example demonstrates that the FOREST procedure can score an input data table by using a previously saved forest model, which was saved using the OUTMODEL= option in a previous procedure run. If you want to properly score a new data table, you must not modify the mycas.forest_model table, because doing so could invalidate the constructed forest model. As with any scoring of new data, the variables that are used in the model creation must be present in order for you to score a new table.

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”
Example 10.2: Generating an Isolation Forest

This example illustrates how to create an isolation forest and generate an anomaly score for each observation. The score estimates how atypical the observation is.

The following DATA step generates a data table and loads the table into your CAS session. The table consists of 456 observations in 13 clumps, which are arranged in a spiral that is defined by the two input variables, \( x_1 \) and \( x_2 \). The farther the clump is along the spiral curve from the center, the larger the clump. The data also contain a target variable (\( y \)) and an identification variable (\( id \)).

```sas
data mycas.spiral;
  keep x1 x2 y id;
  nNow = 200;
  sdNow = 0.2;
  four_pi = 4 * constant("pi");
  call streaminit(3331333);
  y = 0;
  id = 0;
  do theta = 0 to 12;
    r = 1.0 - theta / four_pi;
    mu1 = cos(theta);
    mu2 = sin(theta);
    ss = r*r / (mu1*mu1 + mu2*mu2);
    s = sqrt(ss);
    mu1 = s*mu1;
    mu2 = s*mu2;
    nNow = nNow * 0.7;
    sdNow = sdNow * 0.75;
    do i = 1 to nNow;
      x1 = rand('normal', mu1, sdNow);
      x2 = rand('normal', mu2, sdNow);
      y = 1 - y;
      id = id + 1;
      output;
    end;
  end;
run;
```

These statements assume that your CAS engine libref is named `mycas`, but you can substitute any appropriately defined CAS engine libref.

The `ISOLATION` option in the `PROC FOREST` statement of the following code requests that the procedure create an isolation forest. The `OUTPUT` statement scores the training data and saves the results to a new table named `score`, which contains the anomaly score and the predictions for each observation. The predictions are unreliable because the trees are generally shallow and are generated by using random splits.

The following code runs the `FOREST` procedure with the `ISOLATION` option to generate an isolation forest. The results are saved in the table `mycas.score`, and all of the input variables are copied to the output table. Finally, the table is returned from CAS to the SAS client when the `SORT` procedure is used. In general, you might not want to return data from CAS to the SAS client, especially in the case of large data tables.
proc forest data=mycas.spiral isolation seed=12345;
  input x1 x2 /level=interval;
  id id;
  output out=mycas.score copyvars=(_ALL_);
run;

proc sort data=mycas.score out=score;
  by id;
run;

The following code plots the data with a color ramp from cyan to red, based on the anomaly score. The most anomalous observations are colored red.

proc template;
  define statgraph anomalyPlot;
    begingraph;
      layout overlay;
      scatterplot y=x2 x=x1 / name='color'
        markerattrs=(symbol=circlefilled)
        colormodel=(cyan ligr red)
        colorresponse=_Anomaly_
        continuouslegend 'color'/ title='_Anomaly_';
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=score template=anomalyPlot;
run;

Output 10.2.1 shows that the most anomalous points are either near the exterior boundary of the data or in a small clump in the center.
References


Output 10.2.1  Spiral Data Colored by Anomaly Score
# Chapter 11
## The GMM Procedure

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: GMM Procedure</td>
<td>208</td>
</tr>
<tr>
<td>GMM Procedure Features</td>
<td>208</td>
</tr>
<tr>
<td>GMM Procedure Results</td>
<td>208</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>209</td>
</tr>
<tr>
<td>Getting Started: GMM Procedure</td>
<td>210</td>
</tr>
<tr>
<td>Syntax: GMM Procedure</td>
<td>215</td>
</tr>
<tr>
<td>PROC GMM Statement</td>
<td>215</td>
</tr>
<tr>
<td>DISPLAY Statement</td>
<td>217</td>
</tr>
<tr>
<td>DISPLAYOUT Statement</td>
<td>218</td>
</tr>
<tr>
<td>INPUT Statement</td>
<td>219</td>
</tr>
<tr>
<td>SAVESTATE Statement</td>
<td>219</td>
</tr>
<tr>
<td>SCORE Statement</td>
<td>220</td>
</tr>
<tr>
<td>Details: GMM Procedure</td>
<td>221</td>
</tr>
<tr>
<td>Gaussian Mixture Model</td>
<td>221</td>
</tr>
<tr>
<td>Dirichlet Process</td>
<td>222</td>
</tr>
<tr>
<td>Missing Values</td>
<td>222</td>
</tr>
<tr>
<td>Random Seed</td>
<td>222</td>
</tr>
<tr>
<td>Variational Bayes Method</td>
<td>223</td>
</tr>
<tr>
<td>Covariance Matrix Type</td>
<td>223</td>
</tr>
<tr>
<td>Score Output</td>
<td>223</td>
</tr>
<tr>
<td>ClusterSumOut Output</td>
<td>224</td>
</tr>
<tr>
<td>ClusterCovOut Output</td>
<td>224</td>
</tr>
<tr>
<td>Displayed Output</td>
<td>224</td>
</tr>
<tr>
<td>Number of Observations</td>
<td>225</td>
</tr>
<tr>
<td>Descriptive Statistics</td>
<td>225</td>
</tr>
<tr>
<td>Model Information</td>
<td>225</td>
</tr>
<tr>
<td>ODS Table Names</td>
<td>225</td>
</tr>
<tr>
<td>Examples: GMM Procedure</td>
<td>226</td>
</tr>
<tr>
<td>Example 11.1: Performing Cluster Analysis</td>
<td>226</td>
</tr>
<tr>
<td>Example 11.2: Performing Cluster Analysis and Storing the Model</td>
<td>230</td>
</tr>
<tr>
<td>References</td>
<td>231</td>
</tr>
</tbody>
</table>
Overview: GMM Procedure

The GMM procedure performs clustering—a common step in data exploration—on quantitative data in SAS Viya.

You can use this procedure to read and write data in distributed form, as well as to perform clustering and scoring in parallel by making full use of multicore computers or a distributed computing environment.

PROC GMM 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.

GMM Procedure Features

The GMM procedure has the following basic features:

- can execute clustering in parallel and is highly multithreaded
- 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 221.
- 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 222.
- 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 Method” on page 223.

GMM Procedure Results

The GMM procedure produces results of the cluster analysis in four CAS data tables:

- The CLUSTERCOVOUT= data table is produced by the PROC GMM statement. This data table contains the Gaussian covariance matrix of each cluster. It can be used to examine the shape of each cluster in the input data space. For more information, see the section “ClusterCovOut Output” on page 224.
- The CLUSTERSUMOUT= data table is produced by the PROC GMM statement. This data table contains the Gaussian center of each cluster. It also contains the number of observations in each cluster and the neighbor cluster of each cluster. For more information, see the section “ClusterSumOut Output” on page 224.
The OUT= data table is produced by the SCORE statement. This data 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 distribution over all clusters for each observation is given, PROC GMM shows not only the predicted cluster scores but also how it makes the predictions. For more information, see the section “Score Output” on page 223.

The RSTORE= data table is produced by the SAVESTATE statement. This data table saves the predicted Gaussian mixture model that PROC GMM generates so that the model can be used for later data clustering. For more information, see the section “SAVESTATE Statement” on page 219.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```sas
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```sas
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```sas
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Getting Started: GMM Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example shows how to use the GMM procedure to do soft clustering for observations in an input CAS table.

Suppose you want to group the observations in the input CAS table `mycas.triangle`, in which the variables are raw measures on interval scales.

The following DATA step creates the input data table `mycas.triangle` in your CAS session. This data table contains three variables: `x` and `y` are the input variables, and `c` is the cluster index.

```sas
data mycas.triangle;
  title 'Using PROC GMM for Soft Clustering';
  drop n;
  do n=1 to 100;
    x=rannor(123);
    y=rannor(123);
    c=1;
    output;
  end;
  do n=1 to 100;
    x=-3/2+rannor(123);
    y=sqrt(27)/2+rannor(123);
    c=2;
    output;
  end;
  do n=1 to 100;
    x=3/2+rannor(123);
    y=sqrt(27)/2+rannor(123);
    c=3;
    output;
  end;
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 PROC GMM and output the results to ODS tables and CAS tables:

```sas
proc gmm
  data=mycas.triangle
  seed=123
  nThreads=32
  maxClusters=100
  alpha=1
  inference=VB (maxVbIter=100 covariance=diagonal threshold=0.01)
  clusterSumOut=mycas.clustersum
  clusterCovOut=mycas.clustercov;
```

```sas
Note: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example shows how to use the GMM procedure to do soft clustering for observations in an input CAS table.

Suppose you want to group the observations in the input CAS table `mycas.triangle`, in which the variables are raw measures on interval scales.

The following DATA step creates the input data table `mycas.triangle` in your CAS session. This data table contains three variables: `x` and `y` are the input variables, and `c` is the cluster index.

```sas
data mycas.triangle;
  title 'Using PROC GMM for Soft Clustering';
  drop n;
  do n=1 to 100;
    x=rannor(123);
    y=rannor(123);
    c=1;
    output;
  end;
  do n=1 to 100;
    x=-3/2+rannor(123);
    y=sqrt(27)/2+rannor(123);
    c=2;
    output;
  end;
  do n=1 to 100;
    x=3/2+rannor(123);
    y=sqrt(27)/2+rannor(123);
    c=3;
    output;
  end;
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 PROC GMM and output the results to ODS tables and CAS tables:

```sas
proc gmm
  data=mycas.triangle
  seed=123
  nThreads=32
  maxClusters=100
  alpha=1
  inference=VB (maxVbIter=100 covariance=diagonal threshold=0.01)
  clusterSumOut=mycas.clustersum
  clusterCovOut=mycas.cluster cov;
```
input x y;
score out=mycas.score copyvars=(x y c);
ods select nObs descStats modelInfo;
run;

Figure 11.1 shows the number of observations read and number of observations used. Because the mycas.triangle data table contains no missing values, the number of observations read and number of observations used are equal in this example. When the input data table contains missing values, PROC GMM ignores the observations (rows) with missing values in the input data table and does not use them in the analysis.

**Figure 11.1** Number of Observations

**Using PROC GMM for Soft Clustering**

The GMM Procedure

<table>
<thead>
<tr>
<th>Number of Observations Read</th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Used</td>
<td>300</td>
</tr>
</tbody>
</table>

Figure 11.2 shows statistics for each variable in the INPUT statement, including the mean, standard deviation, minimum, and maximum of each input variable.

**Figure 11.2** Input Variable Statistics

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>-0.037144</td>
<td>1.591540</td>
<td>-4.123222</td>
<td>3.955673</td>
</tr>
<tr>
<td>y</td>
<td>1.691436</td>
<td>1.620588</td>
<td>-3.078746</td>
<td>5.400477</td>
</tr>
</tbody>
</table>

Figure 11.3 shows the values of the parameters that are used in the clustering. The number of clusters is estimated using the Dirichlet process in a nonparametric way. In this example, three clusters are correctly detected. Figure 11.3 also shows the parameter values for the other options.

**Figure 11.3** Model Information

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustering Algorithm</td>
</tr>
<tr>
<td>Inference Method</td>
</tr>
<tr>
<td>Random Number Seed</td>
</tr>
<tr>
<td>Clusters Maximum</td>
</tr>
<tr>
<td>Dirichlet Process Mass</td>
</tr>
<tr>
<td>Covariance Matrix Type</td>
</tr>
<tr>
<td>VB Iterations</td>
</tr>
<tr>
<td>VB Threshold</td>
</tr>
<tr>
<td>VB Converge</td>
</tr>
<tr>
<td>Clusters Discovered</td>
</tr>
</tbody>
</table>

The following statements extract the first 10 observations from the output score table and display them in the table in Figure 11.4. This figure provides not only the predicted cluster score but also the probability distribution over all clusters for each observation. As a result, you can get a sense of how the GMM
procedure makes the cluster score predictions. To be specific, in Figure 11.4 PROC GMM is less confident in its prediction for the first observation than in its prediction for the others. Also note that the cluster indices in scored output are different from the ones in the input variable. For instance, cluster index 2 in _PREDICTED_CLUSTER_ matches to 1 in input c.

```
proc print noobs data=mycas.score(obs=10);
run;
```

**Figure 11.4 Cluster Score Table**

Using PROC GMM for Soft Clustering

| x       | y       | c | _CLUSTER_1_ | _CLUSTER_2_ | _CLUSTER_3_ | _PREDICTED_CLUSTER_
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.32659</td>
<td>1.54244</td>
<td>1</td>
<td>0.269141085</td>
<td>0.6053353352</td>
<td>0.1255235798</td>
<td>2</td>
</tr>
<tr>
<td>-0.02210</td>
<td>-0.77265</td>
<td>1</td>
<td>0.0021388046</td>
<td>0.9976941814</td>
<td>0.0001670139</td>
<td>2</td>
</tr>
<tr>
<td>1.04173</td>
<td>-1.85990</td>
<td>1</td>
<td>0.0000367332</td>
<td>0.9999487105</td>
<td>0.0000145563</td>
<td>2</td>
</tr>
<tr>
<td>1.67034</td>
<td>-0.47339</td>
<td>1</td>
<td>0.0001843824</td>
<td>0.9900625343</td>
<td>0.0009753074</td>
<td>2</td>
</tr>
<tr>
<td>-0.10178</td>
<td>0.27807</td>
<td>1</td>
<td>0.020096045</td>
<td>0.9741773484</td>
<td>0.0057266066</td>
<td>2</td>
</tr>
<tr>
<td>0.16708</td>
<td>-0.26922</td>
<td>1</td>
<td>0.0040562605</td>
<td>0.9945713346</td>
<td>0.0013724049</td>
<td>2</td>
</tr>
<tr>
<td>0.34102</td>
<td>-2.16919</td>
<td>1</td>
<td>0.0000728113</td>
<td>0.9999259857</td>
<td>1.20298286-6</td>
<td>2</td>
</tr>
<tr>
<td>-0.25232</td>
<td>-1.61289</td>
<td>1</td>
<td>0.0006497335</td>
<td>0.9993459577</td>
<td>4.6684448E-6</td>
<td>2</td>
</tr>
<tr>
<td>-0.29096</td>
<td>-1.17275</td>
<td>1</td>
<td>0.0016408131</td>
<td>0.9983344651</td>
<td>0.0000247217</td>
<td>2</td>
</tr>
<tr>
<td>-0.33846</td>
<td>-3.07875</td>
<td>1</td>
<td>0.000492516</td>
<td>0.9999507405</td>
<td>7.9404955E-9</td>
<td>2</td>
</tr>
</tbody>
</table>

You can intuitively examine the soft-clustering effect of the GMM procedure by using the probability distributions of the observations over the clusters in the data table mycas.score. The following PROC SGPLOT step generates the scatter plot of all observations in the input data table, whose shapes are the circle, triangle, and square, which represent their cluster scores (memberships). More interesting is that the color of each observation is a mixture color of green, blue, and red, and the mixing proportion is its probability distribution over the three clusters. Thus the variations in the colors of the observations show PROC GMM’s soft-clustering result (see Figure 11.5).

```
options cmplib=sasuser.funcs;
proc fcmp outlib=sasuser.funcs.color;
    subroutine RGBtoHSV(r, g, b, h, s, v);
        outargs h, s, v;
        r1=r/255;
        g1=g/255;
        b1=b/255;
        cmax=max(r1, max(g1, b1));
        cmin=min(r1, min(g1, b1));
        del=cmax-cmin;
        put cmax cmin del;
        if del=0 then h=0;
        else if cmax=r1 then h=60*mod((g1-b1)/del, 6);
        else if cmax=g1 then h=60*((b1-r1)/del + 2);
        else h=60*((r1-g1)/del+4);
        if h < 0 then h=360+h;
    endsub;
run;
quit;
```
data mycas.score2;
  set mycas.Score;
  call RGBtoHSV(_cluster_1_*255, _cluster_2_*255, _cluster_3_*255, h, s, v);
run;
ods graphics / reset width=4in height=4in imagename='Clusters' attrpriority=none;
title 'Using PROC GMM for Soft Clustering';
proc sgplot data=mycas.score2 noautolegend;
  styleattrs datasymbols=(circlefilled trianglefilled squarefilled);
  scatter x=x y=y / group=_predicted_Cluster_ colorresponse=h markerattrs=(size=10)
    colormodel=(cx00ff00 cx0000ff cxff0000 cx00ff00) name='s';
  xaxis display=(NOLABEL); yaxis display=(NOLABEL);
  legenditem name="c1" type=marker / label="Cluster 1"
    markerattrs=(symbol=circlefilled color=cx0000ff size=10);
  legenditem name="c2" type=marker / label="Cluster 2"
    markerattrs=(symbol=trianglefilled color=cx00ff00 size=10);
  legenditem name="c3" type=marker / label="Cluster 3"
    markerattrs=(symbol=squarefilled color=cxff0000 size=10);
  keylegend 'c1' 'c2' 'c3';
run;

Figure 11.5 Soft Clustering

In Figure 11.5, you can see that the colors of the observations that lie on the borders between the clusters are heavily blended, because of their large probabilities compared with other clusters. For example, the observations along the border between the blue and red clusters are purple. This soft-clustering result tells more about the properties of the input data than the cluster memberships in the hard-clustering results do.
The following statements produce the cluster summary table in Figure 11.6:

```
proc print noobs data=mycas.clustersum;
run;
```

**Figure 11.6** Cluster Summary Table

**Using PROC GMM for Soft Clustering**

<table>
<thead>
<tr>
<th><em>CLUSTER_ID</em></th>
<th><em>SIZE</em></th>
<th><em>NEIGHBOR</em></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>88</td>
<td>2</td>
<td>-1.74786</td>
<td>2.50806</td>
</tr>
<tr>
<td>2</td>
<td>110</td>
<td>1</td>
<td>-0.05153</td>
<td>0.05330</td>
</tr>
<tr>
<td>3</td>
<td>102</td>
<td>2</td>
<td>1.47123</td>
<td>2.76116</td>
</tr>
</tbody>
</table>

In Figure 11.6 the column _SIZE_ shows the number of observations in each cluster. The column _NEIGHBOR_ shows the cluster closest to each cluster. Figure 11.6 also shows the Gaussian mean for each cluster.

The following statements produce the cluster covariance table in Figure 11.7:

```
proc print noobs data=mycas.clustercov;
run;
```

**Figure 11.7** Cluster Covariance Table

**Using PROC GMM for Soft Clustering**

<table>
<thead>
<tr>
<th><em>CLUSTER_ID</em></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.88440</td>
<td>0.00000</td>
</tr>
<tr>
<td>1</td>
<td>0.00000</td>
<td>1.21012</td>
</tr>
<tr>
<td>2</td>
<td>0.78642</td>
<td>0.00000</td>
</tr>
<tr>
<td>2</td>
<td>0.00000</td>
<td>1.12311</td>
</tr>
<tr>
<td>3</td>
<td>0.99437</td>
<td>0.00000</td>
</tr>
<tr>
<td>3</td>
<td>0.00000</td>
<td>0.79790</td>
</tr>
</tbody>
</table>

Figure 11.7 shows the Gaussian covariance matrices of the Gaussian mixture.
Syntax: GMM Procedure

The following statements are available in the GMM procedure:

```
PROC GMM <options> ;
  DISPLAY <table-list> </options> ;
  DISPLAYOUT <table-spec-list> </options> ;
  INPUT variables < INTERVAL > ;
  SCORE OUT=CAS-libref.data-table <options> ;
  SAVESTATE RSTORE=CAS-libref.data-table <option> ;
```

The PROC GMM statement and an INPUT statement are required. The PROC GMM takes only interval input variables. Any nominal variables that you specify in the INPUT statement are ignored.

The following sections describe the PROC GMM statement and then describe the other statements in alphabetical order.

PROC GMM Statement

```
PROC GMM <options> ;
```

The PROC GMM statement invokes the procedure. Table 11.1 summarizes the options available in the PROC GMM statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA=</td>
<td>Specifies the input data table</td>
</tr>
<tr>
<td>NTHREADS=</td>
<td>Specifies the number of threads to use for the computation</td>
</tr>
<tr>
<td>ALPHA=</td>
<td>Specifies the mass parameter of the Dirichlet process</td>
</tr>
<tr>
<td>CLUSTERCOVOUT=</td>
<td>Specifies the output data table to contain the Gaussian covariance matrices</td>
</tr>
<tr>
<td>CLUSTERSUMOUT=</td>
<td>Specifies the output data table to contain the cluster summary</td>
</tr>
<tr>
<td>INFERENCEN=</td>
<td>Specifies the inference method for the Gaussian mixture model</td>
</tr>
<tr>
<td>MAXCLUSTERS=</td>
<td>Specifies the maximum number of possible clusters</td>
</tr>
<tr>
<td>SEED=</td>
<td>Specifies the seed to use for pseudorandom number generation</td>
</tr>
</tbody>
</table>

You can specify the following options:
ALPHA=number
specifies the mass parameter of the Dirichlet process. The value of number must be a positive real number. When you specify a larger number, the Dirichlet process tends to discover more clusters in the input data. By default, ALPHA=1.

CLUSTERCOVOUT=CAS-libref.data-table
creates the output data table that contains the summary of the clustering results. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 209.

This data table includes the covariance matrices of the Gaussian distributions for the clusters that PROC GMM discovers. The table contains the cluster ID (_CLUSTER_ID_) and the covariance matrices of the Gaussian distributions of the discovered clusters, which consist of the nonconstant variables specified in the INPUT statement. For more information, see the section “ClusterCovOut Output” on page 224.

CLUSTERSUMOUT=CAS-libref.data-table
creates the output data table that contains the summary of the clustering results. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 209.

The summary consists of the following items: the cluster ID (_CLUSTER_ID_), the numbers of observations in the clusters (_SIZE_), the clusters closest to the clusters (_NEIGHBOR_), and the cluster centroids (the means of the Gaussian distributions, which consist of the nonconstant variables specified in the INPUT statement). For more information, see the section “ClusterSumOut Output” on page 224.

DATA=CAS-libref.data-table
names the input data table for PROC GMM to use. The default is the most recently created data table. CAS-libref.data-table is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about CAS-libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 209.

data-table specifies the name of the input data table.

INFERENCE=VB < suboptions >
performs the inference by using the variational Bayesian (VB) algorithm; for more information see the section “Variational Bayes Method” on page 223.

You can specify one or more of the following suboptions:

COVARIANCE=keyword specifies the type of covariance matrices in the Gaussian mixture model. For more information, see the section “Covariance Matrix Type” on page 223. You can specify the following keywords:
DIAGONAL specifies the covariance matrices of the Gaussian distributions as diagonal matrices.

FULL specifies the covariance matrices of the Gaussian distributions as full matrices.

By default, COVARIANCE=DIAGONAL.

MAXVBITER=number specifies the maximum number of VB iterations. The VB iterations stop either when the number of iterations reaches the 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 by THRESHOLD=number. When the VB iterations stop with the threshold criterion satisfied, the VB reaches its convergence. By default, MAXVBITER=100.

THRESHOLD=number specifies the threshold to define the convergence of the VB algorithm. By default, THRESHOLD=0.01.

MAXCLUSTERS=number specifies the maximum number of clusters that PROC GMM searches to find the best number of clusters by using the Dirichlet process. By default, MAXCLUSTERS=100.

NTHREADS=number-of-threads specifies the number of threads to use for the computation. The default value is the number of CPUs available in the machine.

SEED=number specifies an integer to use to start the pseudorandom number generator. If you do not specify a seed or if you specify a value less than or equal to zero, the seed is generated by reading the time of day from the computer’s clock. For more information, see the section “Random Seed” on page 222.

DISPLAY Statement

DISPLAY < table-list > < / options > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see SAS Output Delivery System: Procedures Guide.

You can specify the table-list as a list of table names, paths, partial pathnames, and regular expressions.
The table names that you can specify are listed in the section “ODS Table Names” on page 225. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path Bygroup1.Summary:SelectionSummary. A partial pathname does not include all groups; for example, SelectionSummary and Summary:SelectionSummary are partial pathnames for Bygroup1.Summary:SelectionSummary.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both SelectionSummary and Summary:SelectionSummary select Bygroup1.Summary:SelectionSummary.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the Bygroup1.Summary:SelectionSummary table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the Bygroup1.Summary:SelectionSummary table is not selected.

You can specify the following options after a slash (/):

- **CASESENSITIVE**
  
  performs a case-sensitive comparison of table names in the table-list to display table names when tables are subsetted for display. To preserve case, you must enclose table names in the table-list in quotation marks.

- **EXCLUDE**
  
  displays all display tables except those that you specify in the table-list.

- **EXCLUDEALL**
  
  suppresses display of all tables. This option takes precedence over the other options.

- **TRACE**
  
  displays the display table names, labels, and paths.

---

**DISPLAYOUT Statement**

```
DISPLAYOUT table-spec-list < / options > ;
```

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see SAS Output Delivery System: Procedures Guide.

The table-spec-list specifies a list of CAS output tables to create. Each entry in the list has either a key=value format or a key format:

- **key=value** specifies key as the ODS table name, path, or partial pathname, and specifies value as the CAS output table name.

- **key** specifies key as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “ODS Table Names” on page 225. You cannot specify the ODS table named OutputCasTables in the table-spec-list.

Table names and partial pathnames are discussed under the DISPLAY statement. The DISPLAYOUT statement does not support regular expressions.
You can specify the following *options* after a slash (/):

**INCLUDEALL**
creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the `table-spec-list` specification is ignored.

**NOREPLACE**
does not replace any existing CAS output table of the same name.

**REPEATED**
replicates all CAS output tables on all nodes.

---

**INPUT Statement**

```input
INPUT variables < INTERVAL > ;
```

The INPUT statement specifies the names of the *variables* to use in clustering. The GMM procedure uses only interval input variables for its clustering. Any nominal variables that you specify in the INPUT statement are ignored.

Any input interval variable that has a standard deviation of zero indicates a constant variable (column) in the input data table. Such constant variables do not contribute to the clustering and are ignored.

---

**SAVESTATE Statement**

```save
SAVESTATE RSTORE=CAS-libref.data-table < option > ;
```

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can then use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 4, “The ASTORE Procedure.”

You must specify the following option:

**RSTORE=** *CAS-libref.data-table*
specifies a data table in which to save the analytic store for the model. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the `DATA=` option and the section “Using CAS Sessions and CAS Engine Librefs” on page 209.
SCORE Statement

```
SCORE OUT=CAS-libref.data-table < option > ;
```

```
OUTPUT OUT=CAS-libref.data-table < option > ;
```

The SCORE statement writes the cluster membership information of each observation to the output data table that is specified in the OUT= option. This information includes the variables that are specified in the COPYVARS= option, the predicted cluster scores, and the probability distributions over the discovered clusters. The probability value can be interpreted as the probability of that observation being a member of that cluster. This information enables you to use the GMM procedure as a soft-clustering technique. The details are described in the section “Score Output” on page 223.

You must specify the following option:

```
OUT=CAS-libref.data-table
```

names the output data table for PROC GMM to use. You must specify this option before any other options. `CAS-libref.data-table` is a two-level name, where

- **CAS-libref** refers to a collection of information that is defined in the LIBNAME statement and includes the `caslib`, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 209.

- **data-table** specifies the name of the output data table.

The output data table contains the scored data. When you specify this option, the table contains all variables that are specified in the COPYVARS= option, followed by the probability distribution over the discovered clusters, and then the predicted cluster score.

You can also specify the following option:

```
COPYVAR=variable
COPYVARS=(variables)
```

lists one or more variables to transfer from the input data table to the scored output data table, provided that the output data table produces one or more records per input observation. By default, the GMM procedure does not transfer any variables from the input data table to the output data table. The COPYVARS= option accepts numeric and character variables. You can also use `COPYVARS=(_ALL_)` to include all the input variables.
**Details: GMM Procedure**

The GMM procedure performs soft clustering by using the Gaussian mixture model on the data in the input data table. The number of clusters among the input data is inferred by using the Dirichlet process in a nonparametric style. PROC GMM computes the following statistics as the model output:

- the input variable statistics, including the mean, standard deviation, minimum, and maximum
- a summary of the clustering results, including the number of clusters, the number of observations in the clusters, and the neighbor clusters
- the means, which indicate the centers, and the covariances, which indicate the shape of the discovered clusters
- the predicted cluster scores and probability distributions over discovered clusters for the observations

You can use these statistics to analyze the clustering results.

**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 procedure, 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 \( \mu \) is the Gaussian mean; \( \Sigma \) is the Gaussian covariance matrix; \( \pi \) 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 \( \alpha_0 \) 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 procedure 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 PROC GMM, 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 PROC GMM, 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; $N = \sum_k n_k$ is the total number of observations in the input data table; and $\alpha_0$ and $H_0$ are the mass and base measure, respectively, of the Dirichlet process:

- Priors:
  $$H \sim \text{DP}(\alpha_0, H_0)$$

- Posterior:
  $$H|n_1, n_2, \ldots, n_K \sim \text{DP}(\alpha_0 + N, \frac{\alpha_0}{\alpha_0 + N} H_0 + \sum_k \frac{n_k}{\alpha_0 + N})$$

Missing Values

The GMM procedure does not perform missing data imputation. If the input data table contains any missing data, PROC GMM simply ignores the observations (rows) that contain missing values. You can use any data preprocessing steps, such as the VARIMPUTE procedure, to impute the missing data before you use the input data table in the GMM procedure.

Random Seed

You can specify the random seed by using the SEED= option in the PROC GMM statement. The seed value is used in the random number generator, which initializes the cluster assignment to start the iterations of the variational Bayesian algorithm. If a random seed is not specified or if the value is invalid (zero or negative), then the GMM procedure generates a random number by using the time of day from the computer’s clock. Because the random seed determines the initial state of the cluster assignment, different seeds can produce different clustering results.
Variational Bayes Method

The GMM procedure 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 with computable alternative distributions. The VB algorithm that is applied in PROC GMM is the mean-field VB, 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:

\[
q = \arg\min_q KL(q \| p) = \arg\min_q \int q \log \left( \frac{q}{p} \right)
\]

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).

Covariance Matrix Type

The VB inference that PROC GMM uses provides a choice of two covariance matrix types for the Gaussian distributions in the Gaussian mixture model: diagonal and full.

The full type of covariance matrix represents a complete form of the Gaussian covariance matrices. The model’s time complexity is quadratic with respect to the number of variables that the model uses.

The diagonal type of covariance matrix represents a diagonal form of the Gaussian covariance matrices. The model’s time complexity is linear with respect to the number of variables that the model uses. In some cases, when the input variables are independent of each other, the diagonal type can yield good results and a faster processing time than the full type.

Score Output

The GMM procedure writes the cluster membership information, including both the predicted cluster scores and the probability distributions over the clusters, to the output CAS table that is specified in the OUT= option in the SCORE statement. This information also includes the variables that are specified in the COPYVARS= option and the following variables:

- `_CLUSTER_1_` to `_CLUSTER_K_` (the probability distribution over all the \( K \) clusters)
- `_PREDICTED_CLUSTER_` (the predicted cluster ID)
ClusterSumOut Output

The GMM procedure presents the summary of the clustering results in the CAS table that is specified in the CLUSTERSUMOUT= option in the PROC GMM statement. In this table, each row contains the following results for a cluster: the mean of the Gaussian distribution of the cluster, the number of observations in the cluster, and the neighbor cluster whose Gaussian mean is closest to the Gaussian mean of the cluster.

This CAS table contains all the variables that the Gaussian mixture model uses to indicate the Gaussian means (the constant variables with zero standard deviation 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_)

ClusterCovOut Output

The GMM procedure presents the covariance matrices of the Gaussian distributions in the CAS table that is specified in the CLUSTERCOVOUT= option in the PROC GMM statement. The covariance matrix of a cluster indicates the shape of the cluster; 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. You can further see the clustering advantage of using the GMM procedure in Example 11.1.

This CAS 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 with the same cluster ID that form the covariance matrix of a cluster.

Displayed Output

The GMM procedure displays various tables for use in cluster 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 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) that contain missing values in the input data table.

Descriptive Statistics

The “Descriptive Statistics” table displays the mean, standard deviation, minimum, and maximum for each interval variable that is specified in the INPUT statement. 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, variational Bayesian (VB) iterations, VB threshold, VB convergence, and number of clusters discovered.

ODS Table Names

Each ODS table that the GMM procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. These names are listed in Table 11.2.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>DescStats</td>
<td>Descriptive statistics for interval input variables</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>PROC GMM</td>
<td>Default output</td>
</tr>
<tr>
<td>NObs</td>
<td>Number of observations</td>
<td>PROC GMM</td>
<td>Default output</td>
</tr>
</tbody>
</table>
Examples: GMM Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

Example 11.1: Performing Cluster Analysis

This example uses the *MouseEars* data table that is generated by the following SAS code to demonstrate how to use the GMM procedure to perform cluster analysis. There are three clusters in the *MouseEars* data: a bigger cluster of the “head,” with a radius of 0.9, and two smaller clusters of “ears,” with a radius of 0.5.

These data are a simple example of clusters of varying shapes, but they are sufficient to demonstrate the advantage of the GMM procedure over the *k*-means clustering algorithm. For this example, because PROC GMM is equipped with the covariances of Gaussian distributions, it performs well in separating the three varying clusters. By contrast, the *k*-means algorithm cuts parts of the head cluster and assigns those parts to the two ear clusters.

The following statements generate the *MouseEars* data table, which has three Gaussian clusters (one face and two ears):

```sas
data mycas.mickey;
  title "Using PROC GMM for Varying Clusters";
  drop n;
  seed=12345;
  do n=1 to 200;
    x=2.5+rannor(seed)*0.9;
    y=rannor(seed)*0.9;
    c=1;
    output;
  end;
  do n=1 to 50;
    x=rannor(seed)*0.5;
    y=2.5+rannor(seed)*0.5;
    c=2;
    output;
  end;
  do n=1 to 50;
    x=5+rannor(seed)*0.5;
    y=2.5+rannor(seed)*0.5;
    c=2;
    output;
  end;
run;
```

These statements assume that your CAS engine libref is named *mycas*, but you can substitute any appropriately defined CAS engine libref.
Example 11.1: Performing Cluster Analysis

The following statements run the GMM procedure and output the results to CAS tables and ODS tables. In this example, COVARIANCE= FULL is specified to capture the possible variations between the clusters. Also, the MAXVBITER= option is specified using a large value and the THRESHOLD= option is specified using a small value to allow a sufficient VB convergence.

```plaintext
proc gmm
  data=mycas.mickey
  seed=12345
  nThreads=32
  maxClusters=100
  alpha=1
  inference=VB (maxVbIter=1000 covariance=FULL threshold=0.001)
  clusterSumOut=mycas.clustersum
  clusterCovOut=mycas.clustercov;
  input x y;
  score out=mycas.score copyvars=(x y c);
  ods select nObs descStats modelInfo;
run;
```

The GMM procedure generates three ODS tables, which are shown in Output 11.1.1 through Output 11.1.3.

**Output 11.1.1**  Number of Observations

**Using PROC GMM for Varying Clusters**

The GMM Procedure

| Number of Observations Read | 300 |
| Number of Observations Used | 300 |

**Output 11.1.2**  Descriptive Statistics

<table>
<thead>
<tr>
<th>Input Variable Statistics</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Mean</td>
<td>Standard Deviation</td>
<td>Min</td>
<td>Max</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>2.460096</td>
<td>1.601650</td>
<td>-0.901323</td>
<td>5.715064</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0.792385</td>
<td>1.459254</td>
<td>-2.635585</td>
<td>3.899748</td>
<td></td>
</tr>
</tbody>
</table>
Output 11.1.3 Model Information

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustering Algorithm</td>
</tr>
<tr>
<td>Inference Method</td>
</tr>
<tr>
<td>Random Number Seed</td>
</tr>
<tr>
<td>Clusters Maximum</td>
</tr>
<tr>
<td>Dirichlet Process Mass</td>
</tr>
<tr>
<td>Covariance Matrix Type</td>
</tr>
<tr>
<td>VB Iterations</td>
</tr>
<tr>
<td>VB Threshold</td>
</tr>
<tr>
<td>VB Converge</td>
</tr>
<tr>
<td>Clusters Discovered</td>
</tr>
</tbody>
</table>

In this example, the GMM procedure generates the cluster score table `mycas.score`; For more information, see the section “Score Output” on page 223. The following PROC SGПLOT statements produce the scatter plot of `mycas.score`. PROC GMM performs well in separating the head cluster from the two ear clusters, as shown in Output 11.1.4. Only cluster scores are shown in Output 11.1.4.

```plaintext
ods graphics / reset width=4in height=4in imagename='Clusters';
proc sgplot data=mycas.score;
    scatter x=x y=y / group=_predicted_cluster_
    markerattrs=(symbol=CircleFilled size=9);
run;
```

Output 11.1.4 Cluster Scores

Using PROC GMM for Varying Clusters

_Predicted Cluster_ 1 2 3
The GMM procedure generates the cluster summary table `mycas.clusterSum`. The following statements print this table, which is shown in Output 11.1.5. For more information, see the section “ClusterSumOut Output” on page 224. In this table, you can see that PROC GMM performs well in assigning the observations to the correct clusters and also finds the correct cluster centers.

```sas
proc print noobs data=mycas.clustersum;
run;
```

**Output 11.1.5** Cluster Summary Table

Using PROC GMM for Varying Clusters

<table>
<thead>
<tr>
<th><em>CLUSTER_ID</em></th>
<th><em>SIZE</em></th>
<th><em>NEIGHBOR</em></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>195</td>
<td>3</td>
<td>2.41892</td>
<td>-0.10389</td>
</tr>
<tr>
<td>2</td>
<td>54</td>
<td>1</td>
<td>4.79783</td>
<td>2.38858</td>
</tr>
<tr>
<td>3</td>
<td>51</td>
<td>1</td>
<td>0.08574</td>
<td>2.42015</td>
</tr>
</tbody>
</table>

The GMM procedure also generates the cluster covariance table `mycas.clusterCov`. The following statements print this table, which is shown in Output 11.1.6. For more information, see the section “ClusterCovOut Output” on page 224. In this table, you can see that PROC GMM successfully captures the variation in the shapes of the clusters.

```sas
proc print noobs data=mycas.clustercov;
run;
```

**Output 11.1.6** Cluster Covariance Table

Using PROC GMM for Varying Clusters

<table>
<thead>
<tr>
<th><em>CLUSTER_ID</em></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.66864</td>
<td>0.00222</td>
</tr>
<tr>
<td>1</td>
<td>0.00222</td>
<td>0.82879</td>
</tr>
<tr>
<td>2</td>
<td>0.82989</td>
<td>0.28631</td>
</tr>
<tr>
<td>2</td>
<td>0.28631</td>
<td>0.48236</td>
</tr>
<tr>
<td>3</td>
<td>0.24677</td>
<td>-0.02387</td>
</tr>
<tr>
<td>3</td>
<td>-0.02387</td>
<td>0.40868</td>
</tr>
</tbody>
</table>

The GMM procedure also generates the analytic store CAS table by using the SAVESTATE statement. The analytic store table saves the Gaussian mixture model for later use after PROC GMM finishes running on the MouseEars data, as shown in Example 11.2.
**Example 11.2: Performing Cluster Analysis and Storing the Model**

This example uses the Fish data set in the Sashelp library to demonstrate how to use the GMM procedure to perform cluster analysis and how to use the analytic store to save the 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.

Because the statistics of fish in different species overlap, the clusters that PROC GMM discovers do not exactly match the fish species in the Species variable. However, PROC GMM provides a good analysis of the inhomogeneity in the data. Besides, this example shows you how to save the Gaussian mixture model in the analytic store so that you can apply this saved model to other data.

The following DATA step divides the mycas.Fish data into training and testing data tables. PROC GMM 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.

```plaintext
data mycas.fish_train;
   title "Using PROC GMM for Cluster Prediction";
   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 defined CAS engine libref.

The following statements run PROC GMM on the training data and save the trained model in the CAS table mycas astore:

```plaintext
proc gmm
data=mycas.fish_train
   nThreads=32
   seed=1234567890
   maxClusters=100
   alpha=1
   inference=VB (maxVbIter=1000 covariance=DIAGONAL threshold=0.001)
   clusterSumOut=mycas.clustersum
   clusterCovOut=mycas.clustercov;
   input _NUMERIC_;
   score out=mycas.score copyvars=(_ALL_);
   ods select nObs descStats modelInfo;
   savestate rstore=mycas astore(replace=yes);
run;
```

The following statements run the ASTORE procedure with the saved model on the testing data, and output the cluster scores in the mycas.newscore CAS table:
The following statements use PROC PRINT to display the predictions about the testing data in the mycas.newscore CAS table, as shown in Output 11.2.1. 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.

```plaintext
proc astore;
    score data=mycas.fish_test out=mycas.newscore copyvars=(_ALL_) rstore=mycas.astore;
run;
```

```plaintext
proc print noobs data=mycas.newscore (obs=9);
run;
```

**Output 11.2.1** Cluster Scores on Testing Data

**Using PROC GMM for Cluster Prediction**

<table>
<thead>
<tr>
<th><em>CLUSTER_1</em></th>
<th><em>CLUSTER_2</em></th>
<th><em>CLUSTER_3</em></th>
<th><em>CLUSTER_4</em></th>
<th><em>CLUSTER_5</em></th>
<th><em>CLUSTER_6</em></th>
<th><em>CLUSTER_7</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.00000</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.00000</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.00000</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.00000</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.00000</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.00000</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.1102E-12</td>
<td>1.00000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><em>CLUSTER_8</em></th>
<th><em>PREDICTED_CLUSTER</em></th>
<th>Species</th>
<th>Weight</th>
<th>Length1</th>
<th>Length2</th>
<th>Length3</th>
<th>Height</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1102E-12</td>
<td>7</td>
<td>Smelt</td>
<td>8.7</td>
<td>10.8</td>
<td>11.3</td>
<td>12.6</td>
<td>1.9782</td>
<td>1.2852</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>7</td>
<td>Smelt</td>
<td>9.8</td>
<td>11.4</td>
<td>12.0</td>
<td>13.2</td>
<td>2.2044</td>
<td>1.1484</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>7</td>
<td>Smelt</td>
<td>12.2</td>
<td>12.1</td>
<td>13.0</td>
<td>13.8</td>
<td>2.2770</td>
<td>1.2558</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>7</td>
<td>Smelt</td>
<td>10.0</td>
<td>11.3</td>
<td>11.8</td>
<td>13.1</td>
<td>2.2139</td>
<td>1.2838</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>7</td>
<td>Smelt</td>
<td>12.2</td>
<td>11.5</td>
<td>12.2</td>
<td>13.4</td>
<td>2.0904</td>
<td>1.3936</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>7</td>
<td>Smelt</td>
<td>19.7</td>
<td>13.2</td>
<td>14.3</td>
<td>15.2</td>
<td>2.8728</td>
<td>2.0672</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>7</td>
<td>Smelt</td>
<td>9.9</td>
<td>11.3</td>
<td>11.8</td>
<td>13.1</td>
<td>2.2139</td>
<td>1.1659</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>7</td>
<td>Smelt</td>
<td>13.4</td>
<td>11.7</td>
<td>12.4</td>
<td>13.5</td>
<td>2.4300</td>
<td>1.2690</td>
</tr>
<tr>
<td>1.1102E-12</td>
<td>7</td>
<td>Smelt</td>
<td>19.9</td>
<td>13.8</td>
<td>15.0</td>
<td>16.2</td>
<td>2.9322</td>
<td>1.8792</td>
</tr>
</tbody>
</table>

**References**


Chapter 12
The GRADBOOST Procedure

Contents
Overview: GRADBOOST Procedure .................................................. 234
   Using CAS Sessions and CAS Engine Librefs ................................ 235
Getting Started: GRADBOOST Procedure ...................................... 236
Syntax: GRADBOOST Procedure .................................................. 240
   PROC GRADBOOST Statement ................................................ 240
   AUTOTUNE Statement ......................................................... 247
   CODE Statement .............................................................. 254
   CROSSVALIDATION Statement ............................................. 255
   ID Statement ................................................................. 255
   INPUT Statement ............................................................ 255
   OUTPUT Statement .......................................................... 256
   PARTITION Statement ....................................................... 257
   SAVESTATE Statement ...................................................... 258
   TARGET Statement .......................................................... 258
   TRANSFERLEARN Statement ................................................ 259
   VIICODE Statement .......................................................... 259
   WEIGHT Statement .......................................................... 260
Details: GRADBOOST Procedure .................................................. 260
   Subsampling the Data ........................................................ 260
   Training a Decision Tree .................................................... 261
   Splitting Nominal Values .................................................... 261
   Imposing Monotonicity Constraints ........................................ 262
   Boosting ........................................................................... 262
   Early Stopping .................................................................... 263
   Measuring Prediction Error ..................................................... 264
   Handling Missing Values ....................................................... 265
      Strategies ....................................................................... 265
      Specifics ......................................................................... 265
   Measuring Variable Importance .............................................. 266
      Residual Sum of Squares Importance Method ...................... 266
      Random Branch Assignment Importance Method ................. 268
   Transfer Learning ............................................................... 269
   Hyperparameter Tuning ........................................................ 269
   \( k \)-fold Cross Validation ..................................................... 270
   Displayed Output ............................................................... 271
      Model Information .......................................................... 271
Overview: GRADBOOST Procedure

The GRADBOOST procedure creates a predictive model called a gradient boosting model in SAS Viya. A gradient boosting model consists of multiple decision trees. A predictive model defines a relationship between input variables and a target variable. The purpose of a predictive model is to predict a target value from inputs. The GRADBOOST procedure creates the model by using training data in which the target values are known. The model can then be applied to observations in which the target is unknown. If the predictions fit the new data well, the model is said to generalize well. Good generalization is the primary goal of predictive tasks. A predictive model might fit the training data well but generalize poorly.

A decision tree is a type of predictive model that has been developed independently in the statistics and artificial intelligence communities. Based on the boosting method in Hastie, Tibshirani, and Friedman (2001) and Friedman (2001), the GRADBOOST procedure creates a predictive model by fitting a set of additive trees.

For more information about training a gradient boosting model, see the section “Boosting” on page 262.
Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```sas
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```sas
   cas mysess;
   libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```sas
   cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Getting Started: GRADBOOST Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

A common use of gradient boosting models is to predict whether a mortgage applicant will default on a loan. The home equity data table `Hmeq`, which is in the `Sampsio` library that SAS provides, contains observations for 5,960 mortgage applicants. A variable named `Bad` indicates whether the applicant, after being approved for a loan, paid off or defaulted on the loan.

This example uses the `Hmeq` data table to build a gradient boosting model that is used to score the data and can be used to score data about new loan applicants. Table 12.1 describes the variables in `Hmeq`.

**Table 12.1 Variables in the Home Equity (Hmeq) Data Table**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Role</th>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad</td>
<td>Response</td>
<td>Binary</td>
<td>1 = applicant defaulted on the loan or is seriously delinquent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = applicant paid off the loan</td>
</tr>
<tr>
<td>CLAge</td>
<td>Predictor</td>
<td>Interval</td>
<td>Age of oldest credit line in months</td>
</tr>
<tr>
<td>CLNo</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of credit lines</td>
</tr>
<tr>
<td>DebtInc</td>
<td>Predictor</td>
<td>Interval</td>
<td>Debt-to-income ratio</td>
</tr>
<tr>
<td>Delinq</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of delinquent credit lines</td>
</tr>
<tr>
<td>Derog</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of major derogatory reports</td>
</tr>
<tr>
<td>Job</td>
<td>Predictor</td>
<td>Nominal</td>
<td>Occupational category</td>
</tr>
<tr>
<td>Loan</td>
<td>Predictor</td>
<td>Interval</td>
<td>Requested loan amount</td>
</tr>
<tr>
<td>MortDue</td>
<td>Predictor</td>
<td>Interval</td>
<td>Amount due on mortgage</td>
</tr>
<tr>
<td>nInq</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of recent credit inquiries</td>
</tr>
<tr>
<td>Reason</td>
<td>Predictor</td>
<td>Binary</td>
<td>'DebtCon' = debt consolidation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'HomeImp' = home improvement</td>
</tr>
<tr>
<td>Value</td>
<td>Predictor</td>
<td>Interval</td>
<td>Value of property</td>
</tr>
<tr>
<td>YoJ</td>
<td>Predictor</td>
<td>Interval</td>
<td>Years at present job</td>
</tr>
</tbody>
</table>

The following statements load the `mycas.hmeq` data into your CAS session. For this example, the statements assume that your CAS engine libref is named `mycas`, but you can substitute any appropriately defined CAS engine libref.

```sas
data mycas.hmeq;
  length Bad Loan MortDue Value 8 Reason Job $7
  YoJ Derog Delinq CLAge nInq CLNo DebtInc 8;
  set sampsio.hmeq;
run;

proc print data=mycas.hmeq(obs=10); run;
```

Output 12.1 shows the first 10 observations of `mycas.hmeq`. 

---

### Output 12.1

The first 10 observations of `mycas.hmeq` show the data for mortgage applicants. Each observation includes information such as `Bad` (default status), `CLAge` (age of oldest credit line), `CLNo` (number of credit lines), and so on. The `Bad` variable indicates whether the applicant defaulted on the loan, with 1 indicating default and 0 indicating no default.
PROC GRADBOOST treats numeric variables as interval inputs unless you specify otherwise. Character variables are always treated as nominal inputs. The following statements run PROC GRADBOOST and save the model in a table named mycas.savedModel:

```
proc gradboost data=mycas.hmeq outmodel=mycas.savedModel seed=12345;
  input Delinq Derog Job nInq Reason / level = nominal;
  input CLAge CLNo DebtInc Loan Mortdue Value YoJ / level = interval;
  target Bad / level = nominal;
  ods output FitStatistics=fitstats;
run;
```

No parameters are specified in the PROC GRADBOOST statement; therefore, the procedure uses all default values. For example, the number of trees in the boosting model is 100, and the number of bins for interval input variables is 20.

The INPUT and TARGET statements are required in order to run PROC GRADBOOST. The INPUT statement indicates which variables to use to build the model, and the TARGET statement indicates which variable the procedure predicts.

Figure 12.2 displays the “Model Information” table. This table shows the values of the training parameters in the first six rows, in addition to some basic information about the trees in the boosting model.
### Figure 12.2 Model Information

#### The GRADBOOST Procedure

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Trees: 100</td>
</tr>
<tr>
<td>Learning Rate: 0.1</td>
</tr>
<tr>
<td>Subsampling Rate: 0.5</td>
</tr>
<tr>
<td>Number of Variables Per Split: 12</td>
</tr>
<tr>
<td>Number of Bins: 50</td>
</tr>
<tr>
<td>Number of Input Variables: 12</td>
</tr>
<tr>
<td>Maximum Number of Tree Nodes: 31</td>
</tr>
<tr>
<td>Minimum Number of Tree Nodes: 15</td>
</tr>
<tr>
<td>Maximum Number of Branches: 2</td>
</tr>
<tr>
<td>Minimum Number of Branches: 2</td>
</tr>
<tr>
<td>Maximum Depth: 4</td>
</tr>
<tr>
<td>Minimum Depth: 4</td>
</tr>
<tr>
<td>Maximum Number of Leaves: 16</td>
</tr>
<tr>
<td>Minimum Number of Leaves: 8</td>
</tr>
<tr>
<td>Maximum Leaf Size: 2728</td>
</tr>
<tr>
<td>Minimum Leaf Size: 5</td>
</tr>
<tr>
<td>Seed: 12345</td>
</tr>
<tr>
<td>Lasso (L1) penalty: 0</td>
</tr>
<tr>
<td>Ridge (L2) penalty: 1</td>
</tr>
<tr>
<td>Actual Number of Trees: 100</td>
</tr>
<tr>
<td>Average Number of Leaves: 14.52</td>
</tr>
</tbody>
</table>

Figure 12.3 displays the “Number of Observations” table, which shows how many observations were read and used. If you specify a PARTITION statement, the “Number of Observations” table also displays the number of observations that were read and used per partition.

#### Figure 12.3 Number of Observations

<table>
<thead>
<tr>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read: 5960</td>
</tr>
<tr>
<td>Number of Observations Used: 5960</td>
</tr>
</tbody>
</table>

Figure 12.4 displays the estimates of variable importance. The rows in this figure are sorted by the importance measure. A conclusion from fitting the boosting model to these data is that Debt/Inc is the most important predictor of loan default.
**Figure 12.4** Variable Importance

<table>
<thead>
<tr>
<th>Variable</th>
<th>Importance</th>
<th>Std Dev</th>
<th>Relative Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>DebtInc</td>
<td>27.7997</td>
<td>69.9384</td>
<td>1.0000</td>
</tr>
<tr>
<td>Delinq</td>
<td>6.0413</td>
<td>4.5083</td>
<td>0.2173</td>
</tr>
<tr>
<td>Value</td>
<td>5.3591</td>
<td>3.5879</td>
<td>0.1928</td>
</tr>
<tr>
<td>CLAge</td>
<td>4.8846</td>
<td>3.2411</td>
<td>0.1757</td>
</tr>
<tr>
<td>nInq</td>
<td>4.0248</td>
<td>1.8294</td>
<td>0.1448</td>
</tr>
<tr>
<td>Derog</td>
<td>3.9666</td>
<td>3.4149</td>
<td>0.1427</td>
</tr>
<tr>
<td>CLNo</td>
<td>3.3469</td>
<td>2.0543</td>
<td>0.1204</td>
</tr>
<tr>
<td>Job</td>
<td>2.9932</td>
<td>1.9706</td>
<td>0.1077</td>
</tr>
<tr>
<td>MortDue</td>
<td>2.8410</td>
<td>1.5882</td>
<td>0.1022</td>
</tr>
<tr>
<td>YoJ</td>
<td>2.8289</td>
<td>1.8376</td>
<td>0.1018</td>
</tr>
<tr>
<td>Loan</td>
<td>2.4465</td>
<td>1.8437</td>
<td>0.0880</td>
</tr>
<tr>
<td>Reason</td>
<td>0.4069</td>
<td>1.0537</td>
<td>0.0146</td>
</tr>
</tbody>
</table>

**Figure 12.5** Fit Statistics

<table>
<thead>
<tr>
<th>Number of Trees</th>
<th>Training Average Square Error</th>
<th>Training Misclassification Rate</th>
<th>Training Log Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1452</td>
<td>0.1995</td>
<td>0.458</td>
</tr>
<tr>
<td>2</td>
<td>0.1343</td>
<td>0.1995</td>
<td>0.428</td>
</tr>
<tr>
<td>3</td>
<td>0.1251</td>
<td>0.1995</td>
<td>0.405</td>
</tr>
<tr>
<td>4</td>
<td>0.1175</td>
<td>0.1995</td>
<td>0.385</td>
</tr>
<tr>
<td>5</td>
<td>0.1111</td>
<td>0.1961</td>
<td>0.369</td>
</tr>
<tr>
<td>6</td>
<td>0.1060</td>
<td>0.1601</td>
<td>0.356</td>
</tr>
<tr>
<td>7</td>
<td>0.1012</td>
<td>0.1438</td>
<td>0.343</td>
</tr>
<tr>
<td>8</td>
<td>0.0973</td>
<td>0.1299</td>
<td>0.333</td>
</tr>
<tr>
<td>9</td>
<td>0.0942</td>
<td>0.1203</td>
<td>0.324</td>
</tr>
<tr>
<td>10</td>
<td>0.0914</td>
<td>0.1124</td>
<td>0.316</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>0.0469</td>
<td>0.0639</td>
<td>0.168</td>
</tr>
<tr>
<td>92</td>
<td>0.0466</td>
<td>0.0626</td>
<td>0.167</td>
</tr>
<tr>
<td>93</td>
<td>0.0461</td>
<td>0.0614</td>
<td>0.165</td>
</tr>
<tr>
<td>94</td>
<td>0.0458</td>
<td>0.0612</td>
<td>0.164</td>
</tr>
<tr>
<td>95</td>
<td>0.0456</td>
<td>0.0611</td>
<td>0.163</td>
</tr>
<tr>
<td>96</td>
<td>0.0454</td>
<td>0.0616</td>
<td>0.163</td>
</tr>
<tr>
<td>97</td>
<td>0.0452</td>
<td>0.0599</td>
<td>0.162</td>
</tr>
<tr>
<td>98</td>
<td>0.0449</td>
<td>0.0599</td>
<td>0.161</td>
</tr>
<tr>
<td>99</td>
<td>0.0446</td>
<td>0.0597</td>
<td>0.160</td>
</tr>
<tr>
<td>100</td>
<td>0.0444</td>
<td>0.0592</td>
<td>0.160</td>
</tr>
</tbody>
</table>

Figure 12.5 shows the first 10 and last 10 observations of the fit statistics. PROC GRADBOOST computes fit statistics on a per-tree basis. As the number of trees increases, the fit statistics usually improve (decrease) at first and then level off and fluctuate within a small range.
Syntax: GRADBOOST Procedure

The following statements are available in the GRADBOOST procedure:

```plaintext
PROC GRADBOOST <options> ;
  AUTOTUNE <options> ;
  CODE <options> ;
  CROSSVALIDATION <options> ;
  ID variables ;
  INPUT variables < /options> ;
  OUTPUT OUT=CAS-libref.data-table <option> ;
  PARTITION partition-option ;
  SAVESTATE RSTORE=CAS-libref.data-table ;
  TARGET variable </LEVEL=NOMINAL | INTERVAL> ;
  TRANSFERLEARN variable < /options> ;
  VIICODE <options> ;
  WEIGHT variable ;
```

The PROC GRADBOOST, INPUT, and TARGET statements are required. The INPUT statement can appear multiple times.

The rest of this section provides detailed syntax information about each of the preceding statements, beginning with the PROC GRADBOOST statement. The remaining statements are described in alphabetical order.

PROC GRADBOOST Statement

```plaintext
PROC GRADBOOST <options> ;
```

The PROC GRADBOOST statement invokes the procedure. Table 12.2 summarizes the options in the PROC GRADBOOST statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic Options</strong></td>
<td></td>
</tr>
<tr>
<td>ADDTREES</td>
<td>Adds trees to an already generated boosting model, which is specified in the INMODEL= option</td>
</tr>
<tr>
<td>BINMETHOD=</td>
<td>Specifies how to bin interval inputs prior to training</td>
</tr>
<tr>
<td>DISTRIBUTION=</td>
<td>Specifies the distribution of the objective function</td>
</tr>
<tr>
<td>EARLYSTOP=</td>
<td>Specifies how to perform early stopping during model training</td>
</tr>
<tr>
<td>INMODEL=</td>
<td>Specifies a saved gradient boosting model to use to score new a table</td>
</tr>
<tr>
<td>LASSO=</td>
<td>Specifies the L1 norm regularization parameter</td>
</tr>
<tr>
<td>LEARNINGRATE=</td>
<td>Specifies the learning rate for each tree</td>
</tr>
<tr>
<td>NOPRINT</td>
<td>Suppresses ODS output</td>
</tr>
<tr>
<td>NTHREADS=</td>
<td>Specifies the number of threads to use in the computation</td>
</tr>
<tr>
<td>NTREES=</td>
<td>Specifies the number of trees to grow in the boosting model</td>
</tr>
</tbody>
</table>
### Table 12.2

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBIN=</td>
<td>Specifies the number of bins for continuous variables</td>
</tr>
<tr>
<td>OFFSET=</td>
<td>Specifies an offset variable to use with DISTRIBUTION=POISSON or TWEEDIE</td>
</tr>
<tr>
<td>OUTMODEL=</td>
<td>Specifies the data table to which the gradient boosting model is to be saved</td>
</tr>
<tr>
<td>PRINTTARGET</td>
<td>Creates tables that contain information about columns in the output</td>
</tr>
<tr>
<td>RBAIMP</td>
<td>Creates a variable importance table by using random branch assignment</td>
</tr>
<tr>
<td>RIDGE=</td>
<td>Specifies the L2 norm regularization parameter</td>
</tr>
<tr>
<td>SAMPLINGRATE=</td>
<td>Specifies the fraction of the training data to use for growing each tree</td>
</tr>
<tr>
<td>SEED=</td>
<td>Specifies the random number seed to use for model building</td>
</tr>
<tr>
<td>VARS_TO_TRY=</td>
<td>Specifies the number of variables to examine at each node split</td>
</tr>
<tr>
<td>VII=</td>
<td>Calculates the importance of the specified types of variable interactions</td>
</tr>
</tbody>
</table>

#### Splitting Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGNMISSING=</td>
<td>Specifies how to handle missing values in a predictor variable</td>
</tr>
<tr>
<td>MAXBRANCH=</td>
<td>Specifies the maximum number of splits per node</td>
</tr>
<tr>
<td>MAXDEPTH=</td>
<td>Specifies the maximum tree depth</td>
</tr>
<tr>
<td>MINLEAFSIZE=</td>
<td>Specifies the minimum number of observations per leaf</td>
</tr>
<tr>
<td>MINUSEINSEARCH=</td>
<td>Specifies the minimum number of observations to use with the USEINSEARCH policy for handling missing values</td>
</tr>
<tr>
<td>NOMSEARCH=</td>
<td>Specifies the method to use to find a split on a nominal variable</td>
</tr>
</tbody>
</table>

You also specify the following options:

**ADDTREES**

**ADDTREE**

adds trees to an already generated gradient boosting model, which is specified in the INMODEL= option. The resulting model will consist of all the trees in the original model, plus a number of new trees, where the number of new trees is specified in the NTREES= option. When you update a model, you can use the original training data or you can use new data. If you use new data, then the new data must have all the variables that the original training data have.

**ASSIGNMISSING=NONE | MACSMALL | USEINSEARCH**

specifies how to handle missing values during training and creates a splitting rule to handle missing values and unknown levels during scoring. An unknown level is a level of a categorical predictor variable that does not exist in the training data but is encountered during scoring.

During model training, PROC GRADBOOST searches for the best splitting rule for each node, as described in the section “Training a Decision Tree” on page 261. During model scoring, observations are assigned to a node in a tree based on the best splitting rule if that rule’s variable is not missing. If the variable is missing for the observation, then the default splitting that is specified by this option is used. The default splitting rule enables all data to be scored, even if the best splitting rule cannot be used on a particular observation.
You can specify one of the following values:

**NONE**

during training, excludes observations that have any missing variables from the model. In the scoring phase, this default rule assigns observations that have missing values of an interval predictor variable to the branch with the smallest predictor values, and assigns observations that have unknown and missing nominal levels to the branch with the most training observations.

**MACSMALL**

during training, treats a missing value as a separate, legitimate value in the search for a split for the primary splitting rule. Missing values in interval predictor variables are treated as less than any other number. In the scoring phase, this default rule assigns observations that have missing values of interval predictor variables to the leftmost branch of the split, and assigns observations that have unknown nominal levels to the largest branch in the split.

**USEINSEARCH**

during training, treats a missing value as a separate, legitimate value in the search for a split for the primary splitting rule. Missing values in interval predictor variables are treated as a special level that is used during the split process. In the scoring phase, this default rule assigns observations that have missing values of interval predictor variables to the branch determined during the model growing, and assigns observations that have unknown nominal levels to the largest branch in the split.

By default, ASSIGNMISSING=USEINSEARCH.

**BINMETHOD=BUCKET | QUANTILE**

specifies how to bin interval input variables prior to growing the gradient boosting model. The number of bins that are created is determined by the NUMBIN= option.

You can specify one of the following values:

**BUCKET**

bins interval input variables into fixed-width bins. The width of each bin for a particular variable is calculated by subtracting the smallest value among all observations from the largest value among all observations, and then dividing that result by the number of bins.

**QUANTILE**

bins interval input variables into bins according to their quantile. The width of the bins for a particular variable is not fixed, but the number of observations in each bin is approximately equal.

By default, BINMETHOD=QUANTILE.

**DATA=CAS-libref.data-table**

names the input data table for PROC GRADBOOST to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

- **CAS-libref** refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 235.

- **data-table** specifies the name of the input data table.
DISTRIBUTION=BINARY | GAUSSIAN | MULTINOMIAL | POISSON | TWEEDIE(POWER=p)
specifies the distribution of the objective function. The default distribution is GAUSSIAN, BINARY, or MULTINOMIAL for an interval, binary, or nominal target, respectively. The POISSON distribution is appropriate for count data. The TWEEDIE distribution is useful for modeling total losses in insurance. The default value of POWER=p is 1.5, and $1 \leq p \leq 2$.

EARLYSTOP(suboption ...)
specifies options for stopping the gradient boosting model training early. For more information, see the section “Early Stopping” on page 263. Two approaches to early stopping are available: stagnation and threshold. You can specify one or more of the following suboptions:

METRIC=MCR | LOGLOSS
specifies whether to use the misclassification rate or the log-loss prediction error in early stopping for a binary or nominal target.

By default, METRIC=MCR. Average square error is always used for an interval target.

MINIMUM=NO | YES
specifies whether the stagnation approach should count iterations starting from the iteration that has the smallest error.

By default, MINIMUM=NO.

STAGNATION=number
specifies the number of iterations in the gradient boosting model to consider for early stopping, where number must be a nonnegative integer. If STAGNATION=0, then the stagnation approach is not used. If you want to use autotuning by specifying the AUTOTUNE statement, but you do not want autotuning to use early stopping, then specify STAGNATION=0.

By default, STAGNATION=0. When you specify the AUTOTUNE statement, STAGNATION=4 by default.

THRESHOLD=\tau
specifies the threshold value that stops training when the error equals or exceeds it.

By default, THRESHOLD=0 and the threshold approach to early stopping is not used.

THRESHOLDITER=T
specifies the minimum number of training iterations to run before the threshold approach is invoked.

TOLERANCE=number
specifies the number to be used as the tolerance for the stagnation approach to early stopping, where number must be nonnegative.

By default, TOLERANCE=0.

INMODEL=CAS-libref.data-table
specifies the data table that you previously saved as a gradient boosting model by using the OUTMODEL= option in a previous run of PROC GRADBOOST. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 235.
You can specify the INMODEL= option for the following purposes:

- To score new data. In this case, both the DATA= option and the OUTPUT statement are required, and any other options, except for NOPRINT, are ignored.
- To add trees to an existing gradient boosting model. In this case, the GRADBOOST procedure uses the originally saved model and adds new trees to it by using the same or new data. If new data are used, then they must contain all the variables of the original data. When you use the INMODEL= option to add trees to an existing gradient boosting model, you must also specify the ADDTREE option. In this case, all other training options are also honored.

The data table used in the INMODEL= option must include the attributes that are associated with the table produced by the OUTMODEL= option, or the GRADBOOST procedure will error.

\[
\text{LASSO} = \text{number}
\]

\[
\text{L1} = \text{number}
\]

specifies the L1 norm regularization parameter, where \text{number} must be nonnegative.

By default, LASSO=0. This value can be tuned with the AUTOTUNE statement.

\[
\text{LEARNINGRATE} = \text{number}
\]

specifies the learning rate for the gradient boosting algorithm, where \text{number} must be between 0 and 1, inclusive.

By default, LEARNINGRATE=0.1. This value can be tuned with the AUTOTUNE statement.

\[
\text{MAXBRANCH} = b
\]

specifies the maximum number of children per node in the tree. PROC GRADBOOST tries to create this number of children unless it is impossible (for example, if a split variable does not have enough levels).

By default, MAXBRANCH=2.

\[
\text{MAXDEPTH} = \text{number}
\]

specifies the maximum depth of the tree to be grown. The number of levels in a tree is equal to the depth plus one.

By default, MAXDEPTH=4.

\[
\text{MINLEAFSIZE} = \text{number}
\]

specifies the minimum \text{number} of observations that each child of a split must contain in the training data table in order for the split to be considered. The count of observations includes those that are excluded from training as a consequence of specifying a value less than one in the SAMPLINGRATE= option.

By default, MINLEAFSIZE=5. This value can be tuned with the AUTOTUNE statement.

\[
\text{MINUSEINSEARCH} = \text{number}
\]

specifies a threshold for using missing values in the split search when ASSIGNMISSING=USEINSEARCH. If the number of observations in which the splitting variable has missing values is greater than or equal to \text{number}, then PROC GRADBOOST uses the USEINSEARCH policy to handle missing values for that variable.

By default, MINUSEINSEARCH=1.
**NOMSEARCH**(*suboption ...*)

specifies search methods for splitting on a nominal variable. For more information, see the section “Splitting Nominal Values” on page 261. If you do not specify this option, the default values of the *suboptions* determine the search method. You can specify one or more of the following *suboptions*:

- **CLUSTER=** *number*
  specifies the minimum cardinality in the node of the nominal variable for using the cluster method. The cluster method finds a split by \( k \)-means clustering of the categories.

  By default, CLUSTER=100.

- **SHRINKAGE=** *number*
  specifies how much weight to give the average gradient when you combine it with the average gradient within a category. You use this option only when sorting the categories. The value of *number* must be nonnegative.

  By default, SHRINKAGE=10.

- **SORT=** *number*
  specifies the minimum cardinality in the node of the nominal variable for using the sort method. The sort method sorts the categories and then examines binary splits that preserve the ordering. The value of *number* must be a nonnegative integer. If *number* is greater than or equal to the value of CLUSTER=*number* option, then the sort method is not used.

  By default, SORT=10.

**NOPRINT**

suppresses ODS output.

**NTHREADS=** *number-of-threads*

specifies the number of threads to use in the computation. The default value is the number of CPUs available on the machine.

**NTREES=** *number*

specifies the *number* of trees to grow in the gradient boosting model.

By default, NTREES=100. This value can be tuned with the AUTOTUNE statement.

**NUMBIN=** *number*

specifies the *number* of bins in which to bin the interval input variables. PROC GRADBOOST bins continuous predictors to a fixed bin size. This option controls the number of bins and thereby also the size of the bins.

By default, NUMBIN=50.

**OFFSET=** *variable*

specifies an offset variable to be used with DISTRIBUTION=POISSON or TWEEDIE.
OUTMODEL=\texttt{CAS-libref.}\texttt{.data-table}
specifies the data table to which to save the gradient boosting model. \texttt{CAS-libref.}\texttt{data-table} is a two-level name, where \texttt{CAS-libref} refers to the caslib and session identifier, and \texttt{data-table} specifies the name of the output data table. For more information about this two-level name, see the \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 235.

The data table that results from this option contains information about each node and each tree in the gradient boosting model, including the splitting variables, the child nodes, the number of observations at each node, and the predicted response at each node. This table also has attributes attached that contain information about the input variables, the target variable, and the gradient boosting model.

PRINTTARGET
outputs tables that indicate generated columns in the \texttt{OUT=} table from the \texttt{OUTPUT} statement. For a continuous response, PROC GRADBOOST generates an output table named \texttt{PredName}, which indicates the name of the predicted value column. For a categorical response, PROC GRADBOOST generates an output table named \texttt{PredIntoName}, which indicates the name of the predicted value column, and also an output table named \texttt{PredProbName}, which indicates the names of the predicted probability columns.

RBAIMP
creates a variable importance table by using random branch assignment (RBA). This table is created in addition to the normal variable importance table that is calculated using the residual sum of squares (RSS) error. For more information about RBA and RSS variable importance, see the section “Measuring Variable Importance” on page 266.

\texttt{RIDGE=}\texttt{number}
\texttt{L2=}\texttt{number}
specifies the L2 norm regularization parameter on prediction. The value of \texttt{number} must be nonnegative.

By default, \texttt{RIDGE}=1. This value can be tuned with the \texttt{AUTOTUNE} statement.

\texttt{SAMPLINGRATE=}\texttt{number}
specifies the fraction of the training data to be used for growing each tree in the boosting model.

By default, \texttt{SAMPLINGRATE}=0.5. This value can be tuned with the \texttt{AUTOTUNE} statement.

\texttt{SEED=}\texttt{number}
specifies the initial seed for random number generation for model building. The value of \texttt{number} must be an integer. If you do not specify a seed or you specify a value less than or equal to 0, the seed is generated by reading the time of day from the computer’s clock.

\texttt{VARS\_TO\_TRY=}\texttt{m}
\texttt{M=}\texttt{m}
specifies the number of input variables to consider splitting on in a node, where \texttt{m} ranges from 1 to the number of input variables.

By default, \texttt{m} is the number of input variables. This value can be tuned with the \texttt{AUTOTUNE} statement.
calculates the variable interaction importance, which is described in the section “Interaction Importance” on page 267. You can specify the following values:

- **2** calculates the importance of all two-way variable interactions.
- **3** calculates the importance of all three-way and all two-way variable interactions.

If you do not specify this option, then the variable interaction importance is not calculated.

---

**AUTOTUNE Statement**

\[
\text{AUTOTUNE } <\text{ options}> ;
\]

The AUTOTUNE statement searches for the best combination of values of the `LASSO=`, `LEARNINGRATE=`, `MAXDEPTH=`, `MINLEAFSIZE=`, `NTREES=`, `NUMBIN=`, `RIDGE=`, `SAMPLINGRATE=`, and `VARS_TO_TRY=` options in the PROC GRADBOOST statement. You cannot specify both the AUTOTUNE statement and the CROSSVALIDATION statement in the same procedure run.

Table 12.3 summarizes the **options** that you can specify in the AUTOTUNE statement. For more information about all options except the TUNINGPARAMETERS= option, see the option’s description in the section “AUTOTUNE Statement” on page 12 in Chapter 3, “Shared Concepts.” The TUNINGPARAMETERS= option is described following Table 12.3.

**Table 12.3 AUTOTUNE Statement Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>APPENDLOOKUP</code></td>
<td>Specifies that the table specified in the HISTORYTABLE= option contain the rows from the table specified in the LOOKUPTABLE= option</td>
</tr>
<tr>
<td><code>EVALHISTORY=</code></td>
<td>Specifies how to report the evaluation history of the tuner</td>
</tr>
<tr>
<td><code>FRACTION=</code></td>
<td>Specifies the fraction of observations to use for validation</td>
</tr>
<tr>
<td><code>HISTORYTABLE=</code></td>
<td>Specifies the CAS table that contains the evaluation history</td>
</tr>
<tr>
<td><code>KFOLD=</code></td>
<td>Specifies the number of folds for k-fold cross validation</td>
</tr>
<tr>
<td><code>LIVEUPDATE</code></td>
<td>Specifies that the table specified in the HISTORYTABLE= option be updated at every evaluation</td>
</tr>
<tr>
<td><code>LOCALSEARCH</code></td>
<td>Enables local search optimization</td>
</tr>
<tr>
<td><code>LOOKUPTABLE=</code></td>
<td>Specifies the CAS table to use for evaluation lookup</td>
</tr>
<tr>
<td><code>MAXBAYES=</code></td>
<td>Specifies the maximum number of points in the kriging model</td>
</tr>
<tr>
<td><code>MAXEVALS=</code></td>
<td>Specifies the maximum number of evaluations</td>
</tr>
<tr>
<td><code>MAXITER=</code></td>
<td>Specifies the maximum number of iterations when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td><code>MAXTIME=</code></td>
<td>Specifies the maximum time for all iterations</td>
</tr>
<tr>
<td><code>MAXTRAINTIME=</code></td>
<td>Specifies the maximum time for a model training</td>
</tr>
<tr>
<td><code>NCONVITER=</code></td>
<td>Specifies the number of convergence iterations</td>
</tr>
<tr>
<td><code>NOGRIDSHUFFLE</code></td>
<td>Requests that the grid points not be shuffled</td>
</tr>
</tbody>
</table>
Table 12.3  continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPARALLEL=</td>
<td>Specifies the number of parallel sessions</td>
</tr>
<tr>
<td>NSUBSESSIONWORKERS=</td>
<td>Specifies the number of workers in parallel sessions</td>
</tr>
<tr>
<td>OBJECTIVE=</td>
<td>Specifies the objective function</td>
</tr>
<tr>
<td>POPSIZE=</td>
<td>Specifies the population size when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>SAMPLESIZE=</td>
<td>Specifies the sample size when SEARCHMETHOD=LHS or SEARCHMETHOD=RANDOM</td>
</tr>
<tr>
<td>SEARCHMETHOD=</td>
<td>Specifies the search method that the optimizer uses</td>
</tr>
<tr>
<td>SECONDOBJECTIVE=</td>
<td>Specifies the second objective to use for tuning</td>
</tr>
<tr>
<td>SELECTINITPOINT</td>
<td>Specifies that the tuner select the best evaluation from the lookup table</td>
</tr>
<tr>
<td>TARGETEVENT=</td>
<td>Specifies the target event for ROC-based calculations</td>
</tr>
<tr>
<td>TRAIFRACTION=</td>
<td>Specifies the fraction of observations to use for training</td>
</tr>
<tr>
<td>TUNINGPARAMETERS=</td>
<td>Specifies the custom tuning parameters</td>
</tr>
<tr>
<td>USEPARAMETERS=</td>
<td>Specifies how to handle the TUNINGPARAMETERS= option</td>
</tr>
</tbody>
</table>

TUNINGPARAMETERS=(suboption | ... | < suboption >)

TUNEPARMS=(suboption | ... | < suboption >)

specifies which parameters to tune and which ranges to tune over. If USEPARAMETERS=STANDARD, this option is ignored.

You can specify one or more of the following suboptions:

**LASSO (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**

specifies information about the L1 regularization to use for tuning the gradient boosting model. For more information, see the LASSO= option in the PROC GRADBOOST statement.

You can specify the following additional suboptions:

**LB=number**

specifies the minimum L1 regularization value to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=0.

**UB=number**

specifies the maximum L1 regularization value to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=10.

**VALUES=value-list**

specifies a list of L1 regularization values to consider during tuning, where value-list is a space-separated list of numbers greater than or equal to 0. If you specify this suboption, you cannot specify either the LB= or UB= suboption.
**INIT=number**

specifies the initial L1 regularization value for the tuner to use.

By default, INIT=0.

**EXCLUDE**

excludes L1 regularization from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**LEARNINGRATE (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**

specifies information about the learning rate to use for tuning the gradient boosting model. For more information, see the LEARNINGRATE= option in the PROC GRADBOOST statement.

You can specify the following additional suboptions:

**LB=number**

specifies the minimum learning rate to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=0.01.

**UB=number**

specifies the maximum learning rate to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=1.

**VALUES=value-list**

specifies a list of learning rates to consider during tuning, where value-list is a space-separated list of numbers greater than 0 and less than or equal to 1. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=number**

specifies the initial learning rate for the tuner to use.

By default, INIT=0.1.

**EXCLUDE**

excludes the learning rate from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**MAXDEPTH (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**

specifies information about the maximum depth of trees to use for tuning the gradient boosting model. For more information, see the MAXDEPTH= option in the PROC GRADBOOST statement.

You can specify the following additional suboptions:
Chapter 12: The GRADBOOST Procedure

**LB=number**
specifies the minimum value of the MAXDEPTH= option in the PROC GRADBOOST statement to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=1.

**UB=number**
specifies the maximum value of the MAXDEPTH= option in the PROC GRADBOOST statement to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=6.

**VALUES=value-list**
specifies a list of values of the MAXDEPTH= option in the PROC GRADBOOST statement to consider during tuning, where value-list is a space-separated list of positive integers. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=number**
specifies the initial value of the MAXDEPTH= option in the PROC GRADBOOST statement for the tuner to use.

By default, INIT=4.

**MINLEAFSIZE (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**
specifies information about tuning the leaf size option for training the decision tree. This option is not tuned by default. If it is specified without any suboptions, then tuning uses the following values: 1, 5, 10, 20, 40, 80, 160, 320. For more information, see the MINLEAFSIZE= option in the PROC GRADBOOST statement.

You can specify the following additional suboptions:

**LB=number**
specifies the minimum leaf size value to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=1.

**UB=number**
specifies the maximum leaf size value to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=320.

**VALUES=value-list**
specifies a list of leaf size values to consider during tuning, where value-list is a space-separated list of positive integers. If you specify this suboption, you cannot specify either the LB= or UB= suboption.
**INIT=number**

specifies the initial leaf size for the tuner to use.

By default, INIT=5.

**EXCLUDE**

excludes the leaf size from the tuning process.

**EXCLUDE**

excludes the MAXDEPTH= option from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**NTREES (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**

specifies information about the number of trees to use for tuning the gradient boosting model. For more information, see the NTREES= option in the PROC GRADBOOST statement.

You can specify the following additional suboptions:

**LB=number**

specifies the minimum number of trees to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=20.

**UB=number**

specifies the maximum number of trees to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=150.

**VALUES=value-list**

specifies a list of numbers of trees to consider during tuning, where value-list is a space-separated list of positive integers. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=number**

specifies the initial number of trees for the tuner to use.

By default, INIT=100.

**EXCLUDE**

excludes the number of trees from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**NUMBIN (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**

specifies information about the number of bins to consider during the tuning of the options for PROC GRADBOOST. For more information, see the NUMBIN= option in the PROC GRADBOOST statement.

You can specify the following additional suboptions:
**LB=number**
specifies the minimum number of bins to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=20.

**UB=number**
specifies the maximum number of bins to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=100.

**VALUES=value-list**
specifies a list of the numbers of bins to consider during tuning, where value-list is a space-separated list of positive integers. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=number**
specifies the initial number of bins for the tuner to use.

By default, INIT=50.

**EXCLUDE**
excludes the number of bins from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**RIDGE (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**
specifies information about the L2 regularization to use for tuning the gradient boosting model. For more information, see the RIDGE= option in the PROC GRADBOOST statement.

You can specify the following additional suboptions:

**LB=number**
specifies the minimum L2 regularization value to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=0.

**UB=number**
specifies the maximum L2 regularization value to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=10.

**VALUES=value-list**
specifies a list of L2 regularization values to consider during tuning, where value-list is a space-separated list of numbers greater than or equal to 0. If you specify this suboption, you cannot specify either the LB= or UB= suboption.
**INIT=**number

specifies the initial L2 regularization value for the tuner to use.

By default, INIT=0.

**EXCLUDE**

excludes L2 regularization from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**SAMPLINGRATE (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**

specifies information about the portion of the training data for each boosted tree to use for tuning the gradient boosting model. For more information, see the SAMPLINGRATE= option in the PROC GRADBOOST statement.

You can specify the following additional suboptions:

**LB=**number

specifies the minimum sampling rate to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=0.1.

**UB=**number

specifies the maximum sampling rate to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=1.

**VALUES=value-list**

specifies a list of sampling rates to consider during tuning, where value-list is a space-separated list of numbers greater than 0 and less than or equal to 1. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=**number

specifies the initial sampling rate for the tuner to use.

By default, INIT=0.5.

**EXCLUDE**

excludes the sampling rate from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**VARS_TO_TRY (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**

specifies information about the number of variables to consider at each split during tree growth. For more information, see the VARS_TO_TRY= option in the PROC GRADBOOST statement.

You can specify the following additional suboptions:
**LB=number**  
specifies the minimum number of variables to consider during tuning. If you specify this  
suboption, you cannot specify the VALUES= suboption.  

By default, LB=1.

**UB=number**  
specifies the maximum number of variables to consider during tuning. If you specify this  
suboption, you cannot specify the VALUES= suboption.  

The default is the number of input variables.

**VALUES=value-list**  
specifies a list of numbers of variables to consider during tuning, where value-list is a  
space-separated list of positive integers. If you specify this suboption, you cannot specify  
either the LB= or UB= suboption.

**INIT=number**  
specifies the initial number of variables for the tuner to use.  

The default is the total number of input variables.

**EXCLUDE**  
excludes the number of variables from the tuning process. If you specify this suboption, any  
specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

---

**CODE Statement**

**CODE < options > ;**

The CODE statement writes SAS DATA step code for computing predicted values of the fitted model to a  
file, to a catalog entry, or to a CAS table. To score new data, you can then include the file or the catalog entry  
in a DATA step, or you can specify the CAS table in the runCodeTable action in the dataStep action set  
(for more information, see SAS Viya: System Programming Guide).

Table 12.4 summarizes the options available in the CODE statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENT</td>
<td>Adds comments to the generated code</td>
</tr>
<tr>
<td>FILE=</td>
<td>Names the file in which to save the generated code</td>
</tr>
<tr>
<td>FORMATWIDTH=</td>
<td>Specifies the numeric format width for the regression coefficients</td>
</tr>
<tr>
<td>INDENTSIZE=</td>
<td>Specifies the number of spaces to indent the generated code</td>
</tr>
<tr>
<td>LABELID=</td>
<td>Specifies a number used to construct names and labels</td>
</tr>
<tr>
<td>LINESIZE=</td>
<td>Specifies the line size for the generated code</td>
</tr>
<tr>
<td>NOTRIM</td>
<td>Compares formatted values, including blank padding</td>
</tr>
<tr>
<td>OUT=</td>
<td>Names an output CAS table in which to save the generated code</td>
</tr>
</tbody>
</table>
CROSSVALIDATION Statement

CROSSVALIDATION < options > ;

The CROSSVALIDATION statement performs \( k \)-fold cross validation to find the average estimated validation error. You cannot specify this statement if you specify either the AUTOTUNE statement or the PARTITION statement.

You can specify the following options:

**KFOLD=number**
- specifies the \( number \) of partition folds in the cross validation process, where \( number \) must be between 2 and 20, inclusive.
- By default, KFOLD=5.

**NOPARALLEL**
- requests that \( k \)-fold cross validation not be run in parallel. By default, the process runs in parallel.

**NSUBSESSIONWORKERS=number**
- specifies the \( number \) of worker nodes to use in parallel subsessions.
- By default, the value of the NSUBSESSIONWORKERS= option is determined automatically.

ID Statement

ID variables ;

The ID statement lists one or more variables that are to be copied from the input data table to the output data tables that are specified in the OUT= option in the OUTPUT statement and the RSTORE= option in the SAVESTATE statement.

INPUT Statement

INPUT variables </ options> ;

The INPUT statement names input variables that share common options. You can specify the INPUT statement multiple times.

You can specify the following options:
LEVEL=INTERVAL | NOMINAL
specifies the level of measurement of the variables. You can specify the following values:

INTERVAL specifies that the level of measurement of the variables is interval.
NOMINAL specifies that the level of measurement of the variables is nominal.

By default, LEVEL=INTERVAL for numeric variables and LEVEL=NOMINAL for categorical variables.

MONOTONIC=DECREASING | INCREASING
requests that the gradient boosting model be built by requiring monotonic constraints on the model with respect to the variables. You can specify the following values:

DECREASING requests that decreasing monotonic constraints be placed on the model with respect to the variables.
INCREASING requests that increasing monotonic constraints be placed on the model with respect to the variables.

For more information, see the section “Imposing Monotonicity Constraints” on page 262.
By default, no monotonic constraints are placed on model with respect to the variables.

OUTPUT Statement

OUTPUT OUT=CAS-libref.data-table < option > ;

The OUTPUT statement creates an output data table that contains the results of running PROC GRADBOOST.
You must specify the following option:

OUT=CAS-libref.data-table
names the output data table for PROC GRADBOOST to use. You must specify this option before any other options. CAS-libref.data-table is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about CAS-libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 235.
data-table specifies the name of the output data table.

You can also specify the following option:
COPYVAR=variable
COPYVARS=(variables)

lists one or more variables from the input data table to be transferred to the output data table.

ROLE=<name>
generates a numeric variable that indicates the role played by each observation in fitting the model. By default, the variable is named _ROLE_. You can add an optional =name to change the name of this generated variable. For each observation, the interpretation of this variable is shown in Table 12.5.

Table 12.5 Role Interpretation

<table>
<thead>
<tr>
<th>Value</th>
<th>Observation Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>1</td>
<td>Training</td>
</tr>
<tr>
<td>2</td>
<td>Validation</td>
</tr>
<tr>
<td>3</td>
<td>Testing</td>
</tr>
</tbody>
</table>

If you do not partition the input data by using a PARTITION statement, then the value of the role variable is 1 for all observations.

PARTITION Statement

PARTITION partition-option ;

The PARTITION statement specifies how observations in the input data set are logically partitioned into disjoint subsets for model training, validation, and testing. For more information, see the section “Using Validation and Test Data” on page 21 in Chapter 3, “Shared Concepts.” Either you can designate a variable in the input data table and a set of formatted values of that variable to determine the role of each observation, or you can specify proportions to use for randomly assigning observations to each role.

You must specify exactly one of the following partition-options:

FRACTION(<TEST=fraction> <VALIDATE=fraction> <SEED=number>)

randomly assigns specified proportions of the observations in the input data table to the roles. You specify the proportions for testing and validation by using the TEST= and VALIDATE= suboptions. If you specify both the TEST= and VALIDATE= suboptions, then the sum of the specified fractions must be less than 1 and the remaining fraction of the observations are assigned to the training role. The SEED= option specifies an integer that is used to start the pseudorandom number generator for random partitioning of data for training, testing, and validation. If you do not specify SEED=number or if number is less than or equal to 0, the seed is generated by reading the time of day from the computer’s clock.
ROLE=variable (<TEST='value'> <TRAIN='value'> <VALIDATE='value'>)

ROLEVAR=variable (<TEST='value'> <TRAIN='value'> <VALIDATE='value'>)

names the variable in the input data table whose values are used to assign roles to each observation. This variable cannot also appear as an analysis variable in other statements or options. The TEST=, TRAIN=, and VALIDATE= suboptions specify the formatted values of this variable that are used to assign observation roles. If you do not specify the TRAIN= suboption, then all observations whose role is not determined by the TEST= or VALIDATE= suboption are assigned to the training role.

SAVESTATE Statement

SAVESTATE RSTORE=CAS-libref.data-table ;

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 4, “The ASTORE Procedure.”

You must specify the following option:

RSTORE=CAS-libref.data-table

specifies a data table in which to save the analytic store for the model. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 235.

TARGET Statement

TARGET variable < / LEVEL=NOMINAL | INTERVAL > ;

The TARGET statement names the variable whose values PROC GRADBOOST predicts.

You can specify the following option:

LEVEL=NOMINAL | INTERVAL

specifies the level of measurement. You can specify the following values:

NOMINAL specifies that the level of measurement of the variables is nominal.
INTERVAL specifies that the level of measurement of the variables is interval.

By default, LEVEL=INTERVAL for numeric variables and LEVEL=NOMINAL for categorical variables.
TRANSFERLEARN Statement

TRANSFERLEARN variable < / options> ;

The TRANSFERLEARN statement enables you to train the gradient boosting model by using auxiliary data that are added to your training data. For more information, see the section “Transfer Learning” on page 269.

The TRANSFERLEARN statement names the variable whose value indicates whether the observation belongs to the training data or auxiliary data. A value between 0 and 1 indicates training data, and a value greater than or equal to 1 indicates auxiliary data. Observations that have missing values or negative values are ignored.

You can specify the following options:

**BURN=number**

specifies the number of trees to create before down-weighting any observation in the auxiliary data.

By default, BURN=0.

**SHRINKAGE=number**

specifies the number to apply as the weighting factor for down-weighting auxiliary data, where number must be between 0 and 1, exclusive.

By default, SHRINKAGE=0.9.

**TRIMMING=number**

specifies the number to use as a fraction of the distribution of gradients on the training data beyond which auxiliary observations are down-weighted, where number must be greater than 0 and less than or equal to 1/2.

By default, TRIMMING=0.01.

VIICODE Statement

VIICODE < options> ;

The VIICODE statement writes SAS DATA step code to a file or to a catalog entry. The SAS DATA step code creates new variables on the basis of the detected variable interactions.

You can specify the following options in addition to all options in the CODE statement. For more information about the syntax of the CODE statement, see the section “CODE Statement” on page 18 in Chapter 3, “Shared Concepts.”

**ADD**

requests that the newly created variables be of the form $V + W$. 
LIMIT=number

specifies the maximum number of new variables to create. By default, LIMIT=200.

MISS

requests that the generated code handle missing values.

MULTIPLY

requests that the newly created variables be of the form $V \times W$.

SUBTRACT

requests that the newly created variables be of the form $V - W$.

THRESHOLD=number

requests that interactions with an importance less than number times the maximum interaction importance be ignored, where number must be between 0 and 1. By default, THRESHOLD=0.0001.

If the VII= option is not specified in the PROC GRADBOOST statement, then the VIICODE statement is ignored. For more information about variable interaction importance, see the section “Interaction Importance” on page 267.

WEIGHT Statement

WEIGHT variable ;

The variable in the WEIGHT statement is used as a weight to perform a weighted analysis of the data. Observations that have nonpositive or missing weights are not included in the analysis. If a WEIGHT statement is not included, all observations that are used in the analysis are assigned a weight of 1.

Details: GRADBOOST Procedure

Subsampling the Data

A decision tree in a gradient boosting model trains on new training data that are derived from the original training data presented to the GRADBOOST procedure. Using different data to train different trees during the boosting process reduces the correlation of the predictions of the trees, which in turn should improve the predictions of the boosting model.

The GRADBOOST procedure samples the original data without replacement to create the training data for an individual tree. The GRADBOOST procedure performs the action of sampling multiple times throughout a run, and each set of training data created is referred to as a subsample.

The SAMPLINGRATE= option in the PROC GRADBOOST statement specifies the fraction of observations to sample without replacement.
Training a Decision Tree

The GRADBOOST procedure trains a decision tree by splitting the subsampled data, then splitting each resulting segment, and so on recursively until some constraint is met.

Splitting involves the performing following tasks in order:

1. selecting candidate inputs
2. computing the association of each input with the target
3. searching for the best split that uses the most highly associated inputs

PROC GRADBOOST randomly selects VARS_TO_TRY=m candidate input variables independently in every node. A split search is performed on all m variables, and the best rule is kept to split the node.

The split search seeks to maximize the reduction in the gain for a nominal target and the reduction in variance of an interval target.

Splitting Nominal Values

PROC GRADBOOST uses three methods for splitting nominal values:

- enumeration: all possible splits are examined for inputs with cardinality smaller than would qualify for an alternative method
- sorting: splits that preserve a sort order are examined (using the NOMSEARCH(SORT=) suboption in the PROC GRADBOOST statement)
- clustering: k-means clustering to a binary split (using the NOMSEARCH(CLUSTER=) suboption in the PROC GRADBOOST statement)

The number of categories $K$ that are present in a node of an input variable $X$ determines the method. Different methods might be used for different variables in the same node, and for the same variable in different nodes. Let $s$ denote the value of the SORT= suboption of the NOMSEARCH option, and let $c$ denote the value of the CLUSTER= suboption. The following rules determine which method PROC GRADBOOST uses:

- When $K \leq \min(15, s, c)$, enumeration is used.
- When $\min(s, 16) \leq K < c$, sorting is used.
- When $c \leq K$, clustering is used.

The enumeration method examines all possible splits. The number of splits grows exponentially with the number of categories $K$.

The sorting method first sorts the categories and then finds the best split that preserves the sort. The sort is on $z_k$, which is defined as
\[ z_k = \frac{\sum_{i \in k} g_i + r \bar{g}}{n_k + r} \]

where

- \( k \) is a category
- \( \sum \) sums over observations in category \( k \)
- \( g_i \) is the gradient of observation \( i \)
- \( r \) is the NOMSEARCH(SHRINKAGE=) suboption value
- \( \bar{g} \) is the average gradient across all categories
- \( n_k \) is the number of observations in category \( k \)

The clustering method clusters the categories by using a 2-means clustering algorithm on the gradients.

---

**Imposing Monotonicity Constraints**

Sometimes it makes sense to impose a monotonic relationship between an input and the target. For example, someone might believe that the more stuff a person buys, the happier that person is. Assuming this, it would make sense to constrain splits on the amount of stuff a person buys to splits that increase a measure of happiness with increasing amounts of stuff bought. Although this approach does not guarantee that the relationship is exhibited everywhere in the final model, it typically is.

The MONOTONIC= option in the INPUT statement specifies a monotonicity constraint for interval inputs. The constraint is ignored for nominal inputs.

Monotonicity constraints can miss productive splits and sometimes produce no splits at all, thereby degrading accuracy. Increasing the value of the NUMBIN= option can mitigate the effect.

---

**Boosting**

A description of gradient boosting for decision trees can be found in Hastie, Tibshirani, and Friedman (2001) and Friedman (2001).

The GRADBOOST procedure creates a series of decision trees that together form a single predictive model. The trees are built sequentially. Each tree uses a subsample of the data and is built as described in the section “Training a Decision Tree” on page 261. The sequence of trees and how each tree affects a subsequent tree are discussed in Hastie, Tibshirani, and Friedman (2001), Friedman (2001), and Chen and Guestrin (2016).

A tree ensemble that seeks to predict an observed target, \( y_i \), by using the input, \( x_i \), can be written as

\[ \hat{y}_i = F(x_i) = \sum_{k=1}^{K} f_k(x_i) \]

where \( f_k \) is the functional representation of the \( k \)th tree.
To learn the set of functions (the trees) that are used in the gradient boosting model, PROC GRADBOOST seeks to minimize the regularized objective function

\[
L(F) = \Sigma_i l(\hat{y}_i, y_i) + \Sigma_k \Omega(f_k)
\]

where \(l(\hat{y}_i, y_i)\) is the loss function that measures the difference between the observed target, \(y_i\), and the predicted value of the target, \(\hat{y}_i\). The GRADBOOST procedure uses one of three loss functions as follows:

- For continuous targets, the loss function is the L2 loss.
- For binary nominal targets, the loss function is the logistic loss.
- For nonbinary nominal targets, the loss function is the multiclass logistic loss.

The term, \(\Omega(f_k)\), in the objective function is the regularization term, which acts as a penalty on the model complexity. You can specify the regularization in PROC GRADBOOST in the RIDGE= and LASSO= options.

**Early Stopping**

In some cases, gradient boosting models can be overtrained and thus perform poorly on validation or test data. One method to combat overtraining in gradient boosting is early stopping. An additional advantage to early stopping is reduced training time in cases where the stopping criterion is met well before the specified maximum number of iterations occurs.

Early stopping takes advantage of the fact that boosting is an iterative process. This means that the prediction error can be measured on a validation data set at each iteration of the process. When the prediction error on the validation data meets specified criterion, the gradient boosting process stops, yielding a model that is less overtrained than if the boosting process were allowed to continue until completion.

For an interval target, the prediction error is average square error. For a nominal target, the METRIC= suboption of the EARLYSTOP option in the PROC GRADBOOST statement specifies the prediction error as misclassification rate or log loss. The error is measured on validation data that are specified in the PARTITION statement. Without validation data, early stopping is usually ineffective.

The GRADBOOST procedure uses two approaches to early stopping:

- stagnation: stops when consecutive decreases of the error satisfy a criterion
- threshold: stops when the error exceeds a threshold

The stagnation approach is used when the value of the STAGNATION=number suboption is greater than 0. If MINIMUM=NO and the value of the TOLERANCE=\(\tau\) suboption is 0, then training stops after number iterations of nondecreasing error. If MINIMUM=YES, then training stops when the smallest error is not improved upon in number subsequent iterations. When \(\tau > 0\), then if the relative change in error from iteration \(i-1\) to iteration \(i\) is less than \(\tau\),

\[
\left| \frac{E_{\tau i-1} - E_{\tau i}}{E_{\tau i-1}} \right| < \tau
\]

for number iterations, the boosting process stops and returns the model from the last iteration in which the relative error difference was greater than or equal to \(\tau\).
The threshold approach is used when the value of the THRESHOLD=\(\theta\) suboption is greater than 0 and the value of the THRESHOLDITER=\(T\) suboption is greater than 0. Training proceeds for \(T\) iterations and then stops when the error is greater than \(\theta\). The model that has the smallest error after at least \(T\) iterations is chosen.

### Measuring Prediction Error

The GRADBOOST procedure computes the average square error measure of prediction error. For a nominal target, the procedure also computes the misclassification rate and the log loss.

The average square error for an interval, the average square error for a nominal target, the misclassification rate, and the log loss are defined, respectively, as

\[
\text{ASE}_{\text{int}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2
\]

\[
\text{ASE}_{\text{cat}} = \frac{1}{JN} \sum_{i=1}^{N} \sum_{j=1}^{J} (\delta_{ij} - \hat{p}_{ij})^2
\]

\[
\text{MISC} = \frac{1}{N} \sum_{i=1}^{N} 1(y_i \neq \hat{y}_i)
\]

\[
\text{LogLoss} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{J} \delta_{ij} \log(\hat{p}_{ij})
\]

where \(\hat{y}_i\) is the target prediction of observation \(i\), \(\delta_{ij}\) equals 1 if the nominal target value \(j\) occurs in observation \(i\) or 0 if it does not, \(\hat{p}_{ij}\) is the predicted probability of nominal target value \(j\) for observation \(i\), \(N\) is the number of observations, \(J\) is the number of nominal target values (classes), and \(\hat{p}_{ij}\) is \(\hat{p}_{ij}\) truncated away from 0 and 1,

\[
\hat{p}_{ij} = \max(\min(\hat{p}_{ij}, 1 - 10^{-10}), 10^{-10})
\]

The definitions are valid whether \(\hat{y}_i\) is the usual model prediction or the out-of-bag prediction. The \(\text{ASE}_{\text{int}}\) that is based on the usual model predictions of the original training data is usually optimistic and smaller than what its value will be for future data.
Handling Missing Values

Strategies

Tree-based models use observations that have missing input values. The GRADBOOST procedure offers the following strategies for handling missing values:

- The simple strategy is to regard a missing value as a special nonmissing value. For a nominal input, a missing value simply constitutes a new categorical value. For an input whose values are ordered, each missing value constitutes a special value that is assigned a place in the ordering that yields the best split. That place is usually different in different nodes of the tree.

This strategy is beneficial when missing values are predictive of certain target values. For example, people who have large incomes might be more reluctant to disclose their income than people who have ordinary incomes. If income were predictive of a target, then a missing income value would be predictive of the target and the missing values would be regarded as a special large-income value. The strategy seems harmless when the distribution of missing values is uncorrelated with the target because no choice of branch for the missing values would help predict the target.

A linear regression could use the same strategy by adding binary indicator variables to designate whether a value is missing. Alternatively, and much more commonly, a linear regression could simply remove observations in which any input is missing. Let $p$ denote the probability that a variable value is missing, and let $v$ denote the number of input variables. The probability that an observation has one or more missing values is $1 - (1 - p)^v$ (assuming missingness is independent and identically distributed among the inputs). If $p = 0.1$ and $v = 10$, then 65% of the observations would have missing values and would be removed from linear regression.

- The alternative strategy for decision trees is to exclude from the search algorithm any observations that have a missing value in the single input variable that defines the splitting rule. If $p = 0.1$ and $v = 10$, then only 10% instead of 65% of the observations are excluded. Although this compares favorably with common linear regression, using observations that have missing values might still be better.

Specifics

If the value of a target variable is missing, the observation is excluded from training and from evaluating the model. If the value of an input variable is missing, PROC GRADBOOST uses the missing value as a legitimate value either by default or when ASSIGNMISSING=USEINSEARCH and the number of observations in which the splitting variable has missing values is at least as large as the value of the MINUSEINSEARCH= option. When ASSIGNMISSING=USEINSEARCH and the number of observations in which the splitting value has missing values is less than the value of the MINUSEINSEARCH= option, the splitting rule assigns observations that have missing values to the largest branch.
Measuring Variable Importance

The importance of a variable is the contribution it makes to the success of the model. For a predictive model, success means good prediction. Often the prediction relies mainly on a few variables. A good measure of importance reveals those variables. The better the prediction, the more closely the model represents reality, and the more plausible it is that the important variables represent the true cause of prediction. Some people prefer a simple model so that they can understand it. However, a simple model usually relinquishes details of reality. Sometimes it is better to first find a good model and then ask which variables are important than to first ask which model is good for variable importance and then train that model.

Van der Laan (2006) asks whether a predictive model is appropriate at all. He believes that if variable importance is your goal, then you should predict importance directly instead of fitting a model. If your goal is to select suspicious genes for further study in a laboratory or to find variables in an industrial process that might influence the quality of the product, then his argument is persuasive. However, the purpose of many predictive models is to make predictions. In these cases, gaining insight into causes can be useful.

Variable importance is also useful for selecting variables for a subsequent model. The comparative importance of the selected variables does not matter. Researchers often seek speed and simplicity from the first model and seek accuracy from the subsequent model.

The GRADBOOST procedure implements two methods for computing variable importance, which are described in the following subsections. By default, the variable importance is calculated by using the change in the residual sum of square errors. You can request that the GRADBOOST procedure also calculate the variable importance by random branch assignment (RBA) by specifying the RBAIMP option.

Residual Sum of Squares Importance Method

The residual sum of squares (RSS) for regression trees is defined as

$$RSS = \sum_\lambda \sum_{i \in \lambda} (y_i - \hat{y}_\lambda^T)^2$$

where

- $i$ is an observation on leaf $\lambda$
- $y_i$ is the predicted value of the response variable of observation $i$
- $\hat{y}_\lambda^T$ is the actual value of the response variable on leaf $\lambda$

The residual sum of squares (RSS) for classification trees is defined as

$$RSS = \sum_\lambda \sum_\Phi N_\Phi^\lambda \left[ \sum_{\tau \neq \Phi} (P_\tau^\lambda)^2 + (1 - P_\Phi^\lambda)^2 \right]$$

where

- $\Phi$ is the actual response level
- $N_\Phi^\lambda$ is the number of observations on leaf $\lambda$ that have response level $\Phi$
Measuring Variable Importance

- $P_\lambda^\tau$ is the posterior probability for the response level $\tau$ on leaf $\lambda$
- $P_\lambda^\Phi$ is the posterior probability for the actual response level $\Phi$ on leaf $\lambda$

For a single tree in the boosting model, the RSS-based metric measures variable importance based on the change in RSS when a split is found at a node. The change for variable $v$ is

$$\Delta_d = \text{RSS}_d - \sum_i \text{RSS}_i^d$$

where

- $d$ denotes the node
- $i$ denotes the index of a child that this node includes
- $\text{RSS}_d$ is the RSS if the node is treated as a leaf
- $\text{RSS}_i^d$ is the RSS of the node after it has been split

If the change in RSS is negative (which is possible when you use the validation set), then the change is set to 0.

The RSS-based importance for a single variable, $v$, in a single tree is then defined as

$$\sum_{d=1}^{D} \Delta_d^v$$

where $D$ is the total number of nodes in which $v$ was used as the splitting variable.

The RSS variable importance for the boosting model is the average of the RSS variable importance across all trees in the boosting model.

**Interaction Importance**

In some cases, interactions of variables are of more interest than a single variable. When you specify the $\text{VII=}$ option in the PROC GRADBOOST statement, the procedure computes variable interaction importance as follows.

The two-way interaction importance for variables $v$ and $w$ is

$$\sum_{d,e} \frac{\Delta_d^v + \Delta_e^w}{\Delta_d^v}$$

where

- $d$ and $e$ denote nodes, where node $d$ is the parent of node $e$
- $v$ and $w$ are variables, where $v$ is the splitting variable for node $d$ and $w$ is the splitting variable for node $e$
- This summation is across all parent-child node combinations with respect to the variables $v$ and $w$. 
The three-way interaction importance for variables $v$, $w$, and $x$ is

$$\sum_{d,e,f} \frac{\Delta_{d}^{v} + \Delta_{e}^{w} + \Delta_{f}^{x}}{\Delta_{d}^{v} + \Delta_{e}^{w}}$$

where

- $d$, $e$, and $f$ denote nodes, where node $d$ is the parent of node $e$, which is the parent of node $f$
- $v$, $w$, and $x$ are variables, where $v$ is the splitting variable for node $d$, $w$ is the splitting variable for node $e$, and $x$ is the splitting variable for node $f$
- This summation is across all parent-child-grandchild node combinations with respect to the variables $v$, $w$, and $x$.

When comparing variable importance values, you should compare a two-way interaction only with another two-way interaction, or a three-way interaction only with another three-way interaction. Comparing a two-way interaction to a three-way interaction is not meaningful, because the values for the importance are not on the same scale.

In addition to calculating the variable interaction importance, you can also generate SAS DATA step code to create interaction variables and save the code to a file by using the VIICODE statement. When you run the generated SAS DATA step code on the original data, or new data, interaction variables are created.

Created interaction variables are of the form $W + V$, $W - V$, or $W \times V$, where the values used in the calculations come from the variables whose interactions are determined to be important. The values used are as follows:

- For continuous variables, the raw value of the variable is used.
- For categorical variables, the proportion of observations in the training set that is assigned to a specific branch is used.

**Random Branch Assignment Importance Method**

The random branch assignment (RBA) method computes the importance of an input variable $v$ by comparing how well the data fit the predictions before and after they are modified. To modify the predictions, the GRADBOOST procedure replaces all splitting rules that use variable $v$ by a rule that randomly assigns an observation to a branch. The probability of assigning an observation to a branch is proportional to the number of observations that are assigned to the branch in the current data. The current data are the training data when RBA is computed during training. Otherwise, the current data are those being scored on an existing model.

The RBA importance can be expressed mathematically as

$$I_{RBA}(v) \propto \sum_{i=1}^{n} \text{Loss}(y_i, \hat{y}_{i}) - \sum_{i=1}^{n} \text{Loss}(y_i, \hat{y}_{i})$$

where $\hat{y}_{i}$ is the modified prediction for observation $i$ and $\hat{y}_{i}$ is the standard prediction.

For an interval target, PROC GRADBOOST computes the RBA importance of squared error loss. For a nominal target, PROC GRADBOOST uses the misclassification rate as the loss function.
Transfer Learning

Some applications of statistical and machine learning have only a scarce amount of training data available. One method of building models in situations that have small training sets is to supplement the scarce training data with more abundant auxiliary data. The auxiliary data should have the same target and input variables, but can come from a different context than the training data. For more discussion, see Dai et al. (2007).

In the GRADBOOST procedure, you can use the auxiliary data to increase the number of observations for training the model. In order to prevent the model from being overly biased toward the auxiliary data, the GRADBOOST procedure down-weights auxiliary observations that are dissimilar from the training observations. The factor that is applied to down-weight observations is controlled by the SHRINKAGE= option in the TRANSFERLEARN statement. To determine whether an observation is dissimilar, the GRADBOOST procedure computes a gradient for each observation. An observation in the auxiliary data that is dissimilar to the training data will have a gradient that is in either a low quantile or a high quantile of the gradients of the training data. The TRIMMING= option in the TRANSFERLEARN statement controls the fraction of training gradients beyond which auxiliary observations are down-weighted.

The GRADBOOST procedure begins down-weighting auxiliary observations only after a number of trees has been grown, as specified in the BURN= option in the TRANSFERLEARN statement.

To use transfer learning in the GRADBOOST procedure, you must concatenate your training data and your auxiliary data, and specify this combined data table in the DATA= option in the PROC GRADBOOST statement. To distinguish the two types of data, you must create an indicator variable as follows:

- When the value of the indicator variable is greater than or equal to 0 and less than 1, PROC GRADBOOST considers the observation to belong to the training data.
- When the value of the indicator variable is greater than or equal to 1, PROC GRADBOOST considers the observation to belong to the auxiliary data.
- When the value of the indicator variable is missing or is less than 0, PROC GRADBOOST skips the observation during model training.

Hyperparameter Tuning

The options that control the training algorithm are often called hyperparameters to distinguish them from the parameters of the trained model. The main parameters in tree-based models are the splitting rules. Optimal hyperparameter settings depend on the data and are never known for certain. Nevertheless, here are some recommendations:

**BINMETHOD=QUANTILE**

Using quantile bins instead of bins of equal width often results in a better fit, especially when some inputs have outliers.
NUMBIN=number
Using more bins sometimes improves and rarely degrades accuracy. However, training takes longer
with more bins, often much longer with many more bins.

MAXDEPTH=number
A modeling algorithm that fits the training data well will generalize well if most of the data are signal,
but it will generalize poorly if most of the data are noise. In the latter case for boosting, it is better to
make small trees, make each tree contribute less, and make more trees. Accordingly, reduce the values
of the MAXDEPTH= and LEARNINGRATE= options and increase the value of the NTREES= option
when the data have a weak signal.

RIDGE=1
This option helps prevent overfitting splits.

For more information about hyperparameter tuning, see the section “Hyperparameter Tuning” on page 23 in
Chapter 3, “Shared Concepts.”

You can use the AUTOTUNE statement to tune the following options in the PROC GRADBOOST statement:

- LASSO= option for the L1 regularization parameter
- RIDGE= option for the L2 regularization parameter
- LEARNINGRATE= option for the learning rate parameter
- MAXDEPTH= option for the maximum depth of trees
- NTREES= option for the number of trees
- NUMBIN= option for the number of bins
- SAMPLINGRATE= option for the proportion of the training data to sample
- VARS_TO_TRY= option for the number of variables to randomly select at each node split for each tree

---

**k-fold Cross Validation**

The CROSSVALIDATION statement performs a k-fold cross validation process to find the average estimated
validation error (misclassification error for nominal targets or average square error for interval targets) for
the trained model. During cross validation, all data are divided into k subsets (folds), where k is the value
of the KFOLD= option. For each fold, a new model is trained on the (k–1) folds, and then validated using
the selected (hold-out) fold. The validation error estimates are then averaged over each set of training and
scoring executions to obtain a single value. The CROSSVALIDATION statement returns a table that contains
a single data row that shows the average validation error.
Displayed Output

The GRADBOOST procedure displays the parameters that are used to train the model, the fit statistics of the trained model, and other information. The output is organized into various tables, which are discussed here in order of their appearance.

Model Information

The “Model Information” table contains the settings of the training parameters. This table also contains some basic information about the trees in the resulting boosting model. This table is produced by default.

Number of Observations

The “Number of Observations” table contains the number of observations that are read from the input data table and the number of observations that are used in the analysis. When you specify the PARTITION statement, the table also indicates the number of observations that are used in each partition. This table is produced by default.

Variable Importance

The “Variable Importance” table displays variable importance based on residual sum of square errors, which is explained in the section “Measuring Variable Importance” on page 266. This table is produced by default.

RBA Variable Importance

The “RBA Variable Importance” table displays variable importance based on the random branch assignment (RBA) method, which is explained in the section “Random Branch Assignment Importance Method” on page 268. This table is produced by the RBAIMP option in the PROC GRADBOOST statement.

Fit Statistics

The “Fit Statistics” table contains statistics that measure the model’s goodness of fit. The fit of the model to the data improves as the number of trees in the boosting model increases. Successive rows in the table contain fit statistics for a boosting model that has more trees. Fit statistics are described in the section “Measuring Prediction Error” on page 264. This table is produced by default.

Tuner Information

The “Tuner Information” table displays the setup values that the tuner uses. This table is produced by the AUTOTUNE statement.

Tuner Summary

The “Tuner Summary” table displays statistics about the tuning process. This table is produced by the AUTOTUNE statement.
Tuner Timing

The “Tuner Timing” table displays the total time spent on different tasks while tuning. This table is produced by the AUTOTUNE statement.

Best Configuration

The “Best Configuration” table displays the hyperparameters and objective function values for the best configuration. This table is produced by the AUTOTUNE statement.

Tuner Results

The “Tuner Results” table displays the values of the hyperparameters, the objective function for the default configuration (Iteration 0), and up to 10 best found configurations. This table is produced by the AUTOTUNE statement.

Cross-Validation Fit Statistics

The “Cross-Validation Fit Statistics” table contains the per fold and average assessment metrics of $k$-fold cross validation.

Evaluation History

The “Evaluation History” tables displays the values of the hyperparameters and the objective function for all configurations. This table is produced by the AUTOTUNE statement, either by default or when EVALHISTORY=ALL.

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.

Predicted Probability Names

The “Predicted Probability Names” table indicates the names of the created variables in either the score code or the data table that is specified in the OUT= option in the OUTPUT statement. This table is produced when you specify the PRINTTARGET option in the PROC GRADBOOST statement.
Each table that the GRADBOOST procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. The name of each table and a short description of the contents are listed in Table 12.6.

### Table 12.6  ODS Tables Produced by PROC GRADBOOST

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>BestConfiguration</td>
<td>Hyperparameters and objective function values for the best configuration</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>CrossValidateMLFitStat</td>
<td>Per fold and average assessment metrics of $k$-fold cross validation</td>
<td>CROSSVALIDATION</td>
<td>Default</td>
</tr>
<tr>
<td>EvaluationHistory</td>
<td>Values of the hyperparameters and the objective function for all configurations</td>
<td>AUTOTUNE</td>
<td>Default / EVALHISTORY=ALL</td>
</tr>
<tr>
<td>FitStatistics</td>
<td>Fit statistics from the model</td>
<td>PROC GRADBOOST</td>
<td>Default</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>PROC GRADBOOST</td>
<td>Default</td>
</tr>
<tr>
<td>Nobs</td>
<td>Number of observations</td>
<td>PROC GRADBOOST</td>
<td>Default</td>
</tr>
<tr>
<td>OutputCASTables</td>
<td>See the section “OutputCasTables Table” on page 272</td>
<td>PROC GRADBOOST / OUTPUT</td>
<td>OUTMODEL=</td>
</tr>
<tr>
<td>PredName</td>
<td>Predicted name information for interval targets</td>
<td>PROC GRADBOOST</td>
<td>PRINTTARGET</td>
</tr>
<tr>
<td>PredProbName</td>
<td>Predicted probability name information for nominal targets</td>
<td>PROC GRADBOOST</td>
<td>PRINTTARGET</td>
</tr>
<tr>
<td>PredIntoName</td>
<td>Predicted name information for nominal targets</td>
<td>PROC GRADBOOST</td>
<td>PRINTTARGET</td>
</tr>
<tr>
<td>RBAImportance</td>
<td>Random branch assignment variable importance</td>
<td>PROC GRADBOOST</td>
<td>RBAIMP</td>
</tr>
<tr>
<td>TunerInfo</td>
<td>Setup values used by the tuner</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
</tbody>
</table>
Table 12.6  continued

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>TunerResults</td>
<td>Values of the hyperparameters, the objective function for the default configuration (Iteration 0), and up to 10 best found configurations</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerSummary</td>
<td>Statistics about the tuning process</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerTiming</td>
<td>Total time spent on different tasks while tuning</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>VariableImportance</td>
<td>Residual sum of squares variable importance</td>
<td>PROC GRADBOOST</td>
<td>Default</td>
</tr>
</tbody>
</table>

Example: GRADBOOST Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

Example 12.1: Scoring New Data by Using a Previous Boosting Model

This example illustrates how you can use the OUTMODEL= option to save a model table, and later use the model table to score a data table. It uses the JunkMail data set in the Sashelp library.

The JunkMail data set comes from a study that classifies whether an email is junk email (coded as 1) or not (coded as 0). The data set contains 4,601 observations with 59 variables. The response variable is a binary indicator of whether an email is considered spam or not. There are 57 predictor variables that record the frequencies of some common words and characters and the lengths of uninterrupted sequences of capital letters in emails.

You can load the Sashelp.JunkMail data set into your CAS session by naming your CAS engine libref in the first statement of the following DATA step:

```sas
data mycas.junkmail;
   set sashelp.junkmail;
run;
```

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined libref.
Example 12.1: Scoring New Data by Using a Previous Boosting Model

The following statements train a boosting model and score the training data table. The OUTPUT statement scores the training data and saves the results to a new table named *fit_at_runtime*.

```plaintext
proc gradboost data=mycas.junkmail outmodel=mycas.gradboost_model seed=12345;
   input 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 / level = interval;
   target class /level=nominal;
   output out=mycas.score_at_runtime;
   ods output FitStatistics=fit_at_runtime;
run;
```

The preceding statements produce the table shown in Output 12.1.1. The table shows the training statistics.

<table>
<thead>
<tr>
<th>Fit Statistics, Fit at Run Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Trees</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>91</td>
</tr>
<tr>
<td>92</td>
</tr>
<tr>
<td>93</td>
</tr>
<tr>
<td>94</td>
</tr>
<tr>
<td>95</td>
</tr>
<tr>
<td>96</td>
</tr>
<tr>
<td>97</td>
</tr>
<tr>
<td>98</td>
</tr>
<tr>
<td>99</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>

The following statements use a previously saved model to score new data:
proc gradboost data=mycas.junkmail inmodel=mycas.gradboost_model;
  output out=mycas.score_later;
  ods output FitStatistics=fit_later;
run;

When you specify the INMODEL= option to use a previously created boosting model, you see the statistics for the scored data if the target exists in the newly scored data table. In this example, the scored data are the same as the training data, so you can see that the statistics in Output 12.1.2 match those previously seen in Output 12.1.1.

<table>
<thead>
<tr>
<th>Number of Trees</th>
<th>Average Square Error</th>
<th>Misclassification Rate</th>
<th>Log Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2087</td>
<td>0.3940</td>
<td>0.6079</td>
</tr>
<tr>
<td>2</td>
<td>0.1842</td>
<td>0.2334</td>
<td>0.5569</td>
</tr>
<tr>
<td>3</td>
<td>0.1645</td>
<td>0.1098</td>
<td>0.5154</td>
</tr>
<tr>
<td>4</td>
<td>0.1479</td>
<td>0.0926</td>
<td>0.4793</td>
</tr>
<tr>
<td>5</td>
<td>0.1330</td>
<td>0.0880</td>
<td>0.4460</td>
</tr>
<tr>
<td>6</td>
<td>0.1216</td>
<td>0.0830</td>
<td>0.4197</td>
</tr>
<tr>
<td>7</td>
<td>0.1113</td>
<td>0.0815</td>
<td>0.3950</td>
</tr>
<tr>
<td>8</td>
<td>0.1024</td>
<td>0.0767</td>
<td>0.3728</td>
</tr>
<tr>
<td>9</td>
<td>0.0952</td>
<td>0.0769</td>
<td>0.3539</td>
</tr>
<tr>
<td>10</td>
<td>0.0888</td>
<td>0.0739</td>
<td>0.3368</td>
</tr>
<tr>
<td>91</td>
<td>0.0276</td>
<td>0.0367</td>
<td>0.1049</td>
</tr>
<tr>
<td>92</td>
<td>0.0273</td>
<td>0.0356</td>
<td>0.1040</td>
</tr>
<tr>
<td>93</td>
<td>0.0271</td>
<td>0.0361</td>
<td>0.1033</td>
</tr>
<tr>
<td>94</td>
<td>0.0270</td>
<td>0.0350</td>
<td>0.1030</td>
</tr>
<tr>
<td>95</td>
<td>0.0268</td>
<td>0.0343</td>
<td>0.1021</td>
</tr>
<tr>
<td>96</td>
<td>0.0267</td>
<td>0.0341</td>
<td>0.1017</td>
</tr>
<tr>
<td>97</td>
<td>0.0266</td>
<td>0.0346</td>
<td>0.1012</td>
</tr>
<tr>
<td>98</td>
<td>0.0264</td>
<td>0.0343</td>
<td>0.1006</td>
</tr>
<tr>
<td>99</td>
<td>0.0261</td>
<td>0.0322</td>
<td>0.0995</td>
</tr>
<tr>
<td>100</td>
<td>0.0259</td>
<td>0.0319</td>
<td>0.0989</td>
</tr>
</tbody>
</table>

This example demonstrates that the GRADBOOST procedure can score an input data table by using a previously saved boosting model, which was saved using the OUTMODEL= option in a previous procedure run. If you want to properly score a new data table, you must not modify the table mycas.gradboost_model, because doing so could invalidate the constructed boosting model. As with any scoring of new data, the variables that are used in the model creation must be present in order for you to score a new table.
Example 12.2: Transfer Learning

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example illustrates transfer learning. Transfer learning augments the training data with auxiliary data and attempts to down-weight the influence of observations that are not representative of the original training data. The original training data typically come from a target population from which data are hard to get. For simplicity, this example refers to auxiliary observations that are similar to the target population as friends, and the other auxiliary observations as aliens. PROC GRADBOOST tries to identify and down-weight the aliens. This example runs PROC GRADBOOST twice without using transfer learning: once with all the data, and once without the aliens. All models are evaluated with data from the target population that are not part of the training data. The model incorporating transfer learning should provide a fit than one without it, though not as good as the one in which the alien population is removed.

The following DATA step generates three data sets: one for training that includes the friends and the aliens, one without the aliens, and a third data set with test observations held out from training.

```sas
data mycas.train mycas.noAlien mycas.test;
  array x(2) x1 - x2;
  keep datarole y x: constantZero;
  constantZero = 0;
  call streaminit(3331333);
  do datarole = -1 to 2;
    select(datarole);
    when(-1) do; * test data;
      y =1;
      n = 1000;
      end;
    when(0) do; * target population;
      y =1;
      n = 1000;
      end;
    when(1) do; * friend population;
      y =1;
      n = 2000;
      end;
    when(2) do; * alien population;
      y= -1;
      n = 1000;
      end;
  end;
  nhalf = n/2;
  mu = 0.5;
  do i = 1 to n;
    do j = 1 to 2;
      x(j) = rand('normal', mu);
    end;
  end;
```

```sas
Example 12.2: Transfer Learning ♦ 277
```
The following macro invokes PROC GRADBOOST twice: first to train the model, and then again to apply the model to the test data and output fits statistics. The TRANSFERLEARN statement specifies the variable that identifies the auxiliary observations. When that variable is zero for all observations, transfer learning is not done. This happens when constantZero is passed to the macro.

The BURN=10 option in the TRANSFERLEARN statement delays down-weighting until tree 11.

The following code combines the average square error from the three models into a single table.
The following code plots the average square error for each model by the number of trees in the model:

```sas
proc template;
  define statgraph transferLearning;
  begingraph;
    layout overlay;
      scatterplot y=train_ase
        x=trees / markerattrs=(color=blue)
        name='with'
        legendlabel="With Transfer Learning";
      scatterplot y=noTransfer_ase
        x=trees / markerattrs=(color=red)
        name='without'
        legendlabel="Without Transfer Learning";
      scatterplot y=noAlien_ase
        x=trees / markerattrs=(color=brown)
        name='noAliens'
        legendlabel="Without Aliens";

      discretelegend 'without' 'with' 'noAliens';
    endlayout;
  endgraph;
end;
run;

proc sgrender data=result template=transferLearning;
run;
```

Output 12.2.1 shows that the fit with transfer learning is better than without it, though not as good as when all the alien observations are removed from the data. The fit with transfer learning is identical to the fit without it for the first 10 trees because down-weighting does not begin until tree 11 in this example.

**Output 12.2.1** ASE versus Number of Trees for Three Models
References


Chapter 13
The GVARCLUS Procedure

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: GVARCLUS Procedure</td>
<td>282</td>
</tr>
<tr>
<td>PROC GVARCLUS Features</td>
<td>282</td>
</tr>
<tr>
<td>PROC GVARCLUS Compared to PROC VARCLUS</td>
<td>283</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>283</td>
</tr>
<tr>
<td>Getting Started: GVARCLUS Procedure</td>
<td>284</td>
</tr>
<tr>
<td>Syntax: GVARCLUS Procedure</td>
<td>286</td>
</tr>
<tr>
<td>PROC GVARCLUS Statement</td>
<td>287</td>
</tr>
<tr>
<td>DISPLAY Statement</td>
<td>290</td>
</tr>
<tr>
<td>DISPLAYOUT Statement</td>
<td>291</td>
</tr>
<tr>
<td>FREQ Statement</td>
<td>292</td>
</tr>
<tr>
<td>INPUT Statement</td>
<td>292</td>
</tr>
<tr>
<td>Details: GVARCLUS Procedure</td>
<td>292</td>
</tr>
<tr>
<td>Missing Values</td>
<td>293</td>
</tr>
<tr>
<td>Graphical Lasso</td>
<td>293</td>
</tr>
<tr>
<td>Graphical Representation of Precision Matrix</td>
<td>294</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>294</td>
</tr>
<tr>
<td>Selection</td>
<td>295</td>
</tr>
<tr>
<td>Displayed Output</td>
<td>295</td>
</tr>
<tr>
<td>Number of Observations</td>
<td>295</td>
</tr>
<tr>
<td>Cluster Summary</td>
<td>295</td>
</tr>
<tr>
<td>ODS Table Names</td>
<td>295</td>
</tr>
<tr>
<td>Examples: GVARCLUS Procedure</td>
<td>296</td>
</tr>
<tr>
<td>Example 13.1: Analyzing United States Senate Voting Data</td>
<td>296</td>
</tr>
<tr>
<td>Example 13.2: Output a Covariance Matrix to a SAS Data File</td>
<td>298</td>
</tr>
<tr>
<td>Example 13.3: Output the Covariance Matrix in LIL Format</td>
<td>300</td>
</tr>
<tr>
<td>References</td>
<td>302</td>
</tr>
</tbody>
</table>
Overview: GVARCLUS Procedure

The GVARCLUS procedure performs variable clustering and graphical modeling in SAS Viya. The procedure divides a set of variables into disjoint clusters and creates tables that contain the edge and vertex information for defining an undirected graph. Variables in different clusters are conditionally independent given their own clusters. The procedure also provides the edge and vertex information for an undirected graphical model; this information expresses the relationships among all the variables. A regularization parameter is used to control the sparsity of connections among variables. Tuning the regularization parameter from low to high increases the number of disconnected components and splits larger clusters into smaller ones. Those divided clusters naturally form a hierarchical structure during this process.

The GVARCLUS procedure helps mine the relationships among variables and remove redundant variables. Removing redundant variables is especially important for high-dimensional data, which are increasingly common in the modern data mining world.

PROC GVARCLUS Features

The GVARCLUS procedure does the following:

- supports multiple INPUT statement for both continuous and categorical inputs
- provides a FREQ statement for grouped analysis
- provides the OUTTREE option to output the hierarchical clustering results in a tree structure
- provides the OUTEDGE and OUTVERT options to output the edges and vertices, respectively, for defining an undirected graph
- provides the OUTCP option to output a covariance matrix
- performs graphical lasso based on Friedman, Hastie, and Tibshirani (2008), which estimates the inverse covariance (IC) matrix at a specified regularization parameter. The inverse covariance matrix interprets the partial correlation between variables given other variables
- performs sequential steps of estimating the inverse covariance matrix by using regularization parameters from large to small, which impose different sparsity constraints on the estimation. The sequential steps eventually produce a set of nested clusters that are organized as a hierarchical tree
- performs covariance thresholding based on Friedman, Hastie, and Tibshirani (2008) as preprocessing, which thresholds the entries of the sample covariance matrix at the regularization parameter and decomposes the matrix into connected components

Because the GVARCLUS procedure runs on CAS, it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode
PROC GVARCLUS Compared to PROC VARCLUS

This section compares the GVARCLUS procedure with the VARCLUS procedure in SAS/STAT software.

PROC GVARCLUS interprets the conditional dependency among variables by estimating the inverse covariance matrix. The off-diagonal elements of an inverse covariance matrix correspond to partial correlations, so the zero elements imply conditional independence between the pair of variables. The conditional independence provides a better model for understanding the direct link between variables than does simple correlation analysis, which models each pair of variables without considering other variables.

PROC VARCLUS performs variable clustering based on latent components. It assigns a variable to its most correlated latent component, so eventually the variables within a cluster have high correlation.

The primary difference between the two procedures is that PROC GVARCLUS considers the association between variables after the other effects have been factored out.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```plaintext
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```plaintext
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:
cas mysess terminate;

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”

---

### Getting Started: GVARCLUS Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example demonstrates how you can use the GVARCLUS procedure to cluster variables into homogeneous groups and construct a hierarchy.

The following DATA step generates a data table and loads the table into your CAS session. The table consists of 100 observations and 20 continuous variables (x1–x20). There are two groups of five variables, and 10 variables that are randomly generated from an uniform distribution. The initial variables for each group, x1 and x6, are independently generated from a normal distribution. Two of the variables from each group are linearly correlated to their initial variables, and the other two variables have a nonlinear relationship.

```sas
data mycas.getStarted;
   array x{20} x1-x20;
   do i = 1 to 100;
      x1=rannor(12334); x2=0.9*x1+0.2; x3=0.7*x1+0.6; x4=x3*x2*x1; x5=x1*x1*x1;
      x6=rannor(56757); x7=0.9*x6+0.2; x8=0.7*x6+0.6; x9=x8*x6; x10=x6*x6*x6;
      do j= 11 to 20;
         x{j} = ranuni(10);
      end;
   output;
   end;
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 execute the GVARCLUS algorithm on the mycas.getStarted data table:

```sas
proc gvarclus data=mycas.getStarted mincluster=2 maxsteps=8
   rho=0.8 outtree=mycas.dendrogram;
   input x1-x20 / level=interval;
run;
```

The INPUT statement defines the input variables x1 to x20 as interval variables. The MINCLUSTER= option in the PROC GVARCLUS statement specifies the minimum number of clusters as 2, so the clustering process stops when the number of clusters is less than or equal to 2. The MAXSTEPS= option specifies the maximum number of steps in the clustering process as 8. The RHO= option specifies a value of 0.8, which determines the sequence of regularization parameters \([0.8^1, 0.8^2, 0.8^3, \ldots]\) that are used in each step sequentially. The OUTTREE= option creates the mycas.dendrogram table to contain information about the tree structure of hierarchical clustering.
The output from this analysis is presented in Figure 13.1 and Figure 13.2.

Figure 13.1 displays the “Number of Observations” tables. This table shows that all 100 observations in the data table are used in the analysis.

![Figure 13.1 Number of Observations](image)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
<th>Step 5</th>
<th>Step 6</th>
<th>Step 7</th>
<th>Step 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>1</td>
<td>19</td>
<td>36</td>
<td>52</td>
<td>68</td>
<td>82</td>
<td>96</td>
<td>108</td>
</tr>
<tr>
<td>x2</td>
<td>2</td>
<td>19</td>
<td>36</td>
<td>52</td>
<td>68</td>
<td>82</td>
<td>96</td>
<td>108</td>
</tr>
<tr>
<td>x3</td>
<td>3</td>
<td>20</td>
<td>36</td>
<td>52</td>
<td>68</td>
<td>82</td>
<td>96</td>
<td>108</td>
</tr>
<tr>
<td>x4</td>
<td>4</td>
<td>21</td>
<td>37</td>
<td>53</td>
<td>69</td>
<td>83</td>
<td>96</td>
<td>108</td>
</tr>
<tr>
<td>x5</td>
<td>5</td>
<td>22</td>
<td>38</td>
<td>54</td>
<td>68</td>
<td>82</td>
<td>96</td>
<td>108</td>
</tr>
<tr>
<td>x6</td>
<td>6</td>
<td>23</td>
<td>39</td>
<td>55</td>
<td>70</td>
<td>84</td>
<td>97</td>
<td>109</td>
</tr>
<tr>
<td>x7</td>
<td>6</td>
<td>23</td>
<td>39</td>
<td>55</td>
<td>70</td>
<td>84</td>
<td>97</td>
<td>109</td>
</tr>
<tr>
<td>x8</td>
<td>6</td>
<td>23</td>
<td>39</td>
<td>55</td>
<td>70</td>
<td>84</td>
<td>97</td>
<td>109</td>
</tr>
<tr>
<td>x9</td>
<td>7</td>
<td>24</td>
<td>40</td>
<td>56</td>
<td>71</td>
<td>85</td>
<td>97</td>
<td>109</td>
</tr>
<tr>
<td>x10</td>
<td>8</td>
<td>25</td>
<td>41</td>
<td>57</td>
<td>70</td>
<td>84</td>
<td>97</td>
<td>109</td>
</tr>
<tr>
<td>x11</td>
<td>9</td>
<td>26</td>
<td>42</td>
<td>58</td>
<td>72</td>
<td>86</td>
<td>98</td>
<td>110</td>
</tr>
<tr>
<td>x12</td>
<td>10</td>
<td>27</td>
<td>43</td>
<td>59</td>
<td>73</td>
<td>87</td>
<td>99</td>
<td>111</td>
</tr>
<tr>
<td>x13</td>
<td>11</td>
<td>28</td>
<td>44</td>
<td>60</td>
<td>74</td>
<td>88</td>
<td>100</td>
<td>112</td>
</tr>
<tr>
<td>x14</td>
<td>12</td>
<td>29</td>
<td>45</td>
<td>61</td>
<td>75</td>
<td>89</td>
<td>101</td>
<td>113</td>
</tr>
<tr>
<td>x15</td>
<td>13</td>
<td>30</td>
<td>46</td>
<td>62</td>
<td>76</td>
<td>90</td>
<td>102</td>
<td>114</td>
</tr>
<tr>
<td>x16</td>
<td>14</td>
<td>31</td>
<td>47</td>
<td>63</td>
<td>77</td>
<td>91</td>
<td>103</td>
<td>115</td>
</tr>
<tr>
<td>x17</td>
<td>15</td>
<td>32</td>
<td>48</td>
<td>64</td>
<td>78</td>
<td>92</td>
<td>104</td>
<td>116</td>
</tr>
<tr>
<td>x18</td>
<td>16</td>
<td>33</td>
<td>49</td>
<td>65</td>
<td>79</td>
<td>93</td>
<td>105</td>
<td>117</td>
</tr>
<tr>
<td>x19</td>
<td>17</td>
<td>34</td>
<td>50</td>
<td>66</td>
<td>80</td>
<td>94</td>
<td>106</td>
<td>118</td>
</tr>
<tr>
<td>x20</td>
<td>18</td>
<td>35</td>
<td>51</td>
<td>67</td>
<td>81</td>
<td>95</td>
<td>107</td>
<td>119</td>
</tr>
</tbody>
</table>

Figure 13.2 shows the “Cluster Summary” table, which shows the cluster structure in each step. The variable (or level for nominal variables) is assigned to a unique cluster in each step, and the clusters from sequential steps are nested to form a hierarchical structure.

You can define a dendrogram in the Graph Template Language (GTL) and display the hierarchical clustering results from the GVARCLUS algorithm.

The following code copies mycas.dendrogram into the Work library:

```sas
data outtree;
  set mycas.dendrogram;
run;
```

The following code defines the parent node, child node and the height in the dendrogram through the DENDROGRAM statement in GTL. The dendrogram is then plotted using the SGRENDER procedure.
proc template;
   define statgraph dendrogram;
   begingraph;
   layout overlay;
   dendrogram nodeID=_CHILD_ parentID=_PARENT_ clusterheight=_HEIGHT_;
   endlayout;
   endgraph;
end;

proc sgrender data=outtree template=dendrogram;
run;

Output 13.3 shows the dendrogram.

Figure 13.3 Hierarchical Clustering

Syntax: GVARCLUS Procedure

The following statements are available in the GVARCLUS procedure:

PROC GVARCLUS <options> ;
   DISPLAY <table-list> </options> ;
   DISPLAYOUT table-spec-list </options> ;
   INPUT variables <LEVEL= INTERVAL | NOMINAL> ;
   FREQ variable ;

The PROC GVARCLUS statement and at least one IPUT statement are required.

The following sections describe the PROC GVARCLUS statement and then describe the other statements in alphabetical order.
PROC GVARCLUS Statement

PROC GVARCLUS <options> ;

The PROC GVARCLUS statement invokes the procedure. Table 13.1 summarizes the important options in the PROC GVARCLUS statement by function. The options are then described fully in alphabetical order.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Options</td>
<td>דו Specifications the input data table</td>
</tr>
<tr>
<td>DATA</td>
<td>Specifies the maximum number of members within each final cluster</td>
</tr>
<tr>
<td>MAXMEMBER</td>
<td>Specifies the maximum number of members within each final cluster</td>
</tr>
<tr>
<td>MAXSTEP</td>
<td>Specifies the maximum number of clustering steps</td>
</tr>
<tr>
<td>MINCLUSTER</td>
<td>Specifies the minimum number of clusters</td>
</tr>
<tr>
<td>RHO</td>
<td>Specifies the initial value of the regularization parameter</td>
</tr>
<tr>
<td>SELECT</td>
<td>Selects the optimal clustering on the basis of the specified criterion</td>
</tr>
<tr>
<td>STOP</td>
<td>Requests that the clustering process stop if the clustering structure does not change in the previous number of specified steps</td>
</tr>
<tr>
<td>TOL</td>
<td>Specifies the threshold of convergence</td>
</tr>
<tr>
<td>Options Related to Variable Clustering</td>
<td>Performs graphical variable clustering without preprocessing</td>
</tr>
<tr>
<td>OUTCP</td>
<td>Outputs the covariance matrix from the last step</td>
</tr>
<tr>
<td>OUTEDGE</td>
<td>Outputs the edges for defining an undirected network</td>
</tr>
<tr>
<td>OUTTREE</td>
<td>Outputs the hierarchical clustering results in a tree structure</td>
</tr>
<tr>
<td>OUTVERT</td>
<td>Outputs the vertices and their size for defining an undirected network</td>
</tr>
</tbody>
</table>

You can specify the following options:

**DATA=** *CAS-libref.data-table*

denames the input data table for PROC GVARCLUS to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

*CAS-libref* refers to a collection of information that is defined in the LIBNAME statement and includes the *caslib*, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 283.

*data-table* specifies the name of the input data table.
EXACT
performs graphical variable clustering without preprocessing by thresholding the sample covariance into connected components. By default, the preprocessing step is performed.

MAXITER=n
MAXITERS=n
specifies the maximum number of iterations for estimating the sparse precision matrix by using coordinate descent. By default, MAXITER=50.

MAXMEMBER=n
MAXMEMBERS=n
stops PROC GVARCLUS when the number of members within any cluster is greater than or equal to n.

MAXSTEP=n
MAXSTEPS=n
stops PROC GVARCLUS after it runs n steps. By default, MAXSTEP=3.

MINCLUSTER=n
MINCLUSTERS=n
stops PROC GVARCLUS when the number of clusters is less than or equal to n. By default, MINCLUSTER=3.

OUTCP=CAS-libref.data-table < / LIST < EPS=number >>
creates a data table that contains a covariance matrix that depicts the relationships among variables and also creates a set of statistics about the input data table and variables. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 283.

When you specify the LIST option, the symmetric matrix is output in the list-of-list (LIL) format. In this format, the matrix is represented as a set of tuples \((i, j, x)\), where \(x\) is an entry in the matrix and \(i\) and \(j\) denote its row and column indices, respectively. LIL format can be used when the output contains too many columns to fit in a data table. For example, in most database systems, the maximum number of columns in a table is usually limited to several thousand. If an output matrix contains more columns than the limit, you must use the LIST option in order to avoid errors that would arise from writing too many columns to the table. When LIL format is used, all entries in the matrix that have a value of 0 are ignored in the output.

When you specify EPS=number in the LIST suboption, matrix entries whose absolute value is smaller than \(number\) are ignored in the output. This feature helps omit unreliable estimations and generate a compact representation of the matrix. When you do not specify the EPS= option, only the 0 entries in the matrix are ignored in the output.

OUTEDGE=CAS-libref.data-table
creates a data set for use with the hypergroup action. This table contains the information that defines the edges in the network, _FROM_, _TO_, and _WEIGHT_. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 283.
OUTTREE=\texttt{CAS-libref.data-table}
creates a data table 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. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the caslib and session identifier, and \texttt{data-table} specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 283.

OUTVERT=\texttt{CAS-libref.data-table}
creates a data set for use with the hypergroup action. This table contains the vertices in the network and their size. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the caslib and session identifier, and \texttt{data-table} specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 283.

RHO=\texttt{fraction}

specifies the value of $\rho$ that determines the sequence of regularization parameters, $\{\rho^1, \rho^2, \rho^3, \ldots\}$, that are used on sequential clustering steps. The higher the $\rho$ value, the more regularization and the sparser the inverse covariance. By default, RHO=0.8.

SELECT=NONE | PENALIZED

selects the optimal clustering on the basis of the specified criterion. You can specify the following values:

- NONE performs no selection, and the output tables of the edges and vertices of the network reflect the clustering from the last step.

- PENALIZED performs selection through minimizing the penalized log likelihood, as shown in the section “Selection” on page 295.

By default, SELECT=NONE.

STOP=n

requests that the action stop if the clustering results do not change in the previous $n$ consecutive steps. By default, STOP=3.

TOLERANCE=\texttt{fraction}  
TOL=\texttt{fraction}

specifies the minimal absolute tolerance at which an iteration stops. The tolerance number must be equal to or greater than 1.0E–12. By default, TOLERANCE=1.0E–3.
DISPLAY Statement

DISPLAY < table-list > < / options > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see SAS Output Delivery System: Procedures Guide.

You can specify the table-list as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “ODS Table Names” on page 295. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path Bygroup1.Summary.SelectionSummary. A partial pathname does not include all groups; for example, Selection-Summary and Summary.SelectionSummary are partial pathnames for Bygroup1.Summary.SelectionSummary.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both SelectionSummary and Summary.SelectionSummary select Bygroup1.Summary.SelectionSummary.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the Bygroup1.Summary.SelectionSummary table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the Bygroup1.Summary.SelectionSummary table is not selected.

You can specify the following options after a slash (/):

CASESENSITIVE
performs a case-sensitive comparison of table names in the table-list to display table names when tables are subsetted for display. To preserve case, you must enclose table names in the table-list in quotation marks.

EXCLUDE
displays all display tables except those that you specify in the table-list.

EXCLUDEALL
suppresses display of all tables. This option takes precedence over the other options.
TRACE
  displays the display table names, labels, and paths.

DISPLAYOUT Statement

**DISPLAYOUT** *table-spec-list* </*options*> ;

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

The *table-spec-list* specifies a list of CAS output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

- **key=value** specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the CAS output table name.
- **key** specifies *key* as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “ODS Table Names” on page 295. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the DISPLAY statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

- **INCLUDEALL** creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the *table-spec-list* specification is ignored.

- **NOREPLACE** does not replace any existing CAS output table of the same name.

- **REPEATED** replicates all CAS output tables on all nodes.
### FREQ Statement

**FREQ** variable ;

The *variable* in the FREQ statement identifies a numeric variable in the input data table that contains the frequency of occurrence of each observation. PROC GVARCLUS treats each observation as if it appears *f* times, where *f* is the value of the FREQ *variable* for the observation. If *f* is not an integer, it is truncated to an integer. If *f* is less than 1 or missing, the observation is not used in the analysis. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

### INPUT Statement

**INPUT** variables / `<LEVEL=INTERVAL | NOMINAL>` ;

The INPUT statement specifies the names of *variables* to be used in training. Only interval, binary, and nominal variables are accepted. If you want to use different options for different variables, you can specify multiple INPUT statements.

You can specify the following *option* after a slash (/):

**LEVEL=INTERVAL | NOMINAL**

specifies whether the specified input *variables* are continuous or categorical. You can specify the following values:

- **INTERVAL** specifies that the input *variables* are continuous.
- **NOMINAL** specifies that the input *variables* are categorical.

By default, LEVEL=INTERVAL for numeric variables and LEVEL=NOMINAL for categorical variables. Binary variables are considered to be categorical variables.

### Details: GVARCLUS Procedure

The GVARCLUS procedure analyzes the 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. Zero elements 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 one. The divided clusters from sequential steps naturally form a hierarchical structure during the process.
### 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

PROC GVARCLUS uses the graphical lasso (Friedman, Hastie, and Tibshirani 2008) to estimate the inverse covariance matrix. Assume that the observations have a multivariate Gaussian distribution with mean $\mu$ and covariance $\Sigma$. 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 $p$, $X \sim N(\mu, \Sigma)$. 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 $\Theta$:

$$
\log \det \Theta - \text{Trace}(S\Theta) - \lambda \|\Theta\|_1
$$

Here $\det$ denotes the determinant and $\|\Theta\|_1$ is the $L_1$ norm, the sum of the absolute values of the elements of $\Sigma^{-1}$. The scalar parameter $\lambda$ controls the size of the penalty. The penalty term is a proxy for the number of nonzero elements in $\Theta$.

Let $W$ be the estimate of $\Sigma$. 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:

$$
\min \left\{ \frac{1}{2} \left\| W_{11}^{1/2} \beta - b \right\|^2 + \lambda \|\beta\|_1 \right\}
$$

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 $j = 1, 2, \ldots, p$, 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 $X_k$ though some chain of arcs, then $X_i$ is called a connected component of $X_k$.

The regularization parameter in the penalized log likelihood can be seen as a quasi-measure for the strength of connection between $X_i$ and $X_j$, because as $\lambda$ 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, PROC GVARCLUS creates a threshold for the entries of the sample covariance matrix at a regularization parameter $\lambda$ and decomposes the covariance matrix into connected components. In other words, if an entry of sample covariance is less than $\lambda$ 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 $\lambda$. 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 named 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.
Selection

To select the optimal $\lambda$ in the penalized log likelihood of the graphical lasso, PROC GVARCLUS minimizes the following fit statistics over all the steps,

$$- \log \det \hat{\Theta} + \text{Trace}(S\hat{\Theta}) + \sum_{1 \leq i < j \leq p} I(\hat{\theta}_{ij} \neq 0)$$

where $\hat{\Theta} = (\hat{\theta}_{ij})_{1 \leq i, j \leq p}$ denotes the estimates that are obtained at each step.

Displayed Output

The following sections describe the output that PROC GVARCLUS produces by default. The output is organized into various tables, which are discussed in the order of their appearance.

Number of Observations

The “Number of Observations” table displays the number of observations read from the input data table and the number of observations used in the analysis.

Cluster Summary

The “Cluster Summary” table displays the cluster structure for each step.

ODS Table Names

Each table that the GVARCLUS procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. The name of each table and a short description of its contents are listed in Table 13.2.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>NObs</td>
<td>Number of observations read and used; number of events and trials, if applicable</td>
<td>PROC GVARCLUS</td>
<td>Default</td>
</tr>
<tr>
<td>ClusterSummary</td>
<td>Summary of clustering results</td>
<td>PROC GVARCLUS</td>
<td>Default</td>
</tr>
<tr>
<td>SelectionSummary</td>
<td>Selection summary</td>
<td>PROC GVARCLUS</td>
<td>Default</td>
</tr>
<tr>
<td>ClassInfo</td>
<td>Level information for classification variables</td>
<td>PROC GVARCLUS</td>
<td>Default</td>
</tr>
</tbody>
</table>
Examples: GVARCLUS Procedure

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

Example 13.1: Analyzing United States Senate Voting Data

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:

```sas
data mycas.senate;
  set work.senate;
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 invoke the GVARCLUS procedure:

```sas
proc gvarclus data=mycas.senate rho=0.8 maxstep=5 outtree=mycas.tree
  outedge=mycas.edge outvert=mycas.vert;
  input _ALL_/LEVEL=NOMINAL;
run;
```

By specifying the MAXSTEP= option to be more than 1, you can let PROC GVARCLUS perform variable clustering at multiple $\rho$ values, which decrease at each step. RHO=0.8 specifies the initial $\rho$ value for step 1, and then $\rho$ is $0.8^2$ for step 2, $0.8^3$ for step 3, and so on. The regularization parameter $\rho$ 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 provided by PROC GVARCLUS are interpretable with tree output. The OUTTREE= option saves the tree to a CAS table, and you can plot the dendrogram using the SAS Graph Template language (GTL). PROC GVARCLUS also outputs the edge and vertex information for representing the partial correlations in an undirected graph. The OUTEDGE= option saves the edges in the network to a CAS table named mycas.edge, and the OUTVERT= option 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:

```sas
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, child node, and the height in the dendrogram. The dendrogram is then plotted using the SGRENDER procedure.

```sas
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 13.1.1 shows that there are two large clusters in the dendrogram. All the Democratic Senate votes are united and in one cluster. Most of the Republican Senate votes are in the other cluster, but there are some small cliques.
Example 13.2: Output a Covariance Matrix to a SAS Data File

This example shows how to output a covariance matrix to a SAS data file. The `OUTCP=` option creates an output data table named `mycas.cov`.

The following DATA step generates a data table that has 2,000 observations and contains both interval variables (x1–x2) and nominal variables (a, c1, and c2):

```sas
data mycas.data1;
  array x{2};
  array c{2};
  do i=1 to 2000;
    a=int(ranuni(1)*2);
    do j=1 to 2;
      x{j}=ranuni(1);
      c{j}=int(ranuni(1)*2);
  end;
run;
```
Example 13.2: Output a Covariance Matrix to a SAS Data File

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

The following statements invoke the GVARCLUS procedure:

```
title "Output the Covariance Matrix";

proc gvarclus data=mycas.data1 maxiter=4 outcp=mycas.cov;
   input c1 c2 /level=nominal;
   input x1 x2 /level=interval;
run;

proc print data=mycas.cov;
run;
```

Output 13.2.1 shows the content of the data file that PROC GVARCLUS generates.

Output 13.2.1 Output the Covariance Matrix

```
<table>
<thead>
<tr>
<th>Obs</th>
<th>ID</th>
<th>TYPE</th>
<th>VAR</th>
<th>ID</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>V5</th>
<th>V6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>MEAN</td>
<td></td>
<td></td>
<td>0.50</td>
<td>0.50</td>
<td>999.00</td>
<td>1001.00</td>
<td>975.00</td>
<td>1025.00</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>N</td>
<td></td>
<td></td>
<td>2000.00</td>
<td>2000.00</td>
<td>2000.00</td>
<td>2000.00</td>
<td>2000.00</td>
<td>2000.00</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>COV</td>
<td>x1</td>
<td>v1</td>
<td>165.55</td>
<td>-1.49</td>
<td>0.32</td>
<td>-0.32</td>
<td>5.24</td>
<td>-5.24</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>COV</td>
<td>x2</td>
<td>v2</td>
<td>-1.49</td>
<td>166.41</td>
<td>1.31</td>
<td>-1.31</td>
<td>1.32</td>
<td>-1.32</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>COV</td>
<td>c1</td>
<td>v3</td>
<td>0.32</td>
<td>1.31</td>
<td>500.00</td>
<td>-500.00</td>
<td>2.99</td>
<td>-2.99</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>COV</td>
<td>c1</td>
<td>v4</td>
<td>-0.32</td>
<td>-1.31</td>
<td>-500.00</td>
<td>500.00</td>
<td>-2.99</td>
<td>2.99</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>COV</td>
<td>c2</td>
<td>v5</td>
<td>5.24</td>
<td>1.32</td>
<td>2.99</td>
<td>-2.99</td>
<td>499.69</td>
<td>-499.69</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>COV</td>
<td>c2</td>
<td>v6</td>
<td>-5.24</td>
<td>-1.32</td>
<td>-2.99</td>
<td>2.99</td>
<td>-499.69</td>
<td>499.69</td>
</tr>
</tbody>
</table>
```

The _VAR_ column displays the names of all variables and the levels of the nominal variables. Assuming that you have n effects (the total number of interval variables and the levels of categorical variables), the _ID_ column contains n markers, v1 to vn, where vi denotes the ith effect. The column _TYPE_ defines the role of each row. When the _TYPE_ column displays MEAN/FREQ, the corresponding row contains either the mean for an interval variable or the frequency for a level of a nominal variable. When the _TYPE_ column displays N, the corresponding row contains the number of samples. And when the _TYPE_ column displays COV, the corresponding row contains a row of the covariance matrix. In this example, the covariance matrix is 4 × 4, and it resides in the table in rows 3–6 and columns 7–10.
Example 13.3: Output the Covariance Matrix in LIL Format

This example shows how to output a covariance matrix in list-of-list (LIL) format.

The following DATA step generates a data table that has 2,000 observations and contains both interval variables ($x_1$–$x_2$) and a categorical variable (a):

```r
data mycas.data2;
  array x(2);
  do i=1 to 2000;
    a=int(ranuni(1)*2);
    do j=1 to 2;
      x(j)=ranuni(1);
    end;
    output;
  end;
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 invoke the GVARCLUS procedure:

```r
title "Output the Covariance Matrix in LIL Format";

proc gvarclus data=mycas.data2 maxiter=4 outcp=mycas.cov_lil/list eps=0.01;
  input a /level=nominal;
  input x1-x2 /level=interval;
run;

proc print data=mycas.cov_lil;
run;
```

The OUTCP= option creates an output data table named `mycas.cov_lil`, which contains the covariance matrix. Output 13.3.1 shows the correlation matrix in LIL format.
Output 13.3.1 Output the Correlation Matrix in LIL Format

Output the Covariance Matrix in LIL Format

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>TYPE</em></th>
<th><em>ID</em></th>
<th><em>NAME1</em></th>
<th><em>NAME2</em></th>
<th><em>VAL</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S</td>
<td>1</td>
<td>samples</td>
<td></td>
<td>2000.00</td>
</tr>
<tr>
<td>2</td>
<td>S</td>
<td>2</td>
<td>nVar</td>
<td></td>
<td>3.00</td>
</tr>
<tr>
<td>3</td>
<td>S</td>
<td>3</td>
<td>nEff</td>
<td></td>
<td>4.00</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>1</td>
<td>x1</td>
<td></td>
<td>0.49</td>
</tr>
<tr>
<td>5</td>
<td>M</td>
<td>2</td>
<td>x2</td>
<td></td>
<td>0.50</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>3</td>
<td>a 0</td>
<td></td>
<td>979.00</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>4</td>
<td>a 1</td>
<td></td>
<td>1021.00</td>
</tr>
<tr>
<td>8</td>
<td>V</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>168.27</td>
</tr>
<tr>
<td>9</td>
<td>V</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0.58</td>
</tr>
<tr>
<td>10</td>
<td>V</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>166.44</td>
</tr>
<tr>
<td>11</td>
<td>V</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>8.37</td>
</tr>
<tr>
<td>12</td>
<td>V</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>0.46</td>
</tr>
<tr>
<td>13</td>
<td>V</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>499.78</td>
</tr>
<tr>
<td>14</td>
<td>V</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>-8.37</td>
</tr>
<tr>
<td>15</td>
<td>V</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>-0.46</td>
</tr>
<tr>
<td>16</td>
<td>V</td>
<td>9</td>
<td>4</td>
<td>3</td>
<td>-499.78</td>
</tr>
<tr>
<td>17</td>
<td>V</td>
<td>10</td>
<td>4</td>
<td>4</td>
<td>499.78</td>
</tr>
</tbody>
</table>

The column _TYPE_ defines the type of each row:

- When the _TYPE_ column displays S, the corresponding row contains the statistics of the data table. More specifically, when the _TYPE_ column displays S and the _NAME1_ column displays samples, the _VAL_ column in the corresponding row contains the number of samples in the data table. Similarly, when the _TYPE_ column displays S and the _NAME1_ column displays nVar, the _VAL_ column contains the number of variables. And when the _TYPE_ column displays S and the _NAME1_ column displays nEff, the _VAL_ column in the corresponding row contains the number of effects.

- When the _TYPE_ column displays F, the row contains the frequency of a level of a nominal variable. In this case, the _NAME1_ column contains the name and level of the nominal variable.

- When the _TYPE_ column displays M, the row contains the mean of an interval variable. In this case, the _NAME1_ column contains the name of the variable and the _NAME2_ column is empty.

- When the _TYPE_ column displays R, the row contains an entry in the correlation matrix. In this case, the _NAME1_ column contains the row ID, the _NAME2_ column contains the column ID, and the _VAL_ column contains the value.

- When the _TYPE_ column displays V or P, the corresponding row contains an entry of a COV matrix or an SSCP matrix, respectively.

Only entries in the lower triangle of the correlation matrix are written to the file, because the correlation matrix is symmetric. Also because EPS=0.01 is specified, any entry of the matrix whose value is smaller than 0.01 is ignored in the output; this saves storage space.
References


## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: KPCA Procedure</td>
<td>304</td>
</tr>
<tr>
<td>PROC KPCA Features</td>
<td>304</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>305</td>
</tr>
<tr>
<td>Getting Started: KPCA Procedure</td>
<td>306</td>
</tr>
<tr>
<td>Syntax: KPCA Procedure</td>
<td>310</td>
</tr>
<tr>
<td>PROC KPCA Statement</td>
<td>310</td>
</tr>
<tr>
<td>DISPLAY Statement</td>
<td>312</td>
</tr>
<tr>
<td>DISPLAYOUT Statement</td>
<td>313</td>
</tr>
<tr>
<td>ID Statement</td>
<td>314</td>
</tr>
<tr>
<td>INPUT Statement</td>
<td>314</td>
</tr>
<tr>
<td>KERNEL Statement</td>
<td>314</td>
</tr>
<tr>
<td>LRAPPROXIMATION Statement</td>
<td>315</td>
</tr>
<tr>
<td>OUTPUT Statement</td>
<td>316</td>
</tr>
<tr>
<td>SAVESTATE Statement</td>
<td>316</td>
</tr>
<tr>
<td>Details: KPCA Procedure</td>
<td>317</td>
</tr>
<tr>
<td>Details: The Exact Method</td>
<td>317</td>
</tr>
<tr>
<td>Low-Rank Approximation of KPCA Training Using the Nyström Method</td>
<td>318</td>
</tr>
<tr>
<td>Fast Scoring of KPCA</td>
<td>319</td>
</tr>
<tr>
<td>Displayed Output</td>
<td>319</td>
</tr>
<tr>
<td>ODS Table Names</td>
<td>320</td>
</tr>
<tr>
<td>Output Data Tables</td>
<td>320</td>
</tr>
<tr>
<td>Example: KPCA Procedure</td>
<td>321</td>
</tr>
<tr>
<td>Example 14.1: Letter Recognition</td>
<td>321</td>
</tr>
<tr>
<td>References</td>
<td>323</td>
</tr>
</tbody>
</table>
Overview: KPCA Procedure

Kernel principal component analysis (kernel PCA) is a nonlinear form of principal component analysis (Schölkopf, Smola, and Müller 1998). It uses the same basic idea as PCA; that is, it seeks to project the set of data onto a low-dimensional subspace that captures the highest possible amount of variance in the data. Whereas PCA performs a linear projection of the data onto a subset of the original space, kernel PCA uses a mapping function to embed the data in a high-dimensional reproducing kernel Hilbert space (RKHS) called $\mathcal{F}$ by a linear dimensionality reduction through the “kernel trick” in that space. Different kernels correspond to different mapping functions. This way, a nonlinear manifold (with respect to the original input space) can be found that contains the data. The applications of kernel PCA include nonlinear dimensionality reduction, nonlinear data classification, kernel principal component regression, image denoising, novelty detection, and so on.

The KPCA procedure performs kernel principal component analysis and stores the results in output tables that are produced by the EIGENVAL=, EIGENVEC=, and CENTROIDS= options. Also, the projection of the training data into the kernel principal components is stored by using the OUT= option in the OUTPUT statement.

PROC KPCA Features

The KPCA procedure has the following features:

- reads input data in parallel when the data source is on a distributed system
- is multithreaded during all phases of analytic execution
- supports large-scale training data
- enables you to choose among linear, polynomial, and radial basis function (Gaussian) kernels
- enables you to perform fast training and fast scoring of KPCA models
- enables you to perform the analysis on selected columns of the data
- enables you to project new data onto principal components that have captured the nonlinear relationship in the data
Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Chapter 14: The KPCA Procedure

Getting Started: KPCA Procedure

**Note:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example shows how to use the KPCA procedure to project two concentric data sets onto the first two kernel principal components. The following code shows how to generate these data sets by using the IML procedure and plot it by using the SGPLOT procedure. The output is shown in Figure 14.1.

```sas
proc iml;
    start Linspace(a, b, n);
        if n<2 then return( b );
        incr = (b-a) / (n-1);
        return( do(a, b, incr) );
    finish;

    start makecircles(X, y, n_samples, noise, random_state, factor);
        pi = constant("pi");
        lins=Linspace(0, 2*pi, floor(n_samples/2)+1);
        n_2=floor(n_samples/2);
        lins=remove(lins,n_2+1);
        outer_circ_x=cos(lins);
        outer_circ_y=sin(lins);
        inner_circ_x = outer_circ_x#factor;
        inner_circ_y = outer_circ_y#factor;

        X=(outer_circ_x||inner_circ_x)`||(outer_circ_y||inner_circ_y)``;
        y=(j(1,n_2,0)||j(1,n_2,1))``;
        Xy=X||y;

        /* shuffle observations */
        call randseed(random_state);
        /* sample size=5, rows chosen from 1:NumRows */
        obsIdx = sample(1:nrow(Xy), nrow(Xy), "NoReplace");
        X=Xy[obsIdx,1:ncol(Xy)-1];
        y=Xy[obsIdx,ncol(Xy)];
        Xn=j(nrow(X),2);
        if noise^=. then
            call randgen(Xn, "Normal", 0, noise);
            X=X+Xn;
    finish;

n_samples=1000; /* sample size */
noise=0.05;
random_state=0;
factor=0.3; /* radius ratio of inner circle to the outer circle */
```
run makecircles(X, y, n_samples, noise, random_state, factor);

circles=X||y;

create circles from circles[colname={"x" "y" "group"}];
append from circles;
close circles;
quit;

proc sgplot data=circles;
    styleattrs datasymbols=(Circle X);
    scatter x=x y=y / group=group markerattrs=(size=7px);
run;

Figure 14.1 Plot of the Two Concentric Circles

You can load the Circles data table into your CAS session by naming your CAS engine libref in the first statement of the following DATA step:
data mycas.circles;
set circles;
run;

The following code uses the KPCA procedure to project the data in the Circles data table onto the first two kernel principal components and then use PROC SGPLOT to plot these data in the reduced dimensions. The output is shown in Figure 14.2.

proc KPCA data=mycas.circles method=exact;
  input x y;
  output out=mycas.scored copyvar=group npc=2;
run;

proc sgplot data=mycas.scored;
  styleattrs datasymbols=(Circle X);
  scatter x=_PCS_1 y=_PCS_2 / group=group markerattrs=(size=7px);
run;

Figure 14.2 Projection of the Two Concentric Circles onto the First Two Kernel Principal Components
In Figure 14.2, you can see that after you project the data onto the first two kernel principal components, the two circles are linearly separable. However, as you can see in Figure 14.3, the projection of the data onto the first two regular principal components is not linearly separable. (Note that when you use the linear kernel type, a regular PCA is performed.) The following code runs the regular PCA on the Circles data table:

```sas
proc KPCA data=mycas.circles method=exact;
   input x y;
   kernel linear;
   output out=mycas.scoredpca copyvar=group npc=2;
run;
```

```sas
proc sgplot data=mycas.scoredpca;
   styleattrs datasymbols=(Circle X);
   scatter x=_PCS_1 y=_PCS_2 / group=group markerattrs=(size=7px);
run;
```

**Figure 14.3** Projection of the Two Concentric Circles onto the First Two Regular Principal Components
Chapter 14: The KPCA Procedure

Syntax: KPCA Procedure

The following statements are available in the KPCA procedure:

```
PROC KPCA <options> ;
   DISPLAY <table-list> </options> ;
   DISPLAYOUT table-spec-list </options> ;
   ID variables ;
   INPUT variables ;
   KERNEL kernel-type / kernel-parameters ;
   LRAPPROXIMATION <options> ;
   OUTPUT OUT=CAS-libref.data-table <option> ;
   SAVESTATE RSTORE=CAS-libref.data-table <option> ;
```

The INPUT statement is required. The following sections describe the PROC KPCA statement and then describe the other statements in alphabetical order.

PROC KPCA Statement

```
PROC KPCA <options> ;
```

The PROC KPCA statement invokes the procedure. Table 14.1 summarizes the options available in the PROC KPCA statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Data Table Option</strong></td>
<td></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the input data table</td>
</tr>
</tbody>
</table>

| **KPCA Options** |                                 |
| CENTER           | Centers the input data                  |
| METHOD=          | Specifies the method of performing KPCA      |
| NTHREADS=        | Specifies the number of threads to use on each computation node |
| RANKTHRESHOLD=   | Specifies the epsilon value to determine the rank of the kernel matrix |
| SCALE            | Scales the input data                    |

| **Output Table Options** |                                 |
| CENTROIDS=          | Specifies the name of the centroids output table |
| EIGENVAL=           | Specifies the name of the eigenvalue output table |
| EIGENVEC=           | Specifies the name of the eigenvector output table |

You can specify the following options:
**CENTER**  
centers the observations by the mean of each column.

**CENTROIDS=**CAS-libref.data-table  
specifies the name of the output table for the centroids that are generated by clustering. This table is generated if METHOD=APPROXIMATE.

*CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 305.

**DATA=**CAS-libref.data-table  
names the input data table for PROC KPCA to use. The default is the most recently created data table.  
*CAS-libref.data-table* is a two-level name, where  

*CAS-libref* refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 305.

*data-table* specifies the name of the input data table.

**EIGENVAL=**CAS-libref.data-table  
specifies the name of the output table that contains the eigenvalues.

*CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 305.

**EIGENVEC=**CAS-libref.data-table  
specifies the name of the output table that contains the eigenvectors.

*CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 305.

**METHOD=**APPROXIMATE | EXACT  
specifies the method to use in performing KPCA. You can specify the following values:

- **APPROXIMATE** uses the low-rank approximation method.
- **EXACT** uses the exact method.

By default, METHOD=APPROXIMATE.
**NTHREADS=** number-of-threads

specifies the number of threads per computation node. The default value is the lesser of 16 and the number of threads available per computation node.

**RANKTHRESHOLD=** number

specifies the value of epsilon for determining the rank of the kernel matrix. The default value is $10^{-8}$.

**SCALE**

scales the observations by the standard deviation of each column. If a constant variable exists (where all observations have the same value), the observations of this column are not scaled.

---

**DISPLAY Statement**

**DISPLAY** < table-list > < / options > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the **table-list** as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “ODS Table Names” on page 320. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path Bygroup1.Summary.SelectionSummary. A partial pathname does not include all groups; for example, SelectionSummary and Summary.SelectionSummary are partial pathnames for Bygroup1.Summary.SelectionSummary.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both SelectionSummary and Summary.SelectionSummary select Bygroup1.Summary.SelectionSummary.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the Bygroup1.Summary.SelectionSummary table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the Bygroup1.Summary.SelectionSummary table is not selected.

You can specify the following **options** after a slash (/):
CASESENSITIVE
performs a case-sensitive comparison of table names in the table-list to display table names when tables are subsetted for display. To preserve case, you must enclose table names in the table-list in quotation marks.

EXCLUDE
displays all display tables except those that you specify in the table-list.

EXCLUDEALL
suppresses display of all tables. This option takes precedence over the other options.

TRACE
displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT table-spec-list < / options > ;

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see SAS Output Delivery System: Procedures Guide.

The table-spec-list specifies a list of CAS output tables to create. Each entry in the list has either a key=value format or a key format:

key=value specifies key as the ODS table name, path, or partial pathname, and specifies value as the CAS output table name.

key specifies key as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “ODS Table Names” on page 320. You cannot specify the ODS table named OutputCasTables in the table-spec-list.

Table names and partial pathnames are discussed under the DISPLAY statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following options after a slash (/):

INCLUDEALL
creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the table-spec-list specification is ignored.

NOREPLACE
does not replace any existing CAS output table of the same name.
REPEATED
replicates all CAS output tables on all nodes.

**ID Statement**

```
ID variables ;
```

The ID statement lists one or more variables to be copied from the input data table to the output data tables that are specified in the OUTPUT statement and the EIGENVEC= option in the PROC KPCA statement, and also to the file that is specified in the RSTORE= option in the SAVESTATE statement.

**INPUT Statement**

```
INPUT variables ;
```

The INPUT statement specifies the names of variables to be considered in the KPCA procedure. The procedure accepts numeric variables as input. Nonnumeric variables are ignored. This statement is required.

**KERNEL Statement**

```
KERNEL kernel-type / kernel-parameter ;
```

The KERNEL statement specifies the type of kernel and any associated parameters to use during training. You can specify one of the following kernel-types and its associated kernel-parameters:

- **LINEAR** specifies that the kernel type is linear. When the kernel type is linear, regular PCA is performed.

- **POLYNOMIAL / DEGREE=number INTERCEPT=number** specifies that the kernel type is in the polynomial form. The degree of the polynomial is a positive integer value that is specified in the DEGREE= option. By default, DEGREE=3. The intercept is a numeric value that is specified in the INTERCEPT= option. By default, INTERCEPT=1.

- **RBF / BW=s** uses a radial basis function (also known as the Gaussian kernel function) during training. The kernel is defined as

\[ K(x, y) = \exp \left( -\frac{\|x - y\|^2}{2s^2} \right) \]

where \( x \) and \( y \) are vectors and \( s \) is the bandwidth parameter. You can specify the bandwidth parameter in the form BW=s, where \( s \) must be a positive nonzero real number. The processing time usually increases as the value of \( s \) decreases. At very small values of \( s \), the processing time can be very long. By default, the value of BW = \( \sqrt{\frac{0.25}{n(n-1)} \sum_{i,j=1}^{n} \|x_i - x_j\|^2} \), where \( n \) is the number of rows in the training data. This choice of default value comes from the idea in Kwok and Tsang (2004). The default parameter is not guaranteed to produce good results. It is recommended that you tune this parameter by using some systematic approach, such as cross validation.
LRAPPROXIMATION Statement

LRAPPROXIMATION <options> ;

The LRAPPROXIMATION statement specifies the parameters of the low-rank approximation method.

You can specify the following options:

**CCRITERION=number**  
specifies a double value for the convergence criterion in \( k \)-means clustering. By default, CCRITERION=0.02.

**CLUSMETHOD=FC | KMPP | RANDOM**  
specifies the method of seed initialization in \( k \)-means clustering to generate the centroids. You can specify one of the following seed initialization options:

**FC**  
stands for fast clustering and spreads out the initial seeds in a heuristic way.

**KMPP / <RANDOMSEED=number>**  
spreads out the initial seeds by introducing a weighted probability distribution in seed selection. For more information, see Arthur and Vassilvitskii (2007).

**RANDOM / <RANDOMSEED=number NRESTART=number>**  
selects initial seeds by pure randomization.

Note that if CLUSMETHOD=KMPP or RANDOM, you can specify an integer value for the RANDOMSEED= option. If you do not specify a value for the RANDOMSEED= option, the seed is generated by reading the time of day from the computer’s clock. If you want reproducible results, specify an integer for number. Also, if CLUSMETHOD=RANDOM, you can specify the number of restarts by using the NRESTART= option, whose value should be an integer. By default, NRESTART=5. Also, by default, CLUSMETHOD=KMPP.

**MAXCLUS=number**  
specifies the maximum number of clusters to use in \( k \)-means clustering. By default, MAXCLUS=100.

**MAXITER=number**  
specifies the maximum number of iterations to use in \( k \)-means clustering. By default, MAXITER=50.
Chapter 14: The KPCA Procedure

**OUTPUT Statement**

```
OUTPUT OUT=CAS-libref.data-table < option > ;
```

The OUTPUT statement creates an output data table to contain the results of the procedure run.

You must specify the following option:

**OUT=CAS-libref.data-table**
	names the output data table for PROC KPCA to use. You must specify this option before any other options. `CAS-libref.data-table` is a two-level name, where

- `CAS-libref` refers to a collection of information that is defined in the LIBNAME statement and includes the `caslib`, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 305.

- `data-table` specifies the name of the output data table.

You can also specify the following **option**:

**NPC=number**

specifies the number of principal components to display.

---

**SAVESTATE Statement**

```
SAVESTATE RSTORE=CAS-libref.data-table < option > ;
```

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 4, “The ASTORE Procedure.”

You must specify the following option:

**RSTORE=CAS-libref.data-table**

specifies the name of the table in which to save the analytic store. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the `caslib` and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the `DATA=` option and the section “Using CAS Sessions and CAS Engine Librefs” on page 305.

You can also specify the following **option**:

**EXACTSCORE**

specifies the exact method of scoring. If this option is not specified and `METHOD=APPROXIMATE` in the PROC KPCA statement, then fast (approximate) scoring is used. However, if this option is not specified and `METHOD=EXACT` in the PROC KPCA statement, then exact scoring is used. For more information about fast scoring, see the section “Fast Scoring of KPCA” on page 319.

You can use the ASTORE procedure to specify the number of principal components to project the scoring observations. To do this, specify the option `KPCA_NPC` in the SETOPTION statement in PROC ASTORE.
Details: KPCA Procedure

Kernel principal component analysis (KPCA) is a nonlinear form of principal component analysis. In this section, the methods that PROC KPCA implements are described.

Details: The Exact Method

This section describes the exact KPCA method. For more information, see Schölkopf, Smola, and Müller (1998).

Suppose that the mean of the data in a reproducing kernel Hilbert space (RKHS) is

$$\mu = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i) = 0$$

where $x_i$ is a $q \times 1$ data point in the original space and $\phi(\cdot)$ is the mapping function that maps the original data to some high-dimensional RKHS. Then, assuming that the data vector $x_i$ is already centered, the covariance matrix is

$$C = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i)\phi(x_i)^T$$

The eigensystem of $C$ is

$$Cv = \lambda v$$

Schölkopf, Smola, and Müller (1998) show that the eigenvectors can be expressed as a linear combination of features in RKHS:

$$v = \sum_{i=1}^{n} \alpha_i \phi(x_i)$$

Therefore, finding the eigenvectors is equivalent to finding the coefficients $\alpha_i$.

By substituting the preceding equation into the equation immediately before it, for the $j$th eigenvector, you get

$$\frac{1}{n} \sum_{i=1}^{n} \phi(x_i)\phi(x_i)^T \left( \sum_{l=1}^{n} \alpha_{jl} \phi(x_l) \right) = \lambda_j \sum_{l=1}^{n} \alpha_{jl} \phi(x_l)$$

Then, plugging in the kernel and rearranging the equation, you get

$$K^2 \alpha_j = n \lambda_j K \alpha_j$$

where the $n \times n$ matrix $K$ is called the kernel matrix, whose $k/1$th element is the value of kernel function $k(x_k, x_l) = \phi(x_k)^T \phi(x_l)$; and $\alpha_j$ is an $n \times 1$ vector, whose $l$th element is the coefficient $\alpha_{jl}$.
Next, factor $K$ from both sides is removed, and finally the eigensystem can be written in terms of kernel matrix $K$ as

$$K\alpha_j = n\lambda_j \alpha_j$$

Then, a normalization condition is applied such that the eigenvector $v_j$ in the RKHS satisfies

$$v_j^T v_j = 1 \implies \sum_{k=1}^{n} \sum_{l=1}^{n} \alpha_{jl} \alpha_{jk} \phi(x_l)^T \phi(x_k) = 1 \implies \alpha_j^T K \alpha_j = 1$$

By left-multiplying $K\alpha_j = n\lambda_j \alpha_j$ by $\alpha_j^T$ and using the normalization condition, you get

$$n\lambda_j \alpha_j^T \alpha_j = 1 \quad \forall \, j$$

The projection of a new or old data point $x$ onto the $j$th principal component is

$$\phi(x)^T v_j = \sum_{i=1}^{n} \alpha_{ji} \phi(x)^T \phi(x_i) = \sum_{i=1}^{n} \alpha_{ji} K(x, x_i)$$

---

**Low-Rank Approximation of KPCA Training Using the Nyström Method**

To train an exact KPCA model on the input matrix $M$ of size $n \times m$, the full kernel matrix $K \in \mathbb{R}^{n \times n}$ needs to be constructed, and the computationally expensive eigendecomposition operation, with a computational complexity of $O(n^3)$, must be applied to $K$. Thus, for large values of $n$, the exact KPCA method can be very slow. To address this issue, an approximation method called the Nyström method was proposed by Baker (1977). This method is based on sampling from the rows of the input matrix. Instead of applying the eigendecomposition method to the full kernel matrix $K$, you apply it to a smaller matrix of size $c \times c$, where $c$ is the sampling size.

The sampling scheme is important in the performance of the Nyström method. In Kwok and Tsang (2004), an error analysis is conducted, and the resulting error bound suggests using $k$-means clustering as the sampling scheme. The greater the number of clusters that are chosen in $k$-means clustering, the more accurate the reconstruction of $K$ will be. If the number of clusters equals the number of observations, the reconstruction will be exact. In $k$-means clustering, the algorithm for choosing the initial values (or seeds) has significant impact on the centroids that are obtained. PROC KPCA includes several algorithms for clustering initialization, among them $k$-means++ (Arthur and Vassilvitskii 2007). Low-rank approximation reduces the memory complexity of KPCA training from $O(n^2)$ to $O(nc)$ and reduces the computational complexity of kernel PCA from $O(n^3)$ to $O(nc^2)$, where $n$ is the data size and $c$ is the number of clusters.
Fast Scoring of KPCA

In addition to enabling fast and memory-efficient KPCA training, the low-rank approximation method also provides a fast and memory-efficient way to do scoring—namely, calculating KPCA score values for test data. In exact KPCA, to score test data the whole training data set needs to be retained and the kernel matrix between test data and training data needs to be calculated. However, in the low-rank approximation method, only the centroids that result from the $k$-means clustering of the training data need to be retained, and only the kernel matrix between the test data and these centroids needs to be calculated. This reduces both the amount of memory required and the computational complexity of KPCA scoring from $O(n)$ to $O(c)$, where $n$ is the data set size and $c$ is the number of clusters.

Displayed Output

The KPCA procedure displays various tables that are related to input and results. The following sections describe the output tables in the order of their appearance.

Model Information

The “Model Information” table displays basic information about the parameters that are used in the procedure. This information includes the number of interval variables, the kernel type and kernel parameters, the threshold for nonzero eigenvalues, and the number of principal components that are generated.

$k$-means Clustering Information

The “$k$-means Clustering Information” table displays the parameters that are used in the $k$-means clustering method. This table is displayed only if METHOD=APPROXIMATE in the PROC KPCA statement. These parameters include the number of clusters, random seed, seed initialization method for $k$-means clustering, maximum number of iterations, and convergence criterion.

Number of Observations

The “Number of Observations” table displays the number of observations that are read and used in the model. Observations that have missing values for at least one of the input variables are ignored.

Descriptive Statistics

The “Descriptive Statistics” table displays the mean and standard deviation of the input variables.

Task Timing

The “Task Timing” table displays the time in seconds and the percentage of time consumed by the various steps of the KPCA method. These steps include kernel matrix construction, $k$-means clustering, eigendecomposition, and scoring the training data.
## ODS Table Names

Each table that the KPCA procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. The names of each table and a short description of the contents are listed in Table 14.2.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClusteringInfo</td>
<td>k-means parameters</td>
<td>PROC KPCA</td>
<td>Default</td>
</tr>
<tr>
<td>DescStatsInt</td>
<td>Input variable information</td>
<td>PROC KPCA</td>
<td>Default</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>PROC KPCA</td>
<td>Default</td>
</tr>
<tr>
<td>NObs</td>
<td>Number of observations</td>
<td>PROC KPCA</td>
<td>Default</td>
</tr>
<tr>
<td>TaskTiming</td>
<td>Breakdown of the time spent in each step</td>
<td>PROC KPCA</td>
<td>Default</td>
</tr>
</tbody>
</table>

## Output Data Tables

The KPCA procedure creates output tables for eigenvalues (if you specify the EIGENVAL= option in the PROC KPCA statement), eigenvectors (if you specify the EIGENVEC= option in the same statement), or centroids (if you specify the CENTROIDS= option in the same statement). Also, you can score the training data and store the results in the table that you specify in the OUT= option in the OUTPUT statement. Table 14.3 lists details about these data tables.

<table>
<thead>
<tr>
<th>Data Table</th>
<th>Content</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eigenval</td>
<td>Eigenvalues of the kernel matrix</td>
<td>Number of eigenvalues × 1</td>
</tr>
<tr>
<td>Eigenvvec</td>
<td>Eigenvectors of the kernel matrix</td>
<td>Number of observations × number of eigenvalues</td>
</tr>
<tr>
<td>Centroids</td>
<td>Centroids obtained from clustering</td>
<td>Number of centroids × number of variables</td>
</tr>
<tr>
<td>Out</td>
<td>Projection of training observations onto the kernel principal components</td>
<td>Number of observations × number of eigenvalues</td>
</tr>
</tbody>
</table>
Example: KPCA Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

Example 14.1: Letter Recognition

In this example, the KPCA procedure is used as a preprocessor to the linear discriminant model for letter recognition.

The letter recognition data set that this example uses is from the UCI Machine Learning Repository (Dua and Graff 2019). The data set includes a large number of black-and-white pixel images of rectangular shape, each of them corresponding to one of the 26 capital letters in the English alphabet. These letter pictures have 20 different fonts, and each picture was randomly altered to generate a total of 20,000 unique instances. Each instance was then transformed into 16 statistical attributes (edge counts and statistical moments). All these attributes were further scaled to a range of integer values from 0 to 15 (Frey and Slate 1991). The objective is to classify each pixel image as one of the 26 capital letters. In this example, a training set of size 16,000 is randomly selected, and the remaining 4,000 instances are used as a test set. Because this is a big data set, fast KPCA is a more appropriate method to use for training while still achieving performance comparable to that of the exact method. To apply fast KPCA, 200 centroids are selected. After the model is trained, the projection values of training and scoring data onto the kernel principal components undergo a multilabel linear discriminant analysis (LDA) for classification.

In the following code, the PROC KPCA statement applies exact and fast KPCA to the data table mycas.letter_train. The statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref. Next, the PROC DISCRIM statement applies linear discriminant analysis to the projections onto the principal component directions. Here letter_train and letter_test are the names of the training data set and test data set, respectively.

```plaintext
data letter;
  infile datalines delimiter=',';
  input capital $ var1-var16;
datalines;
T,2,8,3,5,1,8,13,0,6,6,10,8,0,8,0,8

  ... more lines ...

data letter;
  set letter;
  obsid=_n_;
run;

/* Split data set into train (16000) and test(4000) */
proc surveyselect data=letter
```
Chapter 14: The KPCA Procedure

```plaintext
method=srs n=16000
  seed=100  out=letter_train;
run;

proc sql;
create table letter_test as
select * from letter
where obsid not in (select obsid from letter_train);
quit;
/* Load data to CAS server */
data mycas.letter_train; set letter_train; run;
data mycas.letter_test; set letter_test; run;

/* Apply exact KPCA to training data, extract 200 principal components*/
proc kpca data=mycas.letter_train method=exact;
  input var1-var16;
  kernel RBF/bw=7.071;
  output out=mycas.score copyvars=(capital obsid) NPC=200;
  savestate rstore=mycas.state;
run;
/* Score test data */
proc astore;
  setoption kpca_npc 200;
  score data=mycas.letter_test rstore=mycas.state
  out=mycas.outscore copyVars=(capital obsid);
quit;
/* Apply linear discriminant analysis and get multilabel classification error */
proc discrim data=mycas.score method=normal pool=yes short
testdata=mycas.outscore;
  class capital;
  testclass capital;
run;

/* Apply fast KPCA to training data, extract 200 principal components*/
proc kpca data=mycas.letter_train method=approximate;
  input var1-var16;
  lrapproximation clusmethod=KMPP maxclus= 200;
  kernel RBF/bw=7.071;
  output out=mycas.score_fast copyvars=(capital obsid) NPC=200;
  savestate rstore=mycas.state_fast;
run;
/* Fast score test data */
proc astore;
  setoption kpca_npc 200;
  score data=mycas.letter_test rstore=mycas.state_fast
  out=mycas.outscore_fast copyVars=(capital obsid);
quit;
```
In this example, the multilabel classification errors that are obtained by fast KPCA and exact KPCA are close: 0.1662 for fast KPCA and 0.1593 for exact KPCA. However, the training time of fast KPCA is only 10.57 seconds, compared to 272.78 seconds for exact KPCA. The example demonstrates the efficiency of fast KPCA in significantly reducing the run time while not compromising much on the quality of the principal components that it generates. Moreover, in exact KPCA, all the training data along with the eigenvector matrix in the state file must be stored for scoring the test data, whereas in fast KPCA only the $k$-means centroids and eigenvector matrix must be stored in the state file. This greatly reduces the space requirement to score new data.

You can use the following code to also apply PCA (equivalent to KPCA with linear kernel) to the training data and extract all 16 components to use in linear discriminant analysis:

```sas
/* Apply linear discriminant analysis and get multilabel classification error */
proc discrim data=mycas.score_fast method=normal pool=yes short
testdata=mycas.outscore_fast;
class capital;
testclass capital;
run;
```

References


Chapter 15
The MBANALYSIS Procedure

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: MBANALYSIS Procedure</td>
<td>326</td>
</tr>
<tr>
<td>PROC MBANALYSIS Features</td>
<td>326</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>326</td>
</tr>
<tr>
<td>Getting Started: MBANALYSIS Procedure</td>
<td>327</td>
</tr>
<tr>
<td>Syntax: MBANALYSIS Procedure</td>
<td>328</td>
</tr>
<tr>
<td>PROC MBANALYSIS Statement</td>
<td>328</td>
</tr>
<tr>
<td>CUSTOMER Statement</td>
<td>330</td>
</tr>
<tr>
<td>HIERARCHY Statement</td>
<td>330</td>
</tr>
<tr>
<td>OUTPUT Statement</td>
<td>331</td>
</tr>
<tr>
<td>SAVESTATE Statement</td>
<td>331</td>
</tr>
<tr>
<td>TARGET Statement</td>
<td>332</td>
</tr>
<tr>
<td>Details: MBANALYSIS Procedure</td>
<td>332</td>
</tr>
<tr>
<td>FP-Growth Algorithm</td>
<td>333</td>
</tr>
<tr>
<td>Scoring with PROC ASTORE</td>
<td>334</td>
</tr>
<tr>
<td>Displayed Output</td>
<td>335</td>
</tr>
<tr>
<td>Results</td>
<td>335</td>
</tr>
<tr>
<td>Limitations</td>
<td>335</td>
</tr>
<tr>
<td>Examples: MBANALYSIS Procedure</td>
<td>336</td>
</tr>
<tr>
<td>Example 15.1: Training a Model</td>
<td>336</td>
</tr>
<tr>
<td>References</td>
<td>337</td>
</tr>
</tbody>
</table>
Overview: MBANALYSIS Procedure

The MBANALYSIS (market basket analysis) procedure performs association rule mining on a transaction database. Association rules are in the form of if A then B, where A and B are items in the transaction database. Association rules help determine association or correlation between various items in a database. These rules are used for finding the latest trends and patterns in data and also help in business decision-making processes. The MBANALYSIS procedure uses the frequent-pattern growth (FP-growth) algorithm of Han, Pei, and Yin (2000) for finding frequent item sets and then generates rules that are based on these frequent item sets. For more information about the FP-growth algorithm, see the section “FP-Growth Algorithm” on page 333.

PROC MBANALYSIS Features

The MBANALYSIS procedure has the following features:

- reads input data in parallel when the data source is on a distributed system
- processes data in a single thread as well as in multiple threads on a single node or on multiple nodes
- supports hierarchical transactional data as input
- uses the FP-growth algorithm for frequent item set mining
- supports various measures of interestingness to find useful rules
- supports the SAVESTATE statement for scoring by the ASTORE procedure

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```sas
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:
cas mysess;
libname mycas cas sessref=mysess;

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

cas mysess terminate;

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”

---

**Getting Started: MBANALYSIS Procedure**

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example finds frequent item sets in the sampsio.assocs data set and generates rules for finding association between those frequent item sets. The sampsio.assocs data set contains 1,000 observations, each of which contains products purchased by a customer.

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:

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

These statements assume that your CAS engine libref is named mycas, as in the section “Using CAS Sessions and CAS Engine Librefs” on page 326, but you can substitute any appropriately defined CAS engine libref.

The following statements execute the MBANALYSIS procedure on the mycas.assocs data table and generate output in the mycas.assocs data table:

proc mbanalysis data=mycas.assocs items=3 support=100;
  output out=mycas.out outfreq=mycas.outfreq outrule=mycas.outrule;
  customer Customer;
  target product;
run;

The PROC MBANALYSIS statement specifies mycas.assocs as the input data table and specifies the rule generation criteria: the number of items and level of support. The OUTPUT statement specifies output tables for storing the results. The CUSTOMER statement identifies Customer as identification variable for the transactions. The TARGET statement defines Product as the target variable.
The procedure stores the list of frequent items in the mycas.outfreq table, the frequent item sets in the mycas.out table, and the generated rules in the mycas.outrule table. The mycas.outrule table contains the rules that are generated for the frequent item sets after rule generation criteria are applied.

**Syntax: MBANALYSIS Procedure**

The following statements are available in the MBANALYSIS procedure:

```plaintext
PROC MBANALYSIS options ;
    CUSTOMER variable ;
    HIERARCHY DATA=CAS-libref.data-table < CAS-libref.data-table . . . > ;
    OUTPUT options ;
    SAVESTATE RSTORE=CAS-libref.data-table ;
    TARGET variable ;
```

The PROC MBANALYSIS statement, the CUSTOMER statement, the TARGET statement, and the OUTPUT statement are required.

The following sections describe the PROC MBANALYSIS statement and then describe the other statements in alphabetical order.

**PROC MBANALYSIS Statement**

```plaintext
PROC MBANALYSIS options ;
```

The PROC MBANALYSIS statement invokes the procedure.

You must specify the following option:

**DATA=**CAS-libref.data-table

names the input data table for PROC MBANALYSIS to use. `CAS-libref.data-table` is a two-level name, where

- `CAS-libref` refers to a collection of information that is defined in the LIBNAME statement and includes the `caslib`, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 326.

- `data-table` specifies the name of the input data table.

You must specify either of the following options:

**SUPPORT=**number

specifies the minimum level of support (minimum frequency of an item) for a rule, where `number` must be an integer greater than or equal to 0. This option overrides the specification of the PCTSUP= option.
**PROC MBANALYSIS Statement**

- **PCTSUP=** *number*
  - Specifies the minimum level of support for a rule as a percentage of the number of baskets in the input data table, where *number* must be a real number between 0 and 100, inclusive. This option is ignored if the **SUPPORT=** option is specified.

You can also specify the following **options**:

- **ANTECEDENTLIST=**(string < string . . . >)
  - Specifies the regular expression strings to match in the antecedent (left-hand side) of a rule.

- **CONF=** *number*
  - Specifies the minimum confidence for the rules, where *number* must be a real number between 0 and 100. By default, **CONF=50**.

- **CONSEQUENTLIST=**(string < string . . . >)
  - Specifies the regular expression strings to match in the consequent (right-hand side) of a rule.

- **ITEMS=** *number*
  - Specifies the number of items in a rule, where *number* must be an integer between 1 and 1,000. By default, **ITEMS=2** when you specify either an **OUT=** or **OUTRULE=** option in the **OUTPUT** statement; otherwise, **ITEMS=1** by default.

- **LIFT=** *number*
  - Specifies the minimum lift value necessary to generate a rule, where *number* must be a positive, real number between 0 and 100, inclusive. By default, **LIFT=1**.

- **MAXBSKTSZ=** *number*
  - Specifies the maximum basket size, where *number* must be a positive integer between 0 and 1,000, inclusive. Baskets whose size is larger than *number* are rejected. By default, **MAXBSKTSZ=1000**.

- **MINBSKTSZ=** *number*
  - Specifies the minimum basket size, where *number* must be a positive integer between 0 and 20, inclusive. Baskets whose size is smaller than *number* are rejected. By default, **MINBSKTSZ=1**.

- **NLHS_RANGE=** *(number, number)*
  - Specifies the range of number of items in the left-hand side (LHS) of a rule. By default, **NLHS_RANGE=(1,1000)**.
NORM
normalizes the values of the items in the output tables that are specified in the OUTPUT statement.

NRHS_RANGE=(number, number)
specifies the range of number of items in the right-hand side (RHS) of a rule.
By default, NRHS_RANGE=(1,1000).

SEPARATOR=string
specifies the separator character in the antecedent (left-hand side) or consequent (right-hand side) of a rule.
By default, SEPARATOR="&".

SUP_LIFT=number
specifies the minimum support lift necessary to generate a rule, where number must be a positive, real number.
By default, SUP_LIFT=0.

CUSTOMER Statement
CUSTOMER variable ;
The CUSTOMER statement specifies the variable that is used to group the target variable into a basket. You can also specify this statement as ID variable.

HIERARCHY Statement
HIERARCHY DATA=CAS-libref.data-table < CAS-libref.data-table . . . > ;
The HIERARCHY statement specifies one or more hierarchical input data tables. If this statement is not specified, then PROC MBANALYSIS performs simple association analysis without a hierarchy. A maximum of five levels of hierarchy is supported. Each level of hierarchy is specified in a separate data table.
You must specify the following option:

DATA=CAS-libref.data-table < CAS-libref.data-table . . . >
specifies one or more data tables that contain the hierarchy data. A maximum of five levels of hierarchy is supported. Each level of hierarchy is specified in a separate data table. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 326.
**OUTPUT Statement**

```r
OUTPUT options ;
```

The OUTPUT statement specifies output tables that contain result of the MBANALYSIS procedure.

You must specify at least one of the following options:

- **OUT=** `CAS-libref.data-table` creates a data table that contains information about frequent item sets with their transaction counts and support. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 326.

- **OUTFREQ=** `CAS-libref.data-table` creates a data table that contains information about the unique frequent items with their transaction counts and support. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 326.

- **OUTRULE=** `CAS-libref.data-table` creates a data table that contains information about the rules. This table includes variables that identify the left-hand side and right-hand sides of the rule, the support, and the lift. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 326.

**SAVESTATE Statement**

```r
SAVESTATE RSTORE=CAS-libref.data-table ;
```

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 4, “The ASTORE Procedure.”

You must specify the following option:

- **RSTORE=** `CAS-libref.data-table` specifies a data table in which to save the analytic store for the model. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 326.
**TARGET Statement**

TARGET variable ;

The TARGET statement specifies a single nominal variable to use as the target variable. If this variable is not present in the input data table, then PROC MBANALYSIS exits with an error.

---

**Details: MBANALYSIS Procedure**

The MBANALYSIS procedure performs association rule mining 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 popularly 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. PROC MBANALYSIS 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 below, 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, but it is possible that none of these support levels is large enough to generate a rule. To overcome this problem, the MBANALYSIS procedure 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 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 that 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 rules are useful. One solution to this drawback is generalized association rule mining.

The MBANALYSIS procedure enables you to generate an interestingness 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 based on 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 PROC MBANALYSIS generates simple association rules.

For example, the following rule for 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 rule have been found to be true. Confidence of a rule for $A$ and $B$ is expressed as

$$A \Rightarrow B : P(B|A) = \frac{X}{Z}$$

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

**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 in 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 algorithms can reduce the search space by doing a restricted generation-and-test process. The Apriori algorithm is the one of the most efficient association rule mining algorithms.

The Apriori algorithm of Agrawal and Srikant (1994) 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 links 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.

When items are sorted in support-descending order, more frequent items are arranged closer to the top of the FP-tree and thus are more likely to be shared. This arrangement suggests that an FP-tree is usually a
highly compact projection of the entire data set. However, if the data set 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 sets could be multiple times the size of the original data set.

- **Partition projection** is done in sequence: an observation $T$ is projected to the $X$-projected data set 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 sets at each level is smaller than the original data set.

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

### Scoring with PROC ASTORE

The SAVESTATE statement stores the analytical results in a binary file, which can be used as an input file for scoring with the ASTORE procedure. The transactional data table is in long format and PROC ASTORE accepts input in wide format, so the input data format must be converted from long to wide before the data are provided as input to PROC ASTORE.

The scoring process is supported by three algorithms, which you can specify in the SETOPTION statement in the ASTORE procedure. Select the algorithm you want to use by specifying SETOPTION ARM_SCORING_ALGO number, where number is one of the following values:

1. Uses a rule association algorithm. For a particular input item set, a rule is selected if its antecedent and consequent item sets are included in the input item set.
2. Uses an exclusive recommendation algorithm. For a particular input item set, a rule is selected if its antecedent item set is a subset of the input item set and its consequent item set is not a subset of the input item set.
3. Uses a recommendation algorithm. For a particular input item set, a rule is selected if its antecedent item set is a subset of the input item set.

In order to maintain consistency, the output of PROC ASTORE needs to be converted from wide to long format.
Displayed Output

The MBANALYSIS procedure does not produce any displayed output tables.

Results

The results that the MBANALYSIS procedure produces are stored in tables. You can access these tables by their names, which are shown in Table 15.1.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Procedure</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>Contains general information about frequent item sets</td>
<td>PROC MBANALYSIS</td>
<td>OUT</td>
</tr>
<tr>
<td>outfreq</td>
<td>Contains information about the items’ frequencies</td>
<td>PROC MBANALYSIS</td>
<td>OUTFREQ</td>
</tr>
<tr>
<td>outrule</td>
<td>Contains detailed information about rules</td>
<td>PROC MBANALYSIS</td>
<td>OUTRULE</td>
</tr>
<tr>
<td>savestate</td>
<td>Contains the output binary scoring file</td>
<td>PROC MBANALYSIS</td>
<td>SAVESTATE</td>
</tr>
</tbody>
</table>

Limitations

When the number of observations is greater than 10 million and the number of items in a rule (as specified in the ITEMS= option in the PROC MBANALYSIS statement) is considerably large (more than 10), the amount of memory 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.
Examples: MBANALYSIS Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

**Example 15.1: Training a Model**

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

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 transaction contains information about the customer, time, and item or product. The customer (ID) variable is named `Customer`. The target variable is named `Product`, and the time variable is ignored. To perform association discovery, the input data table must have a separate observation for each product that is purchased by each customer. You must assign the ID role to one variable and the target model role to another variable when you create the data source. 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.

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

The following DATA steps create two hierarchies, `mycas.firstLevel` and `mycas.secondLevel`:

```sas
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;
```
The following statements run the association analysis on the mycas.assocs data table:

```plaintext
proc mbanalysis data=mycas.assocs items=10 support=1 separator="\*"
  nLHS_range=(2,3) nRHS_range=(4,5) antecedentList="("Fruits")
  consequentList="("Non-Veg")
  output out=mycas.out outfreq=mycas.outfreq outrule=mycas.outrule;
  customer Customer;
  target product;
  hierarchy data = mycas.firstLevel mycas.secondLevel;
  savestate rstore=mycas.myStateArm;
run;
```

The DATA= option names the input data table to be analyzed. The ITEMS= option specifies the number of items in a rule. The SUPPORT= option specifies the minimum level of support for a rule. The SEPARATOR= option specifies separator character in the antecedent (left-hand side) or consequent (right-hand side) of a rule. The NLHS_RANGE= option specifies range of number of items in the left-hand side (LHS) of a rule. The NRHS_RANGE= option specifies range of number of items in the right-hand side (RHS) of a rule. The ANTECEDENTLIST= option specifies the regular expression strings to match in the antecedent of a rule. The CONSEQUENTLIST= option specifies the regular expression strings to match in the consequent of a rule. The OUTPUT statement specifies output tables to contain results of the MBANALYSIS procedure. The CUSTOMER statement requests that the Customer variable be used to group the target variable into baskets. The TARGET statement requests that the Product variable be used as the target variable. The HIERARCHY statement specifies the item’s hierarchies. The RSTORE= option in the SAVESTATE statement specifies the binary scoring file to be used for scoring.

References


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 16
The MTLEARN Procedure

Contents

Overview: MTLEARN Procedure .................................................. 340
PROC MTLEARN Features ....................................................... 340
Using CAS Sessions and CAS Engine Librefs .............................. 340
Getting Started: MTLEARN Procedure .................................. 341
Syntax: MTLEARN Procedure .................................................. 345
   PROC MTLEARN Statement .................................................. 345
   AUTOTUNE Statement ....................................................... 347
   DISPLAY Statement ......................................................... 351
   DISPLAYOUT Statement ................................................... 352
   INPUT Statement ........................................................... 353
   OUTPUT Statement ......................................................... 353
   SAVESTATE Statement ..................................................... 354
   TARGET Statement .......................................................... 354
Details: MTLEARN Procedure .................................................. 355
   Examples of Using the GRAPHTYPE= Option ......................... 355
   Missing Values ............................................................. 360
   Displayed Output .......................................................... 360
   ODS Table Names .......................................................... 361
   Output Data Tables ......................................................... 361
Overview: MTLEARN Procedure

The MTLEARN procedure implements the multitask learning technique for least squares loss with $\ell_1$ and graph structure penalizations. It solves multiple related sparse linear regression problems simultaneously. A graph structure encodes the relationships between the problems. PROC MTLEARN 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, PROC MTLEARN also serves as an elastic net regularization solver, which is widely used in feature selection.

PROC MTLEARN Features

The MTLEARN procedure enables you to use parallel execution in a distributed computing environment or on a single machine. The procedure has the following basic features:

- is highly distributed and multithreaded
- returns multiple related linear regression models by using a parallel implementation of the accelerated proximal gradient descent optimization algorithm

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.
If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”

### Getting Started: MTLEARN Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example shows how to use the MTLEARN procedure to obtain 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, 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.

```sas
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 3.842113 3.799427 3.799427
  5 1.563781 -1.04760 -0.298725 21.160617 20.762561
  6 0.426927 1.319816 -0.022467 -2.301208 -2.301208
  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.

```sas
data mycas.toyr;
  input id Y1 Y2 Y3;
  cards;
  0 0 1 -1
run;
```

The following statements run PROC MTLEARN and output the results to ODS tables:
proc mtlearn data = mycas.toydata
   graphType = CUSTOM
   regL1 = 0.5
   regL2 = 1.0
   maxIters = 10
   tolerance = 1e-2
   seed = 123
   graphTable = mycas.toyr
   modelOut = mycas.mtl_outW
   graphOut = mycas.mtl_outR
;
input X1 X2 X3;
target Y1 Y2 Y3;
output out = mycas.mtl_out copyvars = (id X1 X2 X3);
savestate rstore = mycas.mtl_ast;
ods select ModelInfo NObs DescStats OptIterHistory;
run;

Figure 16.1 shows the values of the parameters that are used in multitask learning.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Information</td>
<td></td>
</tr>
<tr>
<td>Seed</td>
<td>123</td>
</tr>
<tr>
<td>L1 Regularization</td>
<td>0.5</td>
</tr>
<tr>
<td>L2 Regularization</td>
<td>1</td>
</tr>
<tr>
<td>Tolerance</td>
<td>0.01</td>
</tr>
<tr>
<td>Maximum Iterations</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 16.2 shows the “Number of Observations” information. Note that missing values are not allowed in the input variables, and any row that has at least one nonmissing target is used in the optimization. Because all rows in the mycas.toyData data table have at least one target that is not missing, the number of observations read and the number of observations used are equal in this example.
Figure 16.2  Number of Observations

<table>
<thead>
<tr>
<th>Number of Observations Read</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Used</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 16.3 shows statistics for each interval variable in the INPUT and TARGET statements, including the mean and standard deviation.

Figure 16.3  Interval Variable Statistics

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>0.486773</td>
<td>0.863056</td>
</tr>
<tr>
<td>X2</td>
<td>0.198444</td>
<td>1.215561</td>
</tr>
<tr>
<td>X3</td>
<td>-0.149870</td>
<td>1.083567</td>
</tr>
<tr>
<td>Y1</td>
<td>4.013309</td>
<td>9.723587</td>
</tr>
<tr>
<td>Y2</td>
<td>3.856637</td>
<td>9.545267</td>
</tr>
<tr>
<td>Y3</td>
<td>4.929953</td>
<td>8.636507</td>
</tr>
</tbody>
</table>

Figure 16.4 shows the iteration history of the objective function.

Figure 16.4  Iteration History

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Objective Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>381.53329458</td>
</tr>
<tr>
<td>1</td>
<td>168.56137127</td>
</tr>
<tr>
<td>2</td>
<td>76.435700052</td>
</tr>
<tr>
<td>3</td>
<td>39.95948095</td>
</tr>
<tr>
<td>4</td>
<td>27.044926296</td>
</tr>
<tr>
<td>5</td>
<td>23.227784499</td>
</tr>
<tr>
<td>6</td>
<td>22.553144392</td>
</tr>
<tr>
<td>7</td>
<td>22.695112662</td>
</tr>
<tr>
<td>8</td>
<td>22.769024386</td>
</tr>
<tr>
<td>9</td>
<td>22.603979523</td>
</tr>
</tbody>
</table>

The following statements use the PRINT procedure to extract the first 10 observations from the output score table. The results are shown in Figure 16.5.

```plaintext
proc print noobs data=mycas.mtl_out(obs=10);
run;
```
The following PROC PRINT statements display the estimated regression weights table, which is shown in Figure 16.6:

```plaintext
proc print noobs data=mycas.mtl_outW;
run;
```

**Figure 16.6 Output Regression Weights Table**

<table>
<thead>
<tr>
<th>Variable</th>
<th><em>W_Y1</em></th>
<th><em>W_Y2</em></th>
<th><em>W_Y3</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>10.0906</td>
<td>9.86395</td>
<td>9.98828</td>
</tr>
<tr>
<td>X2</td>
<td>-4.8786</td>
<td>-4.91644</td>
<td>-4.98455</td>
</tr>
<tr>
<td>X3</td>
<td>-0.0165</td>
<td>0.00000</td>
<td>-0.00036</td>
</tr>
</tbody>
</table>

The following PROC PRINT statements display the graph table, which is shown in Figure 16.7.

```plaintext
proc print noobs data=mycas.mtl_outR;
run;
```

**Figure 16.7 Output Graph Table**

<table>
<thead>
<tr>
<th>id</th>
<th>Y1</th>
<th>Y2</th>
<th>Y3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Syntax: MTLEARN Procedure

The following statements are available in the MTLEARN procedure:

```plaintext
PROC MTLEARN <options>;
   AUTOTUNE <options>;
   DISPLAY <table-list> </options>;
   DISPLAYOUT table-spec-list </options>;
   INPUT variables <LEVEL=INTERVAL>;
   TARGET variables <LEVEL=INTERVAL>;
   OUTPUT OUT=CAS-libref.data-table <options>;
   SAVESTATE RSTORE=CAS-libref.data-table <options>;
```

The PROC MTLEARN statement, the INPUT statement, and the TARGET statement are required. You can specify multiple INPUT statements. You can specify one or multiple targets in the TARGET statement.

The following sections describe the PROC MTLEARN statement and then describe the other statements in alphabetical order.

PROC MTLEARN Statement

```plaintext
PROC MTLEARN <options>;
```

The PROC MTLEARN statement invokes the procedure. Table 16.1 summarizes the options available in the PROC MTLEARN statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Data Table Options</strong></td>
<td>Specifies the input data table</td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the user-defined graph table</td>
</tr>
<tr>
<td>GRAPHTABLE=</td>
<td></td>
</tr>
<tr>
<td><strong>Multitask Learning Options</strong></td>
<td>Specifies the type of graph table</td>
</tr>
<tr>
<td>GRAPHTYPE=</td>
<td>Specifies the maximum number of iterations</td>
</tr>
<tr>
<td>MAXITERS=</td>
<td>Specifies the number of threads to use on each computation node</td>
</tr>
<tr>
<td>NTHREADS=</td>
<td>Specifies the ( \ell_1 ) (LASSO) penalization weight</td>
</tr>
<tr>
<td>REGL1=</td>
<td>Specifies the ( \ell_2 ) graph penalization weight</td>
</tr>
<tr>
<td>REGL2=</td>
<td>Specifies the seed to use for pseudorandom number generation</td>
</tr>
<tr>
<td>SEED=</td>
<td>Specifies the optimization tolerance (absolute ( \ell_2 ) difference of</td>
</tr>
<tr>
<td>TOLERANCE=</td>
<td>solution) as a stopping criterion</td>
</tr>
<tr>
<td><strong>Output Data Table Options</strong></td>
<td>Specifies the output data table in which to save the graph table</td>
</tr>
<tr>
<td>GRAPHOUT=</td>
<td>Specifies the output data table in which to save the estimated multitask</td>
</tr>
<tr>
<td>MODELOUT=</td>
<td>regression weights</td>
</tr>
</tbody>
</table>
|```
You can specify the following options:

**DATA=** *CAS-libref.data-table*

names the input data table for PROC MTLEARN to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

*CAS-libref* refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 340.

*data-table* specifies the name of the input data table.

The data table must include one variable named id as the row ID variable.

**GRAPHOUT=** *CAS-libref.data-table*

specifies the output data table in which to save the graph table. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 340.

**GRAPHTABLE=** *CAS-libref.data-table*

specifies the user-defined graph table. 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 statement. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 340.

**GRAPHTYPE=** CLUSTER | CUSTOM | FUSE | INDEP

specifies the type of graph table.

You can specify one of the following values:

**CLUSTER** generates a cluster graph table, where all tasks are connected (to a virtual mean task).

**CUSTOM** uses the graph table that you specify.

**FUSE** generates a fuse graph table, where each task is connected to the next task in the target list.

**INDEP** generates an independent graph table, where all tasks are independent.

If you specify the **GRAPHTABLE=** option, the value of the **GRAPHTYPE=** option is automatically overwritten with CUSTOM.

By default, **GRAPHTYPE=** CLUSTER.
**MAXITERS=number**  
specifies the maximum number of iterations for the algorithm to perform, where *number* is an integer greater than or equal to 1.

By default, MAXITERS=100. You can tune this value by using the AUTOTUNE statement.

**MODELOUT=CAS-libref.data-table**  
specifies the output data table in which to save the estimated multitask regression weights. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 340.

**NTHREADS=number**  
specifies the number of threads to use for the computation, where *number* is an integer greater than 0. The default value is the maximum number of available threads per computer.

**REGL1=number**  
specifies the $\ell_1$ (LASSO) penalization weight, where *number* is greater than or equal to 0. The default value is 0.01. You can tune this value by using the AUTOTUNE statement.

**REGL2=number**  
specifies the $\ell_2$ graph penalization weight, where *number* is greater than or equal to 0. The default value is 0.01. You can tune this value by using the AUTOTUNE statement.

**SEED=random-seed**  
specifies an integer that is used to start the pseudorandom number generator. This option enables you to reproduce the same sample output, but only when NTHREADS=1. If you do not specify a seed, or if you specify a value less than or equal to 0, the seed is generated from reading the time of day from the computer’s clock.

**TOLERANCE=number**  
specifies the optimization tolerance (absolute $\ell_2$ difference of solution) as a stopping criterion, where *number* is greater than 0. The default value is 1E–6. You can tune this value by using the AUTOTUNE statement.

---

**AUTOTUNE Statement**

```
AUTOTUNE <options> ;
```

The AUTOTUNE statement searches for the best combination of values of the MAXITERS=, REGL1=, REGL2=, and TOLERANCE= options in the PROC MTLEARN statement.

Table 16.2 summarizes the options that you can specify in the AUTOTUNE statement. For more information about all options except the TUNINGPARAMETERS= option, see the option’s description in the section “AUTOTUNE Statement” on page 12 in Chapter 3, “Shared Concepts.” The TUNINGPARAMETERS= option is described following Table 16.2.
### Table 16.2  AUTOTUNE Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDLOOKUP</td>
<td>Specifies that the table specified in the HISTORYTABLE= option contain the rows from the table specified in the LOOKUPTABLE= option</td>
</tr>
<tr>
<td>EVALHISTORY=</td>
<td>Specifies how to report the evaluation history of the tuner</td>
</tr>
<tr>
<td>FRACTION=</td>
<td>Specifies the fraction of observations to use for validation</td>
</tr>
<tr>
<td>HISTORYTABLE=</td>
<td>Specifies the CAS table that contains the evaluation history</td>
</tr>
<tr>
<td>KFOLD=</td>
<td>Specifies the number of folds for ( k )-fold cross validation</td>
</tr>
<tr>
<td>LIVEUPDATE</td>
<td>Specifies that the table specified in the HISTORYTABLE= option be updated at every evaluation</td>
</tr>
<tr>
<td>LOCALESEARCH</td>
<td>Enables local search optimization</td>
</tr>
<tr>
<td>LOOKUPTABLE=</td>
<td>Specifies the CAS table to use for evaluation lookup</td>
</tr>
<tr>
<td>MAXBAYES=</td>
<td>Specifies the maximum number of points in the kriging model</td>
</tr>
<tr>
<td>MAXEVALS=</td>
<td>Specifies the maximum number of evaluations</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>MAXTIME=</td>
<td>Specifies the maximum time for all iterations</td>
</tr>
<tr>
<td>MAXTRAINITER=</td>
<td>Specifies the maximum time for a model training</td>
</tr>
<tr>
<td>NCONVITER=</td>
<td>Specifies the number of convergence iterations</td>
</tr>
<tr>
<td>NOGRIDSHUFFLE</td>
<td>Requests that the grid points not be shuffled</td>
</tr>
<tr>
<td>NPARALLEL=</td>
<td>Specifies the number of parallel sessions</td>
</tr>
<tr>
<td>NSUBSESSIONWORKERS=</td>
<td>Specifies the number of workers in parallel sessions</td>
</tr>
<tr>
<td>OBJECTIVE=</td>
<td>Specifies the objective function</td>
</tr>
<tr>
<td>POPSIZE=</td>
<td>Specifies the population size when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>SAMPLESIZE=</td>
<td>Specifies the sample size when SEARCHMETHOD=LHS or SEARCHMETHOD=RANDOM</td>
</tr>
<tr>
<td>SEARCHMETHOD=</td>
<td>Specifies the search method that the optimizer uses</td>
</tr>
<tr>
<td>SECONDOBJECTIVE=</td>
<td>Specifies the second objective to use for tuning</td>
</tr>
<tr>
<td>SELECTINITPOINT</td>
<td>Specifies that the tuner select the best evaluation from the lookup table</td>
</tr>
<tr>
<td>TARGETEVENT=</td>
<td>Specifies the target event for ROC-based calculations</td>
</tr>
<tr>
<td>TRAINFRACTION=</td>
<td>Specifies the fraction of observations to use for training</td>
</tr>
<tr>
<td>TUNINGPARAMETERS=</td>
<td>Specifies the custom tuning parameters</td>
</tr>
<tr>
<td>USEPARAMETERS=</td>
<td>Specifies how to handle the TUNINGPARAMETERS= option</td>
</tr>
</tbody>
</table>

\[
\text{TUNINGPARAMETERS}=(\text{suboption} \mid \ldots \mid <\text{suboption}>)
\]
\[
\text{TUNEPARAMS}=(\text{suboption} \mid \ldots \mid <\text{suboption}>)
\]

specifies which parameters to tune and which ranges to tune over. If USEPARAMETERS=STANDARD, this option is ignored.

You can specify one or more of the following suboptions:
MAXITERS (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies information about the maximum number of iterations to use for tuning the multitask
learning model. For more information, see the MAXITERS= option in the PROC MTLEARN
statement.

You can specify the following additional suboptions:

**LB=number**
specifies the minimum number of iterations to consider during tuning. If you specify this
suboption, you cannot specify the VALUES= suboption.

By default, LB=100.

**UB=number**
specifies the maximum number of iterations to consider during tuning. If you specify this
suboption, you cannot specify the VALUES= suboption.

By default, UB=5000.

**VALUES=value-list**
specifies a list of numbers of trees to consider during tuning, where value-list is a space-
separated list of positive integers. If you specify this suboption, you cannot specify either
the LB= or UB= suboption.

**INIT=number**
specifies the initial number of iterations for the tuner to use.

By default, INIT=100.

**EXCLUDE**
excludes the number of iterations from the tuning process. If you specify this suboption, any
specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

REGL1 (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies information about the $\ell_1$ penalization weight to use for tuning the multitask
learning model. For more information, see the REGL1= option in the PROC MTLEARN statement.

You can specify the following additional suboptions:

**LB=number**
specifies the minimum $\ell_1$ penalization weight to consider during tuning. If you specify this
suboption, you cannot specify the VALUES= suboption.

By default, LB=0.

**UB=number**
specifies the maximum $\ell_1$ penalization weight to consider during tuning. If you specify this
suboption, you cannot specify the VALUES= suboption.

By default, UB=1000.
VALUES=value-list
specifies a list of values to consider for the $\ell_1$ penalization weight during tuning, where value-list is a space-separated list of numbers greater than or equal to 0. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

INIT=number
specifies the initial $\ell_1$ penalization weight for the tuner to use.

By default, INIT=0.01.

EXCLUDE
excludes the $\ell_1$ penalization weight from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

REGL2 (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies information about the $\ell_2$ penalization weight to use for tuning the multitask learning model. For more information, see the REGL2= option in the PROC MTLEARN statement.

You can specify the following additional suboptions:

LB=number
specifies the minimum $\ell_2$ penalization weight to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=0.

UB=number
specifies the maximum $\ell_2$ penalization weight to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=1000.

VALUES=value-list
specifies a list of values to consider for the $\ell_2$ penalization weight during tuning, where value-list is a space-separated list of numbers greater than or equal to 0. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

INIT=number
specifies the initial $\ell_2$ penalization weight for the tuner to use.

By default, INIT=0.01.

EXCLUDE
excludes the $\ell_2$ penalization weight from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

TOLERANCE (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies information about the optimization tolerance to use for tuning the multitask learning model. For more information, see the TOLERANCE= option in the PROC MTLEARN statement.

You can specify the following additional suboptions:
**LB=** number
specifies the minimum optimization tolerance to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=1E–6.

**UB=** number
specifies the maximum optimization tolerance to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=1E–3.

**VALUES=** value-list
specifies a list of values to consider for the optimization tolerance during tuning, where value-list is a space-separated list of numbers greater than 0. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=** number
specifies the initial optimization tolerance for the tuner to use.

By default, INIT=1E–6.

**EXCLUDE**
excludes the optimization tolerance from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

---

**DISPLAY Statement**

**DISPLAY <table-list> < / options> ;**

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the **table-list** as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “ODS Table Names” on page 361. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path **Bygroup1.Summary.SelectionSummary**. A partial pathname does not include all groups; for example, **SelectionSummary** and **Summary.SelectionSummary** are partial pathnames for **Bygroup1.Summary.SelectionSummary**.
When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both `SelectionSummary` and `Summary.SelectionSummary` select `Bygroup1.Summary.SelectionSummary`.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the `Bygroup1.Summary.SelectionSummary` table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the `Bygroup1.Summary.SelectionSummary` table is not selected.

You can specify the following `options` after a slash (/):

**CASESENSITIVE**

performs a case-sensitive comparison of table names in the `table-list` to display table names when tables are subsetted for display. To preserve case, you must enclose table names in the `table-list` in quotation marks.

**EXCLUDE**

displays all display tables except those that you specify in the `table-list`.

**EXCLUDEALL**

suppresses display of all tables. This option takes precedence over the other options.

**TRACE**

displays the display table names, labels, and paths.

---

**DISPLAYOUT Statement**

```plaintext
DISPLAYOUT table-spec-list < / options >;
```

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

The `table-spec-list` specifies a list of CAS output tables to create. Each entry in the list has either a `key=value` format or a `key` format:

- `key=value` specifies `key` as the ODS table name, path, or partial pathname, and specifies `value` as the CAS output table name.
- `key` specifies `key` as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “ODS Table Names” on page 361. You cannot specify the ODS table named `OutputCasTables` in the `table-spec-list`.

Table names and partial pathnames are discussed under the `DISPLAY` statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following `options` after a slash (/):
**INCLUDEALL**
creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the *table-spec-list* specification is ignored.

**NOREPLACE**
does not replace any existing CAS output table of the same name.

**REPEATED**
replicates all CAS output tables on all nodes.

---

**INPUT Statement**

**INPUT** `variables < LEVEL=INTERVAL > ;`

The INPUT statement specifies the names of the `variables` to use in the multitask regression. It names one or more input variables that use common options. If you want to use different options for different variables, you can specify multiple INPUT statements.

You can include the following option in each INPUT statement:

---

**LEVEL=INTERVAL**

specifies the level of measurement of the `variables`. You can specify the following value:

- **INTERVAL** specifies that the level of measurement of the `variables` is interval.

You must specify at least one interval input variable. PROC MTLEARN currently accepts only interval input variables.

---

**OUTPUT Statement**

**OUTPUT OUT=** `CAS-libref.data-table < options > ;`

The OUTPUT statement creates an output data table to contain the results of the procedure run.

You must specify the following option:

---

**OUT=** `CAS-libref.data-table`

names the output data table for PROC MTLEARN to use. You must specify this option before any other options. `CAS-libref.data-table` is a two-level name, where

- **CAS-libref** refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 340.

- **data-table** specifies the name of the output data table.

You can also specify the following option:
**COPYVAR**=`variable`  
**COPYVARS**(variables)  

copies one or more variables from the input data table to the output data table.

---

### SAVESTATE Statement

**SAVESTATE** RSTORE=`CAS-libref.data-table <options>` ;

The **SAVESTATE** statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the **ASTORE** procedure to score new data. For more information, see Chapter 4, “The **ASTORE** Procedure.”

You must specify the following option:

**RSTORE**=`CAS-libref.data-table`

specifies a data table in which to save the analytic store for the model. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 340.

---

### TARGET Statement

**TARGET** variables < **LEVEL=INTERVAL** > ;

The **TARGET** statement names the target variables. The targets must be of interval type and must be different from the variables in the **INPUT** statement. You must specify at least one target variable. Missing values in the target variables are allowed. You can specify the following option:

**LEVEL=INTERVAL**

specifies that the level of measurement of the variables is interval.

PROC MTLEARN currently accepts only interval target variables.
Details: MTLEARN Procedure

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 $\epsilon_i$ 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,

$$\hat{W} = \arg \min_W 1 \frac{1}{2} \sum_{i=1}^{K} \|X_i W_i - y_i\|_2^2 + \frac{1}{2} \rho \ell_2 \|WR^T\|_F^2 + \rho \ell_1 \|W\|_1$$

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= Option

The GRAPHTYPE= option values that you can specify are CLUSTER, INDEP, FUSE, and CUSTOM. When the GRAPHTABLE= option is not specified, the MTLEARN procedure can automatically generate a graph table if GRAPHTYPE=CLUSTER, INDEP, or FUSE. If GRAPHTYPE=CUSTOM, the MTLEARN procedure does not generate a graph table and instead expects a graph table to be specified in the GRAPHTABLE= option. If a graph table is specified in GRAPHTABLE= option, PROC MTLEARN uses the user-specified graph table, and the value of the GRAPHTYPE= option is overwritten with CUSTOM.

- When GRAPHTYPE=CLUSTER and no GRAPHTABLE= option is specified, PROC MTLEARN 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 16.8 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 16.8** Cluster Graph Structure with Five Targets (Tasks 1 to 5)

- When `GRAPHTYPE=INDEP` and no `GRAPHTABLE=` option is specified, PROC MTLEARN generates an independent graph table, where all tasks are independent. 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 16.9 shows an example of an independent graph structure of five tasks, whose corresponding graph table matrix is

$$
R = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
$$
Examples of Using the GRAPHTYPE= Option

**Figure 16.9** Independent Graph Structure with Five Targets (Tasks 1 to 5)

![Diagram of Independent Graph Structure with Five Targets](image)

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 GRAPHTYPE=FUSE and no GRAPHTABLE= option is specified, PROC MTLEARN 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 16.10** 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 16.10  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 GRAPHTYPE=CUSTOM, the MTLEARN procedure uses the graph table you specify in the GRAPHTABLE= option. The first column of the custom graph table must contain the row ID variable. The custom graph table must also include all target variables that you specify in the TARGET statement. Figure 16.11 shows an example of a custom graph. Using the definition of $R$ near the beginning of the section “Details: MTLEARN Procedure” on page 355, the corresponding graph table matrix is

$$R = \begin{bmatrix}
1 & -1 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 \\
0 & 0 & 1 & 0 & -1 \\
0 & 0 & 0 & 1 & -1
\end{bmatrix}$$
The following DATA step creates the custom graph table shown in Figure 16.8. This data table contains six variables: id is the row ID variable, and Task1, Task2, Task3, Task4, and Task5 are the target variables.

```plaintext
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 16.12.

```plaintext
proc print noobs data=customR;
run;
```

**Figure 16.12** Example of a Custom Graph Table

<table>
<thead>
<tr>
<th>id</th>
<th>Task1</th>
<th>Task2</th>
<th>Task3</th>
<th>Task4</th>
<th>Task5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Missing Values

In the MTLEARN procedure, 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 $S = \{1, \ldots, K\}$ be the set of all task indices, $S_M$ be the set of indices whose target variables are missing, and $S_N = S \setminus S_M$ be the set of indices whose target variables are present. PROC MTLEARN considers that the observation $x_k \in X_i$, where $i \in S_N$, and $x_k \notin X_j$, where $j \in S_M$.

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 MTLEARN procedure 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, $\lambda_1$ penalization weight, $\lambda_2$ 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. Because this table is not a typical ODS table that contains analytical results, you cannot include it in the table-spec-list in the DISPLAYOUT statement.
ODS Table Names

Each table that the MTLEARN procedure creates has a name associated with it, and you must use this name to refer to the table when you use ODS statements. The name of each table and a short description of its contents are listed in Table 16.3.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statements</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>DescStats</td>
<td>Descriptive statistics for interval variables</td>
<td>INPUT and TARGET</td>
<td>LEVEL=INTERVAL</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>PROC MTLEARN</td>
<td>Default output</td>
</tr>
<tr>
<td>NObs</td>
<td>Number of observations</td>
<td>PROC MTLEARN</td>
<td>Default output</td>
</tr>
<tr>
<td>OptIterHistory</td>
<td>Iteration history</td>
<td>PROC MTLEARN</td>
<td>Default output</td>
</tr>
<tr>
<td>OutputCASTables</td>
<td>See the section “OutputCasTables Table” on page 360</td>
<td>PROC MTLEARN</td>
<td>Default output</td>
</tr>
</tbody>
</table>

Output Data Tables

The MTLEARN procedure 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= option in the PROC MTLEARN statement.

- The output model table lists the value of the estimated multitask regression weights. You specify the name of this data table in the MODELOUT= option in the PROC MTLEARN statement.
Chapter 17
The MWPCA Procedure

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: MWPCA Procedure</td>
<td>364</td>
</tr>
<tr>
<td>PROC MWPCA Features</td>
<td>364</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>364</td>
</tr>
<tr>
<td>Getting Started: MWPCA Procedure</td>
<td>365</td>
</tr>
<tr>
<td>Syntax: MWPCA Procedure</td>
<td>368</td>
</tr>
<tr>
<td>PROC MWPCA Statement</td>
<td>369</td>
</tr>
<tr>
<td>DISPLAY Statement</td>
<td>370</td>
</tr>
<tr>
<td>DISPLAYOUT Statement</td>
<td>371</td>
</tr>
<tr>
<td>ID Statement</td>
<td>372</td>
</tr>
<tr>
<td>INPUT Statement</td>
<td>372</td>
</tr>
<tr>
<td>OUTPUT Statement</td>
<td>372</td>
</tr>
<tr>
<td>RPCA Statement</td>
<td>374</td>
</tr>
<tr>
<td>SVD Statement</td>
<td>375</td>
</tr>
<tr>
<td>Details: MWPCA Procedure</td>
<td>375</td>
</tr>
<tr>
<td>Cumulative Eigenvalue Percentage Tolerance</td>
<td>376</td>
</tr>
<tr>
<td>Displayed Output</td>
<td>376</td>
</tr>
<tr>
<td>ODS Table Names</td>
<td>377</td>
</tr>
<tr>
<td>Examples: MWPCA Procedure</td>
<td>377</td>
</tr>
<tr>
<td>Example 17.1: Anomaly Detection of Parking Lot Floodlights</td>
<td>377</td>
</tr>
<tr>
<td>References</td>
<td>381</td>
</tr>
</tbody>
</table>
Chapter 17: The MWPCA Procedure

Overview: MWPCA Procedure

The MWPCA procedure implements principal component analysis over sliding windows that consist of a specified number of observations. You can use this procedure to capture changes in principal components over time. Also, you can choose to perform robust principal component analysis (RPCA) on each window; in RPCA, the outliers and noise are excluded from each window before the analysis is performed.

One important application of MWPCA is for detecting relative changes in parts of a system compared to the overall system. By tracing the principal components over time, you can determine whether significant changes in the principal components can be a cause for concern.

PROC MWPCA stores the requested principal components for each window in an output table that is produced by the OUTPUT statement.

PROC MWPCA Features

The MWPCA procedure has the following features:

- reads input data in parallel when the data source is on a distributed system
- is multithreaded during all phases of analytic execution
- supports large-scale data
- enables analysis to be done on selected columns of the data
- provides the option to perform robust principal component analysis (RPCA) in addition to the standard principal component analysis (PCA) on each window
- tracks changes of the first principal component over sliding windows

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:
cas mysess;
libname mycas cas sessref=mysess;

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

cas mysess terminate;

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”

### Getting Started: MWPCA Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

The following simple example shows how to use the MWPCA procedure 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  2536.174173  1698.306145  1619.159414

... more lines ...

The following statements plot the amount of energy that each turbine produces over time. Output 17.1 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";
You can load the Turbines 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.

```plaintext
data mycas.turbines;
  set turbines;
run;
```

The following statements run PROC MWPCA:

```plaintext
proc mw pca data=mycas.turbines windowsize=200 stepsize=1;
  id time;
  output out=mycas.windowpcs standardpc pcangles;
run;
```

Note that the STANDARDPC option is used in the OUTPUT statement. Using this option is recommended.

Figure 17.2 displays the “Model Information,” “Dimensions,” and “Results Summary” ODS tables. In the “Model Information” table, you can see the values of the WINDOWSIZE= and STEPSIZE= options and some other options that are set to their default values. A window size of 200 and step size of 1 indicates that the
procedures start 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 procedure 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 mycas.windowpcs data table contains 701 windows, each of which contains 200 observations. You can also see that the solution status is optimal; that is, the PCA algorithm ran successfully for all windows.

You can see the output table windowpcs in your mycas CAS engine library. This table contains the elements of 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 17.3 shows the resulting plot.

```sas
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 Figure 17.3, 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.
**Syntax: MWPCA Procedure**

The following statements are available in the MWPCA procedure:

```
PROC MWPCA STEPSIZE=number WINDOWSIZE=number < options > ;
   ID variable ;
   INPUT variables ;
   RPCA < options > ;
   SVD < options > ;
   OUTPUT OUT=CAS-libref.data-table < options > ;
   DISPLAY < table-list </ options > ;
   DISPLAYOUT table-spec-list </ options > ;
```

The PROC MWPCA and the ID statements are required. The following sections describe the PROC MWPCA statement and then describe the other statements in alphabetical order.
PROC MWPCA Statement

The PROC MWPCA statement invokes the procedure. Table 17.1 summarizes the options available in the PROC MWPCA statement.

### Table 17.1 PROC MWPCA Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Data Table Option</td>
<td></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the input data table</td>
</tr>
<tr>
<td>Other Options</td>
<td></td>
</tr>
<tr>
<td>CENTER</td>
<td>Centers the input data</td>
</tr>
<tr>
<td>CUMEIGPCTTOL=</td>
<td>Specifies the significance level of eigenvalues that determine the rank of the low-rank matrix in each window</td>
</tr>
<tr>
<td>NTHREADS=</td>
<td>Specifies the maximum number of threads to use on each computation node</td>
</tr>
<tr>
<td>ROBUST</td>
<td>Uses robust principal component analysis on each window</td>
</tr>
<tr>
<td>STEPSIZE=</td>
<td>Specifies the step size</td>
</tr>
<tr>
<td>WINDOWSIZE=</td>
<td>Specifies the window size</td>
</tr>
</tbody>
</table>

You must specify the following options:

**STEPSIZE=number**

specifies the step size, which is the number of observations between the beginnings of two consecutive windows.

**WINDOWSIZE=number**

specifies the number of observations in each window.

You can also specify the following options:

**CENTER**

centers the observations by the mean of each column for each window.

**CUMEIGPCTTOL=number**

specifies the significance level of eigenvalues that determine the rank of the low-rank matrix in each window. This value must be between 0 and 1. By default, CUMEIGPCTTOL=1. For more information, see the section “Cumulative Eigenvalue Percentage Tolerance” on page 376.

**DATA=CAS-libref.data-table**

names the input data table for PROC MWPCA to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

**CAS-libref** refers to a collection of information that is defined in the LIBNAME statement and includes the *caslib*, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME
statement. For more information about CAS-libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 364.

\textit{data-table} specifies the name of the input data table.

\textbf{NTHREADS=number-of-threads}

specifies the number of threads to use in the computation. The default value is the lesser of the number of threads available per computation node and 16.

\textbf{ROBUST}

uses the robust principal component analysis (RPCA) method to obtain principal components for each window. You can specify the RPCA options in the RPCA statement. For more information about the RPCA method, see Chapter 19, “The RPCA Procedure.” If you do not specify this option, the regular PCA method is used on each window. It is recommended not to use this option for anomaly detection.

\section*{DISPLAY Statement}

\texttt{DISPLAY < table-list > < / options > ;}

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see \textit{SAS Output Delivery System: Procedures Guide}.

You can specify the \textit{table-list} as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “ODS Table Names” on page 377. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path \texttt{Bygroup1.Summary.SelectionSummary}. A partial pathname does not include all groups; for example, \texttt{SelectionSummary} and \texttt{Summary.SelectionSummary} are partial pathnames for \texttt{Bygroup1.Summary.SelectionSummary}.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both \texttt{SelectionSummary} and \texttt{Summary.SelectionSummary} select \texttt{Bygroup1.Summary.SelectionSummary}.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the \texttt{Bygroup1.Summary.SelectionSummary} table is selected. Specifying “!/tions!/” selects all pathnames that do not contain the substring “tions”; in particular, the \texttt{Bygroup1.Summary.SelectionSummary} table is not selected.

You can specify the following \textit{options} after a slash (/):

\section*{Example}

\begin{verbatim}
PROC MWPCA;
   data-table: mydata;
   NTHREADS: 8;
   ROBUST:;
   DISPLAY: SelectionSummary;
RUN;
\end{verbatim}
CASESENSITIVE performs a case-sensitive comparison of table names in the `table-list` to display table names when tables are subsetted for display. To preserve case, you must enclose table names in the `table-list` in quotation marks.

EXCLUDE displays all display tables except those that you specify in the `table-list`.

EXCLUDEALL suppresses display of all tables. This option takes precedence over the other options.

TRACE displays the display table names, labels, and paths.

---

**DISPLAYOUT Statement**

```
DISPLAYOUT table-spec-list </ options> ;
```

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

The `table-spec-list` specifies a list of CAS output tables to create. Each entry in the list has either a `key=value` format or a `key` format:

- `key=value` specifies `key` as the ODS table name, path, or partial pathname, and specifies `value` as the CAS output table name.
- `key` specifies `key` as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “ODS Table Names” on page 377. You cannot specify the ODS table named OutputCasTables in the `table-spec-list`.

Table names and partial pathnames are discussed under the DISPLAY statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following `options` after a slash (/):

- **INCLUDEALL** creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the `table-spec-list` specification is ignored.

- **NOREPLACE** does not replace any existing CAS output table of the same name.
replicates all CAS output tables on all nodes.

**ID Statement**

```plaintext
ID variable ;
```

The ID statement lists one `variable` from the data set that specifies the order of the observations in the input data set. This variable is not considered as part of the analysis to compute the principal components. Also, the values of `variable` should be unique with a common difference.

The following examples show values of the variables that you can use as ID variables:

- `1, 3, 5, 7, ...`
- `2.2, 2.7, 3.2, 3.7, 4.2, 4.7, ...`

The following examples show values of the variables that you cannot use as ID variables:

- `1, 2, 5, 6, 7, ...` (these values do not have a common difference)
- `1, 2, 3, 3, 4, 5, 6, 7, ...` (these values are not unique)

The ID statement is required. Without this statement, windows cannot be identified.

**INPUT Statement**

```plaintext
INPUT variables ;
```

The INPUT statement specifies the names of `variables` to be considered in the MWPCA procedure. Only numeric `variables` are accepted. If you do not specify this statement, all numeric `variables` in the input data set are considered (except the one used in the ID statement).

**OUTPUT Statement**

```plaintext
OUTPUT OUT=CAS-libref.data-table < options > ;
```

The OUTPUT statement creates an output data table to contain the results of the procedure run.

You must specify the following option:

```plaintext
OUT=CAS-libref.data-table
```

names the output data table for PROC MWPCA to use. You must specify this option before any other options. `CAS-libref.data-table` is a two-level name, where
CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about CAS-libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 364.

data-table specifies the name of the output data table.

You can also specify the following options:

NPC=number
specifies the number of principal components to display for each window in the output table.

PCANGLES
displays the angle change and the absolute angle of the first principal component in the output table. Both the angle change and the absolute angle can be used to track the change of the first principal component over sliding windows. The range of both values is between 0 and 1, because both values are normalized by \( \frac{2}{\pi} \) as shown by the following definitions:

- The angle change is calculated by
  \[
  \theta = \arccos \left( \frac{P_1(k-1) \cdot P_1(k)}{\|P_1(k-1)\| \|P_1(k)\|} \right) \left( \frac{2}{\pi} \right)
  \]
  where \( P_1(k-1) \) and \( P_1(k) \) are the first principal component vectors of two consecutive observations.

- The absolute angle is calculated by
  \[
  \theta = \arccos \left( \frac{v_1 \cdot P_1(k)}{\|v_1\| \|P_1(k)\|} \right) \left( \frac{2}{\pi} \right)
  \]
  where \( v_1 \) is the diagonal vector \([1,1,1,\ldots,1]\) and \( P_1(k) \) is the first principal component vector of the new observation.

STANDARDPC
standardizes the output principal components. If the window matrix is \( W \) and the \( i \)th principal component vector is \( P_i \), then the \( j \)th element of the standard principal component \( P_i^S \) is

\[
P_{ij}^S = \frac{P_{ij} \times \alpha_i}{(W^T W)_{jj}}
\]
  where \( \alpha_i \) is the \( i \)th largest eigenvalue of matrix \( W \).

WINSUMMARY=CAS-libref.data-table
specifies the name of the windows summary table that includes information about each window, such as rank of the low-rank matrix, number of iterations, and solution status.

CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 364.
RPCA Statement

RPCA < options >

If you specify the ROBUST option in the PROC MWPCA statement, then you can use this statement to specify the following RPCA options:

**FIXEDMU**

uses a fixed value for $\mu$ in each iteration of RPCA when METHOD=APG. Otherwise, $\mu$ is updated dynamically in each iteration.

**LAMBDA=number**

specifies a value for $\lambda$, where *number* is a positive real number. The default value is computed by $\frac{1}{\sqrt{n}}$, where $n$ is the greater of the number of observations and the number of variables in the input data set. This parameter affects the sparsity of the sparse matrix. For more information, see Candès et al. (2011).

**LAMBDAWEIGHT=number**

specifies the value of $\lambda_{weight}$. The final value of the $\lambda$ that is used in the RPCA algorithm is calculated by multiplying $\lambda_{weight}$ by $\lambda$. You can use this value to control the sparsity of the sparse matrix. For more information about the sparse matrix in RPCA, see Chapter 19, “The RPCA Procedure.” By default, LAMBDAWEIGHT=1.

**MAXITER=number**

specifies the maximum number of iterations before the RPCA algorithm stops, where *number* is a positive integer. By default, MAXITER=1000.

**METHOD=ALM | APG**

specifies the method to perform RPCA. You can specify the following values:

**ALM** specifies the augmented Lagrange multiplier method.

**APG** specifies the accelerated proximal gradient method.

For more information about each method, see the section “Details: RPCA Procedure” on page 436. By default, METHOD=ALM.

**MU=number**

specifies an initial value of $\mu$ when METHOD=APG. By default, MU=$10^{-3}$.

**TOLERANCE=number**

specifies the convergence criterion for the RPCA algorithm on each window. By default, TOLERANCE=$10^{-7}$.
The SVD statement calls a singular value decomposition (SVD) solver to calculate the principal components for each window. You can specify the following options:

METHOD=EIGEN | ITERATIVE
specifies the type of the SVD solver. You can specify the following values:

EIGEN uses the eigenvalue decomposition method.

ITERATIVE uses the iterative SVD method.

MAXRANK=number
specifies the maximum value for rank considered by the SVD solver. By default, number is the number of input variables.

Details: MWPCA Procedure
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.
Cumulative Eigenvalue Percentage Tolerance

Sometimes when the input has noise, the rank that is reported in the `WINSUMMARY=` table is higher than the true rank. This occurs because the existence of noise in the input can increase the number of eigenvalues. 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 of a window 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.

Displayed Output

The MWPCA procedure displays various tables that are related to input and results. The following sections describe the output tables in the order of their appearance.

Model Information

The “Model Information” table displays basic information about the options that the MWPCA procedure uses. This information includes the data source, window size, step size, RPCA method, SVD method used in RPCA, and values of $\lambda$ and $\lambda_{\text{weight}}$ used in RPCA.

Dimensions

The “Dimensions” table displays the number of observations and variables in the input data set.

Results Summary

The “Results Summary” table displays the summary of the PROC MWPCA results, including the number of windows, solution status, and run time.

The solution status is “optimal” only if optimization ends successfully for all windows. Otherwise, the solution status is “Optimization was unsuccessful in at least one window.”
ODS Table Names

Each table that the MWPCA procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. The name of each table and a short description of the contents are listed in Table 17.2.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions</td>
<td>Dimensions of the input table</td>
<td>PROC MWPCA</td>
<td>Default</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>PROC MWPCA</td>
<td>Default</td>
</tr>
<tr>
<td>Summary</td>
<td>Summary of the results</td>
<td>PROC MWPCA</td>
<td>Default</td>
</tr>
</tbody>
</table>

Examples: MWPCA Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

Example 17.1: Anomaly Detection of Parking Lot Floodlights

This example applies the MWPCA procedure to a data set that consists of energy consumption every five minutes for six floodlight circuits in a parking lot at SAS headquarters in Cary, North Carolina, over a span of about three months. Each of these light circuits has a sensor that measures the energy consumption of the lights.

The following code includes the DATA step that generates the data table mycas.ParkingLotLights. This statement assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```sas
data mycas.parkinglotlights;
  input time e49548 e49549 e49551 e49552 e49553 e49554;
dATALINES;
  ... more lines ...
```

Figure 17.4 displays the energy consumption of each of these light circuits in the first 16 days. The energy consumption throughout the day is zero because the lights are off during the day. In this figure, you can see that the orange lights (e49549) have a dip in energy consumption for a couple of hours on the sixth night, indicating that one of the lights controlled by the circuit was not functioning properly. Also, you can see that
during day 15, all lights have a spike in energy consumption during the day. A storm occurred on that day, creating enough darkness to trigger the lights to come on briefly. There was no malfunction in the lights.

**Figure 17.4** Energy Consumption (Kilowatts) of Floodlights, First 16 Days

![Energy Consumption Chart](image)

Figure 17.5 displays the energy consumption of each light circuit in the last 13 days. In this figure, you can see that the green lights (e49552) start to have many dips in energy consumption in the last 11 nights; several of the lights on this circuit are starting to fail. Also, toward the end, the red lights (e49553) have dips in energy consumption, especially on the second-last night, indicating that problems are occurring in this circuit as well.
The following code uses PROC MWPCA to illustrate how moving-windows principal component analysis can help determine when the lights begin to fail:

```sas
proc MWPCA data=mycas.parkinglotlights windowsize = 240 stepsize = 1;
  id time;
  input e49548 e49549 e49551 e49552 e49553 e49554;
  output out=mycas.output standardpc pcangles;
run;
```

Output 17.1.1 displays the first principal component that is obtained from the output table that results from applying PROC MWPCA to the mycas.ParkingLotLights data table over the sliding windows of data. In this figure, you can observe that the first principal component that is associated with the orange lights starts deviating from the first principal components that are associated with the other lights around the same time that the dips were observed on the sixth night for these lights. Also, toward the end of the data, the first principal component that is associated with the green lights starts deviating from the first principal components that are associated with the other lights again around the same time that the dips for this light were observed towards the end. The same is true of the dips in the energy consumption of the red lights toward the end of the sample. Note that during the storm, when all lights came on unexpectedly (on day 15), the first principal components that are related to the lights do not deviate from each other, and thus there is no signal of failure in this instance. This result is correct: the unexpected behavior of the lights was not caused by a light malfunction. Output 17.1.1 shows how PROC MWPCA detects malfunctions in each light circuit by tracking the first principal components of their energy consumption.
Output 17.1.1 plots the angle change of the first principal component between consecutive windows. When the observation that enters a window contains the outliers, its value of angle change is relatively higher than that of previous windows. By checking the value of the angle change in Output 17.1.2, you can easily detect when a light circuit is not functioning properly.

Output 17.1.2 plots the angle change of the first principal component between consecutive windows. When the observation that enters a window contains the outliers, its value of angle change is relatively higher than that of previous windows. By checking the value of the angle change in Output 17.1.2, you can easily detect when a light circuit is not functioning properly.
Output 17.1.2 Angle Change of the First Principal Component over Sliding Windows

References

Chapter 18
The NNET Procedure

Contents

Overview: NNET Procedure ................................................. 384
PROC NNET Features ...................................................... 384
Using CAS Sessions and CAS Engine Librefs ....................... 385
Getting Started: NNET Procedure ...................................... 386
Syntax: NNET Procedure .................................................. 388
PROC NNET Statement .................................................... 388
ARCHITECTURE Statement .............................................. 390
AUTOTUNE Statement .................................................... 391
CODE Statement .......................................................... 396
CROSSVALIDATION Statement ......................................... 397
HIDDEN Statement ........................................................ 397
INPUT Statement .......................................................... 398
OPTIMIZATION Statement .............................................. 398
OUTPUT Statement ......................................................... 402
PARTITION Statement .................................................... 403
TARGET Statement ......................................................... 403
TRAIN Statement .......................................................... 405
WEIGHT Statement ......................................................... 407
Details: NNET Procedure ................................................ 407
Computational Method .................................................... 407
Hyperparameter Tuning .................................................... 408
k-fold Cross Validation .................................................. 408
Determining the Number of Parallel Evaluations .................. 409
Autoencoder ............................................................... 409
Displayed Output .......................................................... 410
Iteration History .......................................................... 410
Convergence Status ....................................................... 410
Model Information ......................................................... 410
Score Information ........................................................ 410
Tuner Information ........................................................ 410
Tuner Results .............................................................. 410
Evaluation History ....................................................... 410
Best Configuration ......................................................... 411
Tuner Summary ........................................................... 411
TunerTiming ............................................................... 411
HyperparameterImportance .......................................... 411
Overview: NNET Procedure

The NNET procedure trains a multilayer perceptron neural network in SAS Viya. For more information about multilayer perceptron neural networks, see Bishop (1995). PROC NNET can also use a previously trained network to score a data table (referred to as stand-alone scoring), or it can generate SAS DATA step statements that can be used to score a data table.

Training a multilayer perceptron neural network requires the unconstrained minimization of a nonlinear objective function. Because there are currently no practical methods to guarantee finding a global minimum of that objective function, one way to be reasonably sure of finding a good solution is to train the network multiple times by using different sets of initial values for the weights. Thus, even problems with smaller numbers of variables and training observations can benefit from the use of distributed mode.

PROC NNET Features

The NNET procedure was designed with two goals in mind: to perform efficient, high-speed training of neural networks, and to be as easy to use as possible while still creating models that fit the training data well and generalize well.

PROC NNET has the following basic features:

- ability to train and score using distributed mode
- parallel reading of input data and parallel writing of output data
- high degree of multithreading during all phases of training and scoring
- intelligent defaults for most neural network parameters, such as activation and error functions
- either automatic or manual selection and use of a validation data partition to prevent overfitting during training
- automatic termination of training when the validation error stops improving
• automatic searching for best hidden layers and key parameters such as L1 and L2 regularization norm, learning rate, and so on

• k-fold cross validation to estimate average validation error

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Getting Started: NNET Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example shows how to use the NNET procedure to train a neural network to predict the type of iris plant. The iris data published by Fisher (1936) have been widely used for examples in discriminant and cluster analyses. 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 data set is available in the Sashelp library.

You can load the `sashelp.iris` data into your CAS session by naming your CAS engine libref in the first statement of the following DATA step:

```sas
data mycas.iris;
set sashelp.iris;
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 PROC NNET and output the results to ODS tables:

```sas
proc nnet data=mycas.iris;
  input SepalLength SepalWidth PetalLength PetalWidth;
  target Species / level=nominal;
  hidden 2;
  train outmodel=mycas.nnetModel_gs seed=635117188;
  partition fraction(validate=0.3 seed=103873735);
run;
```

*Figure 18.1* shows the model information for the neural network.
Figure 18.1 Model Information

The NNET Procedure

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
</tr>
<tr>
<td>Number of Observations Used</td>
</tr>
<tr>
<td>Number of Observations Read</td>
</tr>
<tr>
<td>Target/Response Variable</td>
</tr>
<tr>
<td>Number of Nodes</td>
</tr>
<tr>
<td>Number of Input Nodes</td>
</tr>
<tr>
<td>Number of Output Nodes</td>
</tr>
<tr>
<td>Number of Hidden Nodes</td>
</tr>
<tr>
<td>Number of Hidden Layers</td>
</tr>
<tr>
<td>Number of Weight Parameters</td>
</tr>
<tr>
<td>Number of Bias Parameters</td>
</tr>
<tr>
<td>Architecture</td>
</tr>
<tr>
<td>Seed for Initial Weight</td>
</tr>
<tr>
<td>Optimization Technique</td>
</tr>
<tr>
<td>Number of Neural Nets</td>
</tr>
<tr>
<td>Objective Value</td>
</tr>
<tr>
<td>Misclassification Rate for Validation</td>
</tr>
</tbody>
</table>

Figure 18.2 shows the misclassification rate for the training sample.

Figure 18.2 Score Information for Training Data

<table>
<thead>
<tr>
<th>Score Information for Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
</tr>
<tr>
<td>Number of Observations Used</td>
</tr>
<tr>
<td>Misclassification Rate</td>
</tr>
</tbody>
</table>

Figure 18.3 shows the misclassification rate for the validation sample.

Figure 18.3 Score Information for Validation Data

<table>
<thead>
<tr>
<th>Score Information for Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
</tr>
<tr>
<td>Number of Observations Used</td>
</tr>
<tr>
<td>Misclassification Rate</td>
</tr>
</tbody>
</table>
Syntax: NNET Procedure

The following statements are available in the NNET procedure:

```
PROC NNET < options > ;
   INPUT variables }/ LEVEL=INTERVAL | NOMINAL } ;
   HIDDEN number </options> ;
   TARGET variables </options> ;
   TRAIN OUTMODEL=CAS-libref.data-table < options > ;
   ARCHITECTURE architecture-options ;
   WEIGHT variable ;
   PARTITION < partition-options > ;
   OPTIMIZATION < options > ;
   AUTOTUNE < options > ;
   CROSSVALIDATION < options > ;
   OUTPUT OUT=CAS-libref.data-table < option > ;
   CODE < options > ;
```

When you train a neural network, the PROC NNET, INPUT, TARGET, and TRAIN statements are required. At least one HIDDEN statement is required unless you use the GLIM architecture; in that case the HIDDEN statement is not allowed.

When you use a previously trained neural network to score a data table, only the PROC NNET, OUTPUT, and CODE statements are allowed.

PROC NNET Statement

```
PROC NNET < options > ;
```

The PROC NNET statement invokes the procedure. You can specify the following options in the PROC NNET statement:

You can specify the following options:

- **DATA=CAS-libref.data-table**
  names the input data table for PROC NNET to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

  - **CAS-libref** refers to a collection of information that is defined in the LIBNAME statement and includes the *caslib*, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 385.

  - **data-table** specifies the name of the input data table.

**NOTE:** The data set options WHERE, TEMPNAMES, and TEMPEXPRESS are not supported.
INMODEL=\texttt{CAS-libref.data-table}

specifies a model for stand-alone scoring or coding. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the caslib and session identifier, and \texttt{data-table} specifies the name of the input data table. For more information about this two-level name, see the \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 385.

LISTNODE=\texttt{INPUT | HIDDEN | OUTPUT}

specifies the values of the layers to be saved in the scored output or code. Values are normalized values after standardization, if any. This option is valid only if you specify the \texttt{OUTPUT} or \texttt{CODE} statement.

You can specify the following values:

\begin{itemize}
  \item \texttt{INPUT} outputs the values of the input layer.
  \item \texttt{HIDDEN} outputs the values of the hidden layer(s).
  \item \texttt{OUTPUT} outputs the values of the output layer.
\end{itemize}

By default, LISTNODE=\texttt{HIDDEN} for autoencoder networks. For non-autoencoder networkers, no layer information is saved if the LISTNODE= option is not specified.

MISSING=\texttt{MAX | MEAN | MIN}

specifies the statistic with which to impute missing values for interval input variables. If you specify this option, missing values for nominal input variables are treated as a valid level. PROC NNET excludes any observations that have missing values in the target variable.

You can specify the following values:

\begin{itemize}
  \item \texttt{MAX} substitutes missing values with the maximum value of the corresponding interval input variable.
  \item \texttt{MEAN} substitutes missing values with the mean value of the corresponding interval input variable.
  \item \texttt{MIN} substitutes missing values with the minimum value of the corresponding interval input variable.
\end{itemize}

By default, observations that have missing values are excluded from the analysis.

NTHREADS=\texttt{number-of-threads}

specifies the number of threads to use for the computation. The default value is the number of CPUs available. The value of \texttt{number-of-threads} can be from 1 to 64, inclusive.

PRINTTARGET

prints out the ODS tables that contain the mapping of the target variable or its events to variables PROC NNET predicts. The ODS names are PredProbName, PredIntoName, and PredName in Table 18.3. This option is valid when TARGET statement is specified with OUTPUT statement or CODE statement.
STANDARDIZE=NONE | STD | MIDRANGE

specifies the method to use for standardizing the interval inputs.

You can specify the following values:

    NONE          does not alter the variables.
    STD           scales the variables such that their mean is 0 and the standard deviation is 1.
    MIDRANGE      scales the variable such that their midrange is 0 and the half-range is 1. That is, the variables have a minimum of –1 and a maximum of 1. You can also use the keyword RANGE for this method.

By default, STANDARDIZE=MIDRANGE.

TARGETSTANDARDIZE=NONE | STD | MIDRANGE

TARGETSTDE=NONE | STD | MIDRANGE

specifies the method to use for standardizing the interval target.

You can specify the following values:

    NONE          does not alter the variables.
    STD           scales the variables such that their mean is 0 and the standard deviation is 1.
    MIDRANGE      scales the variable such that their midrange is 0 and the half-range is 1. That is, the variables have a minimum of –1 and a maximum of 1. You can also use the keyword RANGE for this method.

By default, TARGETSTANDARDIZE=MIDRANGE.

ARCHITECTURE Statement

ARCHITECTURE architecture-option ;

The ARCHITECTURE statement specifies the architecture of the neural network to be trained.

You can specify one of the following architecture-options (GLIM and MLP DIRECT are not allowed when an autoencoder is used; for more information about the autoencoder, see the section “Autoencoder” on page 409):

    GLIM      specifies a neural network architecture that has no hidden layers (this is equivalent to a generalized linear model). If you specify this architecture-option, the HIDDEN statement is not allowed.

    MLP       specifies a multilayer perceptron architecture that has one or more hidden layers. This is the default architecture. This option is the default for autoencoding.
MLP DIRECT

specifies that direct connections between each input and each target neuron be included when the MLP architecture is used.

When you use PROC NNET to train a neural network, this statement is optional. This statement is not allowed when you use PROC NNET to perform stand-alone scoring.

**NOTE:** If you specify the AUTOTUNE statement, PROC NNET uses the architecture suggested by the tuning optimization. As a result, it might use a different architecture from the one specified in the ARCHITECTURE statement.

---

### AUTOTUNE Statement

**AUTOTUNE < options > ;**

The AUTOTUNE statement activates the tuning optimization algorithm, which searches for the best hidden layers and regularization parameters based on the problem and specified options. If ALGORITHM=SGD or ALGORITHM=ADAM, the algorithm also searches for the best values of the learning rate and annealing rate. When you specify the AUTOTUNE statement, PROC NNET might ignore any specified HIDDEN statement depending on the values of USEPARAMETERS= and TUNINGPARAMETERS= options. You cannot specify both the AUTOTUNE statement and the CROSSVALIDATION statement in the same procedure run.

Table 18.1 summarizes the options you can specify in the AUTOTUNE statement. For more information about all options except the TUNINGPARAMETERS= option, see the section “AUTOTUNE Statement” on page 12 in Chapter 3, “Shared Concepts.” The TUNINGPARAMETERS= option is described in detail following Table 18.1.

#### Table 18.1 AUTOTUNE Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDLOOKUP</td>
<td>Specifies that the table specified in the HISTORYTABLE= option contain the rows from the table specified in the LOOKUPTABLE= option</td>
</tr>
<tr>
<td>EVALHISTORY=</td>
<td>Specifies how to report the evaluation history of the tuner</td>
</tr>
<tr>
<td>FRACTION=</td>
<td>Specifies the fraction of observations to use for validation</td>
</tr>
<tr>
<td>HISTORYTABLE=</td>
<td>Specifies the CAS table that contains the evaluation history</td>
</tr>
<tr>
<td>KFOLD=</td>
<td>Specifies the number of folds for k-fold cross validation</td>
</tr>
<tr>
<td>LIVEUPDATE</td>
<td>Specifies that the table specified in the HISTORYTABLE= option be updated at every evaluation</td>
</tr>
<tr>
<td>LOCALSEARCH</td>
<td>Enables local search optimization</td>
</tr>
<tr>
<td>LOOKUPTABLE=</td>
<td>Specifies the CAS table to use for evaluation lookup</td>
</tr>
<tr>
<td>MAXBAYES=</td>
<td>Specifies the maximum number of points in the kriging model</td>
</tr>
<tr>
<td>MAXEVALS=</td>
<td>Specifies the maximum number of evaluations</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>MAXTIME=</td>
<td>Specifies the maximum time for all iterations</td>
</tr>
<tr>
<td>MAXTRAININGTIME=</td>
<td>Specifies the maximum time for a model training</td>
</tr>
<tr>
<td>NCONVITER=</td>
<td>Specifies the number of convergence iterations</td>
</tr>
</tbody>
</table>
### Table 18.1  
**continued**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOGRIDSHUFFLE</td>
<td>Requests that the grid points not be shuffled</td>
</tr>
<tr>
<td>NPARALLEL=</td>
<td>Specifies the number of parallel sessions</td>
</tr>
<tr>
<td>NSUBSESSIONWORKERS=</td>
<td>Specifies the number of workers in parallel sessions</td>
</tr>
<tr>
<td>OBJECTIVE=</td>
<td>Specifies the objective function</td>
</tr>
<tr>
<td>POPSIZE=</td>
<td>Specifies the population size when SEARCHMETHOD=GA or</td>
</tr>
<tr>
<td></td>
<td>SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>SAMPLESIZE=</td>
<td>Specifies the sample size when SEARCHMETHOD=LHS or</td>
</tr>
<tr>
<td></td>
<td>SEARCHMETHOD=RANDOM</td>
</tr>
<tr>
<td>SEARCHMETHOD=</td>
<td>Specifies the search method that the optimizer uses</td>
</tr>
<tr>
<td>SECONDOBJECTIVE=</td>
<td>Specifies the second objective to use for tuning</td>
</tr>
<tr>
<td>SELECTINITPOINT</td>
<td>Specifies that the tuner select the best evaluation from the lookup table</td>
</tr>
<tr>
<td>TARGETEVENT=</td>
<td>Specifies the target event for ROC-based calculations</td>
</tr>
<tr>
<td>TRAINFRACTION=</td>
<td>Specifies the fraction of observations to use for training</td>
</tr>
<tr>
<td>TUNINGPARAMETERS=</td>
<td>Specifies the custom tuning parameters</td>
</tr>
<tr>
<td>USEPARAMETERS=</td>
<td>Specifies how to handle the TUNINGPARAMETERS= option</td>
</tr>
</tbody>
</table>

**TUNINGPARAMETERS=(suboption | . . | < suboption >)**

**TUNEPARMS=(suboption | . . | < suboption >)**

specifies which parameters to tune and which ranges to tune over. If USEPARAMETERS=STANDARD, this option is ignored.

You can specify one or more of the following `suboptions`:

**ANNEALINGRATE (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**

specifies the range of the annealing rates to use in the tuning process. This option is valid only when ALGORITHM=SGD or ADAM in the OPTIMIZATION statement.

You can specify the following additional `suboptions`:

**LB=number**

specifies the minimum annealing rate to use in the tuning process, where `number` is a nonnegative double. If you specify this suboption, you cannot also specify the VALUES= suboption.

By default, LB=1E−13.

**UB=number**

specifies the maximum annealing rate to use in the tuning process, where `number` is a nonnegative double. If you specify this suboption, you cannot also specify the VALUES= suboption.

By default, UB=1E−2.
VALUES=value-list
  specifies a list of annealing rates to use in the tuning process, where value-list is a space-separated list of nonnegative doubles. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

INIT=number
  specifies the initial annealing rate in the tuning process.
  By default, INIT=1E–06.

EXCLUDE
  excludes the annealing rate from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

LEARNINGRATE (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
  specifies the range of the learning rates to use in the tuning process. This option is valid only when ALGORITHM=SGD or ADAM in the OPTIMIZATION statement.
  You can specify the following additional suboptions:

  LB=number
    specifies the minimum learning rate to use for tuning, where number is a nonnegative double. If you specify this suboption, you cannot also specify the VALUES= suboption.
    By default, LB=1E–6.

  UB=number
    specifies the maximum learning rate to use for tuning, where number is a nonnegative double. If you specify this suboption, you cannot also specify the VALUES= suboption.
    By default, UB=1E–1.

  VALUES=value-list
    specifies a list of learning rates to use in the tuning process, where value-list is a space-separated list of nonnegative doubles. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

  INIT=number
    specifies the initial learning rate to use for tuning.
    By default, INIT=1E–03.

  EXCLUDE
    excludes the learning rate from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

NHIDDEN (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
  specifies the tuning range of the number of hidden layers in the network.
  You can specify the following additional suboptions:
**LB=number**
specifies the minimum number of hidden layers, where *number* is an integer between 0 and 5. If you specify this suboption, you cannot also specify the VALUES= suboption.

By default, LB=0.

**UB=number**
specifies the maximum number of hidden layers, where *number* is an integer between 0 and 5. If you specify this suboption, you cannot also specify the VALUES= suboption.

By default, UB=2.

**VALUES=value-list**
specifies a list of numbers of hidden layers to be searched in the tuning process, where *value-list* is a space-separated list of nonnegative integers. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=number**
specifies the initial number of hidden layers for the tuning process.

By default, INIT=0.

**EXCLUDE**
excludes the number of hidden layers from the tuning process. If you specify this suboption, any specified LB= and UB= suboptions are ignored and you cannot specify any NUNITS*i* suboption.

**NUNITS*i* (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**
specifies tuning information for neurons in the *i*th hidden layer, where *i* is any integer 1–5, inclusive. By default, up to two hidden layers are tried during tuning. An NUNITS*i* suboption takes effect only when the value of the UB= suboption in the NHIDDEN suboptions is greater than or equal to *i*.

You can specify the following additional suboptions:

**LB=number**
specifies the minimum number of neurons in the *i*th hidden layer, where *number* must be a nonnegative integer. If you specify this suboption, you cannot also specify the VALUES= suboption.

By default, LB=1.

**UB=number**
specifies the maximum number neurons in the *i*th hidden layer, where *number* must be a nonnegative integer. If you specify this suboption, you cannot also specify the VALUES= suboption.

By default, UB=min (3*n, 100), where *n* is the number of model inputs.
VALUES=value-list
specifies a list of candidate numbers of neurons in the $i$th hidden layer to be searched in
the tuning process, where value-list is a space-separated list of nonnegative integers. If you
specify this suboption, you cannot specify either the LB= or UB= suboption.

INIT=number
specifies the initial number of neurons in the $i$th hidden layer.
By default, INIT=1.

EXCLUDE
excludes the number of neurons in the $i$th hidden layer from the tuning process. If you
specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are
ignored. If you specify this suboption, you cannot specify the NHIDDEN suboption or any
other NUNITS$ i$ option.

REGL1 (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies the range of L1 regularization values in the tuning process.
You can specify the following additional suboptions:

LB=number
specifies the minimum L1 regularization value in the tuning process, where number is a
nonnegative double. If you specify this suboption, you cannot also specify the VALUES= suboption.
By default, LB=0.

UB=number
specifies the maximum L1 regularization value in the tuning process, where number is a
nonnegative double. If you specify this suboption, you cannot also specify the VALUES= suboption.
By default, UB=10.

VALUES=value-list
specifies a list of L1 regularization values to be searched in the tuning process, where
value-list is a space-separated list of nonnegative doubles. If you specify this suboption, you
cannot specify either the LB= or UB= suboption.

INIT=number
specifies the initial L1 regularization value in the tuning process.
By default, INIT=0.

EXCLUDE
excludes L1 regularization from the tuning process. If you specify this suboption, any
specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.
REGL2 (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies the range of L2 regularization values in the tuning process.

You can specify the following additional suboptions:

**LB=number**
specifies the minimum L2 regularization value in the tuning process, where *number* is a nonnegative double. If you specify this suboption, you cannot also specify the VALUES= suboption.

By default, LB=0.

**UB=number**
specifies the maximum L2 regularization value in the tuning process, where *number* is a nonnegative double. If you specify this suboption, you cannot also specify the VALUES= suboption.

By default, UB=10.

**VALUES=value-list**
specifies a list of L2 regularization values to be searched in the tuning process, where *value-list* is a space-separated list of nonnegative doubles. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=number**
specifies the initial L2 regularization value in the tuning process.

By default, INIT=0.

**EXCLUDE**
excludes L2 regularization from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

---

**CODE Statement**

```
CODE <options>;
```

The CODE statement returns the SAS score code that can be used to score data similar to the input data.

You can specify the following options:

**FILE=filename**
specifies the name of the file where PROC NNET is to write the SAS score code.

**NOCOMPPGM**
oms the logic of the option FRACTION option in the PARTITION statement from the score code.

If you do not specify this option, the logic of the FRACTION option in the PARTITION statement is included in the score code.
CROSSVALIDATION Statement

CROSSVALIDATION < options > ;

The CROSSVALIDATION statement performs $k$-fold cross validation to find the average estimated validation error. You cannot specify this statement if you specify either the AUTOTUNE statement or the PARTITION statement.

You can specify the following options:

**Kfold=number**
- Specifies the number of partition folds in the cross validation process, where number must be between 2 and 20, inclusive.
- By default, KFOLD=5.

**NOPARALLEL**
- Requests that $k$-fold cross validation not be run in parallel. By default, the process runs in parallel.

**NSUBSESSIONWORKERS=number**
- Specifies the number of worker nodes to use in parallel subsessions.
- By default, the value of the NSUBSESSIONWORKERS= option is determined automatically.

HIDDEN Statement

HIDDEN number < /options > ;

The HIDDEN statement specifies the number of neurons or units in a hidden layer, which must be a positive integer. You can specify multiple HIDDEN statements; each HIDDEN statement represents a hidden layer.

You can specify the following options:

**ACT=EXP | IDENTITY | LOGISTIC | RECTIFIER | SIN | TANH**
- Specifies the activation function for the hidden layer.
- By default, ACT=TANH.

You can specify the following measures:

**EXP**
- Specifies the exponential function.

**IDENTITY**
- Specifies the identity function.

**LOGISTIC**
- Specifies the logistic function.

**RECTIFIER**
- Specifies the rectifier activation function.

**SIN**
- Specifies the sine function.

**TANH**
- Specifies the hyperbolic tangent function.
COMB=ADD | LINEAR
  specifies the combination function for the hidden layer.

You can specify the following functions:

  ADD       specifies the additive combination function.
  LINEAR    specifies the linear combination function.

By default, COMB=LINEAR.

HIDDEN statements are ignored when you specify the AUTOTUNE statement.

---

**INPUT Statement**

```plaintext
INPUT variables < / LEVEL=INTERVAL | NOMINAL > ;
```

The INPUT statement identifies the variables in the input data table that are input to the neural network. You can specify multiple INPUT statements.

You can specify the following option:

**LEVEL=INTERVAL | NOMINAL**
  specifies the variables in the input data table. You can specify the following methods:

  INTERVAL       specifies that the variables are interval variables, which must be numeric.
  NOMINAL        specifies that the variables are nominal variables, also known as classification variables, which can be numeric or character.

By default, LEVEL=INTERVAL.

---

**OPTIMIZATION Statement**

```plaintext
OPTIMIZATION < options > ;
```

The OPTIMIZATION statement specifies options for the optimization method that is used to train your model.

When you are training your model, the objective function to be minimized is

\[
    f(w) = \frac{1}{n} \sum_{i=0}^{n} L(w; x_i, y_i) + R(w)
\]

where \( L(w; x_i, y_i) \) is the loss associated with observation \( i \) having data \( x_i \) and correct classification \( y_i \), and \( R(w) \) is a regularization term defined by

\[
    R(w) = \lambda_1 \| w \|_1 + \frac{\lambda_2}{2} \| w \|_2^2
\]

You can specify the following options:
specifies the optimization algorithm to use during training. You can specify the following optimization algorithms:

**ADAM** < sgd-options >  
specifies the adaptive moments (ADAM) algorithm, which is one of the variations of the stochastic gradient descent algorithm. It keeps track of the decaying averages of the past gradients and past squared gradients.

You can specify these additional sgd-options:

**ANNEALINGRATE**=number  
specifies the annealing parameter, $\beta$. Annealing is a way to automatically reduce the learning rate as the algorithm progresses, causing smaller steps as the algorithm approaches a solution. Effectively, it replaces the learning rate parameter, $\eta$, with

$$\eta' = \frac{\eta}{1 + \beta t}$$

where $t$ is the number of iterations that the algorithm has performed.

By default, ANNEALINGRATE=1.0E-6. The `number` must be a nonnegative double.

**COMMFREQ**=number  
specifies the number of minibatches that each computational thread processes before weights are synchronized across all threads and nodes.

**LEARNINGRATE**=number  
specifies the learning rate parameter, $\eta$, for the algorithm. New iterates for the algorithm are found by using

$$w_{k+1} = w_k - \frac{\eta}{\|I_k\|} \sum_{(x_i, y_i) \in I_k} \nabla L(w_k; x_i, y_i)$$

where $w_k$ is the current weight vector, $w_{k+1}$ is the new weight vector, $I_k$ is the minibatch used during iteration $k$, and $L(w_k; x_i, y_i)$ is the loss associated with the $i$th observation.

If you see a huge objective value from the algorithm, especially for a small data set, it is likely that the learning rate is set too high.

By default, LEARNINGRATE=0.001. The `number` must be a nonnegative double.

**MINIBATCHSIZE**=number  
specifies the size of the minibatches to use in the algorithm.

By default, MINIBATCHSIZE=10.

**MOMENTUM**=number  
specifies the value for momentum. The `number` must be greater than or equal to 0 and less than or equal to 1. By default, MOMENTUM=0.
**SEED=number**

specifies the seed for random access of observations on each thread for the algorithm. If *number* is less than or equal to 0 or not specified, a random seed is generated by reading the time of day from the computer’s clock.

**USELOCKING**

specifies that computational threads share a common weight vector and update weight vector without race conditions. If you do not specify this option, computational threads update a single weight vector simultaneously. This causes intentional race conditions and nondeterministic behavior but increases performance significantly.

**HF (Experimental)** specifies the Hessian-free algorithm, a nonlinear optimization algorithm that uses the Hessian vector product to build the second-order information. This algorithm is based on a modified conjugate gradient method. Because it does not use the Hessian directly, it can work for large-scale problems.

**LBFGS < lbgsoptions>** specifies the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm.

You can specify the following *lbgsoption*:

**NUMCORRECTIONS=number**

specifies a memory size limit for the LBFGS algorithm, where *number* is an integer between 1 and 200, inclusive.

By default, NUMCORRECTIONS=6.

**SGD < sgdoptions>** specifies the plain stochastic gradient descent (SGD) algorithm.

You can specify the following *sgdoptions*:

**ANNEALINGRATE=number**

specifies the annealing parameter, \( \beta \). Annealing is a way to automatically reduce the learning rate as the algorithm progresses, causing smaller steps as the algorithm approaches a solution. Effectively, it replaces the learning rate parameter, \( \eta \), with

\[
\eta' = \eta \cdot \frac{1}{1 + \beta t}
\]

where \( t \) is the number of iterations that the algorithm has performed.

By default, ANNEALINGRATE=1.0E–6. The *number* must be a nonnegative double.

**COMMFREQ=number**

specifies the number of minibatches that each computational thread processes before weights are synchronized across all threads and nodes.

**LEARNINGRATE=number**

specifies the learning rate parameter, \( \eta \), for the algorithm. New iterates for the algorithm are found by using

\[
w_{k+1} = w_k - \eta \sum_{(x_i, y_i) \in I_k} \nabla L(w_k; x_i, y_i)
\]

where \( w_k \) is the current weight vector, \( w_{k+1} \) is the new weight vector, \( I_k \) is the minibatch used during iteration \( k \), and \( L(w_k; x_i, y_i) \) is the loss associated with the \( i \)th observation.
If you see a huge objective value from the algorithm, especially for a small data set, it is likely that the learning rate is set too high.

By default, LEARNINGRATE=0.001. The *number* must be a nonnegative double.

**MINIBATCHSIZE=number**

specifies the size of the minibatches to use in the algorithm.

By default, MINIBATCHSIZE=10.

**MOMENTUM=number**

specifies the value for momentum. The *number* must be greater than or equal to 0 and less than or equal to 1. By default, MOMENTUM=0.

**SEED=number**

specifies the seed for random access of observations on each thread for the algorithm. If *number* is less than or equal to 0 or not specified, a random seed is generated by reading the time of day from the computer’s clock.

**USELOCKING**

specifies that computational threads share a common weight vector and update weight vector without race conditions. If you do not specify this option, computational threads update a single weight vector simultaneously. This causes intentional race conditions and nondeterministic behavior but increases performance significantly.

By default, ALGORITHM=LBFGS.

**MAXITER=number**

specifies the iteration budget for training. When ALGORITHM=LBFGS, the algorithm stops after **MAXITER=** iterations if convergence has not been achieved. When ALGORITHM=SGD or ADAM, *number* specifies the desired number of training epochs.

By default, MAXITER=250.

**MAXTIME=number**

specifies the maximum time (in seconds) allowed for optimization, where *number* must be greater than or equal to 1. When this value is reached, the optimization terminates the search and returns results. When MAXTIME=0, no maximum time is set.

By default, MAXTIME=0.

**REGL1=number**

specifies the L1 regularization parameter $\lambda_1$ for the model loss function. The *number* must be nonnegative. Note that this value is autotuned when you specify the AUTOTUNE statement.

By default, REGL1=0.

**REGL2=number**

specifies the L2 regularization parameter $\lambda_2$. The *number* must be nonnegative. Note that this value is autotuned when you specify the AUTOTUNE statement.

By default, REGL2=0.
OUTPUT Statement

```plaintext
OUTPUT OUT=CAS-libref.data-table < option > ;
SCORE OUT=CAS-libref.data-table < option > ;
```

The **OUTPUT** statement creates a new data table that contains the prediction results from using the input data and the model.

You must specify the following option:

```plaintext
OUT=CAS-libref.data-table
```

names the output data table for PROC NNET to use. You must specify this option before any other options. *CAS-libref.data-table* is a two-level name, where

- **CAS-libref** refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 385.

- **data-table** specifies the name of the output data table.

You can also specify the following *option*:

```plaintext
COPYVAR=variable
COPYVARS=(variables)
```

lists one or more *variables* from the input data table to be transferred to the output data table.

```plaintext
ROLE=variable
```

specifies the variable name of the partition indicator when the FRACTION option in the PARTITION statement is specified. The default name is `_Fraction_PartInd_`. Table 18.2 shows how this *variable* is interpreted for each observation.

<table>
<thead>
<tr>
<th>Value</th>
<th>Observation Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Training</td>
</tr>
<tr>
<td>2</td>
<td>Validation</td>
</tr>
<tr>
<td>3</td>
<td>Testing</td>
</tr>
</tbody>
</table>
PARTITION Statement

PARTITION < partition-options > ;

The PARTITION statement specifies how observations in the input data table are logically partitioned into disjoint subsets for model training, validation, and testing. Either you can designate a variable in the input data table and a set of formatted values of that variable to determine the role of each observation, or you can specify proportions to use for random assignment of observations for each role. Alternatively, you can use a separate validation data table in the TRAIN statement to do validation.

You can specify the following mutually exclusive partition-options:

ROLEVAR=variable(TRAIN=value VALIDATE=value < TEST=value>)

names the variable in the input data table whose values are used to assign roles to each observation. The formatted values of this variable, which are used to assign observations roles, are specified in the TEST=, TRAIN=, and VALIDATION= suboptions. The VALIDATE= suboptions is required; the TRAIN= and TEST= suboptions are optional. If you do not specify the TRAIN= suboption, the training subset that PROC NNET uses is the complement set of the VALIDATE= suboption, or the complement set of the VALIDATE= and TEST= suboptions if they are both specified.

FRACTION(VALIDATE=fraction TEST=fraction < SEED=random-seed >)

randomly assigns the specified proportions of the observations in the input data table to training and validation roles. You specify the proportions for testing and validation by using the TEST= and VALIDATE= suboptions. The VALIDATE= suboption is required, and the TEST= suboption is optional. If you specify both the TEST= and VALIDATE= suboptions, then the sum of the specified fractions must be less than 1 and the remaining fraction of the observations are assigned to the training role. Otherwise, the PARTITION statement is ignored. The range of the VALIDATE= and TEST= suboptions is from 1E–5 to 1 – (1E–5), inclusive. You can specify the ROLE option to name the partition indicator.

NOTE: The split between training, validation, and test observations can only approximate the requested fraction, because the fraction is used as a cutoff value for a random number generator to determine the actual split. If you require a more accurate split, you must use the ROLEVAR= option to specify the split explicitly.

You cannot use the PARTITION statement along with the CROSSVALIDATION statement.

TARGET Statement

TARGET variable < /options > ;

The TARGET statement specifies the target variable for the neural network. If you do not specify a TARGET statement, PROC NNET trains an autoencoder. For more information, see the section “Autoencoder” on page 409.

You can specify the following options:
**ACT=EXP | IDENTITY | SIN | SOFTMAX | TANH**

specifies the activation function for the target.

You can specify the following values:

- **EXP** specifies the exponential function. You can use ACT=EXP only with ERROR=GAMMA or ERROR=POISSON.
- **IDENTITY** specifies the identity function.
- **SIN** specifies the sine function.
- **SOFTMAX** specifies the softmax function.
- **TANH** specifies the hyperbolic tangent function.

For the GLIM architecture, you can only specify ACT=IDENTITY for an interval target and ACT=SOFTMAX for a nominal target, which are the same by default. For the MLP or MLP DIRECT architecture, the SOFTMAX method is used only with a nominal target, whereas the other methods are used only with an interval target. By default, ACT=IDENTITY for the interval target and ACT=SOFTMAX for the nominal target.

**COMB=ADD | LINEAR**

specifies the combination function for the target layer. You can specify the following combination functions:

- **ADD** specifies the additive combination function.
- **LINEAR** specifies the linear combination function.

By default, COMB=LINEAR.

**ERROR=ENTROPY | GAMMA | NORMAL | POISSON**

specifies the error function. The entropy error function is used only when LEVEL=NOM. You can specify the following error functions:

- **ENTROPY** specifies the cross-entropy function.
- **GAMMA** specifies the gamma error function. This function is usually used when you want to predict the time between events. Only ACT=EXP is valid when ERROR=GAMMA.
- **NORMAL** specifies the normal error function, which is the sum of the squared differences between the network output and the target value.
- **POISSON** specifies the Poisson error function. This function is usually used when you want to predict the number of events per unit time. Only ACT=EXP is valid when ERROR=POISSON.

By default, ERROR=NORMAL when LEVEL=INT, and ERROR=ENTROPY when LEVEL=NOM.
LEVEL=INTERVAL | NOMINAL
specifies the variable type. You can specify the following values:

INTERVAL specifies that the variable is interval, which must be numeric.
NOMINAL specifies that the variable is nominal, also known as a classification variable, which can be numeric or character.

By default, LEVEL=INTERVAL.

TRAIN Statement

TRAIN OUTMODEL=CAS-libref.data-table <options> ;

The TRAIN statement causes the NNET procedure to use the training data that are specified in the PROC NNET statement to train a neural network model whose structure is specified in the ARCHITECTURE, INPUT, TARGET, and HIDDEN statements. The goal of training is to determine a set of network weights that best predicts the targets in the training data while still doing a good job of predicting targets of unseen data (that is, generalizing well and not overfitting).

Training starts with a pseudorandomly generated set of initial weights. PROC NNET then computes the objective function for the training partition, and the optimization algorithm adjusts the weights. This process is repeated until any one of the following conditions is met:

- The objective function that is computed using the training partition stops improving.
- The objective function that is computed using the validation partition stops improving.
- The process has been repeated the number of times specified in the MAXITER= and MAXTIME= options in the OPTIMIZATION statement.

When you are training, you must include exactly one TRAIN statement. The TRAIN statement is not allowed when you are doing stand-alone scoring.

You must specify the following option:

OUTMODEL=CAS-libref.data-table
specifies the final model from training. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 385.

You can use the model data table later to score a different input data table as long as the variable names and types of the variables in the new input data table match those in the training data table.

You can also specify the following options:
DROPOUTHIDDEN=ratio
specifies the dropout ratio of hidden layers. This option is valid only when you specify ALGORITHM=SGD or ADAM in the OPTIMIZATION statement and when all the connections use the linear combination function. The ratio must be between 0 and 1, inclusive.

By default, DROPOUTHIDDEN=0.

DROPOUTINPUT=ratio
specifies the dropout ratio of input layers. This option is valid only when you specify ALGORITHM=SGD or ADAM in the OPTIMIZATION statement and when all the connections use the linear combination function. The ratio must be between 0 and 1, inclusive.

By default, DROPOUTINPUT=0.

NUMTRIES=number
specifies the number of times the network is to be trained using a different starting point. Specifying this option helps ensure that the optimizer finds the table of weights that truly minimizes the objective function and does not return a local minimum. The value of number must be an integer between 1 and 20,000, inclusive. By default, NUMTRIES=1.

**NOTE:** When NUMTRIES > 1, the ODS tables “OptIterHistory” and “ConvergenceStatus” are suppressed.

RESUME
trains with the initial weight that is specified in the INMODEL= option in the PROC NNET statement. If you specify the RESUME option, you must also specify the INMODEL= option.

STAGNATION=number
specifies the number of iterations that result in no improvement for the validation subset before early stopping takes effect during training. This option is valid only when the VALIDATION= option or the PARTITION statement is specified.

By default, STAGNATION=3.

VALIDATION=CAS-libref.data-table
specifies a separate data table for validation during training. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions andCAS Engine Librefs” on page 385.

If you specify both the VALIDATION= option and the PARTITION statement, the PARTITION statement is ignored. The VALIDATION= data table must have the same variables that you specify in the DATA= option in the PROC NNET statement.

VALIDGOAL=number
specifies the number targeted goal of validation error before early stopping takes effect during training. This option is valid only when the VALIDATION= option or the PARTITION statement is specified.

By default, VALIDGOAL=0.
WEIGHT Statement

**WEIGHT** variable;

If you specify a WEIGHT statement, **variable** identifies a numeric **variable** in the input data table that contains the weight to be placed on the prediction error (the difference between the output of the network and the target value specified in the input data table) for each observation during training.

If the **variable** is less than or equal to 0 or is missing, the observation is not used for training. When you perform scoring, **PROC NNET** scores the observation even if the weight is less than or equal to 0 or missing.

The WEIGHT statement is optional. If a WEIGHT statement is not included, all observations are assigned a weight of 1.

Details: NNET Procedure

Computational Method

**PROC NNET** trains a multilayer perceptron neural network that contains one or more hidden layers. For more information about multilayer perceptron neural networks, see Bishop (1995).

The NNET procedure does not have many parameters that you must specify for training. You must specify where the training data are (in the DATA= option in the **PROC NNET** statement), the names and types of the input variables (in the INPUT statement), the names and types of the target variables (in the TARGET statement), the number of hidden layers (in the HIDDEN statement), and the number of neurons in each hidden layer (in the HIDDEN statement).

Optionally, you can also specify where to write the score file that contains targets from the input file and predicted target variables from the trained network and where to write the model file that contains the parameters of the trained network (in the OUTPUT statement). In addition, you can specify where to write the SAS DATA step statements that you can use to score new data tables (in the CODE statement).

The optimization algorithms available in **PROC NNET** are the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm (LBFGS), the stochastic gradient descent algorithm (SGD), the adaptive moments algorithm (ADAM), and the Hessian-free algorithm (HF), which are popular methods of solving large-scale nonlinear optimization problems. The algorithms terminate based on criteria such as convergence tolerance and maximum iterations. In addition, **PROC NNET** stops if the validation error (which is calculated after each line search) lacks improvement a certain number of times in a row.

The most important parameters that you can specify are the number of hidden layers in the network and the number of neurons in each hidden layer. A good strategy is to start with a single hidden layer by specifying...
a single HIDDEN statement with a small number of neurons, and slowly increase the number until the validation error stops improving.

The next most important parameter that you can specify is the number of times the network is to be retrained using different sets of initial weights (in the NUMTRIES= option in the TRAIN statement). Finally, unless your training data table is very large, you should set the MAXITER= option in the OPTIMIZATION statement to a large number, say 1,000 or more to prevent the optimization algorithm from stopping prematurely. The value of the MAXITER= option is only a limit; for example, specifying MAXITER=1000 does not mean that the algorithm runs for 1,000 iterations. Most training runs use far fewer iterations. If you have a large data table, you can start with MAXITER=1 to see how long a single iteration takes, and then increase the MAXITER= value.

Hyperparameter Tuning

For more information about hyperparameter tuning, see the section “Hyperparameter Tuning” on page 23 in Chapter 3, “Shared Concepts.”

You can tune the following hyperparameter values when you specify the AUTOTUNE statement:

- number of hidden layers
- number of hidden units in each hidden layer
- REGL1= option for the L1 regularization parameter
- REGL2= option for the L2 regularization parameter
- ANNEALINGRATE= option for the annealing rate parameter, which is used by the training algorithm of the SGD or ADAM optimizer when you specify ALGORITHM=SGD or ADAM in the OPTIMIZATION statement
- LEARNINGRATE= option for the learning rate parameter, which is used by the training algorithm of the SGD or ADAM optimizer when you specify ALGORITHM=SGD or ADAM in the OPTIMIZATION statement

k-fold Cross Validation

The CROSSVALIDATION statement performs k-fold cross validation to assess the accuracy of a model. During cross validation, all data are divided into k subsets (folds), where k is the value of the KFOLD= option. For each fold, a new model is trained on the (k–1) folds and then validated using the selected (holdout) fold. The assessment metrics are then averaged across all the holdout folds. The CROSSVALIDATION statement creates a table that has k+1 rows. The first k rows contain the assessment metrics for each holdout fold, and the last row contains the average across all the holdout folds.
Determining the Number of Parallel Evaluations

The number of parallel fold evaluations is determined as follows:

1. The number of worker nodes to use in parallel subsessions is determined:
   
   a) If you specify a value greater than 0 in the NSUBSESSIONWORKERS= option to indicate the number of workers to use in parallel subsessions, then that value is used for each subsession.
   
   b) If you do not specify a value in the NSUBSESSIONWORKERS= option, then the number of workers for each subsession is the same as the number of workers used in the parent session.
   
   c) If you specify a value of 0 in the NSUBSESSIONWORKERS= option, or if you specify the default value of 0, then the number of workers to use in each subsession defaults to the number of workers used 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 on the server, then the number of workers for each subsession is reduced to the number of workers on 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+Number of Observations*Number of Columns / 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, then the number of parallel evaluations is limited to 16. Otherwise, the number of parallel evaluations is equal to the number of folds.
   
   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.

Autoencoder

An autoencoder is a neural network that is used for efficient codings and widely used for feature extraction and nonlinear principal component analysis. Architecturally, an autoencoder is similar to a multilayer perceptron neural network because it has an input layer, hidden layers, and an output layer. However, it differs in that the output layer is duplicated from the input layer. Therefore, autoencoders are unsupervised learning models. You can train autoencoder in PROC NNET by omitting the TARGET and ARCHITECTURE statements. The score output of the autoencoder contains the learned features of hidden layers.
Displayed Output

PROC NNET displays basic fit statistics in the SAS log and more detailed information in several ODS tables.

When you are scoring, PROC NNET displays the mean square error for an interval target and the misclassification rate for a nominal target. The scoring summary is based on the entire input data table if there is no validation subset. If a validation or testing subset is used, the scoring is performed for each partition. In addition, PROC NNET generates ODS tables that display detailed information about the model structure, input data, iteration history, and status for optimization solver.

The following sections describe the output that PROC NNET produces.

Iteration History

The “Iteration History” table contains the iteration history from the optimization solver. This table is suppressed if the NUMTRIES option is greater than 1 in the TRAIN statement.

Convergence Status

The “Convergence Status” table contains the returning status of optimization solver. This table is suppressed if the NUMTRIES option is greater than 1 in the TRAIN statement.

Model Information

The “Model Information” table contains information about the neural network model.

Score Information

The “Score Information” table contains the misclassification rate or mean square error for training, validation, or testing partition sets.

Tuner Information

The “Tuner Information” table contains the values of options used by tuner.

Tuner Results

The “Tuner Results” table contains the values of hyperparameters, objective function for the default configuration (Iteration 0), and up to 10 best configurations found from tuner.

Evaluation History

The “Evaluation Results” table contains values of all tuning parameters and the objective function for all model configurations evaluated by the tuner.
Best Configuration

The “Best Configuration” table contains the values of hyperparameters and objective function for the best configuration from tuner.

Tuner Summary

The “Tuner Summary” table contains statistics for the tuning process.

TunerTiming

The “TunerTiming” table contains the run time break down of different tasks during tuning.

HyperparameterImportance

The “HyperparameterImportance” table contains the relative importance of all hyperparameters in the tuning process, in percentage. The most important hyperparameter is assigned a value of 100%; all others are rated relative to the most important one. The values are calculated on the basis of the evaluation history of the autotuning process.

ROCInfo

The “ROCInfo” table contains the ROC curve information for the specified event in the nominal target. For more information, see Chapter 26, “The ASSESS Procedure” (SAS Visual Statistics: Procedures).

FitStat

The “FitStat” table contains various error metrics for supervised learning models. For more information, see Chapter 26, “The ASSESS Procedure” (SAS Visual Statistics: Procedures).

Cross Validation Results

The “Cross Validation Results” table contains the average error rate (misclassification error or average square error) of $k$-fold cross validation.

PredProbName

The “PredProbName” table contains the name mapping between the score variables and the corresponding events in the nominal target.

PredIntoName

The “PredIntoName” table contains the name mapping between the predicted target variable and the original target variable when the target is nominal.
**PredName**

The “PredName” table contains the name mapping between the predicted target variable and the original target variable when the target is interval.

---

**ODS Table Names**

Each table that the NNET procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. The name of each table and a short description of the contents are listed in Table 18.3.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>BestConfiguration</td>
<td>Hyperparameters and objective function values for the best configuration</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>ConvergenceStatus</td>
<td>Convergence status</td>
<td>PROC NNET</td>
<td>Default</td>
</tr>
<tr>
<td>CrossValidateMLFitStat</td>
<td>Error metrics for $k$-fold cross validation</td>
<td>CROSSVALIDATION</td>
<td>Default</td>
</tr>
<tr>
<td>CrossValidationResults</td>
<td>Average error rate (misclassification error or average square error) of $k$-fold cross validation</td>
<td>CROSSVALIDATION</td>
<td>Default</td>
</tr>
<tr>
<td>EvaluationHistory</td>
<td>Values of the hyperparameters, the objective function for the all model configurations evaluated by the tuner</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>FitStat</td>
<td>Error metrics for supervised learning models</td>
<td>AUTOTUNE, TARGET</td>
<td>Default</td>
</tr>
<tr>
<td>HyperparameterImportance</td>
<td>Relative importance of all hyperparameters</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Information about the modeling environment</td>
<td>PROC NNET</td>
<td>Default</td>
</tr>
<tr>
<td>OptIterHistory</td>
<td>Iteration history information</td>
<td>PROC NNET</td>
<td>Default</td>
</tr>
<tr>
<td>PredIntoName</td>
<td>The name mapping between the predicted target variable and the original target variable when the target is nominal</td>
<td>SCORE, CODE, TARGET PRINTTARGET LEVEL=NOMINAL</td>
<td>Default</td>
</tr>
<tr>
<td>Table Name</td>
<td>Description</td>
<td>Statement</td>
<td>Option</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>PredName</td>
<td>The name mapping between the predicted target variable and the original target variable when the target is interval.</td>
<td>SCORE, CODE, TARGET</td>
<td>PRINTTARGET, LEVEL=INTERVAL</td>
</tr>
<tr>
<td>PredProbName</td>
<td>The name mapping between the score variables and the corresponding events in the nominal target.</td>
<td>SCORE, CODE, TARGET</td>
<td>PRINTTARGET, LEVEL=NOMINAL</td>
</tr>
<tr>
<td>ROCInfo</td>
<td>ROC curve information for the specified event in the nominal target.</td>
<td>AUTOTUNE, TARGET</td>
<td>LEVEL=NOMINAL</td>
</tr>
<tr>
<td>ScoreInfo</td>
<td>Misclassification rate or mean square error for partition sets.</td>
<td>SCORE, PARTITION</td>
<td>VALIDATION=</td>
</tr>
<tr>
<td>TunerInfo</td>
<td>Setup values used by the tuner.</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerResults</td>
<td>Values of the hyperparameters, the objective function for the default configuration (Iteration 0), and up to 10 best configurations found.</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerSummary</td>
<td>Statistics about the tuning process.</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerTiming</td>
<td>Total time spent on different tasks during tuning.</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
</tbody>
</table>
Examples: NNET Procedure

**Example 18.1: Binary Target Classification with Partition**

This example demonstrates how to use PROC NNET to predict whether a mortgage applicant will default on a loan. The data table Hmeq, which is in the Sampsio library that SAS provides, contains observations for 5,960 mortgage applicants. A variable named Bad indicates whether the applicant, after being approved for a loan, paid off or defaulted on the loan.

The following DATA steps load the Hmeq data set into a CAS session by naming a CAS engine libref in the first statement of each step:

```sas
data mycas.hmeq;
  set sampsio.hmeq;
run;
```

These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

The following statements run PROC NNET and output the results to ODS tables. Based on the specified percentages, the PARTITION statement randomly splits the hmeq data set into three partitions: training, validation, and testing. The error rates of the validation data set are computed during the training process as one of the stopping criteria in order to prevent overfitting.

```sas
proc nnet data=mycas.hmeq standardize=midrange missing=mean;
  architecture mlp;
  input job reason / level=nominal;
  input debtinc delinq loan mortdue value yoj derog clage clno;
  hidden 7;
  target bad / level=nominal;
  optimization algorithm=lbfgs maxiter=500;
  train outmodel=mycas.nnetmodel1 seed=12345;
  partition fraction(validate=0.2 test=0.1 seed=54321);
run;
```

The PROC NNET call creates the model, nnetModel1, from the training data; this model contains all the weight of the neural network. The example is run on one controller node.

Output 18.1.1 shows the model information for the neural network.
**Output 18.1.1** Model Information

The NNET Procedure

<table>
<thead>
<tr>
<th>Model Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Neural Net</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>4111</td>
</tr>
<tr>
<td>Number of Observations Read</td>
<td>4111</td>
</tr>
<tr>
<td>Target/Response Variable</td>
<td>BAD</td>
</tr>
<tr>
<td>Number of Nodes</td>
<td>28</td>
</tr>
<tr>
<td>Number of Input Nodes</td>
<td>19</td>
</tr>
<tr>
<td>Number of Output Nodes</td>
<td>2</td>
</tr>
<tr>
<td>Number of Hidden Nodes</td>
<td>7</td>
</tr>
<tr>
<td>Number of Hidden Layers</td>
<td>1</td>
</tr>
<tr>
<td>Number of Weight Parameters</td>
<td>140</td>
</tr>
<tr>
<td>Number of Bias Parameters</td>
<td>9</td>
</tr>
<tr>
<td>Architecture</td>
<td>MLP</td>
</tr>
<tr>
<td>Seed for Initial Weight</td>
<td>12345</td>
</tr>
<tr>
<td>Optimization Technique</td>
<td>LBFGS</td>
</tr>
<tr>
<td>Number of Neural Nets</td>
<td>1</td>
</tr>
<tr>
<td>Objective Value</td>
<td>2.0680574741</td>
</tr>
<tr>
<td>Misclassification Rate for Validation</td>
<td>0.1796</td>
</tr>
</tbody>
</table>

Output 18.1.2 shows the misclassification rate of the training data.

**Output 18.1.2** Score Information for Training Data

<table>
<thead>
<tr>
<th>Score Information for Training</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
<td>4111</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>4111</td>
</tr>
<tr>
<td>Misclassification Rate</td>
<td>0.208</td>
</tr>
</tbody>
</table>

Output 18.1.3 shows the misclassification rate of the validation data.

**Output 18.1.3** Score Information for Validation Data

<table>
<thead>
<tr>
<th>Score Information for Validation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
<td>1225</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>1225</td>
</tr>
<tr>
<td>Misclassification Rate</td>
<td>0.1796</td>
</tr>
</tbody>
</table>

Output 18.1.4 shows the misclassification rate of the testing data.

**Output 18.1.4** Score Information for Testing Data

<table>
<thead>
<tr>
<th>Score Information for Testing</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
<td>624</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>624</td>
</tr>
<tr>
<td>Misclassification Rate</td>
<td>0.1827</td>
</tr>
</tbody>
</table>
Example 18.2: Finding the Best Neural Network Configuration

This example illustrates how to use the AUTOTUNE statement to search for the best set of hyperparameters within the domains that you specify. The data set (iris) is the same data set as is used in the section “Getting Started: NNET Procedure” on page 386. The AUTOTUNE statement searches for the best network for iris within two hidden layers (each of which has specified ranges), and it also searches for the best L1 and L2 regularization values based on the specified ranges. Only one controller node is used in the example.

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. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```plaintext
data mycas.iris;
  set sashelp.iris;
run;
```

The following statements run PROC NNET and output the results to ODS tables. The AUTOTUNE statement activates the tuning optimization algorithm, which applies the specified ranges in the local searching process.

```plaintext
proc nnet data=mycas.iris;
  input SepalLength SepalWidth PetalLength PetalWidth;
  target Species / level=nominal;
  train outmodel=mycas.nnetModel2 seed=1517878693;
  autotune useparameters=custom objective=MCE searchmethod=GA
    tuningparameters=(nhidden(LB=1 UB=2 INIT=1)
      nunits1(LB=1 UB=10 INIT=1)
      nunits2(LB=1 UB=15 INIT=2)
      regl1(LB=1e-04 UB=1e-02 INIT=1e-03)
      regl2(LB=1e-04 UB=1e-02 INIT=1e-03)
    );
  optimization algorithm=LBFGS maxiter=100;
run;
```

Output 18.2.1 shows the setup values used by the tuner.

**Output 18.2.1 Tuner Information**

<table>
<thead>
<tr>
<th>The NNET Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuner Information</td>
</tr>
<tr>
<td>Model Type</td>
</tr>
<tr>
<td>Tuner Objective Function</td>
</tr>
<tr>
<td>Search Method</td>
</tr>
<tr>
<td>Population Size</td>
</tr>
<tr>
<td>Maximum Iterations</td>
</tr>
<tr>
<td>Maximum Tuning Time in Seconds</td>
</tr>
<tr>
<td>Validation Type</td>
</tr>
<tr>
<td>Validation Partition Fraction</td>
</tr>
<tr>
<td>Log Level</td>
</tr>
<tr>
<td>Seed</td>
</tr>
<tr>
<td>Number of Parallel Evaluations</td>
</tr>
<tr>
<td>Number of Workers per Subsession</td>
</tr>
</tbody>
</table>
Output 18.2.2 shows the results reported by the NNET procedure. The first row displays results from the default settings, the second row displays the best results found by the tuner, and the third row displays the second-best results found.

### Output 18.2.2 Tuner Results

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Hidden Layers</th>
<th>Neurons in Hidden Layer 1</th>
<th>Neurons in Hidden Layer 2</th>
<th>L1 Regularization</th>
<th>L2 Regularization</th>
<th>Misclassification</th>
<th>Evaluation Time in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.001000</td>
<td>0.001000</td>
<td>0.3333</td>
<td>2.01</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.001179</td>
<td>0.000231</td>
<td>0.0000</td>
<td>1.46</td>
</tr>
<tr>
<td>51</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.001179</td>
<td>0.000231</td>
<td>0.0000</td>
<td>1.66</td>
</tr>
<tr>
<td>52</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.001179</td>
<td>0.000231</td>
<td>0.0000</td>
<td>1.62</td>
</tr>
<tr>
<td>65</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.001179</td>
<td>0.000231</td>
<td>0.0000</td>
<td>1.45</td>
</tr>
<tr>
<td>67</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.001179</td>
<td>0.000231</td>
<td>0.0000</td>
<td>1.55</td>
</tr>
<tr>
<td>81</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.001179</td>
<td>0.000231</td>
<td>0.0000</td>
<td>1.46</td>
</tr>
<tr>
<td>83</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.001179</td>
<td>0.000231</td>
<td>0.0000</td>
<td>1.71</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.001000</td>
<td>0.001000</td>
<td>0.0222</td>
<td>23.28</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>0.001000</td>
<td>0.001000</td>
<td>0.0222</td>
<td>24.02</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>0.010000</td>
<td>0.001000</td>
<td>0.0222</td>
<td>24.02</td>
</tr>
</tbody>
</table>

Output 18.2.3 shows the best values of the tuning parameters from the tuning process.

### Output 18.2.3 Best Configuration

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Hidden Layers</th>
<th>Neurons in Hidden Layer 1</th>
<th>Neurons in Hidden Layer 2</th>
<th>L1 Regularization</th>
<th>L2 Regularization</th>
<th>Misclassification</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.00117874</td>
<td>0.00023089</td>
<td>0</td>
</tr>
</tbody>
</table>

Output 18.2.4 shows the tuner summary.

### Output 18.2.4 Tuner Summary

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Configuration Objective Value</td>
<td>0.3333</td>
</tr>
<tr>
<td>Best Configuration Objective Value</td>
<td>0</td>
</tr>
<tr>
<td>Worst Configuration Objective Value</td>
<td>0.3333</td>
</tr>
<tr>
<td>Initial Configuration Evaluation Time</td>
<td>2.0096</td>
</tr>
<tr>
<td>Best Configuration Evaluation Time</td>
<td>1.4565</td>
</tr>
<tr>
<td>Number of Improved Configurations</td>
<td>4</td>
</tr>
<tr>
<td>Number of Evaluated Configurations</td>
<td>86</td>
</tr>
<tr>
<td>Total Tuning Time in Seconds</td>
<td>34.5837</td>
</tr>
<tr>
<td>Parallel Tuning Speedup</td>
<td>3.9902</td>
</tr>
</tbody>
</table>
Output 18.2.5 shows the run time for each task during the searching process. It is evident that the tuner spent the vast majority of time on training; this behavior is similar to most tuner runs. Therefore, it is important to understand that tuning might take a very long time by nature if the training time is long. Typically, networks that have more neurons or larger training samples take more time; also, if the value of the MAXITER= option is very large and the nonlinear objective function converges slowly, the run time could be very long. In general, tuner performance should not be a concern, because you typically use tuning only once in a while.

<table>
<thead>
<tr>
<th>Task</th>
<th>Seconds</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Training</td>
<td>71.80</td>
<td>52.03</td>
</tr>
<tr>
<td>Model Scoring</td>
<td>42.24</td>
<td>30.61</td>
</tr>
<tr>
<td>Total Objective Evaluations</td>
<td>114.06</td>
<td>82.66</td>
</tr>
<tr>
<td>Tuner</td>
<td>23.93</td>
<td>17.34</td>
</tr>
<tr>
<td>Total CPU Time</td>
<td>138.00</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Output 18.2.6 shows the relative importance of hyperparameter.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Relative Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGL1</td>
<td>1.0000</td>
</tr>
<tr>
<td>NUNITS1</td>
<td>0.9351</td>
</tr>
<tr>
<td>REGL2</td>
<td>0.4244</td>
</tr>
<tr>
<td>NUNITS2</td>
<td>0.0297</td>
</tr>
<tr>
<td>NHIDDEN</td>
<td>0.0192</td>
</tr>
</tbody>
</table>

References


Chapter 19
The RPCA Procedure

Contents

Overview: RPCA Procedure .............................................. 420
PROC RPCA Features ................................................ 420
Using CAS Sessions and CAS Engine Librefs ....................... 421
Getting Started: RPCA Procedure .................................. 422
Syntax: RPCA Procedure ................................................ 426
PROC RPCA Statement ................................................ 426
ANOMALYDETECTION Statement ................................ 430
CODE Statement ...................................................... 431
DISPLAY Statement .................................................. 431
DISPLAYOUT Statement ............................................ 432
ID Statement .......................................................... 433
INPUT Statement ...................................................... 433
OUTDECOMP Statement ............................................. 433
SAVESTATE Statement .............................................. 434
SVD Statement ........................................................ 435
Details: RPCA Procedure .............................................. 436
Augmented Lagrange Multiplier Method ......................... 436
Accelerated Proximal Gradient Method ......................... 437
Scaling and Centering ............................................ 437
Cumulative Eigenvalue Percentage Tolerance .................... 438
Anomaly Detection ............................................... 438
Displayed Output ................................................. 439
ODS Table Names ................................................... 440
Output Data Tables .................................................. 440
Examples: RPCA Procedure ......................................... 441
Example 19.1: Analyzing the Energy Output of the SAS Solar Farm ........................................ 441
References ..................................................................... 444
Overview: RPCA Procedure

The RPCA procedure implements robust principal component analysis (RPCA) in SAS Viya. RPCA can be used in many areas, including image processing, latent semantic indexing, ranking, and matrix completion (Candès et al. 2011).

The RPCA procedure decomposes an input matrix into a sum of two matrices: a low-rank matrix and a sparse matrix. You can use the low-rank matrix to do feature extraction and use the sparse matrix to detect anomalies.

Robustness in RPCA comes from the property that the principal components are computed from observations after removing the outliers—that is, from the low-rank matrix. Many applications of RPCA focus on the sparse matrix. One example is the extraction of moving objects from the background in surveillance videos.

PROC RPCA stores the results of the input matrix decomposition in output tables that are produced by the OUTLOWRANK=, OUTSPARSE=, and OUTERROR= options. Also, PROC RPCA stores the results of the low-rank matrix decomposition in output tables that are produced by the OUTDECOMP statement.

PROC RPCA Features

The RPCA procedure has the following features:

- reads input data in parallel when the data source is on a distributed system
- is multithreaded during all phases of analytic execution
- supports large-scale training data
- enables you to do the analysis on selected columns of the data
- enables you to generate multiple decomposition outputs
- enables you to project new data on principal components as well as to project new data in the low-rank space
- enables you to do anomaly detection
**Using CAS Sessions and CAS Engine Librefs**

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Getting Started: RPCA Procedure

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example shows how to use the RPCA procedure 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 data set rpcaData has 50 observations and three variables: Index, X, and Y. Index is simply an observation number. X is a randomly generated number between 0 and 1. Y is X plus a random term between 0:1 and 0.1. This data set also has three outliers for variable Y in observations 7, 20, and 33.

The following DATA step creates the rpcaData data set:

```sas
data rpcaData;
  input index X Y;
dateline;
  1  0.522  0.510
  2  0.642  0.583
  3  0.628  0.543
  4  0.826  0.875
  5  0.101  0.031
  6  0.310  0.311
  7  0.447  4.421
  8  0.419  0.481
  9  0.861  0.874
 10  0.418  0.334
 11  0.929  1.020
 12  0.946  0.946
 13  0.548  0.567
 14  0.626  0.643
 15  0.616  0.581
 16  0.684  0.622
 17  0.438  0.450
 18  0.264  0.174
 19  0.705  0.607
 20  0.932  3.024
 21  0.866  0.836
 22  0.145  0.138
 23  0.225  0.133
 24  0.577  0.515
 25  0.815  0.832
 26  0.678  0.706
 27  0.844  0.892
 28  0.996  1.033
 29  0.695  0.676
 30  0.988  0.930
 31  0.684  0.614
 32  0.582  0.660
 33  0.048  2.004
```
Getting Started: RPCA Procedure

You can load rpcaData into your CAS session by naming your CAS engine libref in the first statement of the following DATA step:

```plaintext
data mycas.rpcaData;
set rpcaData;
run;
```

This statement assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

The following code runs PROC RPCA and outputs the decomposition results to the mycas CAS library engine:

```plaintext
proc rpca data=mycas.rpcaData
decomp=svd
outlowrank=mycas.lowrankmat
outsparse=mycas.sparsemat;
  id index;
  outdecomp svdleft=mycas.svdleft
    svddiag=mycas.svddiag
    svdright=mycas.svdright;
run;
```

These statements produce the mycas.lowrankmat and mycas.sparsemat tables, which are the decompositions of mycas.rpcaData. They also produce the mycas.svdleft, mycas.svddiag, and mycas.svdright tables, which are the decompositions of the mycas.lowrankmat table. The DECOMP= option produces the SVD decomposition of the low-rank matrix.
Figure 19.1 displays the “Model Information,” “Dimensions,” and “Results Summary” tables. The “Model Information” table shows the default parameters for the RPCA method, the SVD method, Lambda, and LambdaWeight. The “Dimensions” table shows the number of observations and variables in the input table and the number of observations that have missing values. (PROC RPCA ignores the 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 29 iterations. Furthermore, you can observe in this table that the rank of the low-rank matrix is 1, as expected from the fact that variable $Y$ is highly correlated with variable $X$. Note that the sparsity value is 0.16, which indicates that the sparse matrix contains many nonzero values. As you increase the value of the LambdaWeight parameter, the sparsity of the sparse matrix increases.

The default value of the LambdaWeight parameter is 1. You can use the following PROC RPCA statement to change this parameter to 3.5. Figure 19.2 shows that the sparsity of the sparse matrix increases to 0.97 and the rank of the low-rank matrix increases to 2.

```plaintext
proc rpcas data=mycas.rpcadata
   lambdaweight = 3.5
   outsparse=mycas.sparsemat2;
   id index;
run;

proc print data=mycas.sparsemat2;
run;
```
**Figure 19.2** Results Summary When LambdaWeight Is 3.5

<table>
<thead>
<tr>
<th>Results Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparsity of Sparse Matrix</td>
</tr>
<tr>
<td>Rank of Low-Rank Matrix</td>
</tr>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Solution Status</td>
</tr>
<tr>
<td>Run Time (Seconds)</td>
</tr>
<tr>
<td>Total SVD Time (Seconds)</td>
</tr>
</tbody>
</table>

If you look at mycas.sparsemat2 in the CAS engine library, you can see that the only nonzero values in the sparse matrix are related to the outlier values that were introduced for variable Y (in observations 7, 20, and 33).

Alternatively, you can use the **ANOMALYDETECTION** statement in PROC RPCA and then use the **ASTORE** procedure to detect the outliers:

```plaintext
proc rpca data=mycas.rpcaData
   scale center;
   id index;
   anomalydetection;
   savestate rstore=mycas.store;
run;

proc astore;
   setoption rpca_projection_type 2;
   score rstore=mycas.store data=mycas.rpcaData out=mycas.scored;
run;

proc print data=mycas.scored;
run;
```

If you look at mycas.scored in the CAS engine library, you can see that the same outliers (observations 7, 20, and 33) are detected. The two methods that are described in this section are not exactly the same, but with proper settings they both detect outliers.
**Syntax: RPCA Procedure**

The following statements are available in the RPCA procedure:

```plaintext
PROC RPCA <options> ;
  ID variables ;
  INPUT variables ;
  SVD <options> ;
  OUTDECOMP <options> ;
  ANOMALYDETECTION <options> ;
  CODE FILE=filename ;
  DISPLAY <table-list> </options> ;
  DISPLAYOUT table-spec-list </options> ;
  SAVESTATE RSTORE=CAS-libref.data-table ;
```

The following sections describe the PROC RPCA statement and then describe the other statements in alphabetical order.

---

**PROC RPCA Statement**

```plaintext
PROC RPCA <options> ;
```

The PROC RPCA statement invokes the procedure. Table 19.1 summarizes the options available in the PROC RPCA statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Data Table Option</strong></td>
<td></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the input data table</td>
</tr>
<tr>
<td><strong>RPCA Options</strong></td>
<td></td>
</tr>
<tr>
<td>CENTER</td>
<td>Centers the input data</td>
</tr>
<tr>
<td>CUMEIGPCTTOL=</td>
<td>Specifies the significance level of eigenvalues that determine the rank of the low-rank matrix</td>
</tr>
<tr>
<td>DECOMP=</td>
<td>Specifies the low-rank decomposition method</td>
</tr>
<tr>
<td>FIXEDMU</td>
<td>Uses a fixed value for $\mu$</td>
</tr>
<tr>
<td>LAMBDA=</td>
<td>Specifies the parameter $\lambda$</td>
</tr>
<tr>
<td>LAMBDAWEIGHT=</td>
<td>Specifies the parameter $\lambda_{\text{weight}}$</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations</td>
</tr>
<tr>
<td>METHOD=</td>
<td>Specifies the method of solving RPCA</td>
</tr>
<tr>
<td>MU=</td>
<td>Specifies the parameter $\mu$</td>
</tr>
<tr>
<td>NTHREADS=</td>
<td>Specifies the number of threads to use on each computation node</td>
</tr>
<tr>
<td>PCPREFIX=</td>
<td>Specifies a prefix for naming the principal components</td>
</tr>
</tbody>
</table>
You can specify the following options:

**CENTER**

centers the observations by the mean of each column. For more information, see the section “Scaling and Centering” on page 437.

**COLSTATISTICS=** *CAS-libref.data-table*

specifies the name of the output table to contain simple statistics for the variables of the input data table.

*CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 421.

**CUMEIGPCTTOL=** *number*

specifies the significance level of eigenvalues that determine the rank of the low-rank matrix. This value must be between 0 and 1. By default, CUMEIGPCTTOL=1. For more information, see the section “Cumulative Eigenvalue Percentage Tolerance” on page 438.

**DATA=** *CAS-libref.data-table*

names the input data table for PROC RPCA to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

*CAS-libref* refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 421.

*data-table* specifies the name of the input data table.
DECOMP=SVD | PCA | NONE
specifies the type of analysis. If MAXITER=0, this step is applied to the original input data; otherwise, it is applied to the low-rank matrix. You can specify the following values:

SVD performs singular value decomposition.
PCA performs principal component analysis.
NONE performs neither of the two preceding analyses.

By default, DECOMP=NONE.

FIXEDMU
uses a fixed value for $\mu$ in each iteration of the RPCA algorithm when METHOD=APG. Otherwise, the value of $\mu$ is dynamically updated in each iteration.

LAMBDA=number
specifies a value for the parameter $\lambda$, where number is a positive real number. The default value is computed by $\frac{1}{\sqrt{n}}$, where $n$ is the greater of the number of observations and the number of variables in the input data set. This parameter affects the sparsity of the sparse matrix. For more information about this setting, see Candès et al. (2011).

LAMBDAWEIGHT=number
specifies the value of the parameter $\lambda_{\text{weight}}$ to be applied on the $l_1$ norm of the sparse matrix. The final value for the $\lambda$ that is used in the RPCA algorithm is calculated by multiplying $\lambda_{\text{weight}}$ by $\lambda$. You can use this value to control the sparsity of the sparse matrix. A larger number produces a more sparse matrix. For more information, see the section “Details: RPCA Procedure” on page 436.

By default, LAMBDAWEIGHT=1.

MAXITER=number
specifies the maximum number of iterations before the process stops, where number is a positive integer. If MAXITER=0, PROC RPCA runs no iterations. If you also specify the DECOMP= option, the analysis is performed on the original matrix (instead of the low-rank matrix).

If the RPCA procedure does not converge in number of iterations, a solution status of “Failed to Converge” is written to the “Results Summary” table.

By default, MAXITER=1000.

METHOD=ALM | APG
specifies the method to solve RPCA. You can specify the following values:

ALM uses the augmented Lagrange multiplier method.
APG uses the accelerated proximal gradient method.

By default, METHOD=ALM.
**PROC RPCA Statement**

**MU=** *number*

specifies an initial value of $\mu$ when METHOD=APG. By default, MU=0.001.

**NTHREADS=** *number-of-threads*

specifies the number of threads per computation node. The default value is the lesser of 16 and the number of threads available per computation node.

**OUTERROR=** *CAS-libref.data-table*

specifies the name of the output table for the error matrix, which contains the noise in the input data.

*CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 421.

**OUTLOWRANK=** *CAS-libref.data-table*

specifies the name of the output table for the low-rank matrix.

*CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 421.

**OUTSPARSE=** *CAS-libref.data-table*

specifies the name of the output table for the sparse matrix.

*CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 421.

**PCPREFIX=** *name*

specifies a prefix for naming the principal components. By default, the components are named Prin1, Prin2, . . . , Prin$\text{n}$. If you specify PCPREFIX=Abc, the components are named Abc1, Abc2, Abc3, and so on. The number of characters in the prefix plus the number of digits required to designate the variables should not exceed the current name length that is defined by the VALIDVARNAME= system option. For more information about the VALIDVARNAME= system option, see SAS System Options: Reference.

**SCALE**

scales the observations by the standard deviation of each column. If a constant variable exists (where all observations have the same value), the observations of this column are not scaled. For more information, see the section “Scaling and Centering” on page 437.

**TOLERANCE=** *number*

specifies the convergence criterion for the RPCA algorithm. The default value is $10^{-7}$. 
The ANOMALYDETECTION statement specifies the parameters of the anomaly detection methods that are used in the subsequent ASTORE procedure to detect anomalies in the scoring observations.

You can specify the following `options`:

**METHOD=SIGVARS | R4S**

specifies the method of anomaly detection. You can specify the following values:

- **SIGVARS**: uses the SIGVARS method for anomaly detection.
- **R4S**: uses the R4S method for anomaly detection.

For more information about SIGVARS and R4S methods, see the section “Anomaly Detection” on page 438. You can override this parameter by specifying the `RPCA_ANOMALYDETECTION_METHOD` option and the override value in the SETOPTION statement in the ASTORE procedure. By default, `METHOD= SIGVARS`.

**NUMSIGVARS=number**

specifies the minimum number of significant variables in an observation for it to be considered as an anomaly by SIGVARS method, where `number` must be a positive integer value. This option is used when `METHOD=SIGVARS`. You can override this parameter by specifying the `RPCA_NUMSIGVARS` option and the override value in the SETOPTION statement in the ASTORE procedure. By default, `NUMSIGVARS= 1`.

**SIGMACOEF=number**

specifies the threshold on the standardized sparse value in the SIGVARS method or a coefficient that is applied on the threshold in the R4S method. For more information see the section “Anomaly Detection” on page 438. You can override this parameter by specifying the `RPCA_SIGMACOEF` option and the override value in the SETOPTION statement in the ASTORE procedure. By default, `SIGMACOEF= 1`.

**USEMATRIX=INPUT | SPARSE**

specifies which matrix to use to standardize the sparse part of the scoring observations in the anomaly detection methods:

- **INPUT**: uses the standard deviation of the columns of the original input data set.
- **SPARSE**: uses the standard deviation of the columns of the output matrix that is specified in the `OUTSPARSE=` option in the PROC RPCA statement.

You can override this parameter by specifying the `RPCA_USEMATRIX` option and the override value in the SETOPTION statement in the ASTORE procedure. By default, `USEMATRIX= INPUT`. 
**CODE Statement**

```sas
CODE FILE=filename < PROJECTIONTYPE=PCA | LRS > ;
```

The CODE statement generates SAS DATA step code that mimics the computations that are performed. The generated SAS DATA step code can be used to score new observations—that is, to project the new observations onto the principal component space or the low-rank subspace.

The CODE statement is optional. Only one CODE statement is processed. If you specify multiple CODE statements, only the first one is used. If you do not include a CODE statement, no score code is generated.

You must specify the following option:

```sas
FILE=filename
```

specifies the name of the file to write the SAS score code to.

You can also specify the following option:

```sas
PROJECTIONTYPE=PCA | LRS
```

specifies the type of projection. You can specify the following values:

- **PCA** projects the scoring observations onto the principal component space.
- **LRS** projects the scoring observations onto the low-rank subspace.

By default, PROJECTIONTYPE=PCA.

---

**DISPLAY Statement**

```sas
DISPLAY < table-list > < / options > ;
```

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the `table-list` as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “ODS Table Names” on page 440. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path `Bygroup1.Summary.SelectionSummary`. A partial pathname does not include all groups; for example, `SelectionSummary` and `Summary.SelectionSummary` are partial pathnames for `Bygroup1.Summary.SelectionSummary`. 
When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both `SelectionSummary` and `Summary:SelectionSummary` select `Bygroup1.Summary:SelectionSummary`.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the `Bygroup1.Summary:SelectionSummary` table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the `Bygroup1.Summary:SelectionSummary` table is not selected.

You can specify the following `options` after a slash (/):

- **CASESENSITIVE**
  - Performs a case-sensitive comparison of table names in the `table-list` to display table names when tables are subsetted for display. To preserve case, you must enclose table names in the `table-list` in quotation marks.

- **EXCLUDE**
  - Displays all display tables except those that you specify in the `table-list`.

- **EXCLUDEALL**
  - Suppresses display of all tables. This option takes precedence over the other options.

- **TRACE**
  - Displays the display table names, labels, and paths.

---

**DISPLAYOUT Statement**

```
DISPLAYOUT table-spec-list < / options > ;
```

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

The `table-spec-list` specifies a list of CAS output tables to create. Each entry in the list has either a `key=value` format or a `key` format:

```
key=value  specifies key as the ODS table name, path, or partial pathname, and specifies value as the CAS output table name.
key         specifies key as the ODS table name and also as the CAS output table name.
```

The ODS table names that you can specify are listed in the section “ODS Table Names” on page 440. You cannot specify the ODS table named OutputCasTables in the `table-spec-list`.

Table names and partial pathnames are discussed under the `DISPLAY` statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following `options` after a slash (/):
**ID Statement**

**ID** variables ;

The ID statement lists one or more variables to be copied from the input data table to the OUTLOWRANK=, OUTSPARSE=, OUTERROR=, PCSCORES=, and SVDLEFT= output data tables, and to the file that is specified in the RSTORE= option in the SAVESTATE statement. If any of the variables in the ID statement appears in the INPUT statement, that variable is considered to be an ID variable and is not used in the analysis (it is only copied to the output tables and RSTORE= file).

**INPUT Statement**

**INPUT** variables ;

The INPUT statement specifies the names of variables to be considered in the RPCA procedure. Only numeric variables are accepted. If you do not specify this statement, PROC RPCA considers all numeric variables in the input data table.

**OUTDECOMP Statement**

**OUTDECOMP** < options> ;

The OUTDECOMP statement specifies output filenames for the low-rank matrix decomposition tables.

If you specify DECOMP=SVD in the PROC RPCA statement, then you can specify the following three options:

**SVDDIAG**=CAS-libref.data-table

specifies the name of the output table to contain the SVD diagonal vector.

CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 421.
**SVDLEFT**=`CAS-libref.data-table`

specifies the name of the output table to contain the SVD left matrix.

*CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 421.

**SVDRIGHT**=`CAS-libref.data-table`

specifies the name of the output table to contain the SVD right matrix.

*CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 421.

If you specify DECOMP=PCA in the PROC RPCA statement, then you can specify the following two options:

**PCLOADINGS**=`CAS-libref.data-table`

specifies the name of the output table to contain the matrix of principal component loadings.

*CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 421.

**PCSCORES**=`CAS-libref.data-table`

specifies the name of the output table to contain the matrix of principal component scores.

*CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 421.

---

**SAVESTATE Statement**

```plaintext
SAVESTATE RSTORE=`CAS-libref.data-table` ;
```

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 4, “The ASTORE Procedure.”

You must specify the following option:

**RSTORE**=`CAS-libref.data-table`

specifies the name of the table in which to save the analytic store. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 421.
You can use the ASTORE procedure to specify the projection type. Specify the option RPCA_PROJECTION_TYPE in the SETOPTION statement in PROC ASTORE: the value 0 projects the scoring observations onto the principal component space, the value 1 projects the scoring observations onto the low-rank subspace, and the value 2 projects the scoring observations onto the low-rank subspace but stores the sparse part of the scoring data in the scoring results table.

**SVD Statement**

```
SVD <options> ;
```

The SVD statement specifies the parameters of the singular value decomposition (SVD) algorithm that are used in RPCA iterations. You can specify the following options:

- **MAXRANK=** number
  specifies the maximum value for the rank that the SVD solver considers. The default value is the number of input variables.

- **METHOD=EIGEN | ITERATIVE | RANDOM**
  specifies the type of the SVD solver. You can specify the following values:
    - **EIGEN**
      uses the eigenvalue decomposition method.
    - **ITERATIVE**
      uses the iterative SVD method.
    - **RANDOM**
      uses the randomized SVD method.

  When the number of variables in the input data set is very large, it is recommended that you use METHOD=RANDOM. By default, METHOD=EIGEN.

- **POWER=** number
  specifies the parameter power for the SVD solver if METHOD=RANDOM. By default POWER=0.

- **SEED=** number
  specifies the parameter seed for the SVD solver if METHOD=RANDOM. The default value is the current local time in SAS datetime format.
**Details: RPCA Procedure**

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

$$\text{minimize} \quad ||L||_* + \lambda ||S||_1$$
$$\text{subject to} \quad L + S = M$$

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 the outlier values. As a result, many applications of RPCA focus on the sparse matrix.

Two algorithms are implemented in the RPCA procedure: the augmented Lagrange multiplier algorithm and the accelerated proximal gradient algorithm.

**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 augmented Lagrange multiplier (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 $l(L, S_{k+1}, Y_k)$. Next, the Lagrange multiplier $Y_{k+1}$ is updated. For more information, see Candès et al. (2011).

---

1The sparse matrix can also include noise. The sparsity of the sparse matrix depends highly on the settings of the $\lambda$ and $\lambda_{\text{weight}}$ parameters.

2The singular value thresholding operator is based on the singular value decomposition.
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 $\mu$, and the second term is the penalty term of the constraint:

$$\text{minimize} \quad \mu \|L\|_* + \mu \lambda \|S\|_1 + \frac{1}{2} \|M - L - S\|_F^2$$

Lin et al. (2009) applied 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 APG be used when the observations are significantly corrupted by noise.

Scaling and Centering

If you specify the CENTER or SCALE option (or both), the original data $M$ are standardized as

$$M(i, j) = \frac{M(i, j) - \mu_j}{\sigma_j}$$

where $\mu_j$ and $\sigma_j$ are the mean and standard deviation of the $j$th column in $M$, respectively. Therefore, the low-rank matrix ($L$), the sparse matrix ($S$), and the error matrix ($N$) are updated as follows:

$$L(i, j) = L(i, j) \ast (\sigma_j) + \mu_j$$
$$S(i, j) = S(i, j) \ast (\sigma_j)$$
$$N(i, j) = N(i, j) \ast (\sigma_j)$$
Cumulative Eigenvalue Percentage Tolerance

Sometimes when the input data set has noise, the rank that is reported in the ODS summary table 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 table specified in the SVDDIAG= option. 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.

Anomaly Detection

The scoring observations are projected onto the low-rank space. The resulting projection is subtracted from the scoring observation to identify the sparse part. However, not all nonzero elements of the resulting sparse matrix are significant. The following two methods are implemented in the RPCA procedure to postprocess the sparse part of the scoring observation, which is referred to as the $S_{sc}$ vector:

SIGVARS Method

The values of the $S_{sc}$ vector are standardized as

$$SS_{sc,j} = \frac{S_{sc,j}}{\sigma_j}$$

where $j$ refers to variable $j$ in the input data set and $\sigma_j$ is the standard deviation of either column $j$ of the input data set or column $j$ of the OUTSPARSE= output matrix, depending on the value of the USEMATRIX= option.

The absolute value of $SS_{sc,j}$ is considered significant if it is greater than the value of the SIGMACOEF= option. If the number of significant values in the scoring observation is greater than or equal to the value specified in the NUMSIGVARS= option, the scoring observation is considered to be anomalous.

R4S Method

As in the SIGVARS method, the values of $S_{sc}$ are standardized. However, a different criterion is used to detect whether the scoring observation is anomalous.

The root of the sum of squares of the values on $SS_{sc}$, denoted as $RSSSS_{sc}$, is calculated for each scoring observation:

$$RSSSS_{sc} = \sqrt{\sum_{j \in \text{input vars}} (SS_{sc,j})^2}$$
This $RSS_{sc}$ is compared to the following value: the standard deviation of the same measures calculated on OUTSPARSE= output matrix multiplied by the value specified in the SIGMACOEF= option of the ANOMALYDETECTION statement. The scoring observation is considered anomalous if $RSS_{sc}$ is greater than or equal to this value.

### Displayed Output

The RPCA procedure displays various tables that are related to input and results. The following sections describe the output tables in the order of their appearance.

#### Model Information

The “Model Information” table displays basic information about the parameters that are used in the RPCA procedure. This information includes the data source, the RPCA method, the SVD method used in RPCA, and the value of parameters $\lambda$ and $\lambda_{\text{weight}}$. Also, if the SVD method is the randomized method (that is, METHOD=RANDOM in the SVD statement), the seed that is used in the SVD method is displayed.

#### Dimensions

The “Dimensions” table displays the number of observations and variables in the input data set, the number of missing observations, and the number of observations from the input data set that are considered for the RPCA analysis.

#### Results Summary

The “Results Summary” table displays the summary of the RPCA results, including the sparsity of the sparse matrix, the rank of the low-rank matrix, the number of iterations, the solution status, and the run time.

The solution status takes one of these values:

- **Optimal**: RPCA converges and decompositions are successful.
- **SVD Failure**: Internal SVD call failure. This can be caused by running out of memory or SVD not converging.
- **Failed to Converge**: RPCA does not converge with the specified MAXITER= option.
- **Failed**: RPCA failed because it ran out of memory or because of other problems.
Chapter 19: The RPCA Procedure

ODS Table Names

Each table that the RPCA procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. The names of each table and a short description of the contents are listed in Table 19.2.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions</td>
<td>Dimensions of the input table</td>
<td>PROC RPCA</td>
<td>Default</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>PROC RPCA</td>
<td>Default</td>
</tr>
<tr>
<td>Summary</td>
<td>Summary of the results</td>
<td>PROC RPCA</td>
<td>Default</td>
</tr>
</tbody>
</table>

Output Data Tables

The RPCA procedure creates output tables for the low-rank matrix (if the OUTLOWRANK= option is specified), the sparse matrix (if the OUTSPARSE= option is specified), the error matrix (if the OUTERROR= option is specified), and column statistics (if the COLSTATISTICS= option is specified).

Also, depending on the value of the DECOMP=option in the PROC RPCA statement, the low-rank matrix is decomposed in SVD tables (which are specified in the SVDLEFT=, SVDDIAG=, and SVDRIGHT= options) or PCA tables (which are specified in the PCLOADINGS= and PCSCORES= options). Table 19.3 lists details about these data tables.

<table>
<thead>
<tr>
<th>Data Table</th>
<th>Content</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLSTATISTICS</td>
<td>Mean and standard deviation for each variable of the input data table</td>
<td>Number of variables × 3</td>
</tr>
<tr>
<td>OUTERROR</td>
<td>Error matrix</td>
<td>Number of observations × number of variables</td>
</tr>
<tr>
<td>OUTLOWRANK</td>
<td>Low-rank matrix</td>
<td>Number of observations × number of variables</td>
</tr>
<tr>
<td>OUTSPARSE</td>
<td>Sparse matrix</td>
<td>Number of observations × number of variables</td>
</tr>
<tr>
<td>PCLOADINGS</td>
<td>Right-singular vectors (and an extra column for variable names)</td>
<td>Number of variables × (number of singular values + 1)</td>
</tr>
<tr>
<td>PCSCORES</td>
<td>Product of matrix of left-singular vectors and diagonal matrix of singular values</td>
<td>Number of observations × number of singular values</td>
</tr>
</tbody>
</table>
### Table 19.3  continued

<table>
<thead>
<tr>
<th>Data table</th>
<th>Content</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVDDIAG</td>
<td>Singular values in decreasing order</td>
<td>Number of singular values</td>
</tr>
<tr>
<td>SVDLEFT</td>
<td>Left-singular vectors</td>
<td>Number of observations × number of singular values</td>
</tr>
<tr>
<td>SVDRIGHT</td>
<td>Right-singular vectors (and an extra column for variable names)</td>
<td>Number of variables × (number of singular values + 1)</td>
</tr>
</tbody>
</table>

### Examples: RPCA Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

### Example 19.1: Analyzing the Energy Output of the SAS Solar Farm

This example applies the RPCA procedure to data from the solar farm at SAS headquarters in Cary, North Carolina. The data set consists of data from many sensors, at different aggregations, as well as weather data. All together, it contains 80 variables. The goal of the example is to predict the daytime hourly energy output (KWH_Delivered) from the solar farm by using robust principal components.

The following code includes the DATA step that generates the mycas.SolarFarm data table and the PROC RPCA statement that is applied to these data. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```sas
data mycas.solarfarm;
  input date : date9. hour KWH_Delivered A_B_Avg_Volt App_Energy_Tot_Delv App_Pwr_Total App_Pwr_Total_Max B_C_Avg_Volt C_A_Avg_Volt Current_Avg Current_Pct_Unblanc Day_Energy_created Energy_created_10mi Frequency Hr_Energy_created L_L_Avg_Volt ...
  ... more lines ...

Of the observations in the data table, 60% are used for training and 40% are used for testing. You can use the following PARTITION procedure code to randomly divide the observations in the mycas.SolarFarm data table into training and testing data tables:

```sas
  proc partition data = mycas.solarfarm partind samppct=60;
    output out=mycas.solarfarmpart;
  run;
```
data mycas.solarfarmtrain(drop = _partind_);
  set mycas.solarfarmpart;
  if _partind_ eq 1 then output;
run;

data mycas.solarfarmtest(drop = _partind_);
  set mycas.solarfarmpart;
  if _partind_ eq 0 then output;
run;

proc contents data=mycas.solarfarm out=varlist(keep=NAME) noprint;
run;
proc sql noprint;
  select NAME into:inputvars separated by ' ' 
    from varlist where NAME not in ("date","hour","KWH_Delivered");
quit;

proc rpca data=mycas.solarfarmtrain 
  outlowrank=mycas.lowrank 
  decomp=svd method = APG 
  scale center;
  id date hour;
  input &inputvars.;
  outdecomp svddiag=mycas.svddiag;
  code file='rpcascore.sas' projectiontype=PCA;
run;

In the svddiag output table (not shown), the first 13 eigenvalues explain more than 95% of the variability in the data. So you can project the training data on the first 13 principal components and build the regression model to predict the generated energy from these 13 robust components by using the following statements:

data mycas.rpcaprojectedtrain 
  (keep=Prin1 Prin2 Prin3 Prin4 Prin5 Prin6 Prin7 Prin8 
    Prin9 Prin10 Prin11 Prin12 Prin13 KWH_Delivered);
  set mycas.solarfarmtrain;
  %inc 'rpcascore.sas';
run;

proc regselect data=mycas.rpcaprojectedtrain;
  model KWH_Delivered= Prin1 Prin2 Prin3 Prin4 Prin5 Prin6 Prin7 Prin8 Prin9 
    Prin10 Prin11 Prin12 Prin13;
  code file='rpcareg.sas';
run;

Next, you can use the following statements to project the test data into the principal component space and to predict the energy output of the test data by using the regression model that is obtained in the previous step:

data mycas.rpcaprojectedtest 
  (keep= date hour Prin1 Prin2 Prin3 Prin4 Prin5 Prin6 
    Prin7 Prin8 Prin9 Prin10 Prin11 Prin12 Prin13 KWH_Delivered);
  set mycas.solarfarmtest;
  %inc 'rpcascore.sas';
Example 19.1: Analyzing the Energy Output of the SAS Solar Farm

run;

data mycas.EnergyforecastOnTestDataRPCA;
  set mycas.rpcaprojectedtest;
  %inc 'pcareg.sas';
run;

You can repeat the same steps by using standard PCA (with the PCA procedure) as well:

proc pca data=mycas.solarfarmtrain;
  var &inputvars.;
  code file='pcascore.sas';
run;

In the resulting “Eigenvalues” table (not shown), 25 principal components explain 95% of the variability in
the data. So you can project the training data on these 25 principal components and build a regression model
by using the following statements:

data mycas.pcaprojectedtrain
  (keep=Prin1 Prin2 Prin3 Prin4 Prin5 Prin6 Prin7 Prin8 Prin9 Prin10
   Prin11 Prin12 Prin13 Prin14 Prin15 Prin16 Prin17 Prin18 Prin19
   Prin20 Prin21 Prin22 Prin23 Prin24 Prin25 KWH_Delivered);
  set mycas.solarfarmtrain;
  %inc 'pcascore.sas';
run;

proc regselect data=mycas.pcaprojectedtrain;
  model KWH_Delivered= Prin1 Prin2 Prin3 Prin4 Prin5 Prin6 Prin7 Prin8 Prin9
                      Prin10 Prin11 Prin12 Prin13 Prin14 Prin15 Prin16 Prin17
  code file='pcareg.sas';
run;

Next, you can use the following statements to project the test data on these 25 principal components and get
a prediction of the energy that is generated:

data mycas.pcaprojectedtest
  (keep= date hour Prin1 Prin2 Prin3 Prin4 Prin5 Prin6 Prin7 Prin8 Prin9
               Prin10 Prin11 Prin12 Prin13 Prin14 Prin15 Prin16 Prin17
               Prin18 Prin19 Prin20 Prin21 Prin22 Prin23 Prin24 Prin25 KWH_Delivered);
  set mycas.solarfarmtest;
  %inc 'pcascore.sas';
run;

data mycas.EnergyforecastOnTestDataPCA;
  set mycas.pcaprojectedtest;
  %inc 'pcareg.sas';
run;

Now you can calculate the RMSE to compare the model that is built using RPCA to the model that is built
using regular PCA, by using the following statements:
data mycas.rpcaresiduals(keep=date hour rpcaresidual);
  set mycas.EnergyforecastOnTestDataRPCA;
  rpcaresidual= KWH_Delivered - P_KWH_Delivered;
run;

data mycas.pcaresiduals(keep=date hour pcaresidual);
  set mycas.EnergyforecastOnTestDataPCA;
  pcaresidual= KWH_Delivered - P_KWH_Delivered;
run;

data mycas.residuals;
  merge mycas.rpcaresiduals mycas.pcaresiduals;
  by date hour;
run;

proc sql;
  select sqrt(sum(rpcaresidual*rpcaresidual)/count(rpcaresidual))
        as rpcaRMSE from mycas.rpcaresiduals;
  select sqrt(sum(pcaresidual*pcaresidual)/count(pcaresidual))
        as pcaRMSE from mycas.pcaresiduals;
quit;

The results from the last step (not shown) demonstrate that the RMSE of the regression model that is built
using robust principal components is lower than the RMSE of the regression model that is built using standard
principal components. This is no surprise, because the principal components that you obtain from standard
PCA are affected by the outliers in the data.

References

Association for Computing Machinery 58:1–37.

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.

of Electrical and Electronics Engineers.
Overview: SEMISUPLEARN Procedure

The SEMISUPLEARN procedure implements the graph-based semisupervised learning algorithm that relies on label spreading (Zhou et al. 2003) in a distributed framework in SAS Viya. The semisupervised learning algorithm has numerous applications, including web page classification, image recognition, medical imaging, natural language processing, and action recognition.

You can use the SEMISUPLEARN procedure to read and write data in distributed form and to perform search in parallel by making full use of multicore computers or distributed computing environments.

The SEMISUPLEARN procedure operates on a labeled data table and an unlabeled data table. For each observation in the query data table, PROC SEMISUPLEARN returns the predicted labels for the observations in both the unlabeled data table and the labeled data table. The procedure uses the radial basis function (RBF) kernel metric and $k$-nearest neighbor (KNN) metric to compute distances between the pairs of the observations in the unlabeled and labeled data tables, and it assigns the predicted labels to the unlabeled data in an iterative way, according to the similarity measure. PROC SEMISUPLEARN stores the predicted labels for both the labeled data table and the unlabeled data table in the output data table that you specify in the PROC SEMISUPLEARN statement.
PROC SEMISUPLEARN Features

The SEMISUPLEARN procedure enables you to use parallel execution for the graph-based semisupervised learning algorithm that relies on label spreading in a distributed computing environment or on a single machine. PROC SEMISUPLEARN has the following basic features:

- is highly distributed and multithreaded
- returns the predicted labels for both the unlabeled data table and the labeled data table

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example shows how to use the SEMISUPLEARN procedure to predict the labels for both the unlabeled and labeled data, from observations in a set of unlabeled data observations and a set of labeled data observations. In this case, the data are from the hmeq data set. This data set contains information about mortgage applicants. The example selects 3,000 applicants for the unlabeled data table, which is the data table without the target variable, and 200 applicants for the labeled data table, which is the data table with the target variable. PROC SEMISUPLEARN returns the predicted target variables for the applicants in both the unlabeled data table and the labeled data table. The analysis uses 10 variables: loan, mortdue, value, yoj, derog, delinq, clage, ninq, clno, and debtinc. The remaining variables in the data table are not used.

You can load the hmeq 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.

```sas
   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);
   id =_N_;
   run;

   data mycas.label;
   set hmeq(obs=20);
   run;
```

The following statements run PROC SEMISUPLEARN and output the results to ODS tables:

```sas
   proc SEMISUPLEARN data= mycas.unlabel
      label = mycas.label
      gamma = 1000;
      input loan mortdue value yoj derog delinq clage ninq clno debtinc;
      output out = mycas.out copyvar=(id);
      target bad;
   run;
```

The INPUT statement specifies that the variables loan, mortdue, value, yoj, derog, delinq, clage, ninq, clno, and debtinc are to be used as inputs. The OUTPUT statement requests that the predicted labels for the unlabeled and labeled tables be written to the data table mycas.out. Figure 20.1 shows the number of
unlabeled observations, number of labeled observations, number of levels for target variable, gamma value, maximum number of iterations, kernel and the loss used in the computation.

**Figure 20.1** Model Information

The SEMISUPLEARN Procedure

<table>
<thead>
<tr>
<th>Model Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Labeled Observations Used</td>
<td>20</td>
</tr>
<tr>
<td>Unlabeled Observations Used</td>
<td>3000</td>
</tr>
<tr>
<td>Maximum Iterations</td>
<td>3</td>
</tr>
<tr>
<td>Target Number of Levels</td>
<td>2</td>
</tr>
<tr>
<td>Gamma</td>
<td>1000</td>
</tr>
<tr>
<td>Kernel Function</td>
<td>RBF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loss</th>
<th>2.532068</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Output CAS Tables</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS Library</td>
<td>Name</td>
<td>Number of Rows</td>
</tr>
<tr>
<td>CASUSERHDFS(xuzche) OUT</td>
<td>3020</td>
<td>3</td>
</tr>
</tbody>
</table>

The following statements sort the output of PROC SEMISUPLEARN by id and show the observations from 100 to 109:

```plaintext
data out2; set mycas.out; run;
proc sort data=out2; by id; run;
proc print data=out2(firstobs=100 obs=109);
run;
```

**Figure 20.2** shows the id variable, the predicted labels for the unlabeled data, and the indicators for the labeled or unlabeled data.

<table>
<thead>
<tr>
<th>Obs</th>
<th>id</th>
<th>I_BAD</th>
<th><em>WARN</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>80</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>81</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>102</td>
<td>82</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>103</td>
<td>83</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>104</td>
<td>84</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>105</td>
<td>85</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>106</td>
<td>86</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>107</td>
<td>87</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>108</td>
<td>88</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>109</td>
<td>89</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Syntax: SEMISUPLEARN Procedure**

The following statements are available in the SEMISUPLEARN procedure:

```plaintext
PROC SEMISUPLEARN < options > ;
   AUTOTUNE < options > ;
   INPUT variables / < LEVEL=INTERVAL > ;
   KERNEL kernel-type / < kernel-parameters > ;
   OUTPUT OUT=CAS-libref.data-table < options > ;
   TARGET variables / < LEVEL=NOMINAL > ;
```

The PROC SEMISUPLEARN statement, the TARGET statement, and at least one INPUT statement are required. You can specify multiple INPUT statements.

The following sections describe the PROC SEMISUPLEARN statement and then describe the other statements in alphabetical order.

---

**PROC SEMISUPLEARN Statement**

```plaintext
PROC SEMISUPLEARN < options > ;
```

The PROC SEMISUPLEARN statement invokes the procedure. Table 20.1 summarizes the options available in the PROC SEMISUPLEARN statement.

**Table 20.1 PROC SEMISUPLEARN Statement Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unlabeled Data Table Options</td>
<td></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the unlabeled data table</td>
</tr>
<tr>
<td>Labeled Data Table Options</td>
<td></td>
</tr>
<tr>
<td>LABEL=</td>
<td>Specifies the labeled data table</td>
</tr>
<tr>
<td>Semisupervised Learning Algorithm Options</td>
<td></td>
</tr>
<tr>
<td>GAMMA=</td>
<td>Specifies the inverse of the variable when the similarity measure is computed by using the Gaussian kernel</td>
</tr>
<tr>
<td>KERNEL=</td>
<td>Specifies the kernel that is used to compute the similarity between samples</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations</td>
</tr>
<tr>
<td>NTHREADS=</td>
<td>Specifies the number of threads to use on each computation node</td>
</tr>
<tr>
<td>SEED=</td>
<td>Specified the seed number for autotuning</td>
</tr>
</tbody>
</table>

You can specify the following options:
Chapter 20: The SEMISUPLEARN Procedure

**DATA=** `<CAS-libref.data-table>`

names the input data table for PROC SEMISUPLEARN to use. The default is the most recently created data table. `<CAS-libref.data-table>` is a two-level name, where

- `<CAS-libref>` refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `<CAS-libref>`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 446.

- `<data-table>` specifies the name of the input data table.

**GAMMA=** `<number>`

specifies the number of the inverse of the variance for Gaussian kernels in order to calculate similarity between observations, where `<number>` is a positive real number. By default, **GAMMA=20**.

**LABEL=** `<CAS-libref.data-table>`

specifies the input data table that contains the labeled data observations, which are the data with target variables. `<CAS-libref.data-table>` is a two-level name, where `<CAS-libref>` refers to the caslib and session identifier, and `<data-table>` specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 446.

**MAXITER=** `<number>`

specifies the number of iterations for label predictions, where `<number>` is a positive integer. By default, **MAXITER=3**.

**NTHREADS=** `<number>`

specifies the number of threads to use for the computation, where `<number>` is a positive integer. The default value is 0, which uses the maximum number of available threads per computer.

**SEED=** `<number>`

specifies the seed value for autotuning. By default, **SEED=0**.

---

**AUTOTUNE Statement**

```
AUTOTUNE <options> ;
```

The AUTOTUNE statement searches for the best combination of values of the **GAMMA=** parameter and the number of nearest-neighbor parameters (when you specify the KNN option in the KERNEL statement) by minimizing the loss function in the PROC SEMISUPLEARN statement.

Table 20.2 summarizes the **options** that you can specify in the AUTOTUNE statement. For more information about all options except the TUNINGPARAMETERS= option, see the option’s description in the section “AUTOTUNE Statement” on page 12 in Chapter 3, “Shared Concepts.” The TUNINGPARAMETERS= option is described following Table 20.2.
Table 20.2  AUTOTUNE Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDLOOKUP</td>
<td>Specifies that the table specified in the HISTORYTABLE= option contain the rows from the table specified in the LOOKUPTABLE= option</td>
</tr>
<tr>
<td>EVALHISTORY=</td>
<td>Specifies how to report the evaluation history of the tuner</td>
</tr>
<tr>
<td>HISTORYTABLE=</td>
<td>Specifies the CAS table that contains the evaluation history</td>
</tr>
<tr>
<td>LIVEUPDATE</td>
<td>Specifies that the table specified in the HISTORYTABLE= option be updated at every evaluation</td>
</tr>
<tr>
<td>LOCALSEARCH</td>
<td>Enables local search optimization</td>
</tr>
<tr>
<td>LOOKUPTABLE=</td>
<td>Specifies the CAS table to use for evaluation lookup</td>
</tr>
<tr>
<td>MAXBAYES=</td>
<td>Specifies the maximum number of points in the kriging model</td>
</tr>
<tr>
<td>MAXEVALS=</td>
<td>Specifies the maximum number of evaluations</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>MAXTIME=</td>
<td>Specifies the maximum time for all iterations</td>
</tr>
<tr>
<td>MAXTRAINTIME=</td>
<td>Specifies the maximum time for a model training</td>
</tr>
<tr>
<td>NCONVITER=</td>
<td>Specifies the number of convergence iterations</td>
</tr>
<tr>
<td>NOGRIDSHUFFLE</td>
<td>Requests that the grid points not be shuffled</td>
</tr>
<tr>
<td>NPARALLEL=</td>
<td>Specifies the number of parallel sessions</td>
</tr>
<tr>
<td>NSUBSESSIONWORKERS=</td>
<td>Specifies the number of workers in parallel sessions</td>
</tr>
<tr>
<td>POPSIZE=</td>
<td>Specifies the population size when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>SAMPLESIZE=</td>
<td>Specifies the sample size when SEARCHMETHOD=LHS or SEARCHMETHOD=RANDOM</td>
</tr>
<tr>
<td>SEARCHMETHOD=</td>
<td>Specifies the search method that the optimizer uses</td>
</tr>
<tr>
<td>SELECTINITPOINT</td>
<td>Specifies that the tuner select the best evaluation from the lookup table</td>
</tr>
<tr>
<td>TARGETEVENT=</td>
<td>Specifies the target event for ROC-based calculations</td>
</tr>
<tr>
<td>TUNINGPARAMETERS=</td>
<td>Specifies the custom tuning parameters</td>
</tr>
<tr>
<td>USEPARAMETERS=</td>
<td>Specifies how to handle the TUNINGPARAMETERS= option</td>
</tr>
</tbody>
</table>

**TUNINGPARAMETERS=(suboption | . . . | < suboption>)**

**TUNEPARMS=(suboption | . . . | < suboption>)**

specifies which parameters to tune and which ranges to tune over. If USEPARAMETERS=STANDARD, this option is ignored.

You can specify the following **suboptions:**

**GAMMA (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**

specifies information about the number of gamma values to use for tuning the semisupervised learning model. For more information, see the `GAMMA=` option in the PROC SEMISUPLEARN statement.

You can specify the following additional **suboptions:**
LB=number
specifies the minimum number of gamma values to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=0.01.

UB=number
specifies the maximum number of gamma values to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=100000.

VALUES=value-list
specifies a list of values to consider for the number of gamma values during tuning, where value-list is a space-separated list of double numbers greater than 0. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

INIT=number
specifies the initial number of gamma values for the tuner to use.

By default, INIT=20.

EXCLUDE
excludes the number of gamma values from the tuning process. If you specify this suboption, the LB=, UB=, VALUES=, and INIT= suboptions are ignored.

K (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies information about the number of K values (that is, the number of nearest neighbors in the KNN kernel) to use for tuning the semisupervised learning model. For more information, see the K= option in the KERNEL statement.

You can specify the following additional suboptions:

LB=number
specifies the minimum number of K values to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=2.

UB=number
specifies the maximum number of K values to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=200.

VALUES=value-list
specifies a list of values to consider for the number of K values during tuning, where value-list is a space-separated list of integer numbers greater than or equal to 1. If you specify this suboption, you cannot specify either the LB= or UB= suboption.
INIT=number
specifies the initial number of K values for the tuner to use.

By default, INIT=2.

EXCLUDE
excludes the number of K values from the tuning process. If you specify this suboption, the LB=?, UB=?, VALUES=?, and INIT=? suboptions are ignored.

INPUT Statement

INPUT variables /<LEVEL=INTERVAL> ;

The INPUT statement specifies the variables to use for calculating the similarity measure in the semisupervised learning algorithm. The variables must be of interval type. You must specify at least one input variable. Only interval variables are allowed here.

KERNEL Statement

KERNEL kernel-type /<kernel-parameters> ;

The KERNEL statement specifies the type of kernel and any associated parameters to be used during the semisupervised learning.

You can specify the following kernel-types:

KNN < /K=number>
uses a k-nearest-neighbor kernel during training. Specify the number of nearest neighbors as the kernel-parameter, in the form K=number, where number must be a positive integer (the default is 1). For example, specifying KERNEL KNN/K=2 indicates that for each observation, the algorithm finds the two nearest-neighbor samples to calculate the similarity measure, where the distance measure is relying on the Gaussian kernels. The KNN kernel provides a sparse representation of the graph and reduces the memory that is required in order to store the distance matrix for big data.

RBF
uses a radial basis function (RBF) kernel during training. No kernel-parameter is needed.

By default, the kernel-type is RBF.
Chapter 20: The SEMISUPLEARN Procedure

OUTPUT Statement

```plaintext
OUTPUT OUT=CAS-libref.data-table < options >;
```

The OUTPUT statement creates the predicted target variable data table for both the unlabeled data and the labeled data. 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. In the table, the formatted strings are provided in the predicted label.

You must specify the following option:

```
OUT=CAS-libref.data-table
```

names the output data table for PROC SEMISUPLEARN to use. You must specify this option before any other options. `CAS-libref.data-table` is a two-level name, where

- `CAS-libref` refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 446.

- `data-table` specifies the name of the output data table.

You can also specify the following option:

```
COPYVAR=variable
COPYVARS=(variables)
```

lists one or more variables from the unlabeled data table to be transferred to the output data table.

For variables that do not match the input list but also exist in the unlabeled data table and the labeled data table, the output data table transfers the values for the unlabeled data table and shows the values as missing for the labeled data table.

TARGET Statement

```plaintext
TARGET variable / < LEVEL=NOMINAL >;
```

The TARGET statement lists the target variable to be considered from the labeled data table in the PROC SEMISUPLEARN statement. You must specify one variable, which must be nominal. The default target variable type is nominal.
The distance metric in the SEMISUPLEARN procedure is defined as
\[ W_{ij} = \exp\left(-\gamma \frac{1}{2} \sum_{k=1}^{P} (x_{i,k} - x_{j,k})^2 \right) \]
where \( W \) is the Gaussian kernel distance metric, \( P \) is the total number of variables in the data, \( k \) is the feature index for the input observation, \( \gamma \) is the inverse of the variance for the Gaussian kernels, and \( i \) and \( j \) represent the index of the data in the combined input from both the unlabeled data and the labeled data.

Many semi-supervised algorithms rely on the key structure assumption that points within the same structure (such as a cluster or a manifold) are likely to have the same label. Given this assumption, the aim is to use unlabeled data to uncover this structure. Among these algorithms, the graph-based semi-supervised learning algorithms are well known for their good performance and scalability for big data. Assume that you have a point set \( D = \{x_1, \ldots, x_l, x_{l+1}, \ldots, x_n\} \) and the label set \( L = \{l_1, \ldots, l_c\} \), where \( c \) is the number of classes. The first \( l \) data points, \( \{x_1, x_2, \ldots, x_l\} \), are labeled by \( \{y(x_1), y(x_2), \ldots, y(x_l)\} \). The aim is to predict the label of the unlabeled data points by using the information from both the labeled data and the unlabeled data.

Let \( F \) denote the set of \( n \times c \) matrices whose matrix elements are nonnegative. A matrix \( F = [F_1^T, \ldots, F_n^T]^T \) indicates a classification on the data set \( D \) by labeling each point \( x_i \) as \( y_i = \arg \max_{j \leq c} F_{i,j} \). Define an \( n \times c \) matrix \( Y \) with \( Y_{ij} = 1 \) if \( x_i \) is labeled as \( y_i = j \) and \( Y_{ij} = 0 \) otherwise. Conventionally, the edge weight between points \( x_i \) and \( x_j \) is calculated by a Gaussian kernel \( W_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \) if \( i \neq j \) and \( W_{ii} = 0 \). Subsequently construct the normalized similarity matrix as \( S = D^{-1/2} W D^{-1/2} \), where \( D \) is a diagonal matrix whose diagonal element \( (i, j) \) is equal to the sum of the \( i \)th row of \( W \).

As described in Zhou et al. (2003), the optimization algorithm for updating the classification matrix \( F \) is formulated as
\[ F(t + 1) = \alpha SF(t) + (1 - \alpha)Y \]
where \( t \) is the number of iterations and \( \alpha \) is the weight to control the confidence in the initial labels.

Because in most cases the labeled data are relatively small, the SEMISUPLEARN procedure 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 SEMISUPLEARN procedure displays various tables that are related to the graph-based semisupervised learning algorithm. 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 graph-based semisupervised learning algorithm. This information includes the number of the unlabeled observations, number of labeled observations, number of target levels, gamma value, maximum number of iterations, kernel used in the computation, and number of nearest neighbors if the KNN kernel is used in the PROC SEMISUPLEARN statement.

Loss

The “Loss” table computes the value of the optimization function for the graph-based semisupervised learning algorithm with label spreading. As described by Zhou et al. (2003), for a single thread, the value of the loss is calculated as

$$Q(F) = \frac{1}{2} \left( \sum_{i,j=1}^{n} W_{ij} \left( \frac{1}{\sqrt{D_{ii}}} F_i - \frac{1}{\sqrt{D_{jj}}} F_j \right)^2 + \mu \sum_{i=1}^{n} \| F_i - Y_i \|^2 \right)$$

In this equation, the first term on the right-hand side is the smoothness constraint, which shows that a good classifying function should not vary too significantly among nearby points. The second term on the right-hand side is the fitting constraint, which indicates that a good classifying function should not vary too significantly from the initial label assignment. $W_{ij}$ is the value at the intersection of the $i$th row and the $j$th column in the distance matrix. $D$ is a diagonal matrix whose diagonal element $D_{ii}$ is identical to the sum of the $i$th row of $W$. $F_{ij}$ is the output probability distribution for the $i$th sample that belongs to the $j$th class, and $Y_i$ is the $i$th row of the initial label matrix. $\mu$ is a regularization parameter that controls the trade-off between two terms.

In massively parallel processing (MPP) mode, the value of the loss function is calculated by summing up the loss in each thread of the worker machines.

ODS Table Names

Each table that the SEMISUPLEARN procedure creates has a name associated with it, and you must use this name to refer to the table when you use ODS statements. The name of each table and a short description of its contents are listed in Table 20.3.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>PROC SEMISUPLEARN</td>
<td>Default</td>
</tr>
<tr>
<td>Loss</td>
<td>Loss</td>
<td>PROC SEMISUPLEARN</td>
<td>Default</td>
</tr>
</tbody>
</table>
Output Data Tables

The SEMISUPLEARN procedure creates a data table to which it writes the predicted target variables for both the unlabeled data and the labeled data. You specify the name of this data table in the OUTPUT statement. Details about the data table are listed in Table 20.4.

<table>
<thead>
<tr>
<th>Data Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>Lists the predicted labels for target variables for both the labeled data and the unlabeled data</td>
</tr>
</tbody>
</table>

Examples: SEMISUPLEARN Procedure

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

Example 20.1: German Credit Benchmark Data

This example applies the semisupervised learning algorithm to 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. The other input variables are Duration, Amount, InstallP, Resident, and Estior.

The following statements load the unlabeled data table with 800 observations and the labeled data table with 50 observations:

```sas
data mycas.dmagecrunlabel(drop=good_bad);
    set sampsio.dmagecr(obs=800);
    id=_N_;
run;

data mycas.dmagecrlabel;
    set sampsio.dmagecr(obs=50);
run;
```

The following statements run the SEMISUPLEARN procedure on the unlabeled data table mycas.dmagecrUnlabel and the labeled data table mycas.dmagecrLabel:
proc semisuplearn data=mycas.dmagecrunlabel label=mycas.dmagecrlabel;
    input duration amount installp resident existcr;
    kernel KNN/k=5;
    output out = mycas.out copyvars=(amount installp id);
    target good_bad;
run;

Output 20.1.1 shows the number of unlabeled observations, number of labeled observations, number of levels for the target variable, gamma value, maximum number of iterations, kernel used in the computation, number of nearest neighbors and the loss.

Output 20.1.1  Model Information

The SEMISUPLEARN Procedure

<table>
<thead>
<tr>
<th>Model Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Labeled Observations Used</td>
<td>50</td>
</tr>
<tr>
<td>Unlabeled Observations Used</td>
<td>800</td>
</tr>
<tr>
<td>Maximum Iterations</td>
<td>3</td>
</tr>
<tr>
<td>Target Number of Levels</td>
<td>2</td>
</tr>
<tr>
<td>Gamma</td>
<td>20</td>
</tr>
<tr>
<td>Nearest Neighbors</td>
<td>5</td>
</tr>
<tr>
<td>Kernel Function</td>
<td>KNN</td>
</tr>
</tbody>
</table>

Loss
4274.558945

Output CAS Tables

<table>
<thead>
<tr>
<th>CAS Library</th>
<th>Name</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASUSERHDFS(xuzche)</td>
<td>OUT</td>
<td>850</td>
<td>5</td>
</tr>
</tbody>
</table>

The following statements sort the output of PROC SEMISUPLEARN by id and show the observations from 110 to 118:

data out2; set mycas.out; run;
proc sort data=out2; by id; run;
proc print data=out2(firstobs=110 obs=118);
run;

Output 20.1.2 shows the values of the amount, installp, and id variables; the predicted labels for the nine observations; and the indicators for the labeled or unlabeled data.
### References

## Chapter 21
### The SPARSEML Procedure

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: SPARSEML Procedure</td>
<td>461</td>
</tr>
<tr>
<td>PROC SPARSEML Features</td>
<td>462</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>462</td>
</tr>
<tr>
<td>Getting Started: SPARSEML Procedure</td>
<td>463</td>
</tr>
<tr>
<td>Syntax: SPARSEML Procedure</td>
<td>465</td>
</tr>
<tr>
<td>PROC SPARSEML Statement</td>
<td>465</td>
</tr>
<tr>
<td>INPUT Statement</td>
<td>467</td>
</tr>
<tr>
<td>OUTPUT Statement</td>
<td>467</td>
</tr>
<tr>
<td>SAVESTATE Statement</td>
<td>468</td>
</tr>
<tr>
<td>Details: SPARSEML Procedure</td>
<td>468</td>
</tr>
<tr>
<td>Displayed Output</td>
<td>469</td>
</tr>
<tr>
<td>ODS Table Names</td>
<td>470</td>
</tr>
<tr>
<td>Examples: SPARSEML Procedure</td>
<td>471</td>
</tr>
<tr>
<td>Example 21.1: Sparse Data Set Example</td>
<td>471</td>
</tr>
<tr>
<td>Example 21.2: Junk Mail Classification</td>
<td>473</td>
</tr>
<tr>
<td>References</td>
<td>475</td>
</tr>
</tbody>
</table>

### Overview: SPARSEML Procedure

The SPARSEML procedure supports the use of sparse data as input data. It implements the support vector machines (SVM) algorithm in SAS Viya for binary classification. By using the coordinate descent optimization method, the SPARSEML procedure can train large-scale data sets that can be both wide and deep.

Like other predictive modeling tools in SAS Viya, the SPARSEML procedure uses input data to train a model and generates an analytic store that can be deployed through the ASTORE procedure. It can load data from multiple nodes and perform computations in parallel.
Chapter 21: The SPARSEML Procedure

PROC SPARSEML Features

The SPARSEML procedure has the following features:

- supports sparse data format
- supports classification of a binary target
- supports the coordinate descent method
- supports high dimensions of variables
- supports distributed sparse data sets

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

    proc options option=(CASHOST CASPORT);
    run;

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

    cas mysess;
    libname mycas cas sessref=mysess;

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

    cas mysess terminate;

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Getting Started: SPARSEML Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

The following DATA step creates the data table `smldata`. This data set contains one variable named `vars` and 15 observations; `vars` is a character variable. Each observation contains the data information, which includes a target and a sequence of combined column indexes and column values. The target value is either 1 or –1. The column index starts at 1.

```sas
data smldata;
  length vars $ 20.;
  input vars $ 1-20;
  datalines;
  1 1:-1 2:3
  1
  1 1:1 2:1
  1 1:2 2:2
  1 1:3 2:3
  1 1:4 2:4
  1 1:5 2:5
  -1 2:2
  -1 1:1 2:3
  -1 1:2 2:4
  -1 1:3 2:5
  1 1:0 2:-5
  1 1:5
  -1 1:0 2:5
  -1 1:2 2:8
;  
run;
```

You can load the `smldata` data set into your CAS session in the following DATA step:

```sas
data mycas.smldata;
  set smldata;
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 PROC SPARSEML to run the sparse machine learning algorithm on the `mycas.smldata` data table:

```sas
proc sparseml data= mycas.smldata;
  input vars;
run;
```

The INPUT statement defines the input variable `vars`, which contains both target and sparse input values in sparse string format.
PROC SPARSEML generates several ODS tables, some of which are shown in Figure 21.1 through Figure 21.3.

The “Data Information” table in Figure 21.1 shows that the number of observations is 15, the number of features is 2, and the number of sparse elements is 26.

**Figure 21.1** Sparse Data Information

<table>
<thead>
<tr>
<th>The SPARSEML Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Information</strong></td>
</tr>
<tr>
<td>Number of Rows</td>
</tr>
<tr>
<td>Number of Features</td>
</tr>
<tr>
<td>Number of Sparse Elements</td>
</tr>
</tbody>
</table>

The “Misclassification Matrix” table in Figure 21.2 shows that among the total of fifteen observations, nine observations are classified as 1, and six observations are classified as –1. The number of correctly predicted 1 observations is eight, and the number of correctly predicted –1 observations is six. Thus the accuracy is 93.33%, as indicated in the “Fit Statistics” table in Figure 21.3.

**Figure 21.2** Misclassification Matrix

<table>
<thead>
<tr>
<th>Misclassification Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>-1</td>
</tr>
<tr>
<td>Total</td>
</tr>
</tbody>
</table>

**Figure 21.3** Fit Statistics

<table>
<thead>
<tr>
<th>Fit Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>Error</td>
</tr>
<tr>
<td>Sensitivity</td>
</tr>
<tr>
<td>Specificity</td>
</tr>
</tbody>
</table>

A relatively good model means that misclassification is low and both sensitivity and specificity are high.
Syntax: SPARSEML Procedure

The following statements are available in the SPARSEML procedure:

```plaintext
PROC SPARSEML <options> ;
  INPUT variable ;
  OUTPUT OUT=CAS-libref.data-table <option> ;
  SAVESTATE RSTORE=CAS-libref.data-table ;
```

The PROC SPARSEML statement and the INPUT statement are required.

The following sections describe the PROC SPARSEML statement and then describe the other statements in alphabetical order.

PROC SPARSEML Statement

```plaintext
PROC SPARSEML <options> ;
```

The PROC SPARSEML statement invokes the procedure.

You can specify the following options:

- `C=number`
  - specifies the penalty value, where `number` must be a real number greater than 0.
  - By default, C=10.

- `DATA=CAS-libref.data-table`
  - names the input data table for PROC SPARSEML to use. The default is the most recently created data table. `CAS-libref.data-table` is a two-level name, where
    - `CAS-libref` refers to a collection of information that is defined in the LIBNAME statement and includes the `caslib`, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 462.
    - `data-table` specifies the name of the input data table.

- `MAXITERS=number`
  - specifies the maximum number of iterations before the process stops, where `number` is a positive integer.
  - By default, MAXITERS=100. In some cases, you can obtain a good model in fewer than 10 iterations.

- `NTHREADS=number-of-threads`
  - specifies the number of threads to use in the computation. The default value is the number of CPUs available on the machine.
PRINTLEVEL=level-value
specifies a level value from which the ODS information tables are printed. You can specify the following level-values:

  **AGGRESSIVE** generates all information tables.
  **BASIC** generates basic information tables.
  **MODERATE** generates standard information tables.
  **NONE** suppresses information tables.

By default, PRINTLEVEL=MODERATE.

REGL1=number
specifies the L1 norm penalization weight. If number is greater than 0, then the L1 norm penalization is performed.

By default, REGL1=0. If you specify a positive number, the penalty value C is ignored.

**NOTE:** The number must be greater than or equal to 0. You can specify either the REGL1= or REGL2= option, but not both.

REGL2=number
specifies the L2 norm penalization weight. If number is greater than 0, then the L2 norm penalization is performed.

By default, REGL2=0. If you specify a positive number, the penalty value C is ignored.

**NOTE:** The number must be greater than or equal to 0. You can specify either the REGL2= or REGL1= option, but not both.

SEED=value
specifies a non negative integer as the random number seed.

By default, SEED=1.

TOLERANCE=number
specifies the minimal absolute tolerance at which the iteration stops. The number must be greater than 0.

By default, TOLERANCE=1.0E–6. In some cases, you can obtain a good model by using a number greater than or equal to 0.01.
**INPUT Statement**

```input
INPUT variable ;
```

The INPUT statement specifies the name of the input variable to use in training. The input variable is a character string that contains a target and a sequence of space-separated sparse variable values. The format of the input variable is `target column_1:value_1 column_2:value_2 ... column_n:value_n`.

The `target` is either 1 or –1. The value `column_i` is a positive integer that is the column index starting at 1, where `i = 1, 2, ..., n`. The value `value_i` is a double number that is the corresponding column value.

**NOTE:** For more information, see Example 21.1.

---

**OUTPUT Statement**

```output
output out=cas-libref.data-table <option> ;
```

The OUTPUT statement creates an output data table that contains the predicted values of the input data table. You must specify the following option:

```output
out=cas-libref.data-table
```

names the output data table for PROC SPARSEML to use. You must specify this option before any other options. `cas-libref.data-table` is a two-level name, where

- `CAS-libref` refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 462.

- `data-table` specifies the name of the output data table.

You can also specify the following option:

```output
copyvar=variable
copyvars=(variables)
```

lists one or more variables from the input data table that are transferred to the output data table.
SAVESTATE Statement

```
SAVESTATE RSTORE=CAS-libref.data-table;
```

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 4, “The ASTORE Procedure.”

You must specify the following option:

```
RSTORE=CAS-libref.data-table
```

specifies a data table in which to save the analytic store for the model. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 462.

Details: SPARSEML Procedure

PROC SPARSEML uses a linear kernel to compute the support vector machine (SVM) learning classifiers for the binary pattern recognition problem. SVM is a powerful tool for finding robust classifiers; see Vapnik (1995); Burges (1998); Hsieh et al. (2008). Given a data set $X = \{x_1, x_2, \ldots, x_n\}$ and the corresponding labels $Y = \{y_1, \ldots, y_n\}$, the traditional SVM fitting model can be formulated as

$$
\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C_L \sum_{i=1}^{n} \xi_i \\
\text{subject to } y_i (w^T x_i + b) \geq 1 - \xi_i \quad \forall i \\
\xi_i \geq 0 \quad \forall i
$$

where $(w, b)$ determine the classifier and $\xi_i$, for each data point $i$, measures the error based on hinge loss that penalizes the misclassified points according to their distance from the corresponding classification margin.

The preceding model is called the L1-SVM model because the errors are penalized using the L1 norm. Similarly, the L2-SVM model can be formulated as

$$
\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C_L \sum_{i=1}^{n} \xi_i^2 \\
\text{subject to } y_i (w^T x_i + b) \geq 1 - \xi_i \quad \forall i \\
\xi_i \geq 0 \quad \forall i
$$

You can select these models by setting the parameters $C_{L_1}$ (using the REGL1= option) and $C_{L_2}$ (using the REGL2 option), respectively. If you specify the REGL1= option, then the L1-SVM model is used. If you specify the REGL2= option, then the L2-SVM model is used. If neither option is specified, then the L1-SVM model with a default penalty value (10) is assumed.
For performance purposes, the related dual problem would be solved. The following reformulates the L1-SVM model to produce the dual problem,

$$
\min_{\alpha} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i^T x_j) - \sum_{i=1}^{n} \alpha_i
$$

subject to

$$
\sum_{i=1}^{n} \alpha_i y_i = 0
$$

$$
0 \leq \alpha_i \leq C
$$

where the classifier would be recovered by

$$
w = \sum_i \alpha_i y_i x_i
$$

The algorithm that is used to solve the dual problem is referred to as the coordinate descent algorithm. It works on updating each coordinate by finding the $\delta$ that minimizes $f(\alpha + \delta e_i)$, where $e_i$ is the $i$th column of the identity matrix. This is equivalent to solving a one-dimensional problem,

$$
d := \arg \min_{0 \leq \alpha + d e_i \leq C} f(\alpha + d e_i) = \frac{1}{2} \|x_i\|^2 d^2 + (\sum_j \alpha_j (y_i y_j x_i^T x_j) - 1)d + \text{Const}
$$

which is a quadratic that can easily be minimized. Defining $G := (\sum_j \alpha_j (y_i y_j x_i^T x_j) - 1)$, which is referred to as the gradient, would show whether a certain coordinate is suitable to be updated. Especially given the box constraint, the projected gradient (PG) would help determine suitable coordinates by setting

$$
\text{PG} = \begin{cases}
G_i & \text{if } 0 < \alpha_i < C \\
\min(0, G_i) & \text{if } \alpha_i = 0 \\
\max(0, G_i) & \text{if } \alpha_i = C
\end{cases}
$$

thus updating each coordinate at a time. The randomized coordinate descent algorithm is suitable for large-scale sparse SVM models.

**NOTE:** Because the training process involves the randomized coordinated descent algorithm and data shuffling, the generated model can be slightly different between symmetric multiprocessing (SMP) mode and massively parallel processing (MPP) mode, especially when the data set is relatively small.

### Displayed Output

The following sections describe the output that PROC SPARSEML produces. The output is organized into various tables, which are described in the order of their appearance.
Number of Observations

The “Number of Observations” table contains the number of observations that are read and the number of observations that are used.

Data Information

The “Data Information” table contains the number of rows, the number of features, and the number of sparse elements.

Model Information

The “Model Information” table contains the initial training settings, such as task type, optimization technique, and so on.

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.

Run Time Summary

The “Run Time Summary” table contains the time that each task uses.

ODS Table Names

Each table that the SPARSEML procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. The name of each table and a short description of the contents are listed in Table 21.1.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>PRINTLEVEL= Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataInfo</td>
<td>Information for the training data</td>
<td>BASIC</td>
</tr>
<tr>
<td>FitStats</td>
<td>Accuracy information about the training</td>
<td>BASIC</td>
</tr>
<tr>
<td>Misclassification</td>
<td>Misclassification matrix table</td>
<td>MODERATE</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Basic model information for the training</td>
<td>BASIC</td>
</tr>
<tr>
<td>NObs</td>
<td>Observation information about the input data</td>
<td>BASIC</td>
</tr>
<tr>
<td>TaskTime</td>
<td>Time used by each task</td>
<td>AGGRESSIVE</td>
</tr>
</tbody>
</table>

Table 21.1  ODS Tables Produced by PROC SPARSEML
Examples: SPARSEML Procedure

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

Example 21.1: Sparse Data Set Example

This example uses the SPARSEML procedure to train a model by using the same data set, smldata, as in the section “Getting Started: SPARSEML Procedure” on page 463. Here, you can add a column id to the smldata data set to serve as the ID variable, as shown in the following statements:

```sas
data smldata;
  id = _N_; 
  set smldata;
run;
```

You can load the smldata data set into your CAS session by specifying your CAS engine libref in the second statement in the following DATA step:

```sas
data mycas.smldata;
  set smldata;
run;
```

These statements assume that the CAS engine libref is named mycas, but you can substitute any appropriately named CAS engine libref.

The following statements use PROC SPARSEML to run the sparse machine learning algorithm on the mycas.smldata data table and generate the analytic store mycas.mystate:

```sas
proc sparseml data= mycas.smldata
  TOLERANCE=0.01
  C = 5.5
  MAXITERS=10
; 
  input vars;
  savestate rstore=mycas.mystate;
  output out=mycas.trainingout copyvars=(id);
run;
```

The PROC SPARSEML statement has options such as TOLERANCE=, C=, and MAXITERS=. The INPUT statement defines the input variable vars, which includes both target and sparse input values. The SAVESTATE statement specifies the remote data table mycas.mystate, where the analytic store is saved. The OUTPUT statement shows that the training output table is mycas.trainingout, which includes the variable id.

PROC SPARSEML generates several ODS tables, some of which are shown in Output 21.1.1 through Output 21.1.3.

The “Data Information” table in Output 21.1.1 shows that the number of observations is 15, the number of features is 2, and the number of sparse elements is 26.
Output 21.1.1 Sparse Data Information

The SPARSEML Procedure

<table>
<thead>
<tr>
<th>Data Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Rows</td>
</tr>
<tr>
<td>Number of Features</td>
</tr>
<tr>
<td>Number of Sparse Elements</td>
</tr>
</tbody>
</table>

The “Misclassification Matrix” table in Output 21.1.2 shows that among the total of fifteen observations, nine observations are classified as 1, and six observations are classified as -1. The number of correctly predicted 1 observations is four, and the number of correctly predicted -1 observations is six. Thus the accuracy is 66.67%, as indicated in the “Fit Statistics” table in Output 21.1.3.

Output 21.1.2 Misclassification Matrix

<table>
<thead>
<tr>
<th>Misclassification Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Prediction</td>
</tr>
<tr>
<td>Observed</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>-1</td>
</tr>
<tr>
<td>Total</td>
</tr>
</tbody>
</table>

Output 21.1.3 Fit Statistics

<table>
<thead>
<tr>
<th>Fit Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>Error</td>
</tr>
<tr>
<td>Sensitivity</td>
</tr>
<tr>
<td>Specificity</td>
</tr>
</tbody>
</table>

The generated analytic store mycas.mystate can be used by PROC ASTORE to score data. For example, you can score the mycas.smldata data table as follows:

```sas
proc astore;
   score data=mycas.smldata
       out=mycas.smlscore
       rstore=mycas.mystate
       copyvars=(id)
       ;
   run;
quit;
```

The score output data are saved in the mycas.smlscore data table.
Example 21.2: Junk Mail Classification

This example shows how to use the SPARSEML procedure to create a model and use the model to predict whether an email is a junk email or a regular email in the Sashelp.JunkMail data set.

The Sashelp.JunkMail data set comes from a study that classifies whether an email is junk email (coded as 1) or not (coded as 0). The data were collected by Hewlett-Packard Labs and donated by George Forman. The data set Sashelp.JunkMail contains 4,601 observations, with 2 binary variables and 57 continuous explanatory variables. The response variable, Class, is a binary indicator of whether an email is considered spam or not. The partitioning variable, Test, is a binary indicator that is used to divide the data into training and testing sets. The 57 explanatory variables are continuous variables that represent frequencies of some common words and characters and lengths of uninterrupted sequences of capital letters in emails. For more information about the Sashelp.JunkMail data set, see the partition data example in the documentation of the LOGSELECT procedure in SAS Visual Statistics: Procedures.

The following DATA step converts the Sashelp.JunkMail data to the sparse data format so that you can apply it to PROC SPARSEML. The converted table JunkMail contains three variables: Test, Class, and sparseString.

```sas
data JunkMail;
  length sparseString $600;
  length oneVariable $60;
  set Sashelp.JunkMail;
  array vars{*} _NUMERIC_;
  if Class=0 then do;
    sparseString = put(-1, 2.);
  end; else do;
    sparseString = put(+1, 2.);
  end;
  do i=2 to 58;
    if vars{i} ne 0 then do;
      oneVariable = strip(put(i-1, 5.))||':'||strip(put(vars{i}, 10.3));
      sparseString = strip(sparseString)||' '||oneVariable;
    end;
  end;
  keep Test Class sparseString;
run;
```

The following DATA step loads the training data mycas.mailtrain into the CAS session. The training data contain about two-thirds of the data, which have a Test value of 0.

```sas
data mycas.mailtrain;
  id = _N_;  
  set JunkMail;
  where Test=0;
  drop Test;
run;
```

The following DATA step loads the testing data mycas.mailtest into the CAS session. The testing data contain about one-third of the data, which have a Test value of 1.
The following statements perform the model training. In the options of the PROC SPARSEML statement, the penalty value \( C = 0.1 \), and the value of the maximum iteration option \( \text{MAXITERS} = 400 \). The SAVESTATE statement specifies that the trained model is saved in the analytic store mycas.mystate.

```
proc sparseml data=mycas.mailtrain
    C=0.1
    MAXITERS=400;
    input sparseString;
    savestate rstore=mycas.mystate;
run;
```

The sparse data information, misclassification matrix, and fit statistics tables are shown in **Output 21.2.1** through **Output 21.2.3**.

**Output 21.2.1**  Sparse Data Information

<table>
<thead>
<tr>
<th>The SPARSEML Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Information</strong></td>
</tr>
<tr>
<td><strong>Number of Rows</strong></td>
</tr>
<tr>
<td><strong>Number of Features</strong></td>
</tr>
<tr>
<td><strong>Number of Sparse Elements</strong></td>
</tr>
</tbody>
</table>

**Output 21.2.2**  Misclassification Matrix

<table>
<thead>
<tr>
<th>Misclassification Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training</strong></td>
</tr>
<tr>
<td><strong>Prediction</strong></td>
</tr>
<tr>
<td><strong>Observed</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>--------------------------------</td>
</tr>
<tr>
<td>1035</td>
</tr>
<tr>
<td>111</td>
</tr>
<tr>
<td><strong>Total</strong></td>
</tr>
<tr>
<td>1146</td>
</tr>
</tbody>
</table>

**Output 21.2.3**  Fit Statistics

<table>
<thead>
<tr>
<th>Fit Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statistic</strong></td>
</tr>
<tr>
<td><strong>Training</strong></td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>Error</td>
</tr>
<tr>
<td>Sensitivity</td>
</tr>
<tr>
<td>Specificity</td>
</tr>
</tbody>
</table>
To score testing data, you can use PROC ASTORE. The following statements score the testing data table `mycas.mailtest` and save the results in the data table `mycas.scoreout`:

```plaintext
proc astore;
    score data=mycas.mailtest out=mycas.scoreout
        rstore=mycas.mystate copyvars=(id Class);
run;
quit;
```

To calculate the misclassification for the testing data, the following DATA step creates the variable Score to show the predicted value:

```plaintext
data scoreout;
    set mycas.scoreout;
    Score=_I_;
    if Score=-1 then Score=0;
run;
```

You can compare the predicted value `Score` with the true value `Class` by using PROC FREQ. The results of the comparison are displayed in Output 21.2.4.

```plaintext
proc freq data=scoreout;
    table Class*Score;
run;
```

**Output 21.2.4** Crosstabular Frequency Table

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Percent</th>
<th>Row Pct</th>
<th>Col Pct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class(0 - Not Junk, 1 - Junk)</td>
<td>Score</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>881</td>
<td>60</td>
<td>941</td>
</tr>
<tr>
<td>57.36</td>
<td>3.91</td>
<td>61.26</td>
<td></td>
</tr>
<tr>
<td>93.62</td>
<td>6.38</td>
<td>89.99</td>
<td></td>
</tr>
<tr>
<td>89.99</td>
<td>10.77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>98</td>
<td>497</td>
<td>595</td>
</tr>
<tr>
<td>6.38</td>
<td>32.36</td>
<td>38.74</td>
<td></td>
</tr>
<tr>
<td>16.47</td>
<td>83.53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.01</td>
<td>89.23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>979</td>
<td>557</td>
<td>1536</td>
</tr>
<tr>
<td>63.74</td>
<td>36.26</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>

The misclassification rate for the testing data is \((60 + 98)/1536 = 10.29\%\).

**References**


Chapter 22
The SVDD Procedure

Contents

Overview: SVDD Procedure ................................................................. 478
PROC SVDD Features ................................................................. 478
Using CAS Sessions and CAS Engine Librefs ................................. 479
Getting Started: SVDD Procedure ................................................. 480
Syntax: SVDD Procedure .............................................................. 484
PROC SVDD Statement .......................................................... 485
CODE Statement ................................................................. 486
ID Statement ................................................................. 486
INPUT Statement ................................................................. 487
KERNEL Statement ............................................................... 487
SAVESTATE Statement ......................................................... 488
SOLVER Statement .............................................................. 488
WEIGHT Statement .............................................................. 490
Details: SVDD Procedure ............................................................. 491
Mathematical Formulation ......................................................... 491
Radial Basis Function (RBF) Bandwidth Parameter ........................ 494
Fraction Outlier Parameter ....................................................... 497
SVDD Formulation with a Weight Variable .................................. 498
Comparison with One-Class Support Vector Machines ............... 499
SVDD Solvers ........................................................................ 500
Processing Mode and SVDD Results ........................................ 501
Scoring Multiple SVDD Models with the ASTORE Procedure .... 501
ODS Table Names ................................................................... 502
Examples: SVDD Procedure ......................................................... 503
Example 22.1: Analysis of Aircraft Engine Degradation .............. 503
Example 22.2: Automatic RBF Kernel Bandwidth Parameter Value Selection .......................... 508
Example 22.3: Stochastic Subset Solver ...................................... 519
Example 22.4: Analysis of Tennessee Eastman Process Data Using Multiple Bandwidth Values and Consensus-Based Classification .................................................. 526
Example 22.5: Analysis of Chessboard Data Using the Trace Criterion and the Fast Incremental Solver .................................................. 532
Example 22.6: Bandwidth Calculation Using the Peak Criterion Method .................................................. 536
References ............................................................................... 542
Overview: SVDD Procedure

The SVDD procedure 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 some examples of application areas where the majority of the 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, an SVDD model is obtained by building a minimum-radius hypersphere around the one-class training data. The hypersphere provides a compact spherical description of the training data. This training data description can be used 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 with the hypersphere radius. If the distance is more 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 description is nonspherical and conforms to the geometry of the data. PROC SVDD implements only the flexible data description.

SVDD is useful for obtaining a geometric description of data and in most applications also for detecting outliers. SVDD is used in domains where most of the data are in one class. Applications of SVDD include the following:

- **multivariate process control** (Sun and Tsung 2003; Sukchotrat, Kim, and Tsung 2009; Kakde, Peredriy, and Chaudhuri 2017)
- **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; Liao et al. 2018)
- **analysis of health data from wearable devices** (Shin, Lee, and Park 2011; Yang et al. 2010)

PROC SVDD Features

The following list provides important features of the SVDD procedure:

- multithreaded SVDD training
- support for both interval and nominal variables
- a WEIGHT statement that enables you to assign weights to observations
- automatic value selection of the radial basis function (RBF) kernel bandwidth parameter
choice of solvers:

− stochastic subset solver that enables you to quickly obtain an approximate solution
− active-set solver that enables you to obtain a more accurate solution

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```sas
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```sas
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```sas
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Getting Started: SVDD Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

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 table contains 360 observations and 31 variables. The variables represent various process measurements that are taken during the refrigerant analysis. This example uses variables $y_5$ (inlet water temperature 1) and $y_4$ (pressure drop 6) to perform an SVDD training. Variables $y_4$ and $y_5$ 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 refrigerant (which is stored in the Work library by default):

```sas
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
  ...

  The following code standardizes variables $y_4$ and $y_5$:

  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:

```sas
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 22.1 shows the results. The scatter plot indicates four distinct clusters.
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:

```plaintext
data mycas.refrigerant;
  set refrigerant;
run;
```

These statements assume that your CAS engine libref is named mycas, as in the section “Using CAS Sessions and CAS Engine Librefs” on page 479, but you can substitute any appropriately defined CAS engine libref.

The following statements execute the SVDD algorithm on the mycas.refrigerant data table and produce Figure 22.2 through Figure 22.5.

```plaintext
proc svdd data=mycas.Refrigerant;
id y4 y5;
in y4 y5/level=interval;
kernel rbf / bw=0.021;
savestate rstore=mycas.state;
run;
```

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 Figure 22.2 summarizes the key options and the input data variables.
**Figure 22.2** Refrigerant Analysis Model Information

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
</tr>
<tr>
<td>Kernel Type</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth</td>
</tr>
<tr>
<td>Bandwidth Selection Method</td>
</tr>
<tr>
<td>Bandwidth Relative Scale</td>
</tr>
<tr>
<td>Expected Outlier Fraction</td>
</tr>
<tr>
<td>Optimization Tolerance</td>
</tr>
<tr>
<td>Number of Interval Variables</td>
</tr>
<tr>
<td>Number of Nominal Variables</td>
</tr>
</tbody>
</table>

The “Training Results” table in **Figure 22.3** shows the number of support vectors and the $R^2$ threshold value.

**Figure 22.3** Refrigerant Analysis Training Results

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary</td>
</tr>
<tr>
<td>Number of Dropped Observations</td>
</tr>
<tr>
<td>Threshold R Square Value</td>
</tr>
<tr>
<td>Constant ($C_r$) Value</td>
</tr>
<tr>
<td>Run Time (seconds)</td>
</tr>
</tbody>
</table>

**Figure 22.4** displays the number of observations in the training data set.

**Figure 22.4** Refrigerant Analysis Training Observations

<table>
<thead>
<tr>
<th>Number of Observations Read</th>
<th>360</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Used</td>
<td>360</td>
</tr>
</tbody>
</table>

The “Optimization Summary” table in **Figure 22.5** shows whether the solution is optimal, the number of iterations that were required, and the objective function value.

**Figure 22.5** Refrigerant Analysis Optimization Summary

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Objective Value</td>
</tr>
<tr>
<td>Infeasibility</td>
</tr>
<tr>
<td>Optimization Status</td>
</tr>
<tr>
<td>Degenerate</td>
</tr>
</tbody>
</table>

To evaluate the quality of the training results, scoring is performed on a 200 x 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 code scores the mycas.scoreds data by using the score analytic store that was created during training:

   proc astore;
      score data=mycas.scoreds
         out=mycas.score_results
         rstore=mycas.state;
   quit;

The following code copies the score_results into the Work library:

   data work.score_results;
      set mycas.score_results;
   run;

The following code plots 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;

Figure 22.6 shows the scatter plot of the scoring data set. The gray area indicates the observations that are identified as outliers, and the black area indicates observations that are identified as inliers. Comparison between Figure 22.1 and the training data in Figure 22.6 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. This example illustrates that SVDD procedure can be used to describe data with multiple disjoint clusters, without the need to specify the correct number of clusters in advance. The scoring results in Figure 22.6 indicate that the description obtained using SVDD takes essential geometric features of the training data into account. The scoring correctly identified any observations with pressure drop 6 and inlet water temperature 1 values different from the training data values correctly identified as outliers.
Syntax: SVDD Procedure

The following statements are available in the SVDD procedure:

```bash
PROC SVDD <options>;
  CODE FILE=filename;
  ID variables;
  INPUT variables / <LEVEL=INTERVAL | NOMINAL>;
  WEIGHT variable;
  KERNEL kernel-type / <kernel-parameter>;
  SOLVER solver-type / <options>;
  SAVESTATE RSTORE=CAS-libref.data-table;
```

The PROC SVDD statement, the KERNEL statement, and at least one INPUT statement are required.

The following sections describe the PROC SVDD statement and then describe the other statements in alphabetical order.
The PROC SVDD statement invokes the procedure. You can specify the following options:

**DATA=**CAS-libref.data-table

names the input data table for PROC SVDD to use. The default is the most recently created data table. 
`CAS-libref.data-table` is a two-level name, where

- `CAS-libref` refers to a collection of information that is defined in the LIBNAME statement and includes the `caslib`, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 479.

- `data-table` specifies the name of the input data table.

**NBINS=**number

specifies the number of bins to display, where `number` must be an integer greater than 2. By default, NBINS=20.

**NOPRINT**

suppresses the generation of ODS outputs. If you specify this option, no ODS tables are generated.

**NTHREADS=**number-of-threads

specifies the number of threads to be used in the computation. The default value is the number of CPUs available in the machine.

**OUTLIER_FRACTION=**number

specifies the expected fraction of the training data that consists of outliers, where `number` must be a real number between 0 and 1.

By default, OUTLIER_FRACTION=1E–6.

**SCORE**

scores the observations in the data table that is specified in the DATA= option and generates a frequency table of _SVDDDISTANCE_ values.

**STD**

**STANDARDIZE**

standardizes the variables that are specified in the INPUT statement with LEVEL=INTERVAL by subtracting their mean and dividing by their standard deviation.

**UCC**

**USECONSTANTCOL**

uses variables that have constant values for SVDD scoring. (By default, the variables that have constant values are ignored.) This option is useful in scenarios where certain variables have constant value in the training data, but variation in the values of those variables is expected in the scoring data. This option ensures that variables with constant values are used for scoring.
OUTSV=\texttt{CAS-libref.data-table}
\SV=\texttt{CAS-libref.data-table}

specifies the output data to contain the training data observations that are identified as support vectors, the corresponding value of the Lagrange multiplier (column \_SVDDALPHA\_), and an indicator (column \_SVDDOUTLIER\_) that flags the position of the support vectors relative to the data boundary. Support vectors whose \_SVDDOUTLIER\_ value is 0 fall on the data boundary, and support vectors whose \_SVDDOUTLIER\_ value is 1 fall outside the data boundary. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the caslib and session identifier, and \texttt{data-table} specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 479.

---

**CODE Statement**

```
CODE FILE=filename ;
```

The CODE statement generates SAS DATA step code that mimics the computations that are done by the OUTPUT statement.

You must specify the following option:

\textbf{FILE=filename}

specifies the filename of the file to write the SAS score code to.

The CODE statement is optional.

\textbf{NOTE:} The CODE statement is not supported when you specify multiple bandwidth values in the KERNEL statement. For more information about the KERNEL statement, see the section “KERNEL Statement” on page 487.

---

**ID Statement**

```
ID variables ;
```

The ID statement lists one or more variables to be copied from the input data table to the output data tables that are specified in the OUTSV= option in the PROC SVDD statement and the RSTORE= option in the SAVESTATE statement.
INPUT Statement

```
INPUT variables / < LEVEL=INTERVAL | NOMINAL > ;
```

The INPUT statement specifies the names of variables to be used in training. Only interval, binary, and nominal variables are accepted. If you want to use different options for different variables, you can specify multiple INPUT statements.

You can specify the following option after a slash (/):

```
LEVEL=INTERVAL | NOMINAL
```

specifies whether the specified input variables are continuous or categorical. You can specify the following values:

- **INTERVAL** specifies that the input variables are continuous.
- **NOMINAL** specifies that the input variables are categorical.

By default, LEVEL=INTERVAL for numeric variables and LEVEL=NOMINAL for categorical variables. Binary variables are considered to be categorical variables.

KERNEL Statement

```
KERNEL kernel-type / kernel-parameter ;
```

The KERNEL statement specifies the type of kernel and any associated parameters to be used during training. Only the radial basis kernel function is supported.

You must specify the following kernel-type and kernel-parameter:

```
RBF / BW=s
```

```
RBF / BW=(number list) | (n TO m BY p)
```

```
RBF / BW=MEAN (DELTAM=number)
```

```
RBF / BW=MEAN2
```

```
RBF / BW=TRACE (NREP=number)
```

uses a radial basis function (also known as the Gaussian kernel function) during training. The kernel is defined as

```
K(x, y) = \exp \left( -\frac{\|x - y\|^2}{2s^2} \right)
```

where \(x\) and \(y\) are vectors and \(s\) is the bandwidth parameter. The bandwidth parameter is determined in one of the following ways:

- You can specify the bandwidth parameter in the form BW=s, where \(s\) must be a positive nonzero real number. The processing time usually increases as the value of \(s\) decreases. At very small values of \(s\), the processing time can be very long.

- You can specify multiple bandwidth parameter values in the form BW=(number list) or BW=(n TO m BY p). The maximum number of bandwidth parameters that you can specify or iterate over is 64. You can use multiple bandwidth parameters to train multiple SVDD models in a single run.
You can instruct the SVDD procedure to select an appropriate value by specifying BW=MEAN(DELTA=number) to request that the bandwidth parameter be computed using the mean criterion. The DELTA=number suboption specifies the tolerance value for the mean criterion. By default, DELTA=1E–6. For more information about the mean criterion, see the section “Mean Criterion” on page 496.

You can instruct the SVDD procedure to select an appropriate value by specifying BW=MEAN2 to request that the bandwidth parameter be computed using the modified mean criterion. For more information about the modified mean criterion, see the section “Modified Mean Criterion” on page 496.

You can instruct the SVDD procedure to select an appropriate value by specifying BW=TRACE(NREP=number) to request that the bandwidth parameter be computed using the trace criterion. The NREP=number suboption specifies the number of representative points. By default, NREP=5. For more information about the trace criterion, see the section “Trace Criterion” on page 497.

**SAVESTATE Statement**

```
SAVESTATE RSTORE=CAS-libref.data-table ;
```

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 4, “The ASTORE Procedure.”

You must specify the following option:

**RSTORE=CAS-libref.data-table**

specifies the filename of the file in which to save the analytic store. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 479.

**SOLVER Statement**

```
SOLVER solver-type </options> ;
```

The SOLVER statement specifies the type of optimization solver for SVDD training. You can specify following solver-types; the default type is ACTSET.

**ACTSET**

uses the active-set solver for SVDD training. For more information about this solver, see the section “Active-Set Solver” on page 500.
STOCHS
uses a stochastic subset solver for SVDD training. For more information about this solver, see the section “Stochastic Subset Solver” on page 500.

FASTINC
uses a fast incremental solver for SVDD training. For more information about this solver, see the section “Fast Incremental Solver” on page 501.

You can specify the following option after a slash (/) for either solver-type:

MAXTIME=\textcolor{red}{t}
specifies an upper limit of \textcolor{red}{t} seconds for solving the SVDD optimization problem. If you omit this option, the solver does not stop based on the amount of time elapsed. The value of \textcolor{red}{t} can be any positive number; the default value is 1,800 seconds, or 30 minutes. This option puts a time budget only on the solvers, while the actual running time may be longer because of various kinds of overheads.

You can specify the following options after a slash (/) only when the solver-type is ACTSET or STOCHS:

MAXITER=\textcolor{red}{\textit{number}}
specifies the maximum number of iterations of the stochastic solver, where \textcolor{red}{\textit{number}} must be an integer greater than 0. To obtain a solution more quickly, specify a lower value for \textcolor{red}{\textit{number}}. By default, MAXITER=25 for the active-set solver and MAXITER=200 for the stochastic subset solver.

STOL=\textcolor{red}{\textit{number}}
specifies the tolerance value for the solver, where \textcolor{red}{\textit{number}} must be a real number greater than 0. To obtain a solution more quickly, specify a higher value for \textcolor{red}{\textit{number}}. By default, STOL=1E–4.

You can specify the following options after a slash (/) only when the solver-type is FASTINC or STOCHS:

MAXSV=\textcolor{red}{\textit{number}}
specifies the maximum number of support vectors, where \textcolor{red}{\textit{number}} must be an integer greater than 0. To obtain a solution more quickly, specify a lower value for \textcolor{red}{\textit{number}}. By default, MAXSV=3500.

You can specify the following options after a slash (/) only when the solver-type is FASTINC:

OTRESH=\textcolor{red}{\textit{number}}
specifies the threshold for identifying outliers. The fast incremental solver considers an observation \textcolor{red}{z} to be an outlier if

\[
\max_{i} (K(x_i, z)) < \textcolor{red}{\textit{number}}
\]

where \(x_i, i = 1 \ldots n\) is the set of support vectors and \(K(x_i, z) = \exp \frac{-\|x_i - z\|^2}{2s^2}\).

By default, OTHRESH=1E–20. The OTHRESH option is used only for the FASTINC solver and it is ignored for the ACTSET and the STOCHS solvers.

You can specify the following options after a slash (/) only when the solver-type is STOCHS:
CTOL=number
specifies the tolerance value that is used for detecting convergence of the center, where number must be a real number greater than 0. To obtain a solution more quickly, specify a higher value for number. By default, CTOL=1E–2.

NMATCH=number
specifies a convergence criterion for the stochastic solver, where number must be an integer greater than 0. If the radius and center values converge for number of consecutive iterations, then convergence is declared. By default, NMATCH=5. To obtain a solution more quickly, specify a lower value for number.

NSAMP=number
specifies the number of observations that are sampled in each iteration of the stochastic solver, where number must be an integer greater than 0. To obtain a solution more quickly, specify a lower value for number. By default, NSAMP=10.

RTOL=number
specifies the tolerance value that is used to detect convergence of threshold value, where number must be a real number greater than 0. To obtain a solution more quickly, specify a higher value for number. By default, RTOL=1E–2.

SEED=number
specifies a seed value that is used for selecting a random sample, where number must be an integer. The default value of SEED is based on the system clock. If you want reproducible results, specify an integer for number.

WEIGHT Statement

WEIGHT variable;

The WEIGHT statement names a variable in the training data that specifies the weight to be assigned to each observation. The values of variable must be nonnegative. Observations that have higher weight have a better chance of being designated as support vectors.
**Details: SVDD Procedure**

PROC SVDD uses the radial basis kernel function to describe one-class training data and perform outlier detection. For more information about the theory and use of SVDD, see Tax and Duin (2004).

**Mathematical Formulation**

This section describes the mathematical formulation of SVDD.

**Normal Data Description**

The most elemental form of SVDD is a normal data description. The SVDD model for normal data description builds a minimum-radius hypersphere around the data. PROC SVDD does not support this description because it is very simplistic and often leads to higher rates of misclassification. PROC SVDD supports the flexible data description (as described in the section “Flexible Data Description” on page 492). SVDD formulation can be expressed in either of the following forms:

**Primal Form**

The objective function in primal form is

\[
\min R^2 + C \sum_{i=1}^{n} \xi_i
\]

subject to

\[
\|x_i - a\|^2 \leq R^2 + \xi_i \quad \text{for all } i = 1, \ldots, n
\]

\[
\xi_i \geq 0 \quad \text{for all } i = 1, \ldots, n
\]

where

- \(x_i \in \mathbb{R}^m\) for \(i = 1, \ldots, n\) represents the training data
- \(R\) is the radius and represents the decision variable
- \(\xi_i\) is the slack for each variable
- \(a\) is the center
- \(C = \frac{1}{nf}\) is the penalty constant that controls the trade-off between the volume and the errors
- \(f\) is the expected outlier fraction

**Dual Form**

The dual formulation is obtained from primal formulation by using the method of Lagrange multipliers. The objective function is

\[
\max \sum_{i=1}^{n} \alpha_i (x_i \cdot x_i) - \sum_{i,j} \alpha_i \alpha_j (x_i \cdot x_j)
\]
subject to
\[ \sum_{i=1}^{n} \alpha_i = 1 \]
\[ 0 \leq \alpha_i \leq C \quad \text{for all } i = 1, \ldots, n \]

where \( \alpha_i \in \mathbb{R} \) are the Lagrange multipliers and \( C = \frac{1}{n_f} \) is the penalty constant.

**Duality Information**  The solution to the dual formulation of SVDD provides values of Lagrange multipliers \( \alpha_i \) for all training data observations \( i = 1, \ldots, n \). Based on the value of \( \alpha_i \), the position of an observation in the context of the data description can be determined as follows:

- **Inside position:**
  \[ \|x_i - a\| < R \implies \alpha_i = 0 \]

- **Boundary position:**
  \[ \|x_i - a\| = R \implies 0 < \alpha_i < C \]

- **Outside position:**
  \[ \|x_i - a\| > R \implies \alpha_i = C \]

The center of the data description \( a \) can be obtained using the values of \( \alpha_i \) for all \( n \) observations in the training data as follows:

- **Center position:**
  \[ \sum_{i=1}^{n} \alpha_i x_i = a \]

The circular data boundary can include a significant amount of space in which training observations are very sparsely distributed. Scoring with this model can increase the probability of false positives. Hence, instead of a circular shape, a compact bounded outline around the data is often desired. Such an outline should approximate the shape of the one-class training data. Such a data boundary is possible with the use of kernel functions.

**Flexible Data Description**

The support vector data description is made flexible by replacing the inner product \( (x_i \cdot x_j) \) present in the objective function of (“Dual Form” on page 491) with a suitable kernel function \( K(x_i, x_j) \). The radial basis kernel function is defined as

\[ K(x, y) = \exp \left( -\frac{\|x - y\|^2}{2s^2} \right) \]

where \( x \) and \( y \) are any two observations from the training data and \( s \) is the bandwidth parameter. Using a kernel function, the objective function is
\[
\max \sum_{i=1}^{n} \alpha_i K(x_i, x_i) - \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j)
\]

subject to

\[
\sum_{i=1}^{n} \alpha_i = 1 \\
0 \leq \alpha_i \leq C \text{ for all } i = 1, \ldots, n
\]

The results for the center, inside, boundary, and outside positions, which are described in the section “Duality Information” on page 492, hold when the kernel function is used in the mathematical formulation.

The threshold \(R^2\) is calculated as

\[
R^2 = K(x_k, x_k) - 2 \sum_i \alpha_i K(x_i, x_k) + \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j)
\]

using any \(x_k \in SV_{<C}\), where \(SV_{<C}\) is the set of support vectors for which \(\alpha_k < C\).

**Scoring**

For each observation \(z\) in the scoring data set, the distance \(\text{dist}^2(z)\) is calculated as follows:

\[
\text{dist}^2(z) = K(z, z) - 2 \sum_i \alpha_i K(x_i, z) + \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j)
\]

Observations in the scoring data set for which \(\text{dist}^2(z) > R^2\) are designated as outliers.

**Visual Explanation of SVDD**

Figure 22.7 illustrates key SVDD statistics. The unfilled markers indicate the bivariate training data set. The filled green marker indicates the center \(a\). The filled blue markers are support vectors, which define the data boundary with a threshold \(R^2\) value of 0.66. The plot shows contour lines for distance values from 0.76 to 1.06. The red markers indicate the outlier observations from the scoring data; their distance values are greater than the threshold \(R^2\) value. The plot illustrates that as points move away from the training data, their distance value increases. It can be noted that the contours corresponding to different distance values follow the natural geometry of the training data.
Radial Basis Function (RBF) Bandwidth Parameter

The flexible data description is preferred when the data boundary needs to closely follow the shape of the data. The tightness of the boundary is a function of the number of support vectors. In the case of an RBF kernel, it is observed that if the value of the outlier fraction \( f \) is kept constant, the number of support vectors identified by the SVDD algorithm is a function of the RBF bandwidth parameter \( s \). At a very low value of \( s \), the number of support vectors is very high and approaches the number of observations. As the value of \( s \) increases, the number of support vectors decreases. It is also observed that at lower values of \( s \), the data boundary is extremely wiggly. As \( s \) is increased, the data boundary becomes less wiggly, and it starts to follow the shape of the data. At higher values of \( s \), the data boundary starts becoming spherical. The processing time increases as the value of \( s \) decreases. At very small values of \( s \), the processing time can be very long.

Figure 22.8 illustrates the effect of \( s \) on the data boundary.
Figure 22.8  Effect of RBF Bandwidth Parameter on Data Boundary

Subfigure (a) shows the training data. The training data were trained using different values of \( s \). To confirm the shape of the data boundary, a 200 \( \times \) 200 data grid was scored using training results. The scoring results are shown in subfigures (b) through (f). For subfigures (b) through (f), light gray indicates inside points and dark gray indicates outside points. All five results are obtained using \( f = 0.0001 \). At the lower value of \( s = 0.2 \), the data boundary is extremely wiggly. As the value of \( s \) is increased, the data boundary starts becoming less wiggly and conforms to the geometry of the training data. The higher value of \( s = 4.2 \) provides a very spherical data boundary. The data boundary that is obtained using \( s = 0.9 \) provides the best description of the data.

The selection of an appropriate value of \( s \) is tricky, and you must often experiment with several values until you obtain a good data boundary. If labeled data that belong to multiple classes are available, you can use cross validation to select an appropriate value of \( s \). If training data observations belong to a single class, you cannot use cross validation. In such a scenario, you can use the distance to the farthest neighbor (DFN) method (Xiao et al. 2014), the peak criterion method (Kakde et al. 2017; Peredriy, Kakde, and Chaudhuri 2017), the mean or median criterion method (Chaudhuri et al. 2017), the modified mean criterion method (Liao et al. 2018) or the trace criterion method (Chaudhuri et al. 2018b) to select an appropriate value of \( s \). The SVDD procedure supports the mean criterion and modified mean criterion methods of selecting an appropriate value of \( s \). You can use the mean criterion method by specifying \( BW=MEAN \), the modified mean
criterion method by specifying BW=MEAN2 and the trace criterion method by specifying BW=TRACE as described in the **KERNEL** statement section.

**Mean Criterion**

The SVDD results depend on the training data through the kernel matrix. For training data that consist of \( n \) observations, the kernel matrix is an \( n \times n \) matrix whose \( ij \) element is

\[
\exp\left(-\frac{\|x_i - x_j\|^2}{2s^2}\right)
\]

where \( x_i \) and \( x_j \) are any two observations in the training data and \( s \) is the bandwidth parameter. When the kernel matrix is close to the identity matrix, SVDD overfits the data. The mean criterion chooses an \( s \) value such that the kernel matrix is sufficiently different from the identity matrix. The mean criterion provides a closed-form expression for computing the bandwidth value as

\[
\delta = \sqrt{\frac{\tilde{D}^2}{\ln\frac{n-1}{\delta^2}}}
\]

where

\[
\tilde{D}^2 = \frac{\sum_{i<j} \|x_i - x_j\|^2}{\binom{n}{2}}
\]

For more information about the mean criterion, see Chaudhuri et al. (2017). The unsupervised bandwidth tuning is an extremely difficult problem, so it is quite possible that there is a class of data sets for which the mean criterion does not produce good results.

**Modified Mean Criterion**

The modified mean criterion method is an empirical method of bandwidth selection based on the peak method (Kakde et al. 2017) and the mean criterion method (Chaudhuri et al. 2017). This criterion provides the following closed-form expression for determining the value of parameter \( \delta \) that is used in the mean criteria,

\[
\delta = -0.14818008\phi^4 + 0.284623624\phi^3 - 0.252853808\phi^2 + 0.159059498\phi - 0.001381145
\]

where \( \phi = \frac{1}{\ln(n-1)} \) and \( n \) is the number of observations in the training data set.

The modified mean criterion method uses this value of delta and computes the bandwidth value as

\[
\delta = \sqrt{\frac{\tilde{D}^2}{\ln\frac{n-1}{\delta^2}}}
\]

where

\[
\tilde{D}^2 = \frac{\sum_{i<j} \|x_i - x_j\|^2}{\binom{n}{2}}
\]

For more information about the modified mean criterion method, see Liao et al. (2018).
**Trace Criterion**

The trace criterion is an empirical method of bandwidth selection that is based on the low-rank representation of the kernel matrix. The rank of the low-rank kernel matrix can be specified by using the NREP= option, as described in the KERNEL statement. The default value of NREP is 5. If the number of clusters in the training data are known, you can get a better bandwidth value by specifying the number of clusters as the value of the NREP= option. This method has been seen to work well with high-dimensional data. For more information about the trace criterion method, see (Chaudhuri et al. 2018b).

**Bandwidth Relative Scale**

The bandwidth relative scale value is the ratio of the user-specified RBF bandwidth parameter value to the bandwidth parameter value, computed using the mean criteria, with DELTA=1E−6. The bandwidth relative scale value is reported in the ModelInfo ODS table. If your analysis is taking a long time to complete or your model is overfitting the data, you can use the bandwidth relative scale value to determine whether resubmitting the analysis with a different bandwidth value is warranted. A very small value of the bandwidth relative scale (less than one) indicates that the user-specified bandwidth value is too low for the training data. In such a scenario, you can improve the performance or quality of your training results by resubmitting the analysis with a higher bandwidth value.

There are multiple factors (such as, but not limited to, solver type, training data, and solver tolerances) that can affect the processing time and quality of your results. Hence, there might be situations in which adjusting the bandwidth parameter value alone might not improve the performance or quality of your results.

---

**Fraction Outlier Parameter**

As outlined in the section “Mathematical Formulation” on page 491, the penalty constant $C$ controls the trade-off between the volume and accuracy. The constant $C$ is defined as

$$C = \frac{1}{nf}$$

where $n$ is the number of observations in the training data set and $f$ is the fraction of outliers.

Keeping the value of the RBF bandwidth parameter $s$ constant, higher values of $f$ produce a tighter boundary around the training data and thereby lower the enclosed volume. As the value of $f$ is decreased, the enclosed volume tends to increase. Figure 22.9 illustrates the effect of $f$ on the data boundary.
Figure 22.9  Effect of $f$ on Data Boundary

(a) Scatter Plot of Training Data  (b) $f = 0.2$

(c) $f = 0.5$  (d) $f = 0.8$

For subfigures (b) through (d), light gray indicates outside points, dark gray indicates inside points, and black indicates support vectors. All three results are obtained using $s = 2$. As the value of $f$ is increased from 0.2 to 0.8, the data boundary becomes tighter and the enclosed area is reduced. A data description that is obtained using an incorrect value of $f$ can be misleading. If the specified value of $f$ is greater than the true value, it can lead to a higher misclassification rate because observations are incorrectly classified as outliers. The corresponding threshold $R^2$ value is also smaller. On the other hand, a specified value of $f$ that is less than its true value can also lead to a higher misclassification rate because outliers are classified as inliers.

It is recommended that you use domain expertise and select SVDD training data such that the data truly reflect the single class of interest. The corresponding $f$ can then be set to a very small value.

SVDD Formulation with a Weight Variable

You can use the WEIGHT statement to specify observation weights. The weight can be observation frequency. The following expression shows the dual formulation of SVDD with a kernel function and a weight variable:

$$\max \sum_{i=1}^{n} \alpha_i K(x_i, x_i) - \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j)$$

subject to

$$\sum_{i=1}^{n} \alpha_i = 1$$

$$0 \leq \alpha_i \leq \frac{w_i}{(\sum_{i=1}^{n} w_i) f}, \text{ for all } i = 1, \ldots, n$$

where
Comparison with One-Class Support Vector Machines

The one-class support vector machines (OCSVM) is a one-class classification technique similar to the SVDD. Instead of obtaining a bounding hypersphere around the training data, the OCSVM algorithm finds the maximal margin hyperplane that best separates the training data from the origin (Schölkopf et al. 1999). Similar to SVDD, the OCSVM algorithm uses kernel functions to map training data into a higher-dimensional feature space. The dual formulation of OCSVM can be expressed as

\[
\min \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j)
\]

subject to

\[
\sum_{i=1}^{n} \alpha_i = 1
\]

\[
0 \leq \alpha_i \leq \frac{1}{\nu n} \text{ for all } i = 1, \ldots, n
\]

where \( \alpha_i \in \mathbb{R} \) are the Lagrange multipliers, \( \nu \) is a parameter that controls the trade-off between maximizing the distance of the hyperplane from the origin and the number of data points contained by the hyperplane, \( n \) is the number of points in the training data, and \( K(x_i, x_j) \) is the kernel function.

For the radial basis kernel function (RBF)

\[
K(x, y) = \exp \left( -\frac{\|x - y\|^2}{2s^2} \right)
\]

where \( x \) and \( y \) are any two observations from the training data and \( s \) is the bandwidth parameter.

With the RBF kernel function, the first term of the SVDD objective function of the flexible data description, as outlined in “Flexible Data Description” on page 492 evaluates to \( 1 \). Hence the OCSVM formulation is equivalent to the SVDD formulation when the RBF kernel function is used.
SVDD Solvers

You can perform SVDD training using either the active-set solver or the stochastic subset solver as described in the SOLVER Statement.

Active-Set Solver

The active set is defined as the set of indices that satisfy bound constraints as equality. The inactive set is a set of indices that satisfy bound constraints as strict inequality. For SVDD dual formulation, the active set refers to the set indices of observations for which \( \alpha = 0 \) or \( \alpha = C \), and the inactive set refers to set indices of observations for which \( 0 < \alpha < C \). The active-set method seeks to form the solution by determining which indices of bound constraints belong to the active set and by using the inactive set (whose size is relatively small) to solve the optimization problem. Even though there is a great number of ways to partition bound constraint indices into active and inactive sets, the number of combinations tried in practice is usually small. A consequence is that the run time for the active-set method for a particular problem is hard to predict. The computational cost of the active-set method is also controlled by the number of support vectors, and that number is highly dependent on both the problem and the choice of the kernel bandwidth value. As a rule, the run time increases as the kernel bandwidth value decreases and the corresponding number of support vectors increases.

Stochastic Subset Solver

The stochastic subset solver for SVDD training provides an efficient way to train your SVDD model and is recommended for training large data sets. The stochastic solver performs the following steps on training data with a sample size \( n \), Gaussian bandwidth parameter \( s \), fraction outlier \( f \), and convergence criteria parameters that are specified in the SOLVER statement:

1. Initialize a master set of support vectors \( SV^* \) by computing the SVDD of a random sample \( S_0 \). Set \( i = 1 \).
2. Select a random sample \( S_i \). Compute its SVDD. Obtain a set of support vectors \( SV_i \). When the WEIGHT statement is specified, the probability of an observation becoming part of the sample is proportional to the observation’s weight value.
3. Compute the SVDD of \((SV^* \cup SV_i)\). Designate corresponding support vectors as \( SV^* \). Set \( i = i + 1 \).
4. Repeat steps 2 and 3 until convergence of the threshold value \( R^2 \) and center \( a \).

When convergence is obtained, the resulting master set of support vectors \( SV^* \) is set as an input to the active-set solver and a solution is obtained. For more information about the stochastic subset solver, see Chaudhuri et al. (2018a).
Fast Incremental Solver

The fast incremental solver uses incremental updates to the set of support vectors for performing SVDD training. This algorithm builds on an observation that all support vectors on the boundary have the same distance to the center of sphere in a higher-dimensional feature space as mapped by the Gaussian kernel function. This method uses rank-1 updates and a fast bookkeeping operation to update the solution. If a new point does not substantially alter the solution, then it is discarded. For more information about the fast incremental solver, see (Jiang et al. 2019).

Processing Mode and SVDD Results

You can run the SVDD procedure in symmetric multiprocessing mode (SMP) or massively parallel processing mode (MPP). In MPP mode, the training data are distributed across multiple worker nodes. When PROC SVDD performs training, observations move to and from different worker nodes for computation. Which sequence this data movement follows cannot be guaranteed, because it depends on a number of external factors, such as the load on the worker node. Different data movement sequences can lead to minor differences in the training results. You notice these differences when you train the same data set multiple times, keeping all options unchanged. This behavior is expected.

Scoring Multiple SVDD Models with the ASTORE Procedure

The KERNEL statement section outlines how you can specify multiple BW= option values in that statement. You can use the SAVESTATE statement to generate a single analytical store that contains scoring models corresponding to each bandwidth value. If the BW= option contains \( n \) bandwidth values, then the analytical store will contain \( n \) scoring models. The scoring model for each bandwidth value is stored inside the analytic store with a unique value of the modelID variable. When the BW= option contains \( n \) bandwidth values, the value of the modelID variable ranges from 1 to \( n \).

In the ASTORE procedure, you can use a subset of scoring models by specifying the SETOPTION statement with the MODELID option. The following examples show how to use the SETOPTION statement in PROC ASTORE.

The following statements execute the SVDD algorithm on the mycas.refrigerant data table by using three bandwidth values, 0.05, 0.1, and 0.15:

```
proc svdd data=mycas.Refrigerant score nbins=10;
  id y4 y5;
  input y4 y5/level=interval;
  kernel rbf / bw=(0.05 0.1 0.15);
  savestate rstore=mycas.state;
run;
```

The analytic store mycas.state contains three scoring models, with modelID 1, 2, and 3 corresponding to the bandwidth values 0.05, 0.1, and 0.15, respectively.

The following statements execute the ASTORE procedure without the SETOPTION statement. This causes the scoring data to be scored using all three models, corresponding with the bandwidth values 0.05, 0.1, and 0.15.
The following statements execute the ASTORE procedure with the SETOPTION statement. This causes the scoring data to be scored using the model for which MODELID=2, corresponding to the option BW=0.1.

```plaintext
proc astore;
  score data=mycas.scoreds
  out=mycas.score_results
  rstore=mycas.state;
  setoption modelid 2;
quit;
```

You can instruct the ASTORE procedure to keep bandwidth values that correspond to each model in the OUT data table by setting the BWCOL option to 1. You can remove the bandwidth value from the OUT data table by setting the BWCOL option to 0. The default value of the BWCOL option is 0.

The following statements execute the ASTORE procedure with the SETOPTION statement. The BWCOL option is set to 1, which causes the mycas.score_result to include bandwidth columns for each model.

```plaintext
proc astore;
  score data=mycas.scoreds
  out=mycas.score_results
  rstore=mycas.state;
  setoption bwcol 1;
quit;
```

### ODS Table Names

Each table that the SVDD procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. The name of each table and a short description of the contents are listed in Table 22.1.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>DescStatsNom</td>
<td>Number of levels of nominal variables in the input data</td>
<td>PROC SVDD</td>
<td>Default</td>
</tr>
<tr>
<td>Histogram</td>
<td>Frequency table of <em>SVDDDISTANCE</em> values of training data</td>
<td>PROC SVDD</td>
<td>Default</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Basic model information for the training</td>
<td>PROC SVDD</td>
<td>Default</td>
</tr>
<tr>
<td>Nobs</td>
<td>Observation information about the input data</td>
<td>PROC SVDD</td>
<td>Default</td>
</tr>
</tbody>
</table>
Examples: SVDD Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

Example 22.1: Analysis of Aircraft Engine Degradation

Sun and Tsung (2003) introduced an SVDD-based $K$-chart, which is a nonparametric multivariate control chart that is used for statistical process control and can also be used for monitoring equipment health and operating data. A $K$-chart is implemented in two phases: In phase I, observations from normal operations of the process are collected and are used to train a SVDD model and obtain the threshold $R^2$ value. This model of normal operations is then operationalized in phase II for process or equipment monitoring. For each new observation $z$, its distance value $\text{dist}^2(z)$ is computed and compared to the threshold $R^2$ value. Observations for which $\text{dist}^2(z) > R^2$ generally indicate something abnormal in the process.

This example uses a $K$-chart to illustrate the analysis of aircraft engine degradation and uses data from Saxena et al. (2008); Saxena and Goebel (2008). The data that are used in this example consist of the flight history of 15 engines. For each flight, three variables are related to the engine’s operating conditions and 21 variables are for sensor measurements. Because each engine degrades at a different rate, the number of flights until the end of life is different for each engine.

The following code creates a phase I (normal operations) data set (the *engine* data set in the *Work* library) by selecting the first 25% of observations of a random sample of five engines. The analysis assumes that the first 25% of observations represent normal operations of the aircraft engine.

```sas
data engine;
  input Engine cycle x1-x24;
datalines;
... more lines ...
```

The following statements creates a subset of *work.engine* by selecting the first 25% of observations that belong to five randomly selected engines:
data eid(keep=engine);
  set engine;
  by engine;
  if first.engine then output;
run;

data reid;
  retain nleft 5;
  set eid nobs=n;
  r = ranuni(1994);
  if r <= nleft/n then do;
    nleft = nleft - 1;
    output;
  end;
  n = n - 1;
run;

%macro a;

data _null_;  
  set reid;
  call symput('e'||left(_n_), engine);
run;

%do i=1 %to 5;
  %if &i=1 %then %do;
    %let st=&&e&i;
  %end;
  %else %do;
    %let st=&st &&e&i;
  %end;
%end;

data eid_cy;
  set engine;
  by engine;
  if engine in (&st) and last.engine=1 then output;
run;

data eid_cy;
  set eid_cy;
  ds1=floor(0.25*cycle);
run;

/* The following DATA steps create the training data set */
data train;
  set engine;
  if engine in (&st) then output;
run;

data train;
  merge train(in=a) eid_cy(in=b);
  by engine;
  if a=b;
Example 22.1: Analysis of Aircraft Engine Degradation

### Example 22.1: Analysis of Aircraft Engine Degradation

```r
run;

data train;
    set train (where=(cycle le ds1));
run;

/* The following DATA step creates the scoring data set. The scoring data set
contains flight data for all engines, which were not part of the training data set */
data score;
    set engine;
    if engine not in (&st) then output;
run;

%mend a;
%

You can load the work.train 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, as in the section “Using CAS Sessions and CAS Engine Librefs” on page 479, but you can substitute
any appropriately defined CAS engine libref.

data mycas.train;
    set train;
run;

You can load the work.score data set into your CAS session by specifying your CAS engine libref in the
second statement in the following DATA step:

data mycas.score;
    set score;
run;

The following statements execute the SVDD algorithm on the mycas.train data table and produce Figure 22.1.1
through Figure 22.1.2.

    proc svdd data=mycas.train;
        id engine cycle;
        input x1-x24/level=interval;
        kernel rbf / bw=94;
        savestate rstore=mycas.state;
    run;
```

The INPUT statement defines 24 input variables (x1–x24) as interval variables. The KERNEL statement
specifies the radial basis function as the kernel function and specifies a value of 94 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 22.1.1 summarizes the key options and input data variables.
Output 22.1.1  Aircraft Engine Degradation Analysis Model Information

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
</tr>
<tr>
<td>Kernel Type</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth</td>
</tr>
<tr>
<td>Bandwidth Selection Method</td>
</tr>
<tr>
<td>Bandwidth Relative Scale</td>
</tr>
<tr>
<td>Expected Outlier Fraction</td>
</tr>
<tr>
<td>Optimization Tolerance</td>
</tr>
<tr>
<td>Number of Interval Variables</td>
</tr>
<tr>
<td>Number of Nominal Variables</td>
</tr>
</tbody>
</table>

The “Training Results” table in Output 22.1.2 shows the number of support vectors and the threshold $R^2$ value.

Output 22.1.2  Aircraft Engine Degradation Analysis Training Results

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary</td>
</tr>
<tr>
<td>Number of Dropped Observations</td>
</tr>
<tr>
<td>Threshold R Square Value</td>
</tr>
<tr>
<td>Constant (C_r) Value</td>
</tr>
<tr>
<td>Run Time (seconds)</td>
</tr>
</tbody>
</table>

Output 22.1.3 displays the number of observations in the training data set.

Output 22.1.3  Aircraft Engine Degradation Analysis Training Observations

| Number of Observations Read | 225 |
| Number of Observations Used  | 225 |

The “Optimization Summary” table in Output 22.1.4 shows whether the solution is optimal, the number of iterations that were required and the objective function value.

Output 22.1.4  Aircraft Engine Degradation Analysis Optimization Summary

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Objective Value</td>
</tr>
<tr>
<td>Infeasibility</td>
</tr>
<tr>
<td>Optimization Status</td>
</tr>
<tr>
<td>Degenerate</td>
</tr>
</tbody>
</table>

The following PROC ASTORE code scores the mycas.score data, which contain the entire flight history for 10 engines. In the context of $K$-charts, this scoring constitutes the phase II analysis, where new observations are scored to infer any departure from the normal operations.
Example 22.1: Analysis of Aircraft Engine Degradation

```sas
proc astore;
    score data=mycas.score
    out=mycas.test_out
    rstore=mycas.state;
quit;
```

Output 22.1.5 shows the scoring results for Engine 6. The plot indicates the \( \text{dist}^2(z) \) value that is plotted on the Y axis against the flight or cycle number on the X axis. The blue line indicates the \( \text{dist}^2(z) \) value, and the red line indicates a moving average that is computed using five observations. The horizontal reference line indicates the threshold \( R^2 \) value of 0.80378 that was obtained from training, which also acts as an upper control limit (UCL) for the \( \text{dist}^2(z) \) value. The vertical reference line indicates the last cycle (flight) of the first engine. Output 22.1.5 shows that the \( \text{dist}^2(z) \) value serves as a good proxy for engine degradation.
Example 22.2: Automatic RBF Kernel Bandwidth Parameter Value Selection

This example demonstrates how you can use automatic kernel bandwidth selection for the radial basis function (RBF) in the SVDD procedure. In this example, the use of the mean criterion is illustrated first, followed by the use of the modified mean criterion and the trace criterion. The training data in this example are refrigerant analysis data (Heckert and Filliben 2000). These data are also used in the section “Getting Started: SVDD Procedure” on page 480, where the value of the RBF kernel bandwidth parameter is predetermined by the specification of the BW=0.021 option in the KERNEL statement. In the current example, the SVDD procedure computes the RBF kernel bandwidth parameter value by using the automatic bandwidth selection methods, alleviating the need to explicitly specify a value. 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 $y_5$ (inlet water temperature 1) and $y_4$ (pressure drop 6) to perform an SVDD training. The variables $y_4$ and $y_5$ 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 Refrigerant data set (which is stored in the Work library by default):

```r
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 ...
quit;
```

The following code standardizes the variables $y_4$ and $y_5$:

```r
proc sql noprint;
  select min(y4), min(y5), max(y4), max(y5)
  into :miny4, :miny5, :maxy4, :maxy5
  from Refrigerant;
quit;
```

```r
data Refrigerant;
  set Refrigerant;
  y4=(y4-&miny4)/%sysevalf(&maxy4-&miny4);  
  y5=(y5-&miny5)/%sysevalf(&maxy5-&miny5);
run;
```

The following PROC SG PLOT statements plot inlet water temperature 1 against pressure drop 6:

```r
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 22.2.1 shows the results. The scatter plot indicates four distinct clusters.
Example 22.2: Automatic RBF Kernel Bandwidth Parameter Value Selection

Output 22.2.1 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:

```plaintext
data mycas.Refrigerant;
   set Refrigerant;
run;
```

These statements assume that your CAS engine libref is named mycas, as in the section “Using CAS Sessions and CAS Engine Librefs” on page 479, but you can substitute any appropriately defined CAS engine libref.

The following statements execute the SVDD algorithm on the mycas.Refrigerant data table and produce Output 22.2.2 through Output 22.2.6.

```plaintext
proc svdd data=mycas.Refrigerant score nbins=10;
   id y4 y5;
   input y4 y5/level=interval;
   kernel rbf / bw=mean;
   savestate rstore=mycas.state;
run;
```

The SCORE option in the PROC SVDD statement requests that the training data set be scored. The NBINS=10 option creates a histogram data set with 10 bins. 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 the BW=MEAN option instructs the SVDD procedure to select value of the RBF bandwidth parameter by using the mean criterion. 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 22.2.2 summarizes the key options and the input data variables, and it provides the value of the RBF kernel bandwidth parameter, which is computed to be 0.1262 by using the mean criterion.
Output 22.2.2  Refrigerant Analysis Model Information

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
</tr>
<tr>
<td>Kernel Type</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth</td>
</tr>
<tr>
<td>Bandwidth Selection Method</td>
</tr>
<tr>
<td>Bandwidth Relative Scale</td>
</tr>
<tr>
<td>Expected Outlier Fraction</td>
</tr>
<tr>
<td>Optimization Tolerance</td>
</tr>
<tr>
<td>Number of Interval Variables</td>
</tr>
<tr>
<td>Number of Nominal Variables</td>
</tr>
</tbody>
</table>

The “Training Results” table in Output 22.2.3 shows the number of support vectors and the $R^2$ threshold value.

Output 22.2.3  Refrigerant Analysis Training Results

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary</td>
</tr>
<tr>
<td>Number of Outliers</td>
</tr>
<tr>
<td>Number of Dropped Observations</td>
</tr>
<tr>
<td>Threshold R Square Value</td>
</tr>
<tr>
<td>Constant (C_r) Value</td>
</tr>
<tr>
<td>Run Time (seconds)</td>
</tr>
<tr>
<td>Bandwidth Calculation Time (seconds)</td>
</tr>
</tbody>
</table>

Output 22.2.4 displays the number of observations in the training data set.

Output 22.2.4  Refrigerant Analysis Training Observations

| Number of Observations Read | 360 |
| Number of Observations Used  | 360 |

The “Optimization Summary” table in Output 22.2.5 shows whether the solution is optimal, the number of iterations that were required, and the objective function value.

Output 22.2.5  Refrigerant Analysis Optimization Summary

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Objective Value</td>
</tr>
<tr>
<td>Infeasibility</td>
</tr>
<tr>
<td>Optimization Status</td>
</tr>
<tr>
<td>Degenerate</td>
</tr>
</tbody>
</table>

The “Histogram” table in Output 22.2.6 shows the frequency table of _SVDDDISTANCE_ values of observations in the mycas.Refrigerant training data set.
Output 22.2.6  Refrigerant Analysis Histogram Table

<table>
<thead>
<tr>
<th>Bin Midpoint</th>
<th>Count</th>
<th>Percent (%)</th>
<th>Cumulative (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.716108</td>
<td>28</td>
<td>7.78</td>
<td>7.78</td>
</tr>
<tr>
<td>0.721048</td>
<td>45</td>
<td>12.50</td>
<td>20.28</td>
</tr>
<tr>
<td>0.725989</td>
<td>73</td>
<td>20.28</td>
<td>40.56</td>
</tr>
<tr>
<td>0.730929</td>
<td>62</td>
<td>17.22</td>
<td>57.78</td>
</tr>
<tr>
<td>0.735869</td>
<td>48</td>
<td>13.33</td>
<td>71.11</td>
</tr>
<tr>
<td>0.740810</td>
<td>37</td>
<td>10.28</td>
<td>81.39</td>
</tr>
<tr>
<td>0.745750</td>
<td>30</td>
<td>8.33</td>
<td>89.72</td>
</tr>
<tr>
<td>0.750691</td>
<td>12</td>
<td>3.33</td>
<td>93.06</td>
</tr>
<tr>
<td>0.755631</td>
<td>15</td>
<td>4.17</td>
<td>97.22</td>
</tr>
<tr>
<td>0.760572</td>
<td>10</td>
<td>2.78</td>
<td>100.00</td>
</tr>
</tbody>
</table>

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:

```plaintext
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 first statement in the following DATA step:

```plaintext
data mycas.scoreds;
  set scoreds;
run;
```

The following PROC ASTORE code scores the mycas.scoreds data table by using the score analytic store that was created during training:

```plaintext
proc astore;
  score data=mycas.scoreds
    out=mycas.score_results
    rstore=mycas.state;
quit;
```

The following code copies the score_results data table into the Work library:

```plaintext
data work.score_results;
  set mycas.score_results;
run;
```

The following PROC SG PLOT code plots the scoring results:
Chapter 22: The SVDD Procedure

proc sgplot data=score_results(where=(_svddscore_ in (-1,1)));  
styleattrs datacontrastcolors=(ligr black) datasymbols=(circlefilled);  
title "Refrigerant Analysis: Mean Criterion";  
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 22.2.7 shows the scatter plot of the scoring data. The gray area indicates the observations that are identified as outliers, and the black area indicates observations that are identified as inliers. In this example, the mean criterion was used to compute the RBF kernel bandwidth parameter value. Comparison between Output 22.2.7 and the training data in Output 22.2.1 indicates that PROC SVDD identified three out of four clusters in the original data. The training data observations in the two clusters located in the lower left corner were combined into one cluster. In addition, the description of clusters that is obtained in this example is more circular. This is expected because the RBF bandwidth parameter value of 0.1262, which is computed using the mean criterion, is higher than the value 0.021, which is specified in the section “Getting Started: SVDD Procedure” on page 480. Despite these shortcomings, the mean criterion provides a reasonable approximation of the training data. The scatter plot of the scoring data in Output 22.2.7 indicates that PROC SVDD was able to correctly identify a majority of the outliers.

Output 22.2.7 Scoring Results Scatter Plot: Bandwidth Computation Using the Mean Criterion

The use of the modified mean criterion for bandwidth selection is illustrated next.

The following statements execute the SVDD algorithm on the mycas.Refrigerant data table and produce the results shown in Output 22.2.8 through Output 22.2.12. The SVDD algorithm uses the bandwidth value that is computed using the modified mean criterion.
Example 22.2: Automatic RBF Kernel Bandwidth Parameter Value Selection

```plaintext
proc svdd data=mycas.Refrigerant score nbins=10;
  id y4 y5;
  input y4 y5/level=interval;
  kernel rbf / bw=mean2;
  savestate rstore=mycas.state;
run;
```

The SCORE option in the PROC SVDD statement requests that the training data table be scored. The NBINS=10 option creates a histogram data table that contains 10 bins. 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 the BW=MEAN2 option instructs the SVDD procedure to select the value of the RBF bandwidth parameter by using the modified mean criterion. The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in the data table mycas.state. You can use the analytic store later in the ASTORE procedure to score new data. The “Model Information” table in Output 22.2.8 summarizes the key options and the input data variables, and it provides the value of the RBF kernel bandwidth parameter, which is computed using the modified mean criterion to be 0.197.

### Output 22.2.8 Refrigerant Analysis Model Information

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
</tr>
<tr>
<td>Kernel Type</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth</td>
</tr>
<tr>
<td>Bandwidth Selection Method</td>
</tr>
<tr>
<td>Bandwidth Relative Scale</td>
</tr>
<tr>
<td>Expected Outlier Fraction</td>
</tr>
<tr>
<td>Optimization Tolerance</td>
</tr>
<tr>
<td>Number of Interval Variables</td>
</tr>
<tr>
<td>Number of Nominal Variables</td>
</tr>
</tbody>
</table>

The “Training Results” table in Output 22.2.9 shows the number of support vectors and the $R^2$ threshold value.

### Output 22.2.9 Refrigerant Analysis Training Results

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary</td>
</tr>
<tr>
<td>Number of Outliers</td>
</tr>
<tr>
<td>Number of Dropped Observations</td>
</tr>
<tr>
<td>Threshold R Square Value</td>
</tr>
<tr>
<td>Constant (C_r) Value</td>
</tr>
<tr>
<td>Run Time (seconds)</td>
</tr>
<tr>
<td>Bandwidth Calculation Time (seconds)</td>
</tr>
</tbody>
</table>

Output 22.2.10 displays the number of observations in the training data table.
Output 22.2.10  Refrigerant Analysis Training Observations

<table>
<thead>
<tr>
<th>Number of Observations Read</th>
<th>360</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Used</td>
<td>360</td>
</tr>
</tbody>
</table>

The “Optimization Summary” table in Output 22.2.11 shows whether the solution is optimal, the number of iterations that were required, and the objective function value.

Output 22.2.11  Refrigerant Analysis Optimization Summary

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Objective Value</td>
</tr>
<tr>
<td>Infeasibility</td>
</tr>
<tr>
<td>Optimization Status</td>
</tr>
<tr>
<td>Degenerate</td>
</tr>
</tbody>
</table>

The “Histogram” table in Output 22.2.12 shows the frequency table of _SVDDDISTANCE_ values of observations in the mycas.Refrigerant training data table.

Output 22.2.12  Refrigerant Analysis Histogram Table

<table>
<thead>
<tr>
<th>Frequency Distribution for SVDD Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin Midpoint</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>0.656727</td>
</tr>
<tr>
<td>0.663317</td>
</tr>
<tr>
<td>0.669908</td>
</tr>
<tr>
<td>0.676498</td>
</tr>
<tr>
<td>0.683088</td>
</tr>
<tr>
<td>0.689678</td>
</tr>
<tr>
<td>0.696269</td>
</tr>
<tr>
<td>0.702859</td>
</tr>
<tr>
<td>0.709449</td>
</tr>
<tr>
<td>0.716039</td>
</tr>
</tbody>
</table>

The following PROC ASTORE code scores the mycas.scoreds data table by using the score analytic store that was created during training. The mycas.scoreds data table that is used in the following code is the same scoring data used earlier to evaluate the quality of results that were obtained using the SVDD model that was trained with the mean criterion. Using the same data enables you to objectively compare the bandwidth values that are obtained using the mean criterion and the modified mean criterion.

```
proc astore;
   score data=mycas.scoreds
   out=mycas.score_results
   rstore=mycas.state;
quit;
```
The following code copies the `score_results` into the `Work` library:

```sas
data work.score_results;
set mycas.score_results;
run;
```

The following PROC SGPLOT code plots the scoring results:

```sas
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: Modified Mean Criterion";
scatter x=y4 y=y5/group=_svddscore_ markerattrs=(size=3);
run;
```

Output 22.2.13 shows the scatter plot of the scoring data table. The gray area indicates the observations that are identified as outliers, and the black area indicates observations that are identified as inliers. In this example, the modified mean criterion was used to compute the RBF kernel bandwidth parameter value. Comparison between Output 22.2.13 and the training data in Output 22.2.1 indicates that PROC SVDD identified three out of four clusters in the original data. The training data observations in the two clusters that are located in the lower left corner were combined into one cluster. In addition, the description of clusters that is obtained in this example is more circular. This is expected because the RBF bandwidth parameter value of 0.197, which is computed using the modified mean criterion, is higher than the value 0.021, which is specified in the section “Getting Started: SVDD Procedure” on page 480. Comparing the scoring results in Output 22.2.13 and Output 22.2.7 indicates that the description that is obtained using the modified mean criterion is more circular than the description that is obtained using the mean criterion with a default value of DELTA. Despite these shortcomings, the modified mean criterion provides a reasonable approximation of the training data. The scatter plot of the scoring data in Output 22.2.7 indicates that PROC SVDD was able to correctly identify a majority of the outliers.

Output 22.2.13 shows trace results.
Chapter 22: The SVDD Procedure

Output 22.2.13 Scoring Results Scatter Plot: Bandwidth Computation Using the Modified Mean Criterion

The use of the trace criterion for bandwidth selection is illustrated next.

The following statements execute the SVDD algorithm on the mycas.Refrigerant data table and produce the results shown in Output 22.2.14 through Output 22.2.18. The SVDD algorithm uses the bandwidth value that is computed using the trace criterion.

```
proc svdd data=mycas.Refrigerant score nbins=10;
  id y4 y5;
  input y4 y5/level=interval;
  kernel rbf / bw=trace;
  savestate rstore=mycas.state;
run;
```

The SCORE option in the PROC SVDD statement requests that the training data table be scored. The NBINS=10 option creates a histogram data table that contains 10 bins. 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 the BW=TRACE option instructs the SVDD procedure to select the value of the RBF bandwidth parameter by using the trace criterion. The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in the data table mycas.state. You can use the analytic store later in the ASTORE procedure to score new data. The “Model Information” table in Output 22.2.14 summarizes the key options and the input data variables, and it provides the value of the RBF kernel bandwidth parameter, which is computed using the trace criterion to be 0.0132.
Example 22.2: Automatic RBF Kernel Bandwidth Parameter Value Selection

**Output 22.2.14** Refrigerant Analysis Model Information

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
</tr>
<tr>
<td>Kernel Type</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth</td>
</tr>
<tr>
<td>Bandwidth Selection Method</td>
</tr>
<tr>
<td>Bandwidth Relative Scale</td>
</tr>
<tr>
<td>Expected Outlier Fraction</td>
</tr>
<tr>
<td>Optimization Tolerance</td>
</tr>
<tr>
<td>Number of Interval Variables</td>
</tr>
<tr>
<td>Number of Nominal Variables</td>
</tr>
</tbody>
</table>

The “Training Results” table in **Output 22.2.15** shows the number of support vectors and the $R^2$ threshold value.

**Output 22.2.15** Refrigerant Analysis Training Results

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary</td>
</tr>
<tr>
<td>Number of Outliers</td>
</tr>
<tr>
<td>Number of Dropped Observations</td>
</tr>
<tr>
<td>Threshold R Square Value</td>
</tr>
<tr>
<td>Constant ($C_r$) Value</td>
</tr>
<tr>
<td>Run Time (seconds)</td>
</tr>
<tr>
<td>Bandwidth Calculation Time (seconds)</td>
</tr>
</tbody>
</table>

**Output 22.2.16** displays the number of observations in the training data table.

**Output 22.2.16** Refrigerant Analysis Training Observations

<table>
<thead>
<tr>
<th>Number of Observations Read</th>
<th>360</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Used</td>
<td>360</td>
</tr>
</tbody>
</table>

The “Optimization Summary” table in **Output 22.2.17** shows whether the solution is optimal, the number of iterations that were required, and the objective function value.

**Output 22.2.17** Refrigerant Analysis Optimization Summary

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Objective Value</td>
</tr>
<tr>
<td>Infeasibility</td>
</tr>
<tr>
<td>Optimization Status</td>
</tr>
<tr>
<td>Degenerate</td>
</tr>
</tbody>
</table>

The “Histogram” table in **Output 22.2.18** shows the frequency table of _SVDDDISTANCE_ values of observations in the mycas.Refrigerant training data table.
Chapter 22: The SVDD Procedure

Output 22.2.18  Refrigerant Analysis Histogram Table

<table>
<thead>
<tr>
<th>Bin Midpoint</th>
<th>Count</th>
<th>Percent (%)</th>
<th>Cumulative (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.957064</td>
<td>4</td>
<td>1.11</td>
<td>1.11</td>
</tr>
<tr>
<td>0.958111</td>
<td>5</td>
<td>1.39</td>
<td>2.50</td>
</tr>
<tr>
<td>0.959157</td>
<td>5</td>
<td>1.39</td>
<td>3.89</td>
</tr>
<tr>
<td>0.960204</td>
<td>4</td>
<td>1.11</td>
<td>5.00</td>
</tr>
<tr>
<td>0.962298</td>
<td>40</td>
<td>11.11</td>
<td>25.00</td>
</tr>
<tr>
<td>0.963345</td>
<td>26</td>
<td>7.22</td>
<td>32.22</td>
</tr>
<tr>
<td>0.964392</td>
<td>43</td>
<td>11.94</td>
<td>44.17</td>
</tr>
<tr>
<td>0.965439</td>
<td>66</td>
<td>18.33</td>
<td>62.50</td>
</tr>
<tr>
<td>0.966486</td>
<td>135</td>
<td>37.50</td>
<td>100.00</td>
</tr>
</tbody>
</table>

The following PROC ASTORE code scores the mycas.scoreds data table by using the score analytic store that was created during training. The mycas.scoreds data table that is used in the following code is the same scoring data used earlier to evaluate the quality of results that were obtained using the SVDD model that was trained with the mean criterion. Using the same data enables you to objectively compare the bandwidth values that are obtained using the mean criterion, the modified mean criterion, and the trace criterion.

   proc astore;
      score data=mycas.scoreds
         out=mycas.score_results
         rstore=mycas.state;
   quit;

The following code copies the score_results into the Work library:

   data work.score_results;
      set mycas.score_results;
   run;

The following PROC SGPLOT code plots 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: Trace Criterion";
      scatter x=y4 y=y5/group=_svddscore_ markerattrs=(size=3);
   run;

Output 22.2.13 shows the scatter plot of the scoring data table. The gray area indicates the observations that are identified as outliers, and the black area indicates observations that are identified as inliers. In this example, the trace criterion was used to compute the RBF kernel bandwidth parameter value. Comparison between Output 22.2.19 and the training data in Output 22.2.1 indicates that PROC SVDD identified all four clusters in the original data. The RBF bandwidth parameter value of 0.013, which is computed using the trace criterion, is close to the value 0.021, which is specified in the section “Getting Started: SVDD Procedure” on page 480. Comparing the scoring results in Output 22.2.13 and Output 22.2.7 with the scoring results in Output 22.2.19 indicates that the description that is obtained using the trace criterion is more accurate than
Example 22.3: Stochastic Subset Solver

The stochastic subset solver for SVDD training provides an efficient way to train an SVDD model and is recommended for training large data sets. For more information, see the section “Stochastic Subset Solver” on page 500. This example compares the active-set solver with the stochastic subset solver in terms of processing time and quality of results.

The following DATA step creates the two_donut data set, which contains 4,000,000 observations and is stored in the Work library by default:

```sas
%let r_min=2;
%let r_max=5;

data two_donut;
  call streaminit(24215);
  two_pi = 2 * constant("pi");
  a = &r_min * &r_min;
  b = (&r_max + &r_min) * ( &r_max - &r_min ) ;
  do _n_ = 1 to 2000000;
    u1 = rand("uniform");
    r = sqrt( a + u1 * b );
    u2 = rand("uniform");
    t = two_pi * u2;
  end;
run;
```

the descriptions that were obtained using the modified mean criterion and mean criterion with a default value of DELTA. Output 22.2.19 shows trace results.

**Output 22.2.19** Scoring Results Scatter Plot: Bandwidth Computation Using the Trace Criterion

![Scoring Results Scatter Plot](image_url)
x = r * cos(t); y = r * sin(t);
output;
x = x + 10; y = y + 20;
output;
end;
keep x y;
run;

The following statements plot the two_donut data set:

ods graphics/maxobs=48000000;
proc sgplot data=two_donut;
    title "Two Donut Data set";
    scatter x=x y=y/markerattrs=(size=3 symbol=circlefilled);
run;

Output 22.3.1 shows the results.

Output 22.3.1 Scatter Plot of Two-Donut Data

You can load the Work.two_donut data set into your CAS session by specifying your CAS engine libref in the first statement in the following DATA step:

data mycas.two_donut;
    set two_donut;
run;

These statements assume that your CAS engine libref is named mycas, as in the section “Using CAS Sessions and CAS Engine Librefs” on page 479, but you can substitute any appropriately defined CAS engine libref. The following statements execute the SVDD algorithm on the mycas.two_donut data table by using the active-set solver and produce Output 22.3.2 through Output 22.3.5:

proc svdd data=mycas.two_donut std;
    id x y;
    input x y/level=interval;
    kernel rbf/bw=MEAN;
    solver actset/stol=1e-6;
    savestate rstore=mycas.actset;
run;
The STD option in the PROC SVDD statement standardizes the mycas.two_donut data set. The INPUT statement defines the input variables x and y as interval variables. The KERNEL statement specifies the kernel function as a radial basis function (RBF), and the BW=MEAN option requests that the SVDD procedure select the value of the RBF bandwidth parameter by using the mean criterion. The SOLVER statement specifies ACTSET (active-set solver) as the SVDD solver. The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in the mycas.actset data table. You can use the analytic store later in the ASTORE procedure to score new data.

The “Model Information” table in Output 22.3.2 summarizes the key options and the input data variables, and it shows that the value of the RBF kernel bandwidth parameter (which is computed by using the mean criterion) is 0.3055.

### Output 22.3.2  Two Donut Model Information: Active-Set Solver

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
</tr>
<tr>
<td>Kernel Type</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth</td>
</tr>
<tr>
<td>Bandwidth Selection Method</td>
</tr>
<tr>
<td>Bandwidth Relative Scale</td>
</tr>
<tr>
<td>Expected Outlier Fraction</td>
</tr>
<tr>
<td>Optimization Tolerance</td>
</tr>
<tr>
<td>Standardization</td>
</tr>
<tr>
<td>Number of Interval Variables</td>
</tr>
<tr>
<td>Number of Nominal Variables</td>
</tr>
</tbody>
</table>

The “Training Results” table in Output 22.3.3 shows the number of support vectors and the $R^2$ threshold value.

### Output 22.3.3  Two Donut Analysis Training Results: Active-Set Solver

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary</td>
</tr>
<tr>
<td>Number of Dropped Observations</td>
</tr>
<tr>
<td>Threshold R Square Value</td>
</tr>
<tr>
<td>Constant ($C_r$) Value</td>
</tr>
<tr>
<td>Run Time (seconds)</td>
</tr>
<tr>
<td>Bandwidth Calculation Time (seconds)</td>
</tr>
</tbody>
</table>

Output 22.3.4 displays the number of observations in the training data set.

### Output 22.3.4  Two Donut Training Observations: Active-Set Solver

| Number of Observations Read       | 4000000 |
| Number of Observations Used       | 4000000 |

The “Optimization Summary” table in Output 22.3.5 shows whether the solution is optimal, the number of iterations that were required, and the objective function value.
Chapter 22: The SVDD Procedure

**Output 22.3.5** Two Donut Optimization Summary: Active-Set Solver

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Objective Value</td>
</tr>
<tr>
<td>Infeasibility</td>
</tr>
<tr>
<td>Optimization Status</td>
</tr>
<tr>
<td>Degenerate</td>
</tr>
</tbody>
</table>

The following statements execute the SVDD algorithm on the mycas.two_donut data table by using the STOCHS solver and produce Output 22.3.6 through Output 22.3.9.

```plaintext
proc svdd data=mycas.two_donut std;
  id x y;
  input x y/level=interval;
  kernel rbf/bw=MEAN;
  solver stochs/nsamp=10 nmatch=10 rtol=1e-6 ctol=1e-6 stol=1e-6 seed=10;
  savestate rstore=mycas.stochs;
run;
```

The STD option in PROC SVDD statement standardizes the mycas.two_donut data set. The INPUT statement defines the input variables x and y as interval variables. The KERNEL statement specifies the kernel function as a radial basis function (RBF), and the BW=MEAN option requests that the SVDD procedure select the value of the RBF bandwidth parameter by using the mean criterion. The SOLVER statement specifies STOCHS (stochastic subset solver) as the SVDD solver. The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in the mycas.stochs data table. You can use the analytic store later in the ASTORE procedure to score new data.

The “Model Information” table in Output 22.3.6 summarizes the key options and the input data variables, and it shows that the value of the RBF kernel bandwidth parameter (which is computed by using the mean criterion) is 0.3055.

**Output 22.3.6** Two Donut Model Information: Stochastic Subset Solver

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
</tr>
<tr>
<td>Kernel Type</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth</td>
</tr>
<tr>
<td>Bandwidth Selection Method</td>
</tr>
<tr>
<td>Bandwidth Relative Scale</td>
</tr>
<tr>
<td>Sample Size</td>
</tr>
<tr>
<td>Seed</td>
</tr>
<tr>
<td>Optimization Tolerance</td>
</tr>
<tr>
<td>Standardization</td>
</tr>
<tr>
<td>Number of Interval Variables</td>
</tr>
<tr>
<td>Number of Nominal Variables</td>
</tr>
</tbody>
</table>

The “Training Results” table in Output 22.3.7 shows the number of support vectors and the $R^2$ threshold value.
Output 22.3.7  Two Donut Analysis Training Results: Stochastic Subset Solver

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary</td>
</tr>
<tr>
<td>Number of Dropped Observations</td>
</tr>
<tr>
<td>Threshold $R^2$ Value</td>
</tr>
<tr>
<td>Constant ($C_r$) Value</td>
</tr>
<tr>
<td>Run Time (seconds)</td>
</tr>
<tr>
<td>Bandwidth Calculation Time (seconds)</td>
</tr>
</tbody>
</table>

Output 22.3.8 displays the number of observations in the training data set.

Output 22.3.8  Two Donut Training Observations: Stochastic Subset Solver

<table>
<thead>
<tr>
<th>Number of Observations Read</th>
<th>4000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Used</td>
<td>4000000</td>
</tr>
</tbody>
</table>

The “Optimization Summary” table in Output 22.3.9 shows whether the solution is optimal, the number of iterations that were required, and the objective function value.

Output 22.3.9  Two Donut Optimization Summary: Stochastic Subset Solver

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Objective Value</td>
</tr>
<tr>
<td>Infeasibility</td>
</tr>
<tr>
<td>Optimization Status</td>
</tr>
<tr>
<td>Degenerate</td>
</tr>
</tbody>
</table>

The run time, the threshold $R^2$ value, and the number of support vectors for the active-set (ACTSET) solver and the stochastic subset (STOCHS) solver are provided in “Model Information” table in Output 22.3.2 and Output 22.3.6. Comparison of the run time indicates that in this example, the stochastic subset solver performance is orders of magnitude faster than the active-set solver. The threshold $R^2$ values for the two solvers are close. The number of support vectors produced by the stochastic subset solver are less than the number produced by the active-set solver.

To compare the quality of the training results obtained using the ACTSET and STOCHS solvers, scoring is performed on a 200 x 200 data grid. The following DATA step creates a data set named scoreds to contain the scoring observations. Note that the scale of the observations in the scoreds scoring data set matches the scale of the data in the original, nonstandardized two_donut data set. The analytic stores mycas.actset and mycas.stochs contain the code necessary to standardize the scoring data set on the same scale as the training data set.

```sql
proc sql noprint;
    select min(x), min(y), max(x), max(y)
    into :minx, :miny, :maxx, :maxy
    from two_donut;
quit;
```
data scoreds;
   do i=&minx to &maxx by %sysevalf((&maxx-&minx)/200);
      do j=&miny to &maxy by %sysevalf((&maxy-&miny)/200);
         output;
      end;
   end;
   rename i=x j=y;
run;

You can load the scoreds data set into your CAS session by specifying your CAS engine libref in the first statement in the following DATA step:

data mycas.scoreds;
   set scoreds;
run;

The following code scores the mycas.scoreds using the score analytic store that was created using the ACTSET solver during training.

proc astore;
   score data=mycas.scoreds
      out=mycas.scoreact
         rstore=mycas.actset;
run;

The following code copies the mycas.scoreact data table into the Work library:

data work.scoreact;
   set mycas.scoreact;
run;

The following code plots the scoring results, as shown in Output 22.3.10:

proc sgplot data=scoreact(where=(_svddscore_ in (-1,1)));
   styleattrs datacontrastcolors=(ligr black) datasymbols=(circlefilled);
   title "Two Donut Data set";
   title1 "Scatter plot of scoring results - Active set solver";
   scatter x=x y=y/group=_svddscore_ markerattrs=(size=3);
run;
The following code scores the mycas.scoreds data table by using the score analytic store that was created using the STOCHS solver during training:

```sas
proc astore;
   score data=mycas.scoreds
      out=mycas.scorestochs
      rstore=mycas.stochs;
run;
```

The following code copies the scorestochs data table into the Work library:

```sas
data Work.scorestochs;
   set mycas.scorestochs;
run;
```

The following code plots the scoring results:

```sas
proc sgplot data=scorestochs(where=(_svddscore_ in (-1,1)));
   styleattrs datacontrastcolors=(ligr black) datasymbols=(circlefilled);
   title "Two Donut Data set";
   title1 "Scatter plot of scoring results - Stochastic subset solver";
   scatter x=x y=y/group=_svddscore_ markerattrs=(size=3);
run;
```
Chapter 22: The SVDD Procedure

Output 22.3.11 Scoring Results Scatter Plot: Stochastic Subset Solver

Output 22.3.10 and Output 22.3.11 show the scatter plots of the scoring data set that were obtained using the active-set (ACTSET) and the stochastic subset (STOCHS) solvers, respectively. The gray area indicates the observations that are identified as outliers, and the black area indicates observations that are identified as inliers. Comparison between Output 22.3.10 and Output 22.3.11 indicates that quality of the SVDD training that was obtained using active-set (ACTSET) and the stochastic subset (STOCHS) solvers is nearly identical.

In this example, the stochastic subset solver provides an order of magnitude faster performance compared to the active-set solver. The run-time advantage of the stochastic subset solver depends on the number of variables in the training data set. When a training data set has a large number of variables, the active-set solver might be faster than the stochastic subset solver.

Example 22.4: Analysis of Tennessee Eastman Process Data Using Multiple Bandwidth Values and Consensus-Based Classification

The Tennessee Eastman (TE) process is a realistic model of a chemical process that is widely used in the academic community to study multivariate process control problems. The TE process is based on an actual industrial process, with some modifications to protect the proprietary nature of the process (Downs and Vogel 1993). MATLAB simulation code that was provided by Ricker (2002) was used to generate the TE process data. Data were generated for the normal operations of the process and 20 different faulty conditions. Each observation consists of 41 variables, $x_1$–$x_{41}$; 22 variables are measured continuously (every 6 seconds on average), and 19 variables are sampled at a specified interval of either 0.1 or 0.25 hours. The variable $x_{42}$ indicates the operating mode: 0 represents the normal conditions, and 1 to 20 represent the condition when the corresponding fault was present. Table 22.2 shows the total number of observations for normal operation and for each fault condition.
This example analyzes the TE process data by using multiple bandwidth values and demonstrates consensus-based classification of the scoring results.

### Table 22.2  The Tennessee Eastman Data

<table>
<thead>
<tr>
<th>Operation Mode</th>
<th># Obs</th>
<th>Operation Mode</th>
<th># Obs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>900</td>
<td>Fault 11</td>
<td>372</td>
</tr>
<tr>
<td>Fault 1</td>
<td>294</td>
<td>Fault 12</td>
<td>427</td>
</tr>
<tr>
<td>Fault 2</td>
<td>1,007</td>
<td>Fault 13</td>
<td>465</td>
</tr>
<tr>
<td>Fault 3</td>
<td>653</td>
<td>Fault 14</td>
<td>1,729</td>
</tr>
<tr>
<td>Fault 4</td>
<td>807</td>
<td>Fault 15</td>
<td>1,729</td>
</tr>
<tr>
<td>Fault 5</td>
<td>603</td>
<td>Fault 16</td>
<td>1,124</td>
</tr>
<tr>
<td>Fault 6</td>
<td>280</td>
<td>Fault 17</td>
<td>1,953</td>
</tr>
<tr>
<td>Fault 7</td>
<td>217</td>
<td>Fault 18</td>
<td>1,741</td>
</tr>
<tr>
<td>Fault 8</td>
<td>759</td>
<td>Fault 19</td>
<td>1,729</td>
</tr>
<tr>
<td>Fault 9</td>
<td>1,465</td>
<td>Fault 20</td>
<td>1,473</td>
</tr>
<tr>
<td>Fault 10</td>
<td>1,362</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following DATA step creates the TE data set (which is stored in the Work library by default):

```plaintext
data te;
  input x1-x42;
datalines;

... more lines ...
```

The following code creates the training and scoring data sets. The ID variable is created to uniquely identify an observation. The training data set consists of 800 observations that belong to normal operations. The scoring data set contains 100 observations that belong to normal operations and 20,189 observations when the fault conditions 1 to 20 were present.

```plaintext
data TE;
  set TE;
  id=_n_;
run;

data train score;
set TE;
  if id le 800 then output train;
  else if id gt 800 then output score;
run;
```

You can load the Work.train and Work.score data sets into your CAS session by specifying your CAS engine libref in the second statement in the following DATA step:

```plaintext
data mycas.train;
set train;
run;
```
data mycas.score;
set score;
run;

These statements assume that your CAS engine libref is named mycas, as in the section “Using CAS Sessions and CAS Engine Librefs” on page 479, but you can substitute any appropriately defined CAS engine libref.

The following statements execute the SVDD algorithm on the mycas.train data table and produce the results in Output 22.4.1 through Output 22.4.4:

    proc svdd data=mycas.train score nbins=10;
    id id x42;
    input x1-x41/level=interval;
    kernel rbf / bw=(5 to 20 by 1);
    savestate rstore=mycas.scorecat;
    run;

The SCORE option in the PROC SVDD statement requests that the training data table be scored. The NBINS=10 option creates a histogram data table that contains 10 bins. The INPUT statement defines the input variables x1 through x41 as interval variables. The KERNEL statement specifies the kernel function as a radial basis function (RBF), and the BW=(5 to 20 by 1) option instructs the SVDD procedure to train 16 SVDD models that correspond to bandwidth values from 5 to 20. The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in the mycas.scorecat data table. You can use the analytic store later in the ASTORE procedure to score new data.

Output 22.4.1 displays the number of observations in the training data table.

    Output 22.4.1  TE Process Analysis Training Observations
        Number of Observations Read   800
        Number of Observations Used   800

Because the SVDD procedure is executed with 16 bandwidth values, the “Model Information,” “Training Results,” “Optimization Summary,” and “Histogram” tables are generated for each bandwidth value. For the sake of brevity, only tables that correspond to two bandwidth values are provided here.

The “Model Information” table in Output 22.4.2 summarizes the key options and the input data variables for the SVDD model that is trained using the option BW=5.

    Output 22.4.2  TE Process Analysis Model Information (MODELID=1, BW=5)

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
</tr>
<tr>
<td>Kernel Type</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth</td>
</tr>
<tr>
<td>Bandwidth Selection Method</td>
</tr>
<tr>
<td>Bandwidth Relative Scale</td>
</tr>
<tr>
<td>Expected Outlier Fraction</td>
</tr>
<tr>
<td>Optimization Tolerance</td>
</tr>
<tr>
<td>Number of Interval Variables</td>
</tr>
<tr>
<td>Number of Nominal Variables</td>
</tr>
</tbody>
</table>
The “Training Results” table in Output 22.4.3 shows the number of support vectors and the $R^2$ threshold value for the SVDD model that is trained using the option BW=5.

### Output 22.4.3  TE Process Analysis Training Results (MODELID=1, BW=5)

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary</td>
</tr>
<tr>
<td>Number of Outliers</td>
</tr>
<tr>
<td>Number of Dropped Observations</td>
</tr>
<tr>
<td>Threshold R Square Value</td>
</tr>
<tr>
<td>Constant (C_r) Value</td>
</tr>
<tr>
<td>Run Time (seconds)</td>
</tr>
</tbody>
</table>

The “Optimization Summary” table in Output 22.4.4 shows whether the solution is optimal, the number of iterations that were required, and the objective function value for the SVDD model that is trained using the option BW=5.

### Output 22.4.4  TE Process Analysis Optimization Summary (MODELID=1, BW=5)

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Objective Value</td>
</tr>
<tr>
<td>Infeasibility</td>
</tr>
<tr>
<td>Optimization Status</td>
</tr>
<tr>
<td>Degenerate</td>
</tr>
</tbody>
</table>

The “Histogram” table in Output 22.4.5 shows whether the solution is optimal, the number of iterations that were required, and the objective function value for the SVDD model that is trained using the option BW=5.

### Output 22.4.5  TE Process Analysis Histogram (MODELID=1, BW=5)

<table>
<thead>
<tr>
<th>Frequency Distribution for SVDD Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin Midpoint</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>0.983408</td>
</tr>
<tr>
<td>0.983901</td>
</tr>
<tr>
<td>0.984393</td>
</tr>
<tr>
<td>0.984885</td>
</tr>
<tr>
<td>0.985377</td>
</tr>
<tr>
<td>0.985870</td>
</tr>
<tr>
<td>0.986362</td>
</tr>
<tr>
<td>0.986854</td>
</tr>
<tr>
<td>0.987346</td>
</tr>
<tr>
<td>0.987839</td>
</tr>
</tbody>
</table>

The “Model Information” table in Output 22.4.6 summarizes the key options and the input data variables for the SVDD model that is trained using the option BW=20.
Output 22.4.6  TE Process Analysis Model Information (MODELID=16, BW=20)

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method: Active Set</td>
</tr>
<tr>
<td>Kernel Type: RBF</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth: 20</td>
</tr>
<tr>
<td>Bandwidth Selection Method: User specified</td>
</tr>
<tr>
<td>Bandwidth Relative Scale: 2.8970656673</td>
</tr>
<tr>
<td>Expected Outlier Fraction: 1E-6</td>
</tr>
<tr>
<td>Optimization Tolerance: 0.0001</td>
</tr>
<tr>
<td>Number of Interval Variables: 41</td>
</tr>
<tr>
<td>Number of Nominal Variables: 0</td>
</tr>
</tbody>
</table>

The “Training Results” table in Output 22.4.7 shows the number of support vectors and the $R^2$ threshold value for the SVDD model that is trained using the option BW=20.

Output 22.4.7  TE Process Analysis Training Results (MODELID=16, BW=20)

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors: 35</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary: 35</td>
</tr>
<tr>
<td>Number of Outliers: 2</td>
</tr>
<tr>
<td>Number of Dropped Observations: 0</td>
</tr>
<tr>
<td>Threshold R Square Value: 0.88994</td>
</tr>
<tr>
<td>Constant ($C_r$) Value: 0.11006</td>
</tr>
<tr>
<td>Run Time (seconds): 0.01266</td>
</tr>
</tbody>
</table>

The “Optimization Summary” table in Output 22.4.8 shows whether the solution is optimal, the number of iterations that were required, and the objective function value for the SVDD model that is trained using the option BW=5.

Output 22.4.8  TE Process Analysis Optimization Summary (MODELID=16, BW=20)

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations: 1</td>
</tr>
<tr>
<td>Objective Value: 0.1100561016</td>
</tr>
<tr>
<td>Infeasibility: 0.0000785355</td>
</tr>
<tr>
<td>Optimization Status: Optimal</td>
</tr>
<tr>
<td>Degenerate: No</td>
</tr>
</tbody>
</table>

The “Histogram” table in Output 22.4.9 shows whether the solution is optimal, the number of iterations that were required, and the objective function value for the SVDD model that is trained using the option BW=5.
**Output 22.4.9** TE Process Analysis Histogram (MODELID=16, BW=20)

<table>
<thead>
<tr>
<th>Bin Midpoint</th>
<th>Count</th>
<th>Percent (%)</th>
<th>Cumulative (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.869585</td>
<td>2</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>0.871744</td>
<td>2</td>
<td>0.25</td>
<td>0.50</td>
</tr>
<tr>
<td>0.873904</td>
<td>7</td>
<td>0.88</td>
<td>1.38</td>
</tr>
<tr>
<td>0.876063</td>
<td>13</td>
<td>1.63</td>
<td>3.00</td>
</tr>
<tr>
<td>0.878223</td>
<td>25</td>
<td>3.13</td>
<td>6.13</td>
</tr>
<tr>
<td>0.880383</td>
<td>18</td>
<td>2.25</td>
<td>8.38</td>
</tr>
<tr>
<td>0.882542</td>
<td>43</td>
<td>5.38</td>
<td>13.75</td>
</tr>
<tr>
<td>0.884702</td>
<td>163</td>
<td>20.38</td>
<td>34.13</td>
</tr>
<tr>
<td>0.886862</td>
<td>313</td>
<td>39.13</td>
<td>73.25</td>
</tr>
<tr>
<td>0.889021</td>
<td>214</td>
<td>26.75</td>
<td>100.00</td>
</tr>
</tbody>
</table>

The following PROC ASTORE code scores the mycas.score data table by using the score analytic store that was created during training:

```plaintext
proc astore;
  score data=mycas.score
  out=mycas.score_results
  rstore=mycas.scorecat;
quit;
```

The following code copies the score_results into the Work library:

```plaintext
data work.score_results;
  set mycas.score_results;
run;
```

The score_results table has the following columns:

- Variables ID and X42, which are specified using the COPYVARS option
- Variables BW_i, where i = 1 to 16. These indicate the bandwidth values that the SVDD model uses.
- Variables _SVDDDISTANCE_i, where i = 1 to 16. These indicate the value of _SVDDDISTANCE_ that are obtained by scoring against the ith model.
- Variables _SVDDSCORE_i, where i = 1 to 16. These indicate the value of _SVDDSCORE_ that is obtained by scoring against the ith model.

The SVDDScore values in the score_results data table, which correspond to different models, can be used to develop a consensus-based strategy to determine the degree to which observations in the scoring data table are outliers or inliers. The proportion of models that designate an observation as an outlier or inlier can be used to determine this degree.

The following DATA step reads the score_results data table and computes the proportion of models that classify an observation as an outlier or an inlier. This DATA step creates two data sets: score_results_normal, which contains observations that belong to normal operating mode, and score_results_fault, which contains observations that belonging to faulty operating mode.
data score_results_normal score_results_fault;
set score_results;
label p_inside="Proportion Inside" p_outside="Proportion Outside";
array SVDDSCORE{16} _SVDDSCORE_1 - _SVDDSCORE_16 ;
inside=0;
outside=0;
do j=1 to 16;
   if SVDDSCORE[j]=-1 then inside=inside+1;
   if SVDDSCORE[j]=1 then outside=outside+1;
end;
p_inside=inside/16;
p_outside=outside/16;
if x42=0 then output score_results_normal;
else output score_results_fault;
run;

The following PROC SORT code sorts the score_results_normal data in ascending order of the p_inside column. Observations whose p_inside values are higher can be treated as more likely to be inliers than observations whose p_inside values are lower.

    proc sort data=score_results_normal;
    by descending p_inside;
    run;

The following PROC SORT code sorts the score_results_fault data in ascending order of the p_outside column. Observations whose p_outside values are higher can be treated as more likely to be outliers than observations whose p_outside values are lower.

    proc sort data=score_results_fault;
    by descending p_outside;
    run;

This example illustrates how to use multiple bandwidth values and a consensus-based methodology to classify observations in the scoring data table. In such analysis, the bandwidth values can be selected in the neighborhood of the bandwidth value that is obtained using either the mean criterion or the modified mean criterion.

---

Example 22.5: Analysis of Chessboard Data Using the Trace Criterion and the Fast Incremental Solver

This example illustrates the use of the trace criterion method to calculate bandwidth. The chessboard data that are used in this example represent 64 squares arranged in an 8 x 8 grid. The colors of the 64 squares alternate between light and dark. Each observation in the chessboard data consists of three variables: the variables x and y provide the coordinates of a point, and the variable label provides the color. The training data contain observations that correspond to the dark squares, and the scoring data contain observations that correspond to both the dark and the light squares.

The following DATA step creates the chessboard data set (which is stored in the Work library by default):
Example 22.5: Analysis of Chessboard Data

```plaintext
data chess;
  length label $ 5;
  input x y label $;
  datalines;
  0 0 black
  0.01 0 black
  ... more lines ... 
```

The following code plots the chessboard data:

```plaintext
proc sgplot data=chess;
  styleattrs datacontrastcolors=(black ligr ) datasymbols=(circlefilled);
  title "Chessboard Data Analysis";
  scatter x=x y=y/group=label markerattrs=(size=3);
run;
```

Output 22.5.1 shows the results.

**Output 22.5.1** Chessboard Data

![Chessboard Data Analysis](image)

The following code creates the training and scoring data sets. The ID variable is created to uniquely identify an observation. The training data set consists of 5,008 observations that belong to the dark squares. The scoring data set contains 5,008 observations that belong to dark squares and 4,992 observations that belong to the light squares.

```plaintext
data chess;
  set chess;
  id=_n_; 
  run;

  data train score;
    set chess;
    if label="black" then output train;
    output score;
  run;
```
You can load the `Work.train` and `Work.score` data sets into your CAS session by specifying your CAS engine libref (mycas) and data set names (train and score) in the first statement in each of the following DATA steps:

```plaintext
data mycas.train;
set train;
runk;

data mycas.score;
set score;
runk;
```

These statements assume that your CAS engine libref is named mycas, as in the section “Using CAS Sessions and CAS Engine Librefs” on page 479, but you can substitute any appropriately defined CAS engine libref.

The following statements execute the SVDD algorithm on the mycas.train data table and produce the results in Output 22.5.2 through Output 22.5.5:

```plaintext
proc svdd data=mycas.train;
id x y id;
input x y/level=interval;
kernar rbf / bw=trace(nrep=32);
solver fastinc;
savestate rstore=mycas.scorecat;
runk;
```

The INPUT statement defines the input variables `x` and `y` as interval variables. The KERNEL statement specifies the kernel function as a radial basis function (RBF), and the `BW=TRACE(NREP=32)` option instructs the SVDD procedure to compute the bandwidth value by using the trace criterion method with 32 representative points. Here 32 corresponds to the number of black squares in the training data. The SOLVER statement specifies FASTINC (fast incremental solver) as the SVDD solver. The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in the mycas.scorecat data table. You can use the analytic store later in the ASTORE procedure to score new data.

The “Model Information” table in Output 22.5.2 summarizes the key options and the input data variables.

**Output 22.5.2** Chessboard Data Analysis Model Information

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
</tr>
<tr>
<td>Kernel Type</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth</td>
</tr>
<tr>
<td>Bandwidth Selection Method</td>
</tr>
<tr>
<td>Bandwidth Relative Scale</td>
</tr>
<tr>
<td>Outlier Threshold</td>
</tr>
<tr>
<td>Optimization Tolerance</td>
</tr>
<tr>
<td>Number of Interval Variables</td>
</tr>
<tr>
<td>Number of Nominal Variables</td>
</tr>
</tbody>
</table>

The “Training Results” table in Output 22.5.3 shows the number of support vectors and the $R^2$ threshold value.
Output 22.5.3  Chessboard Data Analysis Training Results

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary</td>
</tr>
<tr>
<td>Number of Dropped Observations</td>
</tr>
<tr>
<td>Threshold R Square Value</td>
</tr>
<tr>
<td>Constant (C_r) Value</td>
</tr>
<tr>
<td>Run Time (seconds)</td>
</tr>
<tr>
<td>Bandwidth Calculation Time (seconds)</td>
</tr>
</tbody>
</table>

Output 22.5.4 displays the number of observations in the training data set.

Output 22.5.4  Chessboard Data Analysis Training Observations

| Number of Observations Read         | 5008 |
| Number of Observations Used         | 5008 |

The “Optimization Summary” table in Output 22.5.5 shows whether the solution is optimal, the number of iterations that were required, and the objective function value.

Output 22.5.5  Chessboard Data Analysis Optimization Summary

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective Value</td>
</tr>
<tr>
<td>Infeasibility</td>
</tr>
<tr>
<td>Optimization Status</td>
</tr>
<tr>
<td>Degenerate</td>
</tr>
</tbody>
</table>

The following PROC ASTORE code scores the mycas.score data table by using the score analytic store that was created during training:

```sas
proc astore;
   score data=mycas.score
       out=mycas.score_results
       rstore=mycas.scorecat;
quit;
```

The following code copies the score_results into the Work library:

```sas
data work.score_results;
set mycas.score_results;
run;
```

The following code plots the scoring results:

```sas
proc sgplot data=score_results(where=(_svddscore_ in (-1,1)));
   styleattrs datacontrastcolors=(black ligr) datasymbols=(circlefilled);
   title "Chessboard Data Analysis";
   title1 "Scoring Results";
   scatter x=x y=y/group=_svddscore_ markerattrs=(size=3);
run;
```
Chapter 22: The SVDD Procedure

Output 22.5.6 Chessboard Data: Scoring Results

Output 22.5.6 shows the scatter plot of the scoring data set. The gray area indicates the observations that are identified as outliers, and the black area indicates observations that are identified as inliers. In this example, the number of clusters in the training data set was known to be 32, which corresponded to the number of black squares. Accordingly, the NREP= option was set to 32. The results indicate that the bandwidth value that is obtained using the trace criterion method can provide a good description of the chessboard data.

Example 22.6: Bandwidth Calculation Using the Peak Criterion Method

This example illustrates the peak criterion method (Kakde et al. 2017) for bandwidth selection. The peak criterion method suggests the values of Gaussian bandwidth $s$, where the second derivative of the value of the optimal dual objective function (as defined in the section “Flexible Data Description” on page 492) with respect to $s$ first reaches 0. The bandwidth value that is obtained using the peak criterion method provides a data boundary that closely follows the data shape. For certain data sets, the method provides a range of values where this criterion is met. Any value of $s$ within this range provides a good data boundary.

The peak criterion method involves computing the optimal dual objective function value over a range of bandwidth values. These computations can be performed by specifying multiple bandwidth parameter values in the form of BW=n TO m BY p in the KERNEL statement in order to train multiple SVDD models in a single run.

The training data set in this example is star-shaped. This example first computes the bandwidth value by using the peak criterion method. The bandwidth value thus obtained is subsequently used to train an SVDD model. To evaluate the quality of the training results, scoring is performed on a $200 \times 200$ data grid.

The following DATA step creates the star (which is stored in the Work library by default):

```plaintext
data star;
infile datalines delimiter=',';
input x1 x2 ;
datalines;
0.07197, 6.48334
0.28467, 6.35686
```

```
Example 22.6: Bandwidth Calculation using Peak Criterion Method

0.45698, 6.35741
0.57731, 6.13851

... more lines ...

The following statements plot variable x1 against variable x2:

```latex
proc sgplot data=star;
   title "Star-shaped Training Data";
   scatter x=x1 y=x2/markerattrs=(size=3 symbol=circlefilled);
run;
```

Output 22.6.1 shows the results.

**Output 22.6.1 Star-Shaped Training Data**

You can load the `work.star` data set into your CAS session by specifying your CAS engine libref in the following DATA step:

```latex
data mycas.star;
   set star;
run;
```

These statements assume that your CAS engine libref is named `mycas`, as in the section “Using CAS Sessions and CAS Engine Librefs” on page 479, but you can substitute any appropriately defined CAS engine libref.

The following statements execute the SVDD algorithm on the `mycas.star` data table. The optSummary and modelInfo ODS data sets are stored in the `optimization_summary` and `model_info` data sets, respectively.

```latex
ods output optSummary=optimization_summary modelinfo=model_info;
proc svdd data=mycas.star;
   id x1 x2;
   input x1-x2/level=interval;
   kernel rbf / bw=(0.1 to 5 by 0.1);
   solver actset/stol=1e-10;
   savestate rstore=mycas.scorecat;
run;
```
The INPUT statement defines the input variables \(x_1\) and \(x_2\) as interval variables. The KERNEL statement specifies the kernel function as a radial basis function (RBF), and the BW=(0.1 to 5 by 0.1) option instructs the SVDD procedure to train 50 SVDD models that correspond to bandwidth values from 0.1 to 5. The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in the mycas.scorecat data table.

Output 22.6.2 displays the number of observations in the training data table.

**Output 22.6.2**  Peak Criterion Method: Star-Shaped Data

<table>
<thead>
<tr>
<th>Number of Observations Read</th>
<th>582</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Used</td>
<td>582</td>
</tr>
</tbody>
</table>

The SVDD procedure is executed with 50 bandwidth values, and it generates the “Model Information,” “Training Results,” “Optimization Summary,” and “Histogram” tables for each bandwidth value. For the sake of brevity, only tables that correspond to one bandwidth value are provided here.

The “Model Information” table in **Output 22.6.3** summarizes the key options and the input data variables for the SVDD model that is trained using the option BW=0.1.

**Output 22.6.3**  Peak Criterion Method: Star-Shaped Data (MODELID=1, BW=0.1)

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
</tr>
<tr>
<td>Kernel Type</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth</td>
</tr>
<tr>
<td>Bandwidth Selection Method</td>
</tr>
<tr>
<td>Bandwidth Relative Scale</td>
</tr>
<tr>
<td>Expected Outlier Fraction</td>
</tr>
<tr>
<td>Optimization Tolerance</td>
</tr>
<tr>
<td>Number of Interval Variables</td>
</tr>
<tr>
<td>Number of Nominal Variables</td>
</tr>
</tbody>
</table>

The “Training Results” table in **Output 22.6.4** shows the number of support vectors and the \(R^2\) threshold value for the SVDD model that is trained using the option BW=0.1.

**Output 22.6.4**  Peak Criterion Method: Star-Shaped Data (MODELID=1, BW=0.1)

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary</td>
</tr>
<tr>
<td>Number of Dropped Observations</td>
</tr>
<tr>
<td>Threshold R Square Value</td>
</tr>
<tr>
<td>Constant (C_r) Value</td>
</tr>
<tr>
<td>Run Time (seconds)</td>
</tr>
</tbody>
</table>

The “Optimization Summary” table in **Output 22.6.5** shows whether the solution is optimal, the number of iterations that were required, and the objective function value for the SVDD model that is trained using the option BW=0.1.
Example 22.6: Bandwidth Calculation using Peak Criterion Method

Output 22.6.5  Peak Criterion Method: Star-Shaped Data (MODELID=1, BW=0.1)

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Objective Value</td>
</tr>
<tr>
<td>Infeasibility</td>
</tr>
<tr>
<td>Optimization Status</td>
</tr>
<tr>
<td>Degenerate</td>
</tr>
</tbody>
</table>

The following code uses the optimization_summary and model_info data sets to obtain the optimal objective function and bandwidth values, respectively, for each model. Subsequently, these two data sets are merged to create one data set named peak_input.

```sas
data model_info(keep=model nvalue rename=(nvalue=bw));
set model_info (where=(rowid="BW"));
run;
data optimization_summary(keep=model value rename=(value=obj));
set optimization_summary (where=(rowid="OBJ"));
run;
proc sort data=model_info; by model; run;
proc sort data=optimization_summary; by model; run;
data peak_input;
merge model_info(in=a) optimization_summary(in=b);
by model;
if a=b;
run;
```

The following code computes the first and second derivative of the optimal objective function value with respect to the bandwidth value $s$. The sby macro variable specifies the bandwidth step size of 0.1, as used earlier in the SVDD procedure.

```sas
%let sby=0.1;
data peak_input;
set peak_input;
lag_obj=lag(obj);
lag2_obj=lag2(obj);
run;

data peak_input;
set peak_input;
if _n_ > 1 then d_obj = (obj - lag_obj) / (&sby);
if _n_ > 2 then d2_obj = (obj - 2 * lag_obj + lag2_obj) / ( &sby * &sby ) ;
run;
```

To decide whether the value of the second derivative is 0, a penalized B-spline is fitted to the second derivative by using the TRANSREG procedure in SAS/STAT software. If the 95% confidence interval of the fitted value of second derivative contains 0, the second derivative value is considered to be approximately 0.

```sas
proc transreg data=peak_input;
model identity(d2_obj) = pbspline(bw / nknots=20 evenly sbc);
output out=peak_res predicted cli clm;
run;
```
The following code determines the bandwidth value where the second derivative of the optimal objective function value with respect to \( s \) first reaches 0:

```plaintext
data peak_res;
set peak_res(where=(d2_obj ne .));
if ((CIUd2_obj ge 0) and (CILd2_obj le 0)) then zero_flg=1;
else zero_flg=0;
label d2_obj = "Second Derivative";
run;

proc sql noprint;
select min(bw) into :peak_bw
from peak_res
where zero_flg=1;
quit;

data peak_res;
set peak_res;
if round(bw,0.0001)= &peak_bw then do;
ss=bw;
label="Bandwidth="||trim(left(bw));
end;
run;
```

The following code plots the second derivative of the optimal objective function value against bandwidth value \( s \):

```plaintext
proc template;
define statgraph bandplot;
begingraph;
entrytitle " ";
layout overlay;
bandplot x=bw limitupper=CIUd2_obj
limitlower=CILd2_obj / name="band1" modelname="fit"
legendlabel="95% Confidence Limits";
scatterplot x=bw y=d2_obj / primary=true;
seriesplot x=bw y=Pd2_obj / name="fit"
legendlabel="Fit Line" lineattrs=(thickness=2);
referenceline x=ss / curvelabel=label
lineattrs=(color=gray pattern=dot thickness=1);
referenceline y=0 /lineattrs=(color=gray pattern=solid );
discretelegend "fit" "band1" ;
endlayout;
endgraph;
end;
run;
```

```plaintext
proc sgrender data=peak_res(where=(d2_obj ne .)) template=bandplot;
run;
```

Output 22.6.6 shows the results, which indicate that the value of second derivative of the optimal objective function value with respect to \( s \) first reaches 0 at \( s = 0.9 \). This represents the bandwidth value that is obtained using the peak criterion method.
Example 22.6: Bandwidth Calculation using Peak Criterion Method

**Output 22.6.6** Peak Criterion: Plot of the 2nd Derivative

The following statements execute the SVDD algorithm on the mycas.star data table by using the bandwidth value (0.9) that was obtained using the peak criterion method.

```plaintext
proc svdd data=mycas.star;
  id x1 x2;
  input x1 x2/level=interval;
  kernel rbf / bw=0.9;
  savestate rstore=mycas.state;
run;
```

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:

```plaintext
data scoreds;
  do x1=0 to 10 by 0.05;
    do x2=0 to 10 by 0.05;
      output;
    end;
  end;
run;
```

You can load the scoreds data set into your CAS session by specifying your CAS engine libref in the following DATA step:

```plaintext
data mycas.scoreds;
  set scoreds;
run;
```

The following code scores the mycas.scoreds data by using the score analytic store that was created during training:

```plaintext
proc astore;
  score data=mycas.scoreds
    out=mycas.score_results
    rstore=mycas.state;
```
Chapter 22: The SVDD Procedure

quit;
The following code copies the `score_results` into the Work library:

```sas
data work.score_results;
set mycas.score_results;
run;
```

The following code plots the scoring results:

```sas
proc sgplot data=score_results(where=(_svddscore_ in (-1,1)));
styleattrs datacontrastcolors=(ligr black) datasymbols=(circlefilled);
title "Peak Criterion: Scoring Results";
scatter x=x1 y=x2/group=_svddscore_ markerattrs=(size=3);
run;
```

Output 22.6.7 shows the scatter plot of the scoring data set. The gray area indicates the observations that are identified as outliers, and the black area indicates observations that are identified as inliers. Comparison between Output 22.6.7 and the training data in Output 22.6.1 indicates that the bandwidth that was obtained using the peak criterion method provides an accurate description of the training data.

As outlined in this example, determining the bandwidth that is suggested by the peak criterion method requires that the SVDD solution be computed multiple times for the training data for a list of bandwidth values that lie on a grid. The parameters of the grid such as the starting value, ending value, and increment can affect the quality of results. Also, it is necessary to set the solver tolerance, STOL, to a very low value (such as the value 1E–10 that is used in this example).

![Output 22.6.7 Peak Criterion: Scoring Results](image)

References


### Chapter 23
The SVMACHINE Procedure

#### Contents

**Overview: SVMACHINE Procedure** ................................. 546

**PROC SVMACHINE Features** ........................................ 546

**Using CAS Sessions and CAS Engine Librefs** ...................... 546

**Getting Started: SVMACHINE Procedure** .......................... 547

**Syntax: SVMACHINE Procedure** ..................................... 549

**PROC SVMACHINE Statement** ........................................ 549

**AUTOTUNE Statement** ................................................ 551

**CODE Statement** ..................................................... 554

**ID Statement** ......................................................... 554

**INPUT Statement** ...................................................... 554

**KERNEL Statement** ................................................... 555

**OUTPUT Statement** ................................................... 555

**PARTITION Statement** ................................................ 556

**SAVSTATE Statement** ................................................ 557

**TARGET Statement** ................................................... 557

**Details: SVMACHINE Procedure** .................................... 558

**Support Vector Classification** ..................................... 558

**Support Vector Regression** ......................................... 559

**Interior-Point Method Optimization Technique** .................. 560

**Scoring Process** ..................................................... 560

**Displayed Output** .................................................... 561

**ODS Table Names** .................................................... 562

**Examples: SVMACHINE Procedure** .................................. 563

**Example 23.1: Home Equity Loan Case** ............................ 563

**Example 23.2: Large Simulated Data Table** ....................... 566

**Example 23.3: Support Vector Regression** ......................... 567

**References** .......................................................... 568
Overview: SVMACHINE Procedure

The SVMACHINE procedure implements support vector machines (SVM) algorithms in SAS Viya. The SVM algorithms include support vector classification (SVC) and support vector regression (SVR). As a popular data mining model, SVM algorithms compute support vector machine learning classifiers for the binary pattern recognition problem and regression problem; they have been broadly used in the fields such as image classification, handwriting recognition, financial decision, text mining, and so on.

Like other predictive modeling tools, the SVMACHINE procedure uses input data to train a model and provides information about the model. The SVMACHINE procedure executes an SVM algorithm (applying the interior-point optimization technique during training) and can generate SAS code for scoring future data. PROC SVMACHINE uses both linear and low-degree polynomial kernels to conduct computation, and it can run on multiple threads in a single machine or on multiple threads on multiple machines. It can load data from multiple nodes and perform computation in parallel.

PROC SVMACHINE Features

The SVMACHINE procedure has the following features:

- reads input data in parallel when the data source is on a distributed system
- is highly multithreaded during all phases of analytic execution
- supports large-scale training data
- supports both continuous and categorical inputs
- supports classification of a binary target
- supports regression of an interval target
- supports the interior-point method

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:
proc options option=(CASHOST CASPORT);
run;

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

    cas mysess;
    libname mycas cas sessref=mysess;

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

    cas mysess terminate;

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”

---

**Getting Started: SVMACHINE Procedure**

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

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.


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, as in the section “Using CAS Sessions and CAS Engine Librefs” on page 546, but you can substitute any appropriately defined CAS engine libref.

The following statements execute the SVM algorithm on the mycas.dmagecr data table and produce Figure 23.1 through Figure 23.3.
The first INPUT statement defines the input variables Checking, History, Purpose, Savings, Employed, Marital, Coapp, Property, Other, Job, Housing, Telephone, and Foreign as categorical variables. The second INPUT statement defines the input variables Duration, Amount, Installp, Resident, Existcr, Depends, and Age as continuous variables. The TARGET statement defines good_bad to be the target variable (the variable that is predicted). The “Training Results” table in Figure 23.1 shows that the inner product of weights is 11.6121718, the bias is –2.1296773, and the number of support vectors is 531, of which 481 are on the margin.

**Figure 23.1** German Credit Data Training Results

<table>
<thead>
<tr>
<th>Training Results</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Inner Product of Weights</td>
<td>11.6121718</td>
</tr>
<tr>
<td>Bias</td>
<td>-2.1296773</td>
</tr>
<tr>
<td>Total Slack (Constraint Violations)</td>
<td>492.87883</td>
</tr>
<tr>
<td>Norm of Longest Vector</td>
<td>4.17809329</td>
</tr>
<tr>
<td>Number of Support Vectors</td>
<td>531</td>
</tr>
<tr>
<td>Number of Support Vectors on Margin</td>
<td>481</td>
</tr>
<tr>
<td>Maximum F</td>
<td>2.57131793</td>
</tr>
<tr>
<td>Minimum F</td>
<td>-4.6513481</td>
</tr>
<tr>
<td>Number of Effects</td>
<td>20</td>
</tr>
<tr>
<td>Columns in Data Matrix</td>
<td>61</td>
</tr>
</tbody>
</table>

The “Misclassification Matrix” table in Figure 23.2 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 Figure 23.3.

**Figure 23.2** German Credit Misclassification Matrix

<table>
<thead>
<tr>
<th>Misclassification Matrix</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Prediction</td>
<td></td>
</tr>
<tr>
<td>Observed</td>
<td></td>
</tr>
<tr>
<td>good</td>
<td>bad</td>
</tr>
<tr>
<td>good</td>
<td>626</td>
</tr>
<tr>
<td>bad</td>
<td>142</td>
</tr>
<tr>
<td>Total</td>
<td>768</td>
</tr>
</tbody>
</table>
A relatively good model means that misclassification is low while both sensitivity and specificity are high. In PROC SVMACHINE, you can adjust training parameters and use different kernels to obtain a better model.

**Syntax: SVMACHINE Procedure**

The following statements are available in the SVMACHINE procedure:

```
PROC SVMACHINE < options > ;
  AUTOTUNE < options > ;
  CODE FILE=filename ;
  ID variables ;
  INPUT variables / < LEVEL=INTERVAL | NOMINAL > ;
  KERNEL kernel-type / < kernel-parameter > ;
  OUTPUT OUT=CAS-libref.data-table < option > ;
  PARTITION partition-option ;
  SAVESTATE RSTORE=CAS-libref.data-table ;
  TARGET variable < /options > ;
```

The PROC SVMACHINE statement, the TARGET statement, and at least one INPUT statement are required. The following sections describe the PROC SVMACHINE statement and then describe the other statements in alphabetical order.

**PROC SVMACHINE Statement**

```
PROC SVMACHINE < options > ;
```

The PROC SVMACHINE statement invokes the procedure.

You can specify the following **options**:

```
C=number
```

specifies the penalty value, where *number* must be a real number greater than 0.

By default, C=1.0.
names the input data table for PROC SVMACHINE to use. The default is the most recently created data table. \texttt{CAS-libref.data-table} is a two-level name, where

\texttt{CAS-libref} refers to a collection of information that is defined in the LIBNAME statement and includes the \texttt{caslib}, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about \texttt{CAS-libref}, see the section “Using CAS Sessions and CAS Engine Librefs” on page 546.

\texttt{data-table} specifies the name of the input data table.

\textbf{EARLYSTOP}

generates a model that is based on the validation accuracy. The iterations stop when the validation accuracy stops improving, and the model from the previous iteration is selected. The iteration accuracy from both the training data and the validation data is displayed in the “Iteration Report” table.

\textbf{NOTE:} The EARLYSTOP option is ignored and a warning message is issued if the PARTITION statement is omitted. This option is ignored for SVR training.

\textbf{ITERATIONREPORT}

generates a table named “Iteration Report,” which displays the training accuracy for each iteration. If you also specify the PARTITION statement and the validation data and testing data exist, then the “Iteration Report” table also displays the validation accuracy and testing accuracy, respectively, for each iteration. This option is ignored if you specify the AUTOTUNE statement or perform the SVR training.

\textbf{NOTE:} Generating the accuracy report for each iteration is computationally expensive and requires a significant amount of time.

\textbf{MAXITER=} \textit{number}

specifies the maximum number of iterations before the process stops, where \textit{number} is a positive integer.

By default, MAXITER=25. In some cases, you can obtain a good model in fewer than five iterations.

\textbf{NOPRINT}

suppresses the generation of ODS outputs. If you specify this option, no ODS tables are generated.

\textbf{NOSCALE}

uses the original data during training.

\textbf{NTHREADS=} \textit{number-of-threads}

specifies the number of threads that are used in the computation. The default value is the number of CPUs available in the machine.

\textbf{PRINTTARGET}

generates the table “Predicted Probability Variables,” which displays the target variable and the predicted probability variables, and the table “Predicted Target Variable,” which displays the predicted target variable.

By default, these two tables are not generated. For SVR, only the table “Predicted Target Variable,” is generated.
SCALE
scales the input variables to between 0 and 1 during training.
By default, for SVC, all numerical data are scaled before the training.
NOTE: For SVR, this option is ignored.

TOLERANCE=number
specifies the minimal absolute tolerance at which the iteration stops. The tolerance number must be
equal to or greater than 1.0E-12.
By default, TOLERANCE=1.0E-6. In some cases, you can obtain a good model with TOLERANCE
\geq 0.01.

USEMISS
uses missing values for input variables. Missing is treated as a special level for a categorical variable,
and missing values for a continuous variable are imputed to the mean before training.
By default, all observations that have missing values are dropped during the training process.

AUTOTUNE Statement

AUTOTUNE <options>;

The AUTOTUNE statement searches for the best combination of values of the C= option in the PROC
SVMACHINE statement and the DEGREE= suboption of the POLYNOMIAL option in the KERNEL
statement.

Table 23.1 summarizes the options you can specify in the AUTOTUNE statement. For more information
about all options except the TUNINGPARAMETERS= option, see the option’s description in the section
“AUTOTUNE Statement” on page 12 in Chapter 3, “Shared Concepts.” The TUNINGPARAMETERS option
is described following table Table 23.1.

NOTE: Processing the AUTOTUNE statement is computationally expensive and requires a significant amount
of time.

NOTE: If you specify both AUTOTUNE statement and OUTPUT statement, the PROC SVMACHINE exits
with error message.

NOTE: For SVR, the AUTOTUNE statement is ignored.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDLOOKUP</td>
<td>Specifies that the table specified in the HISTORYTABLE= option contain the rows from the table specified in the LOOKUPTABLE= option</td>
</tr>
<tr>
<td>EVALHISTORY=</td>
<td>Specifies how to report the evaluation history of the tuner</td>
</tr>
<tr>
<td>FRACTION=</td>
<td>Specifies the fraction of observations to use for validation</td>
</tr>
<tr>
<td>HISTORYTABLE=</td>
<td>Specifies the CAS table that contains the evaluation history</td>
</tr>
<tr>
<td>KFOLD=</td>
<td>Specifies the number of folds for k-fold cross validation</td>
</tr>
</tbody>
</table>
Table 23.1  continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIVEUPDATE</td>
<td>Specifies that the table specified in the HISTORYTABLE= option be updated at every evaluation</td>
</tr>
<tr>
<td>LOCALSEARCH</td>
<td>Enables local search optimization</td>
</tr>
<tr>
<td>LOOKUPTABLE=</td>
<td>Specifies the CAS table to use for evaluation lookup</td>
</tr>
<tr>
<td>MAXBAYES=</td>
<td>Specifies the maximum number of points in the kriging model</td>
</tr>
<tr>
<td>MAXEVALS=</td>
<td>Specifies the maximum number of evaluations</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>MAXTIME=</td>
<td>Specifies the maximum time for all iterations</td>
</tr>
<tr>
<td>MAXTRAINTIME=</td>
<td>Specifies the maximum time for a model training</td>
</tr>
<tr>
<td>NOGRIDSHUFFLE</td>
<td>Requests that the grid points not be shuffled</td>
</tr>
<tr>
<td>NSUBSESSIONWORKERS=</td>
<td>Specifies the number of workers in parallel sessions</td>
</tr>
<tr>
<td>OBJECTIVE=</td>
<td>Specifies the objective function</td>
</tr>
<tr>
<td>POPSIZE=</td>
<td>Specifies the population size when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>SAMPLESIZE=</td>
<td>Specifies the sample size when SEARCHMETHOD=LHS or SEARCHMETHOD=RANDOM</td>
</tr>
<tr>
<td>SEARCHMETHOD=</td>
<td>Specifies the search method that the optimizer uses</td>
</tr>
<tr>
<td>SECONDOBJECTIVE=</td>
<td>Specifies the second objective to use for tuning</td>
</tr>
<tr>
<td>SELECTINITPOINT</td>
<td>Specifies that the tuner select the best evaluation from the lookup table</td>
</tr>
<tr>
<td>TARGETEVENT=</td>
<td>Specifies the target event for ROC-based calculations</td>
</tr>
<tr>
<td>TRAINFRACTION=</td>
<td>Specifies the fraction of observations to use for training</td>
</tr>
<tr>
<td>TUNINGPARAMETERS=</td>
<td>Specifies the custom tuning parameters</td>
</tr>
<tr>
<td>USEPARAMETERS=</td>
<td>Specifies how to handle the TUNINGPARAMETERS= option</td>
</tr>
</tbody>
</table>

TUNINGPARAMETERS=(suboption | . . . | < suboption >)
TUNEPARMS=(suboption | . . . | < suboption >)

specifies which parameters to tune and which ranges to tune over. If USEPARAMETERS=STANDARD, this option is ignored.

You can specify one or more of the following suboptions:

C (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)

specifies the penalty values to be used when tuning the SVM model, where number or any value in value-list must be a real number greater than 0. For more information, see the C= option in the PROC SVMACHINE statement.

You can specify the following additional suboptions:
**LB=number**
specifies the minimum penalty number to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=1E-10.

**UB=number**
specifies the maximum penalty number to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=100.

**VALUES=value-list**
specifies a list of penalty numbers to consider during tuning, where value-list is a space-separated list of numbers greater than 0. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=number**
specifies the initial penalty value for the tuner to use.

By default, INIT=1.

**EXCLUDE**
excludes the penalty value from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**DEGREE (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**
specifies the degree that is used in a linear (when number=1) or polynomial (when number=2 or 3) kernel during training, where number and values in value-list can be 1, 2 or 3. For more information, see the POLYNOMIAL option in the KERNEL statement.

You can specify the following additional suboptions:

**LB=number**
specifies a lower bound of the degree to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=1.

**UB=number**
specifies an upper bound of the degree to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=3.

**VALUES=value-list**
specifies a list of values to consider for the degree in the kernel, where value-list is a space-separated list of numbers. If you specify this suboption, you cannot specify either the LB= or UB= suboption.
**Chapter 23: The SVMACHINE Procedure**

**INIT**=number

specifies the initial degree of the kernel in the SVM.

By default, INIT=1.

**EXCLUDE**

excludes the degree from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

---

**CODE Statement**

**CODE FILE**=filename ;

The CODE statement generates SAS DATA step code that mimics the computations that are done by the OUTPUT statement.

You must specify the following option:

**FILE**=filename

specifies the filename of the file to write the SAS score code to.

The CODE statement is optional.

---

**ID Statement**

**ID** variables ;

The ID statement lists one or more variables that are to be copied from the input data table to the output data tables that are specified in the OUT= option in the OUTPUT statement and the RSTORE= option in the SAVESTATE statement.

---

**INPUT Statement**

**INPUT** variables / <**LEVEL**=INTERVAL | NOMINAL> ;

The INPUT statement specifies the names of variables to be used in training. Only interval, binary, and nominal variables are accepted. If you want to use different options for different variables, you can specify multiple INPUT statements.

You can specify the following option after a slash (/):

**LEVEL**=INTERVAL | NOMINAL

specifies whether the specified input variables are continuous or categorical. You can specify the following values:

**INTERVAL**

specifies that the input variables are continuous.

**NOMINAL**

specifies that the input variables are categorical.
By default, LEVEL=INTERVAL for numeric variables and LEVEL=NOMINAL for categorical variables. Binary variables are considered to be categorical variables.

**KERNEL Statement**

```
KERNEL kernel-type / <kernel-parameter>;
```

The KERNEL statement specifies the type of kernel and any associated parameters to be used during training. You can specify the following **kernel-types**:

**LINEAR** uses a linear kernel during training. No **kernel-parameter** is needed. The kernel is defined as

\[ k(x_1, x_2) = \langle x_1, x_2 \rangle \]

where \( x_1 \) and \( x_2 \) are two vectors and \( \langle, \rangle \) is the inner product.

**POLYNOMIAL**

```
POLYNOMIAL < / DEGREE=number >
```

uses a polynomial kernel during training. Specify the polynomial degree as the **kernel-parameter**, in the form **DEGREE=number**, where **number** must be 2 or 3 (the default is 2). For example, specify KERNEL POLYNOMIAL / DEGREE=2. The kernel is defined as

\[ k(x_1, x_2) = (\langle x_1, x_2 \rangle + 1)^p \]

where \( p \) is the degree of the polynomial.

By default, the kernel type is LINEAR.

**OUTPUT Statement**

```
OUTPUT OUT=CAS-libref.data-table < option >;
```

The OUTPUT statement creates an output data table that contains the predicted values of the input data table. You must specify the following option:

**OUT=**

```
OUT=CAS-libref.data-table
```

names the output data table for PROC SVMACHINE to use. You must specify this option before any other options. **CAS-libref.data-table** is a two-level name, where

- **CAS-libref** refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about **CAS-libref**, see the section “Using CAS Sessions and CAS Engine Librefs” on page 546.

- **data-table** specifies the name of the output data table.

You can also specify the following **option**:
COPYVAR=variable
COPYVARS=(variables)
lists one or more variables from the input data table that are transferred to the output data table.

ROLE<=variable>
specifies a numeric variable that indicates the role that each observation plays in fitting the model. The default name is _ROLE_. Table 23.2 shows how this variable is interpreted for each observation.

Table 23.2  Role Interpretation

<table>
<thead>
<tr>
<th>Value</th>
<th>Observation Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>1</td>
<td>Training</td>
</tr>
<tr>
<td>2</td>
<td>Validation</td>
</tr>
<tr>
<td>3</td>
<td>Testing</td>
</tr>
</tbody>
</table>

If you do not partition the input data by specifying a PARTITION statement, then the role variable value is 1 for observations that are used in fitting the model and 0 for observations that have at least one missing or invalid value for the target or input variables.

PARTITION Statement

PARTITION partition-option ;

The PARTITION statement specifies how observations in the input data set are logically partitioned into disjoint subsets for model training, validation, and testing. For more information, see the section “Using Validation and Test Data” on page 21 in Chapter 3, “Shared Concepts.” Either you can designate a variable in the input data table and a set of formatted values of that variable to determine the role of each observation, or you can specify proportions to use for randomly assigning observations to each role.

You must specify exactly one of the following partition-options:

FRACTION(<TEST=fraction> <VALIDATE=fraction> <SEED=number>)
randomly assigns specified proportions of the observations in the input data table to the roles. You specify the proportions for testing and validation by using the TEST= and VALIDATE= suboptions. If you specify both the TEST= and VALIDATE= suboptions, then the sum of the specified fractions must be less than 1 and the remaining fraction of the observations are assigned to the training role. The SEED= option specifies an integer that is used to start the pseudorandom number generator for random partitioning of data for training, testing, and validation. If you do not specify SEED=number or if number is less than or equal to 0, the seed is generated by reading the time of day from the computer’s clock.
**SAVESTATE Statement**

```plaintext
SAVESTATE RSTORE=CAS-libref.data-table ;
```

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 4, “The ASTORE Procedure.”

You must specify the following option:

```plaintext
RSTORE=CAS-libref.data-table
```

specifies a data table in which to save the analytic store for the model. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the `caslib` and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the `DATA=` option and the section “Using CAS Sessions and CAS Engine Librefs” on page 546.

---

**TARGET Statement**

```plaintext
TARGET variable < /options > ;
```

The TARGET statement names the target variable whose values PROC SVMACHINE predicts. The target variable must be binary or interval, and must be different from the variables in the INPUT statement.

You can specify following options after a slash (/):

```plaintext
LEVEL=INTERVAL | NOMINAL
```

specifies whether the specified target is continuous or categorical. You can specify the following values:

```plaintext
INTERVAL    specifies that the target is continuous and performs SVR training.
NOMINAL      specifies that the target is categorical and performs SVC training.
```

By default, LEVEL=NOMINAL.

For SVC training, you can also specify either of the following options to define how to levelize the target values. By default, the target values are levelized in descending order.
ASC
ASCENDING
levelizes the target values in ascending order.

DESC
DESCENDING
levelizes the target values in descending order.

Details: SVMACHINE Procedure

PROC SVMACHINE uses linear or nonlinear kernels to compute support vector machine (SVM) learning classifiers for the binary pattern recognition problem and the regression problem. The binary pattern recognition problem is called support vector classification (SVC), and the regression problem is called support vector regression (SVR). For more information about the theory and use of SVM learning, see Vapnik (1995); Burges (1998); Cristianini and Shawe-Taylor (2000); Smola and Schölkopf (2004).

Support Vector Classification

In the linear kernel case, support vector classification (SVC) computes the parameters $w$ and $\beta$ that define the model function,

$$ f(x) = w^T x + \beta $$

by solving the following quadratic optimization problem

$$ \begin{align*}
\text{minimize} & \quad \frac{1}{2} w^T w + Ce^T z \\
\text{subject to} & \quad DXw + \beta d \geq e - z \\
& \quad z \geq 0
\end{align*} $$

where $X$ denotes the $m \times n$ matrix whose rows correspond to the observations, $D$ 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

$$ \begin{align*}
\text{minimize} & \quad \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\
\text{subject to} & \quad d^T \alpha = 0 \\
& \quad 0 \leq \alpha \leq C
\end{align*} $$

where $Q = DXX^T D$ and $\alpha$ 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 $\alpha$ by the equation $w = X^T D \alpha$. Observations that correspond to nonzero entries in $\alpha$ are called support vectors. Observations that correspond to entries in $\alpha$ 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 the section “KERNEL Statement” on page 555. The corresponding model function is defined in terms of \( \beta, \alpha \), 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 \( X \) corresponds to explicitly projected observations.

---

**Support Vector Regression**

For support vector regression (SVR), the primal problem is defined as

\[
\begin{align*}
  \text{minimize} & & \frac{1}{2} w^T w + C e^T (z + z^*) \\
  \text{subject to} & & d - Xw - \beta e \leq \varepsilon e + z \\
  & & Xw + \beta e - d \leq \varepsilon e + z^* \\
  & & z, z^* \geq 0
\end{align*}
\]

where \( X \) denotes the \( m \times n \) matrix as defined in SVC and \( z \) and \( z^* \) are slack variables.

The corresponding dual optimization problem becomes

\[
\begin{align*}
  \text{minimize} & & \frac{1}{2} (\alpha - \alpha^*)^T X X^T (\alpha - \alpha^*) + \varepsilon e^T (\alpha + \alpha^*) - d^T (\alpha - \alpha^*) \\
  \text{subject to} & & e^T (\alpha - \alpha^*) = 0 \\
  & & 0 \leq \alpha, \alpha^* \leq C e
\end{align*}
\]

where \( \alpha \) and \( \alpha^* \) are the Lagrange multipliers.

With the same model function \( f(x) = w^T x + \beta \), the parameter \( w \) in the primal problem can be calculated as

\[
  w = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) x_i = X^T (\alpha - \alpha^*)
\]
Chapter 23: The SVMACHINE Procedure

**Interior-Point Method Optimization Technique**

A popular approach for accurately obtaining the global solution for an SVM optimization problem is the interior-point method. Interior-point methods are attractive in that the required number of iterations is relatively small and does not grow dramatically with problem size. The cost per iteration for interior-point methods can be quite high for large-scale problems unless there exists an underlying structure that can be exploited. For example, interior-point methods can be extremely efficient for problems where the number of variables are small in comparison to the number of constraints. When both the number of variables and number of constraints is large, interior-point methods become intractable for dense problems.

PROC SVMACHINE applies a primal-dual interior-point method to linear and polynomial kernels of degree 2 and 3. In the polynomial kernel case, $X$ is obtained by explicitly projecting each observation in the design matrix. The resulting number of columns is given by the binomial coefficient

$$\binom{n + p}{p} = \frac{(n + p)!}{p!n!}$$

where $n$ denotes the number of columns in the levelized design matrix and $p$ denotes the polynomial degree. Thus, KERNEL POLYNOMIAL/DEGREE=2 is not recommended when the number of columns in the design matrix is much greater than 100. Similarly, KERNEL POLYNOMIAL/DEGREE=3 is not recommended when the number of columns in the design matrix is much greater than 32.

Primal-dual interior-point methods perturb the optimality conditions in order to create a system of nonlinear equations that satisfy the requirements of Newton’s method. This perturbed system of nonlinear equations has the property that an interior solution (with respect to the inequality constraints) always exists; safeguards are then wrapped around Newton’s method to ensure that the interior (and hence feasible) approximate solution is obtained. Interior-point methods have the additional property that as the perturbation term goes to 0, the approximate solution converges to the true solution.

The dominant cost for each iteration comes from the need to solve a system of linear equations of size $(m+n) \times (m+n)$. Because PROC SVMACHINE assumes that $m$ might be very large, it uses block reduction strategies similar to those described in Gertz and Griffin (2005, 2010) to reduce the size of this system to a matrix of size $n \times n$, where $n$ denotes the number of columns in the design (or projected kernel) matrix. Then it performs dense matrix factorization on the resulting system. For problems where $m \gg n$, the dominant computational cost occurs during the block-row reduction step. To reduce the solution time, this operation is both distributed and threaded.

**Scoring Process**

The scoring process for the interior-point method is straightforward. As long as the training weight parameters and bias are known, the scoring process is just a linear combination. The event or nonevent is decided by the decision function. If the decision function is less than or equal to 0, then the prediction is an event; otherwise, it is a nonevent.

For the interior-point method, the score code is provided for a linear kernel and for a polynomial kernel of degree 2 and 3.
Displayed Output

The following sections describe the output that PROC SVMACHINE 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 contains the model information. The information includes but is not limited to the inner product of weights, bias, maximum and minimum F (the maximum and minimum values, respectively, of the decision function), and 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= option (which specifies the minimal absolute tolerance at which an iteration stops) and the MAXITER= option (which controls the number of iterations).

**NOTE:** For SVR, the “Iteration History” table is not generated.

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.

**NOTE:** For SVR, the “Misclassification Matrix” table is not generated.

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.

**NOTE:** For SVR, the “Fit Statistics” table is not generated.
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.

**NOTE:** For SVR, the “Iteration Report” table is not generated.

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 $P_\_$. 

**NOTE:** For SVR, the “Predicted Probability Variables” table is not generated.

Predicted Target Variable

The “Predicted Target Variable” table contains the predicted target variable. For SVC, the name begins with $I_\_$. For SVR, the name begins with $P_\_$. 

ODS Table Names

Each table that the SVMACHINE procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. The name of each table and a short description of the contents are listed in Table 23.3.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FitStatistics</td>
<td>Accuracy information about the training</td>
<td>PROC SVMACHINE</td>
<td>Default</td>
</tr>
<tr>
<td>IterationReport</td>
<td>Accuracy report for each iteration</td>
<td>PROC SVMACHINE</td>
<td>ITERATIONREPORT</td>
</tr>
<tr>
<td>IterHistory</td>
<td>Iteration history</td>
<td>PROC SVMACHINE</td>
<td>Default</td>
</tr>
<tr>
<td>Misclassification</td>
<td>Misclassification matrix table</td>
<td>PROC SVMACHINE</td>
<td>Default</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Basic model information for the training</td>
<td>PROC SVMACHINE</td>
<td>Default</td>
</tr>
<tr>
<td>NObs</td>
<td>Observation information about the input data</td>
<td>PROC SVMACHINE</td>
<td>Default</td>
</tr>
<tr>
<td>PredIntoName</td>
<td>Predicted target variable</td>
<td>PROC SVMACHINE</td>
<td>PRINTTARGET</td>
</tr>
<tr>
<td>PredProbName</td>
<td>Predicted probability variables</td>
<td>PROC SVMACHINE</td>
<td>PRINTTARGET</td>
</tr>
<tr>
<td>TrainingResult</td>
<td>Displays the training results</td>
<td>PROC SVMACHINE</td>
<td>Default</td>
</tr>
</tbody>
</table>
Examples: SVMACHINE Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

Example 23.1: Home Equity Loan Case

This example shows how you can use PROC SVMACHINE to create scoring code that can be used to score future home equity loan applications. The data set Hmeq, which is in the Sampsio library that SAS provides, contains observations for 5,960 mortgage applicants. A variable named Bad indicates whether the customer has paid on the loan or has defaulted on it. Table 23.4 describes the variables in Hmeq.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Role</th>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad</td>
<td>Response</td>
<td>Binary</td>
<td>1 = customer defaulted on the loan or is seriously delinquent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = customer is current on loan payments</td>
</tr>
<tr>
<td>CLAge</td>
<td>Predictor</td>
<td>Interval</td>
<td>Age of oldest credit line in months</td>
</tr>
<tr>
<td>CLNo</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of credit lines</td>
</tr>
<tr>
<td>DebtInc</td>
<td>Predictor</td>
<td>Interval</td>
<td>Debt-to-income ratio</td>
</tr>
<tr>
<td>Delinq</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of delinquent credit lines</td>
</tr>
<tr>
<td>Derog</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of major derogatory reports</td>
</tr>
<tr>
<td>Job</td>
<td>Predictor</td>
<td>Nominal</td>
<td>Occupational category</td>
</tr>
<tr>
<td>Loan</td>
<td>Predictor</td>
<td>Interval</td>
<td>Requested loan amount</td>
</tr>
<tr>
<td>MortDue</td>
<td>Predictor</td>
<td>Interval</td>
<td>Amount due on existing mortgage</td>
</tr>
<tr>
<td>nInq</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of recent credit inquiries</td>
</tr>
<tr>
<td>Reason</td>
<td>Predictor</td>
<td>Binary</td>
<td>'DebtCon' = debt consolidation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'HomeImp' = home improvement</td>
</tr>
<tr>
<td>Value</td>
<td>Predictor</td>
<td>Interval</td>
<td>Value of current property</td>
</tr>
<tr>
<td>YoJ</td>
<td>Predictor</td>
<td>Interval</td>
<td>Years at present job</td>
</tr>
</tbody>
</table>

You can load the Hmeq data set into your CAS session by specifying your CAS engine libref in the second statement in the following DATA step:

```sas
data mycas.hmeq;
  set sampsio.hmeq;
run;
```

The following statements execute the SVM algorithm on the mycas.hmeq data table:

```sas
filename codefile temp;
proc svmachine data=mycas.hmeq;
  input reason job derog delinq ninq / level=nominal;
  input loan mortdue value yoj clage clno debtinc / level=interval;
```
The first INPUT statement defines the input variables Reason, Job, Derog, Delinq, and Ninq as categorical variables. The second INPUT statement defines the input variables Loan, MortDue, Value, YoJ, CLAge, CLNo, and DebtInc as continuous variables. The TARGET statement defines Bad (which is a binary variable) as the target variable and specifies the order of the target variable as descending. The CODE statement generates DATA step scoring code and stores it in the filename codefile. The scoring code can be used to score other home equity loan applications.

PROC SVMACHINE generates several ODS tables, some of which are shown in Output 23.1.1 through Output 23.1.5.

The “Model Information” table in Output 23.1.1 shows that the kernel function is linear, and the penalty parameter value is 1 (both of which are default values).

```
target bad / desc;
code file=codefile;
run;
```

The observations table in Output 23.1.2 shows that the total number of observations is 5,960 and the number of observations used in the training is 3,364.

The “Training Results” table in Output 23.1.3 shows the inner product of weights, bias, total slack, and so on.
Output 23.1.3  Training Results

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inner Product of Weights</td>
</tr>
<tr>
<td>Bias</td>
</tr>
<tr>
<td>Total Slack (Constraint Violations)</td>
</tr>
<tr>
<td>Norm of Longest Vector</td>
</tr>
<tr>
<td>Number of Support Vectors</td>
</tr>
<tr>
<td>Number of Support Vectors on Margin</td>
</tr>
<tr>
<td>Maximum F</td>
</tr>
<tr>
<td>Minimum F</td>
</tr>
<tr>
<td>Number of Effects</td>
</tr>
<tr>
<td>Columns in Data Matrix</td>
</tr>
</tbody>
</table>

The “Misclassification Matrix” table in Output 23.1.4 displays the original observations and predicted values. Here the true positive is 43, the false negative is 257, the true negative is 3,055, and the false positive is 9.

Output 23.1.4  Misclassification Matrix

<table>
<thead>
<tr>
<th>Misclassification Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Prediction</td>
</tr>
<tr>
<td>Observed 1 0 Total</td>
</tr>
<tr>
<td>1 43 257 300</td>
</tr>
<tr>
<td>0 9 3055 3064</td>
</tr>
<tr>
<td>Total 52 3312 3364</td>
</tr>
</tbody>
</table>

The “Fit Statistics” table in Output 23.1.5 shows information about the accuracy, error, sensitivity, and specificity.

Output 23.1.5  Fit Statistics

<table>
<thead>
<tr>
<th>Fit Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic</td>
</tr>
<tr>
<td>Training</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>Error</td>
</tr>
<tr>
<td>Sensitivity</td>
</tr>
<tr>
<td>Specificity</td>
</tr>
</tbody>
</table>

In addition to these ODS tables, PROC SVMACHINE also generates the “Iteration History” table. The CODE statement in this example generates SAS code and stores at in the filename codefile. Advanced SAS users can use the SAS code to easily score their data.
Example 23.2: Large Simulated Data Table

This example uses a large simulated data table to demonstrate how PROC SVMACHINE can handle relatively large data. The following DATA step generates 10 million observations in the CAS table mycas.bigdata:

```
data mycas.bigdata;
array x{5} x1-x5;
drop i n;
do n=1 to 10000000;
do i=1 to dim(x);
   x{i} = ranbin(10816, 12, 0.6);
   x6 = sum(x2-x4) + ranuni(6068);
end;
if x6 > 0.5 then y = 1;
else if x6 < -0.5 then y = 0;
else y = ranbin(6084, 1, 0.4);
output;
end;
run;
```

The following statements execute the SVM algorithm on the table mycas.bigdata:

```
proc svmachine data=mycas.bigdata;
   input x1-x6 / level=interval;
   target y;
run;
```

The “Misclassification Matrix” table in Output 23.2.1 shows the classification result. The total number of observations in which $y=1$ is 5,631,506, and the total number of observations in which $y=0$ is 4,368,494.

```
Output 23.2.1  Misclassification Matrix

The SVMACHINE Procedure

<table>
<thead>
<tr>
<th>Training Prediction</th>
<th>Observed</th>
<th>1</th>
<th>0</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5164803</td>
<td>466703</td>
<td>5631506</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>248256</td>
<td>4120238</td>
<td>4368494</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>5413059</td>
<td>4586941</td>
<td>10000000</td>
<td></td>
</tr>
</tbody>
</table>
```

The “Fit Statistics” table in Output 23.2.2 shows the accuracy (92.85%) and the error (7.15%) of the model.

```
Output 23.2.2  Fit Statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.9285</td>
</tr>
<tr>
<td>Error</td>
<td>0.0715</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>0.9171</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.9432</td>
</tr>
</tbody>
</table>
```
Example 23.3: Support Vector Regression

In this example, support vector regression training is performed. For the sake of simplicity, the same data set mycas.dmagecr is used. Instead of using the binary target good_bad, amount is chosen as the interval target. The insensitive loss parameter epsilon is set to 5.0, and the LEVEL=INTERVAL option in the TARGET statement requests support vector regression training.

The following statements execute the SVR algorithm on the mycas.dmagecr data table and produce Output 23.3.1 and Output 23.3.2.

```plaintext
proc svmachine data=mycas.dmagecr epsilon=5.0;
  input checking history purpose savings employed marital coapp
      property other job housing telephon foreign/level=nominal;
  input duration installp resident existcr depends age/level=interval;
  target amount / level=INTERVAL;
run;
```

The “Model Information” table is displayed in Output 23.3.1.

```
Output 23.3.1 Model Information

The SVMACHINE Procedure

Model Information

<table>
<thead>
<tr>
<th>Task Type</th>
<th>ESP_REG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Technique</td>
<td>Interior Point</td>
</tr>
<tr>
<td>Scale</td>
<td>NO</td>
</tr>
<tr>
<td>Kernel Function</td>
<td>Linear</td>
</tr>
<tr>
<td>Penalty Method</td>
<td>C</td>
</tr>
<tr>
<td>Penalty Parameter</td>
<td>1</td>
</tr>
<tr>
<td>Insensitive Loss Parameter</td>
<td>5</td>
</tr>
<tr>
<td>Maximum Iterations</td>
<td>25</td>
</tr>
<tr>
<td>Tolerance</td>
<td>1e-06</td>
</tr>
</tbody>
</table>
```

The “Training Results” table is displayed in Output 23.3.2.

```
Output 23.3.2 Training Result

Training Results

<table>
<thead>
<tr>
<th>Inner Product of Weights</th>
<th>128231.898</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bias</td>
<td>912.616613</td>
</tr>
<tr>
<td>Total Slack (Constraint Violations)</td>
<td>1298997.98</td>
</tr>
<tr>
<td>Norm of Longest Vector</td>
<td>87.2467764</td>
</tr>
<tr>
<td>Mean Squared Error</td>
<td>4483863.04</td>
</tr>
<tr>
<td>Number of Effects</td>
<td>19</td>
</tr>
<tr>
<td>Columns in Data Matrix</td>
<td>60</td>
</tr>
</tbody>
</table>
```
References


# Chapter 24
## The TEXTMINE Procedure

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview: TEXTMINE Procedure</strong></td>
<td>570</td>
</tr>
<tr>
<td><strong>PROC TEXTMINE Features</strong></td>
<td>570</td>
</tr>
<tr>
<td><strong>Using CAS Sessions and CAS Engine Librefs</strong></td>
<td>571</td>
</tr>
<tr>
<td><strong>Getting Started: TEXTMINE Procedure</strong></td>
<td>572</td>
</tr>
<tr>
<td><strong>Syntax: TEXTMINE Procedure</strong></td>
<td>576</td>
</tr>
<tr>
<td><strong>PROC TEXTMINE Statement</strong></td>
<td>576</td>
</tr>
<tr>
<td><strong>DOC_ID Statement</strong></td>
<td>577</td>
</tr>
<tr>
<td><strong>PARSE Statement</strong></td>
<td>578</td>
</tr>
<tr>
<td><strong>SAVESTATE Statement</strong></td>
<td>582</td>
</tr>
<tr>
<td><strong>SELECT Statement</strong></td>
<td>582</td>
</tr>
<tr>
<td><strong>SVD Statement</strong></td>
<td>584</td>
</tr>
<tr>
<td><strong>TARGET Statement</strong></td>
<td>588</td>
</tr>
<tr>
<td><strong>VARIABLES Statement</strong></td>
<td>588</td>
</tr>
<tr>
<td><strong>Details: TEXTMINE Procedure</strong></td>
<td>588</td>
</tr>
<tr>
<td><strong>Natural Language Processing</strong></td>
<td>588</td>
</tr>
<tr>
<td><strong>Stemming</strong></td>
<td>588</td>
</tr>
<tr>
<td><strong>Part-of-Speech Tagging</strong></td>
<td>588</td>
</tr>
<tr>
<td><strong>Noun Group Extraction</strong></td>
<td>589</td>
</tr>
<tr>
<td><strong>Entity Identification</strong></td>
<td>589</td>
</tr>
<tr>
<td><strong>Multiword Terms Handling</strong></td>
<td>591</td>
</tr>
<tr>
<td><strong>Language Support</strong></td>
<td>591</td>
</tr>
<tr>
<td><strong>Term and Cell Weighting</strong></td>
<td>592</td>
</tr>
<tr>
<td><strong>Sparse Format</strong></td>
<td>592</td>
</tr>
<tr>
<td><strong>Coordinate List (COO) Format</strong></td>
<td>593</td>
</tr>
<tr>
<td><strong>Singular Value Decomposition</strong></td>
<td>593</td>
</tr>
<tr>
<td><strong>Applications in Text Mining</strong></td>
<td>593</td>
</tr>
<tr>
<td><strong>Computation</strong></td>
<td>594</td>
</tr>
<tr>
<td><strong>SVD-Only Mode</strong></td>
<td>594</td>
</tr>
<tr>
<td><strong>Topic Discovery</strong></td>
<td>594</td>
</tr>
<tr>
<td><strong>Output Data Tables</strong></td>
<td>595</td>
</tr>
<tr>
<td><strong>The OUTCHILD= Data Table</strong></td>
<td>595</td>
</tr>
<tr>
<td><strong>The OUTCONFIG= Data Table</strong></td>
<td>595</td>
</tr>
<tr>
<td><strong>The OUTDOCPRO= Data Table</strong></td>
<td>596</td>
</tr>
<tr>
<td><strong>The OUTPARENT= Data Table</strong></td>
<td>596</td>
</tr>
<tr>
<td><strong>The OUTPOS= Data Table</strong></td>
<td>597</td>
</tr>
<tr>
<td><strong>The OUTTERMS= Data Table</strong></td>
<td>597</td>
</tr>
</tbody>
</table>
Overview: TEXTMINE Procedure

The TEXTMINE procedure integrates natural language processing and statistical analysis to analyze large-scale textual data in SAS Viya. PROC TEXTMINE supports a wide range of fundamental text analysis features, which include tokenizing, stemming, part-of-speech tagging, noun group extraction, default or customized stop lists and start lists, entity parsing, multiword tokens, synonym lists, term weighting, term-by-document matrix creation, dimension reduction with singular value decomposition (SVD), and topic discovery. The procedure leverages the tmMine action of the textMining action set to accomplish these tasks, but it does not surface all of the action’s capabilities. Further functionality is available to you if you call this action directly using PROC CASL.

PROC TEXTMINE Features

The TEXTMINE procedure processes large-scale textual data in parallel in order to achieve efficiency and scalability. The following list summarizes the basic features of PROC TEXTMINE:

- Functionalities that are related to document parsing, term-by-document matrix creation, and dimension reduction are integrated into one procedure in order to process data more efficiently.
- Parsing supports essential natural language processing (NLP) features, which include tokenizing, stemming, part-of-speech tagging, noun group extraction, default or customized stop lists and start lists, entity parsing, multiword tokens, and synonym lists.
- Term weighting and filtering are supported for term-by-document matrix creation.
- Parsing and term-by-document matrix creation are processed in parallel.
- Computation of singular value decomposition (SVD) is parallelized.
- Topic discovery is integrated into the procedure.
- All phases of processing use a high degree of multithreading.
Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
   libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Chapter 24: The TEXTMINE Procedure

Getting Started: TEXTMINE Procedure

The input data must be a table on your CAS server, and a CAS session must be set up. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

The following DATA step creates the getstart data table, which contains 16 observations that have two variables, in your CAS session. The text variable contains the input documents, and the did variable contains the ID of the documents. Each row in the data table represents a document for analysis.

```sas
data mycas.getstart;
  infile datalines delimiter='|' missover;
  length text $150;
  input text$ did;
  datalines;
  Reduces the cost of maintenance. Improves revenue forecast. | 1
  Analytics holds the key to unlocking big data. | 2
  The cost of updates between different environments is eliminated. | 3
  Ensures easy deployment in the cloud or on-site. | 4
  Organizations are turning to SAS for business analytics. | 5
  This removes concerns about maintenance and hidden costs. | 6
  Service-oriented and cloud-ready for many cloud infrastructures. | 7
  Easily apply machine learning and data mining techniques to data. | 8
  SAS Viya will address data analysis, modeling and learning. | 9
  Helps customers reduce cost and make better decisions faster. | 10
  Simple, powerful architecture ensures easy deployment in the cloud. | 11
  SAS is helping industries glean insights from data. | 12
  Solve complex business problems faster than ever. | 13
  Shatter the barriers associated with data volume with SAS Viya. | 14
  Casual business users, data scientists and application developers. | 15
  Serves as the basis for innovation causing revenue growth. | 16
run;
```

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 uses the default stop list to eliminate noisy, noninformative terms:

```sas
proc cas;
  loadtable caslib="ReferenceData" path="en_stoplist.sashdat";
run;
quit;
```

The following statements parse the input collection and use singular value decomposition followed by a rotation to discover topics that exist in the sample collection. The statements specify that all terms in the document collection, except for those on the stop list, are to be kept for generating the term-by-document matrix. The summary information about the terms in the document collection is stored in a data table named mycas.terms. The SVD statement requests that the first three singular values and singular vectors be computed. The topic assignments of the documents are stored in a data table named mycas.docpro, and the descriptive terms that define each topic are stored in a data table named mycas.topics.
The following statements use PROC PRINT in Base SAS to show the contents of the first 10 rows of the sorted mycas.docpro data table that is generated by the TEXTMINE procedure:

```sas
proc print data=mycas.docpro;
run;
```
data docpro;
  set mycas.docpro;
run;
proc sort data=docpro;
  by did;
run;
proc print data = docpro (obs=10);
run;

Figure 24.2 shows the output of PROC PRINT. For information about the output of the OUTDOCPRO= option, see the section “The OUTDOCPRO= Data Table” on page 596.

Figure 24.2 The mycas.docpro Data Table

<table>
<thead>
<tr>
<th>Obs</th>
<th>did</th>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.7460570931</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0.1111856451</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0.0964494952</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.8688770161</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0.4742893251</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0.6276285113</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0.0901933118</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0.0626896657</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>0</td>
<td>0.5236329356</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0.0478786576</td>
<td>0.0703302315</td>
</tr>
</tbody>
</table>

The following statements use a DATA step and PROC PRINT to show the contents of the mycas.topics data table that is generated by the TEXTMINE procedure:

data topics; set mycas.topics; run;
proc print data = topics;
run;

Figure 24.3 shows the output of PROC PRINT. The three discovered topics are listed with four descriptive terms to characterize each topic.

Figure 24.3 The mycas.topics Data Table

<table>
<thead>
<tr>
<th>Obs</th>
<th>topicid</th>
<th>_name</th>
<th>_termCutOff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>easy deployment, deployment, +ensure, easy, cloud</td>
<td>0.135</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>sas, data, viya, analytics, +industry</td>
<td>0.149</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>+cost, maintenance, revenue forecast, forecast, +improve</td>
<td>0.146</td>
</tr>
</tbody>
</table>

The following statements use a DATA step and the SORT and PRINT procedures to show the first 10 observations of the mycas.terms data table that is generated by the TEXTMINE procedure:

data terms; set mycas.terms; run;
proc sort data = terms; by key; run;
proc print data = terms (obs=10);
var term role freq numdocs key parent;
run;
Figure 24.4 shows the output of PROC PRINT, which provides details about the terms that are identified by the TEXTMINE procedure. Only the values of the variables term, role, freq, numdocs, key, and parent are displayed. For information about the output of the OUTTERMS= option, see the section “The OUTTERMS= Data Table” on page 597.

### Figure 24.4 The mycas.terms Data Table

<table>
<thead>
<tr>
<th>Obs</th>
<th>Term</th>
<th>Role</th>
<th>Freq</th>
<th>numdocs</th>
<th>Key</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>simple</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>revenue forecast</td>
<td>npnNounGroup</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>technique</td>
<td>N</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>different environment</td>
<td>npnNounGroup</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>decision</td>
<td>N</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>.</td>
</tr>
<tr>
<td>6</td>
<td>cloud infrastructure</td>
<td>npnNounGroup</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>.</td>
</tr>
<tr>
<td>7</td>
<td>hold</td>
<td>V</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>.</td>
</tr>
<tr>
<td>8</td>
<td>application developer</td>
<td>npnNounGroup</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>analysis</td>
<td>N</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>.</td>
</tr>
<tr>
<td>10</td>
<td>analytics</td>
<td>N</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>.</td>
</tr>
</tbody>
</table>

The following DATA step and statements create data and then score that data with PROC ASTORE.

```sas
data mycas.scoreData;
  infile datalines delimiter='|' missover;
  length text $150;
  input text$ id;
  datalines;
    Deployment in the cloud or on-site. | 1
    SAS for business analytics.         | 2
    Maintenance and hidden costs.       | 3
run;

proc astore;
  score rstore=mycas.aStoreTab
    data=mycas.scoreData
    out= mycas.scoreResults
    copyVars= id;
run;

proc sort data=mycas.scoreResults out=scoreResults;
  by id;
run;
proc print data = scoreResults;
run;
```

Figure 24.5 shows the output of PROC PRINT, which provides the topic score for the documents processed by the ASTORE PROCEDURE.
Figure 24.5  The mycas.scoreResults Data Table

<table>
<thead>
<tr>
<th></th>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.56920</td>
<td>0.00000</td>
<td>0.00000</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.00000</td>
<td>0.41840</td>
<td>0.00000</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.55244</td>
<td>3</td>
</tr>
</tbody>
</table>

Syntax: TEXTMINE Procedure

The following statements are available in the TEXTMINE procedure:

```
PROC TEXTMINE DATA=CAS-libref.data-table <options> ;
  VARIABLES variable ;
  TARGET variable ;
  DOC_ID variable ;
  PARSE <parse-options> ;
  SELECT label-list <GROUP=group-option> KEEP | IGNORE ;
  SVD <svd-options> ;
  SAVESTATE RSTORE=CAS-libref.data-model ;
```

The PROC TEXTMINE statement, the VARIABLES statement, and the DOC_ID statement are required. The following sections describe the PROC TEXTMINE statement and then describe the other statements in alphabetical order.

PROC TEXTMINE Statement

```
PROC TEXTMINE DATA=CAS-libref.data-table <options> ;
```

The PROC TEXTMINE statement invokes the procedure. Table 24.1 summarizes the options in the statement by function. The options are then described fully in alphabetical order.

<table>
<thead>
<tr>
<th>option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic Options</strong></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>DOC=</td>
</tr>
<tr>
<td>LANGUAGE=</td>
<td></td>
</tr>
<tr>
<td>NEWVARNAMES</td>
<td></td>
</tr>
<tr>
<td><strong>Multithreading Options</strong></td>
<td></td>
</tr>
<tr>
<td>NTHREADS=</td>
<td></td>
</tr>
</tbody>
</table>
You must specify the following option:

**DATA**=CAS-libref.data-table

names the input data table for PROC TEXTMINE to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

- **CAS-libref** refers to a collection of information that is defined in the LIBNAME statement and includes the *caslib*, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 571.

- **data-table** specifies the name of the input data table.

Each row of the input data table must contain one text variable and one ID variable that correspond to the text and the unique ID of a document, respectively.

When you specify the SVD statement but not the PARSE statement, PROC TEXTMINE runs in **SVD-only** mode. In this mode, the DATA= option names the input SAS data table that contains the term-by-document matrix that is generated by the OUTPARENT= option in the PARSE statement.

You can also specify the following **options**:

**LANGUAGE**=language

names the language that is used by the documents in the input SAS data table. Languages supported in the current release are Arabic, Chinese, Croatian, Czech, Danish, Dutch, English, Farsi, Finnish, French, German, Greek, Hebrew, Hungarian, Indonesian, Italian, Japanese, Kazakh, Korean, Norwegian, Polish, Portuguese, Russian, Slovak, Slovene, Spanish, Swedish, Thai, Turkish and Vietnamese. By default, LANGUAGE=ENGLISH.

**NEWVARNAMES**

adds leading and trailing blanks to variable names in the input and output tables.

**NTHREADS**=nthreads

specifies the number of threads to be used. By default, the number of threads is the same as the number of CPUs on the CAS server.

---

**DOC_ID Statement**

**DOC_ID** variable ;

The DOC_ID statement specifies the variable that contains the ID of each document. In the input data table, each row corresponds to one document. The ID of each document must be unique; it can be either a number or a string of characters.
The PARSE statement specifies the options for parsing the input documents and creating the term-by-document matrix. Table 24.2 summarizes the `parse-options` in the statement by function. The `parse-options` are then described fully in alphabetical order.

### Table 24.2 PARSE Statement Options

<table>
<thead>
<tr>
<th><code>parse-option</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parsing Options</strong></td>
<td></td>
</tr>
<tr>
<td>ENTITIES=</td>
<td>Specifies whether to extract entities in parsing</td>
</tr>
<tr>
<td>MULTITERM=</td>
<td>Specifies the multiword term list</td>
</tr>
<tr>
<td>NONOUNGROUPS</td>
<td>NONG</td>
</tr>
<tr>
<td>NOSTEMMING</td>
<td>Suppresses stemming in parsing</td>
</tr>
<tr>
<td>NOTAGGING</td>
<td>Suppresses part-of-speech tagging in parsing</td>
</tr>
<tr>
<td>SHOWDROPPEDTERMS=</td>
<td>Includes dropped terms in the OUTTERMS= data table</td>
</tr>
<tr>
<td>START=</td>
<td>Specifies the start list</td>
</tr>
<tr>
<td>STOP=</td>
<td>Specifies the stop list</td>
</tr>
<tr>
<td>SYNONYM</td>
<td>SYN=</td>
</tr>
<tr>
<td><strong>Term-by-Document Matrix Creation Options</strong></td>
<td></td>
</tr>
<tr>
<td>CELLWGT=</td>
<td>Specifies how cells are weighted</td>
</tr>
<tr>
<td>REDUCEF=</td>
<td>Specifies the frequency for term filtering</td>
</tr>
<tr>
<td>TERMWGT=</td>
<td>Specifies how terms are weighted</td>
</tr>
<tr>
<td><strong>Output Options</strong></td>
<td></td>
</tr>
<tr>
<td>OUTCHILD=</td>
<td>Specifies the data table to contain the raw term-by-document matrix. All kept terms, whether or not they are child terms, are represented in this data table along with their corresponding frequency.</td>
</tr>
<tr>
<td>OUTCONFIG=</td>
<td>Specifies the data table to contain the option settings that PROC TEXTMINE uses in the current run</td>
</tr>
<tr>
<td>OUTPARENT=</td>
<td>Specifies the data table to contain the term-by-document matrix. Child terms are not represented in this data table. The frequencies of child terms are attributed to their corresponding parents.</td>
</tr>
<tr>
<td>OUTTERMS=</td>
<td>Specifies the data table to contain the summary information about the terms in the document collection</td>
</tr>
<tr>
<td>OUTPOS=</td>
<td>Specifies the data table to contain the position information about the child terms’ occurrences in the document collection</td>
</tr>
</tbody>
</table>

You can specify the following `parse-options`. 

```
PARSE <parse-options>; 
```
CELLWGT=LOG | NONE
specifies how the elements in the term-by-document matrix are weighted. You can specify the following values:

LOG weights cells by using the log formulation. For information about the log formulation for cell weighting, see the section “Term and Cell Weighting” on page 592.

NONE specifies that no cell weight be applied.

ENTITIES=STD | NONE
determines whether to use the standard LITI file for entity extraction. You can specify the following values:

STD uses the standard LITI file for entity extraction. A term such as “George W. Bush” is recognized as an entity and given the corresponding entity role and attribute. For this term, the entity role is PERSON and the attribute is Entity. Although the entity is treated as the single term, “george w. bush,” the individual tokens “george,” “w,” and “bush” are also included.

NONE does not use the standard LITI file for entity extraction.

By default, ENTITIES=None.

MULTITERM=CAS-libref.data-table
specifies the input SAS data table that contains a list of multiword terms. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571. The multiword terms are case-sensitive and are treated as a single entry by the TEXTMINE procedure. Thus, the terms “Thank You” and “thank you” are processed differently. Consequently, you must convert all text strings to lowercase or add each of the multiterm’s case variations to the list before using the TEXTMINE procedure to create consistent multiword terms. The multiterm data table must have a variable Multiterm and each of its values must be formatted in the following manner:

multiterm: 3: pos

Specifically, the first item is the multiword term itself followed by a colon, the second item is a number that represents the token type followed by a colon, and the third item is the part of speech that the multiword term represents. NOTE: The token type 3 is the most common token type for multiterm lists; it represents compound words.

NONOUNGROUPS NONG
suppresses standard noun group extraction. By default, the TEXTMINE procedure extracts noun groups, returns noun phrases without determiners or prepositions, and (unless the NOSTEMMING option is specified) stems noun group elements.
NOSTEMMING
suppresses stemming of words. By default, words are stemmed; that is, terms such as “advises” and “advising” are mapped to the parent term “advise.” The TEXTMINE procedure uses dictionary-based stemming (also known as lemmatization).

NOTAGGING
suppresses tagging of terms. By default, terms are tagged and the TEXTMINE procedure identifies a term’s part of speech based on context clues. The identified part of speech is provided in the Role variable of the OUTTERMS= data table.

OUTCHILD=CAS-libref.data-table
specifies the output data table to contain a compressed representation of the sparse term-by-document matrix. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571. The term counts are not weighted. The data table saves only the kept, representative terms. The child frequencies are not attributed to their corresponding parent (as they are in the OUTPARENT= data table). For more information about the compressed representation of the sparse term-by-document matrix, see the section “The OUTCHILD= Data Table” on page 595.

OUTCONFIG=CAS-libref.data-table
specifies the output data table to contain configuration information that is used for the current run of PROC TEXTMINE. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571. The primary purpose of this data table is to relay the configuration information from the TEXTMINE procedure to the TMSCORE procedure. The TMSCORE procedure uses options that are consistent with the TEXTMINE procedure. Thus, the data table that is created by using the OUTCONFIG= option becomes an input data table for PROC TMSCORE and ensures that the parsing options are consistent between the two runs. For more information about this data table, see the section “The OUTCONFIG= Data Table” on page 595.

OUTPARENT=CAS-libref.data-table
specifies the output data table to contain a compressed representation of the sparse term-by-document matrix. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571. The term counts can be weighted, if requested. The data table contains only the kept, representative terms, and the child frequencies are attributed to the corresponding parent. To obtain information about the children, use the OUTCHILD= option. For more information about the compressed representation of the sparse term-by-document matrix, see the section “The OUTPARENT= Data Table” on page 596.

OUTPOS=CAS-libref.data-table
specifies the output data table to contain the position information about the child terms’ occurrences in the document collection. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571. For more information about this data table, see the section “The OUTPOS= Data Table” on page 597.
OUTTERMS=\texttt{CAS-libref.data-table}

specifies the output data table to contain the summary information about the terms in the document collection. The maximum output length of a tokenized term is 256 bytes. So tokens consisting of an extremely long sequence of letters, numbers and symbols will be truncated to less than or equal to that maximum value of 256 bytes. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the caslib and session identifier, and \texttt{data-table} specifies the name of the output data table. For more information about this two-level name, see the \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571. For more information about this data table, see the section “Output Data Tables” on page 595.

REDUCEF=$n$

removes terms that are not in at least $n$ documents. The value of $n$ must be a positive integer. By default, REDUCEF=4.

SHOWDROPPEDTERMS

includes the terms that have a keep status of N in the OUTTERMS= data table and the OUTCHILD= data table.

START=\texttt{CAS-libref.data-table}

specifies the input data table that contains the terms that are to be kept for the analysis. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the caslib and session identifier, and \texttt{data-table} specifies the name of the input data table. For more information about this two-level name, see the \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571. These terms are displayed in the OUTTERMS= data table with a keep status of Y. All other terms are displayed with a keep status of N if the SHOWDROPPEDTERMS option is specified or not displayed if the SHOWDROPPEDTERMS option is not specified. The START= data table must have a Term variable and can also have a Role variable. You cannot specify both the START= and STOP= options.

STOP=\texttt{CAS-libref.data-table}

specifies the input data table that contains the terms to exclude from the analysis. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the caslib and session identifier, and \texttt{data-table} specifies the name of the input data table. For more information about this two-level name, see the \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571. These terms are displayed in the OUTTERMS= data table with a keep status of N if the SHOWDROPPEDTERMS option is specified. The terms are not identified as parents or children. The STOP= data table must have a Term variable and can also have a Role variable. You cannot specify both the START= and STOP= options.

SYNONYM=\texttt{CAS-libref.data-table}

specifies the input data table that contains user-defined synonyms to be used in the analysis. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the caslib and session identifier, and \texttt{data-table} specifies the name of the input data table. For more information about this two-level name, see the \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571. The input synonym data table must have either the two variables Term and Parent or the four variables Term, Parent, Termrole, and Parentrole. This data table overrides any relationships that are
identified when terms are stemmed. (Terms are stemmed by default; you can suppress stemming by specifying the NOSTEMMING option.)

**TERMWGT=ENTROPY | MI | NONE**

specifies how terms are weighted. You can specify the following values:

- **ENTROPY** uses the entropy formulation to weight terms.
- **MI** uses the mutual information formulation to weight terms (you must also specify the TARGET statement).
- **NONE** requests that no term weight be applied.

For more information about the entropy formulation and the mutual information formulation for term weighting, see the section “Term and Cell Weighting” on page 592.

---

**SAVESTATE Statement**

```
SAVESTATE RSTORE=CAS-libref.data-model;
```

The SAVESTATE statement saves a text mining model to a binary object contained in a data table. The object is referred to as the analytic store and contains the necessary information for scoring a text mining model by the ASTORE procedure. Only complete text models consisting of both parsing and document projections can be saved to the analytic store by the TEXTMINE procedure.

You must specify the following option:

- **RSTORE=** specifies a data table in which to save the text mining model. **CAS-libref.data-table** is a two-level name, where **CAS-libref** refers to the caslib and session identifier, and **data-table** specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571.

---

**SELECT Statement**

```
SELECT label-list /<GROUP=group-option> KEEP | IGNORE;
```

The SELECT statement enables you to specify the parts of speech or entities or attributes that you want to include in or exclude from your analysis. Exclusion by the SELECT statement is different from exclusion that is indicated by the _keep variable in the OUTTERMS= data table. Terms that are excluded by the SELECT statement cannot be included in the OUTTERMS= data table, whereas terms that have _keep=N can be included in the OUTTERMS= data table if the SHOWDROPPEDTERMS option is specified. Terms excluded by the SELECT statement are excluded from the OUTPOS= data table, but terms that have _keep=N are included in OUTPOS= data table. Table 24.3 summarizes the options you can specify in the SELECT statement. The options are then described fully in syntactic order.
Table 24.3  SELECT Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label-list</td>
<td>Specifies one or more labels of terms that are to be ignored or kept in your analysis</td>
</tr>
<tr>
<td>GROUP=</td>
<td>Specifies whether the labels are parts of speech, entities, or attributes</td>
</tr>
<tr>
<td>IGNORE</td>
<td>Ignores terms whose labels are specified in the label-list</td>
</tr>
<tr>
<td>KEEP</td>
<td>Keeps terms whose labels are specified in the label-list</td>
</tr>
</tbody>
</table>

You must specify a label-list and either the IGNORE or KEEP option:

**label-list**

specifies one or more labels that are either parts of speech or entities or attributes. Each label must be surrounded by double quotation marks and separated by spaces from other labels. Labels are case-insensitive. Terms that have these labels are either ignored during parsing (when the IGNORE option is specified) or kept in the parsing results in the OUTPOS= and OUTTERMS= data tables (when the KEEP option is specified). Table 24.5 shows all possible part-of-speech tags. Table 24.6 shows all valid English entities. The attribute variable in Table 24.11 shows all possible attributes.

**IGNORE**

ignores during parsing all terms whose labels are specified in the label-list, but keeps all other terms in the parsing results (the OUTPOS= and OUTTERMS= data tables).

**KEEP**

keeps in the parsing results (the OUTPOS= and OUTTERMS= data tables) only the terms whose labels are specified in the label-list.

You can also specify the following option:

**GROUP=“ATTRIBUTES” | “ENTITIES” | “POS”**

specifies whether the labels are attributes, entities, or parts of speech. The group type must be surrounded by double quotation marks and is case-insensitive. All labels that are specified in the label-list in the same SELECT statement should belong to the specified group. If you need to select labels from more than one group, you can use multiple SELECT statements (one for each group that you need to select from). You cannot specify multiple SELECT statements for the same group. By default, Num and Punct in the “ATTRIBUTES” group are ignored, but this default is overridden by a SELECT statement that specifies GROUP=“ATTRIBUTES”. By default, GROUP=“POS”.

---

**SELECT Statement**

Table 24.3  SELECT Statement Options
**SVD Statement**

\[
\text{SVD} < \text{svd-options} > ;
\]

The SVD statement specifies the options for calculating a truncated singular value decomposition (SVD) of the large, sparse term-by-document matrix that is created during the parsing phase of PROC TEXTMINE. Table 24.4 summarizes the `svd-options` in the statement by function. The `svd-options` are then described fully in alphabetical order.

<table>
<thead>
<tr>
<th><code>svd-option</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Options</strong></td>
<td></td>
</tr>
<tr>
<td>COL=</td>
<td>Specifies the column variable, which contains the column indices of the term-by-document matrix, which is stored in coordinate list (COO) format</td>
</tr>
<tr>
<td>ROW=</td>
<td>Specifies the row variable, which contains the row indices of the term-by-document matrix, which is stored in COO format</td>
</tr>
<tr>
<td>ENTRY=</td>
<td>Specifies the entry variable, which contains the entries of the term-by-document matrix, which is stored in COO format</td>
</tr>
<tr>
<td><strong>SVD Computation Options</strong></td>
<td></td>
</tr>
<tr>
<td>K=</td>
<td>Specifies the number of dimensions to be extracted</td>
</tr>
<tr>
<td>MAX_K=</td>
<td>Specifies the maximum number of dimensions to be extracted</td>
</tr>
<tr>
<td>TOL=</td>
<td>Specifies the maximum allowable tolerance for the singular value</td>
</tr>
<tr>
<td>RESOLUTION</td>
<td>RES=</td>
</tr>
<tr>
<td><strong>Topic Discovery Options</strong></td>
<td></td>
</tr>
<tr>
<td>NUMLABELS=</td>
<td>Specifies the number of terms to be used in the descriptive label for each topic</td>
</tr>
<tr>
<td>ROTATION=</td>
<td>Specifies the type of rotation to be used for topic discovery</td>
</tr>
<tr>
<td>IN_TERMS=</td>
<td>Specifies the data table that contains the terms for topic discovery in SVD-only mode</td>
</tr>
<tr>
<td>EXACTWEIGHT</td>
<td>Prevents rounding of the topic weights</td>
</tr>
<tr>
<td>NOCUTOFFS</td>
<td>Prevents setting term weights to 0 when they are below the threshold</td>
</tr>
<tr>
<td><strong>Output Options</strong></td>
<td></td>
</tr>
<tr>
<td>SVDU=</td>
<td>Specifies the U matrix, which contains the left singular vectors</td>
</tr>
<tr>
<td>SVDV=</td>
<td>Specifies the V matrix, which contains the right singular vectors</td>
</tr>
<tr>
<td>SVDS=</td>
<td>Specifies the S matrix, whose diagonal elements are the singular values</td>
</tr>
<tr>
<td>OUTDOCPRO=</td>
<td>Specifies the data table to contain the projections of the documents</td>
</tr>
<tr>
<td>OUTTOPICS=</td>
<td>Specifies the data table to contain the topics that have been discovered</td>
</tr>
</tbody>
</table>

You can specify the following `svd-options`:
**COL=** *variable*

specifies the *variable* that contains the column indices of the term-by-document matrix. You must specify this option when you run PROC TEXTMINE in **SVD-only** mode (that is, when you specify the SVD statement but not the PARSE statement).

**ENTRY=** *variable*

specifies the *variable* that contains the entries of the term-by-document matrix. You must specify this option when you run PROC TEXTMINE in **SVD-only** mode (that is, when you specify the SVD statement but not the PARSE statement).

**EXACTWEIGHT**

requests that the weights aggregated during topic derivation not be rounded. By default, the calculated weights are rounded to the nearest 0.001.

**IN_TERMS=** **CAS-libref.data-table**

specifies the input data table that contains information about the terms in the document collection. **CAS-libref.data-table** is a two-level name, where **CAS-libref** refers to the caslib and session identifier, and **data-table** specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571. The data table should have the variables that are described in Table 24.11. The terms are required to generate topic names in the **OUTTOPICS=** data table. This option is only for topic discovery in SVD-only mode. This option conflicts with the PARSE statement, and only one of the two can be specified. If you want to run SVD-only mode without topic discovery, then you do not need to specify this option.

**K=** *k*

specifies the number of columns in the matrices U, V, and S. This value is the number of dimensions of the data table after SVD is performed. If the value of *k* is too large, then the TEXTMINE procedure runs for an unnecessarily long time. This option takes precedence over the **MAX_K=** option. This option also controls the number of topics that are extracted from the text corpus when the **ROTATION=** option is specified.

**MAX_K=** *n*

specifies the maximum value that the TEXTMINE procedure should return as the recommended value of *k* (the number of columns in the matrices U, V, and S) when the **RESOLUTION=** option is specified (to recommend the value of *k*). The TEXTMINE procedure attempts to calculate *k* dimensions (as opposed to recommending it) when it performs SVD. This option is ignored if the **K=** option has been specified. This option also controls the number of topics that are extracted from the text corpus when the **ROTATION=** option is specified.

**NOCUTOFFS**

uses all weights in the U matrix to form the document projections. When topics are requested, weights below the term cutoff (as calculated in the **OUTTOPICS=** data table) are set to 0 before the projection is formed.
NUMLABELS=n
specifies the number of terms to use in the descriptive label for each topic. The descriptive label provides a quick synopsis of the discovered topics. The labels are stored in the OUTTOPICS=data table. By default, NUMLABELS=5.

OUTDOCPRO=CAS-libref.data-table <KEEPVARIABLES=variable-list><NONORMDOC>
OUTDOCPRO=CAS-libref.data-table <KEEPVARS=variable-list><NONORMDOC>
specifies the output data table to contain the projections of the columns of the term-by-document matrix onto the columns of U. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571. Because each column of the term-by-document matrix corresponds to a document, the output forms a new representation of the input documents in a space that has much lower dimensionality.

You can copy the variables from the data table that is specified in the DATA= option in the PROC TEXTMINE statement to the data table that is specified in this option. You can specify the following suboptions:

KEEPVARIABLES=variable-list
attaches the content of the variables that are specified in the variable-list to the output. These variables must appear in the data table that is specified in the DATA= option in the PROC TEXTMINE statement.

NONORMDOC
suppresses normalization of the columns that contain the projections of documents to have a unit norm.

OUTTOPICS= CAS-libref.data-table
specifies the output data table to contain the topics that are discovered. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571.

RESOLUTION=LOW | MED | HIGH
RES=LOW | MED | HIGH
specifies how to calculate the recommended number of dimensions (resolution) for the singular value decomposition. If you specify this option, you must also specify the MAX_K= option. A low-resolution singular value decomposition returns fewer dimensions than a high-resolution singular value decomposition. This option recommends the value of k (the number of columns in the matrices U, V, and S) heuristically based on the value specified in the MAX_K= option. Assume that the MAX_K= option is set to n and a singular value decomposition that has n dimensions accounts for t\% of the total variance. You can specify the following values:

HIGH always recommends the maximum number of dimensions; that is, k = n.
MED recommends a k that explains \((5/6) \times t\%\) of the total variance.
LOW recommends a k that explains \((2/3) \times t\%\) of the total variance.

By default, RESOLUTION=HIGH.
**ROTATION=VARIMAX | PROMAX**
specifies the type of rotation to be used in order to maximize the explanatory power of each topic. You can specify the following values:

**PROMAX** does an oblique rotation on the original left singular vectors and generates topics that might be correlated.

**VARIMAX** does an orthogonal rotation on the original left singular vectors and generates uncorrelated topics.

By default, ROTATION=VARIMAX.

**ROW=variable**
specifies the variable that contains the row indices of the term-by-document matrix. You must specify this option when you run PROC TEXTMINE in SVD-only mode (that is, when you specify the SVD statement but not the PARSE statement).

**SVDS=CAS-libref.data-table**
specifies the output data table to contain the calculated singular values. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571.

**SVDU=CAS-libref.data-table**
specifies the data table to contain the calculated left singular vectors. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571.

**SVDV=CAS-libref.data-table**
specifies the data table to contain the calculated right singular vectors. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 571.

**TOL=ε**
specifies the maximum allowable tolerance for the singular value. Let $A$ be a matrix. Suppose $\lambda_i$ is the $i$th singular value of $A$ and $\xi_i$ is the corresponding right singular vector. The SVD computation terminates when for all $i \in \{1, \ldots, k\}$, $\lambda_i$ and $\xi_i$ satisfy $\|A^T A \xi - \lambda_i \xi\|^2 \leq \epsilon$. The default value of $\epsilon$ is $10^{-6}$, which is more than adequate for most text mining problems.
TARGET Statement

```
TARGET variable ;
```

This statement specifies the `variable` that contains the information about the category that a document belongs to. The target `variable` can be any nominal or ordinal variable; it is used in calculating mutual information term weighting.

VARIABLES Statement

```
VARIABLES variable ;
VAR variable ;
```

This statement specifies the `variable` that contains the text to be processed.

Details: TEXTMINE Procedure

Natural Language Processing

Natural language processing (NLP) techniques can be used to extracting meaningful information from natural language input. The following sections describe features from SAS linguistic technologies that the TEXTMINE procedure implements to support natural language processing.

Stemming

Stemming (a special case of morphological analysis) identifies the possible root form of an inflected word. For example, the word “talk” is the stem of the words “talk,” “talks,” “talking,” and “talked.” In this case “talk” is the parent, and “talk,” “talks,” “talking,” and “talked” are its children. The TEXTMINE procedure uses dictionary-based stemming (also known as lemmatization), which unlike tail-chopping stemmers, produces only valid words as stems. When part-of-speech tagging is on (that is, the NOTAGGING option is not specified), the stem selection process restricts the stem to be of the same part-of-speech as the original term.

Part-of-Speech Tagging

Part-of-speech tagging uses SAS linguistic technologies to identify or disambiguate the grammatical category of a word by analyzing it within its context. For example:

```
I like to bank at the local branch of my bank.
```

In this case, the first “bank” is tagged as a verb (V), and the second “bank” is tagged as a noun (N). Table 24.5 shows all possible part-of-speech tags.
Table 24.5 All Part-of-Speech Tags

<table>
<thead>
<tr>
<th>Part-of-Speech Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Adjective</td>
</tr>
<tr>
<td>ADV</td>
<td>Adverb</td>
</tr>
<tr>
<td>AFX</td>
<td>Affix</td>
</tr>
<tr>
<td>CONJ</td>
<td>Conjunction</td>
</tr>
<tr>
<td>DET</td>
<td>Determiner</td>
</tr>
<tr>
<td>INTJ</td>
<td>Interjection</td>
</tr>
<tr>
<td>N</td>
<td>Noun</td>
</tr>
<tr>
<td>NUM</td>
<td>Number or numeric expression</td>
</tr>
<tr>
<td>PPOS</td>
<td>Preposition</td>
</tr>
<tr>
<td>PTCL</td>
<td>Participle</td>
</tr>
<tr>
<td>PRO</td>
<td>Pronoun</td>
</tr>
<tr>
<td>PN</td>
<td>Proper noun</td>
</tr>
<tr>
<td>PUNC</td>
<td>Punctuation</td>
</tr>
<tr>
<td>V</td>
<td>Verb</td>
</tr>
</tbody>
</table>

Noun Group Extraction

Noun groups provide more relevant information than simple nouns. A noun group is defined as a sequence of nouns and their modifiers. Noun group extraction uses part-of-speech tagging to identify nouns and their adjacent noun and adjective modifiers that together form a noun group. Examples of noun groups are “weeklong cruises” and “Middle Eastern languages.”

Entity Identification

Entity identification uses SAS linguistic technologies to classify sequences of words into predefined classes. These classes are assigned as roles for the corresponding sequences. For example, “nlpPerson,” “nlpPlace,” “nlpOrganization,” and “nlpMeasure” are identified as classes for “George W. Bush,” “Boston,” “SAS Institute,” “2.5 inches,” respectively. Table 24.6 shows all valid entities for English. Not all languages support all entities. Table 24.7 and Table 24.8 indicate the languages that are available for each entity.

Table 24.6 All Valid English Entities

<table>
<thead>
<tr>
<th>Entities</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlpDate</td>
<td>Date</td>
</tr>
<tr>
<td>nlpMeasure</td>
<td>Measurement or measurement expression</td>
</tr>
<tr>
<td>nlpMoney</td>
<td>Currency or currency expression</td>
</tr>
<tr>
<td>nlpNounGroup</td>
<td>Phrases that contain multiple words</td>
</tr>
<tr>
<td>nlpOrganization</td>
<td>Organization or company name</td>
</tr>
<tr>
<td>nlpPercent</td>
<td>Percentage or percentage expression</td>
</tr>
<tr>
<td>nlpPerson</td>
<td>Person’s name</td>
</tr>
<tr>
<td>nlpPlace</td>
<td>Addresses, cities, states, and other locations</td>
</tr>
<tr>
<td>nlpTime</td>
<td>Time or time expression</td>
</tr>
</tbody>
</table>
Table 24.7  Supported Language-Entity Pairs, Part 1

<table>
<thead>
<tr>
<th>Language</th>
<th>nlpDate</th>
<th>nlpMeasure</th>
<th>nlpMoney</th>
<th>nlpNounGroup</th>
<th>nlpOrganization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Chinese</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Croatian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Czech</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Danish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Dutch</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>English</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Farsi</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Finnish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>French</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>German</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Greek</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hebrew</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hindi</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hungarian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Indonesian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Italian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Japanese</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Korean</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Norwegian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Polish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Portuguese</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Romanian</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Russian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Slovak</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Slovene</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Spanish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Swedish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Tagalog</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Thai</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Turkish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 24.8  Supported Language-Entity Pairs, Part 2

<table>
<thead>
<tr>
<th>Language</th>
<th>nlpPercent</th>
<th>nlpPerson</th>
<th>nlpPlace</th>
<th>nlpTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Chinese</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Croatian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Czech</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Danish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Dutch</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>English</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Farsi</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Finnish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>French</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>German</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Greek</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hebrew</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hindi</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hungarian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Indonesian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Italian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Japanese</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Korean</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Norwegian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Polish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Portuguese</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Romanian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Russian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Slovak</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Slovene</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Spanish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Swedish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Tagalog</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Thai</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Turkish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Table 24.8 continued

<table>
<thead>
<tr>
<th>Language</th>
<th>nlpPercent</th>
<th>nlpPerson</th>
<th>nlpPlace</th>
<th>nlpTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dutch</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>English</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Farsi</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Finnish</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>French</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>German</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Greek</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Hebrew</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Hindi</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Hungarian</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Indonesian</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Italian</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Japanese</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Korean</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Norwegian</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Polish</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Portuguese</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Romanian</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Russian</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Slovak</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Slovene</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Spanish</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Swedish</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Tagalog</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Thai</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Turkish</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Multiword Terms Handling

By default, SAS linguistic technologies tokenize the text to individual words and operate at the word level. Multiword terms provide a control that enables you to specify sequences of words to be interpreted as individual units. For example, “greater than,” “in spite of,” and “as well as” can be defined as multiword terms.

Language Support

Languages supported in the current release are Arabic, Chinese, Croatian, Czech, Danish, Dutch, English, Finnish, French, German, Greek, Hebrew, Hindi, Hungarian, Indonesian, Italian, Japanese, Kazakh, Korean, Norwegian, Polish, Portuguese, Romanian, Russian, Slovak, Slovene, Spanish, Swedish, Tagalog, Thai, Turkish and Vietnamese. By turning off some of the advanced parsing functionality, you might be able to use PROC TEXTMINE effectively with other space-delimited languages.
Term and Cell Weighting

The TERMWGT= option and the CELLWGT= option control how to weight the frequencies in the compressed term-by-document matrix. The term weight is a positive number that is assigned to each term based on the distribution of that term in the document collection. This weight can be interpreted as an indication of the importance of that term to the document collection. The cell weight is a function that is applied to every entry in the term-by-document matrix; it moderates the effect of a term that is repeated within a document.

Let $f_{i,j}$ be the entry in the $i$th row and $j$th column of the term-by-document matrix, which indicates the time of appearance of term $i$ in document $j$. Assuming that the term weight of term $i$ is $w_i$ and the cell weight function is $g(x)$, the weighted frequency of each entry in the term-by-document matrix is given by $w_i \times g(f_{i,j})$.

When the CELLWGT=LOG option is specified, the following equation is used to weight cells:

$$g(x) = \log_2(f_{i,j} + 1)$$

The equation reduces the influence of highly frequent terms by applying the log function.

When the TERMWGT=ENTROPY option is specified, the following equation is used to weight terms:

$$w_i = 1 + \sum_j p_{i,j} \frac{\log_2(p_{i,j})}{\log_2(n)}$$

In this equation, $n$ is the number of documents, and $p_{i,j}$ is the probability that term $i$ appears in document $j$, which can be estimated by $p_{i,j} = \frac{f_{i,j}}{g_i}$, where $g_i$ is the global term frequency for term $i$.

When the TERMWGT=MI option is specified, the following equation is used to weight terms:

$$w_i = \max_{C_k} \left( \log \left( \frac{P(t_i, C_k)}{P(t_i) P(C_k)} \right) \right)$$

In this equation, $C_k$ is the set of documents that belong to category $k$, $P(C_k)$ is the percentage of documents that belong to category $k$, and $P(t_i, C_k)$ is the percentage of documents that contain term $t_i$ and belong to category $k$. Let $d_i$ be the number of documents that term $i$ appears in. Then $P(t_i) = \frac{d_i}{n}$.

Sparse Format

A matrix is sparse when most of its elements are 0. The term-by-document matrix that the TEXTMINE procedure generates is a sparse matrix. To save storage space, the TEXTMINE procedure supports the COO format for storing a sparse matrix.
Coordinate List (COO) Format

The COO is also known as the transactional format. In this format, the matrix is represented as a set of triples \((i, j, x)\), where \(x\) is an entry in the matrix and \(i\) and \(j\) denote its row and column indices, respectively. When the transactional style is used, all 0 entries in the matrix are ignored in the output, thereby saving storing space when the matrix is sparse. The COO format is good for incremental matrix construction. For example, it is easy to add new rows and new columns to the matrix by inserting more tuples in the list.

Singular Value Decomposition

Singular value decomposition (SVD) of a matrix \(A\) factors \(A\) into three matrices such that \(A = U\Sigma V^T\). Singular value decomposition also requires that the columns of \(U\) and \(V\) be orthogonal and that \(\Sigma\) be a real-valued diagonal matrix that contains monotonically decreasing, nonnegative entries. The entries of \(\Sigma\) are called singular values. The columns of \(U\) and \(V\) are called left and right singular vectors, respectively. A truncated singular value decomposition calculates only the first \(k\) singular values and their corresponding left and right singular vectors. In information retrieval, singular value decomposition of a term-by-document matrix is also known as latent semantic indexing (LSI).

Applications in Text Mining

Let \(A \in \mathbb{R}^{m \times n}\) be a term-by-document matrix, where \(m\) is the number of terms and \(n\) is the number of documents. The SVD statement has two main functions: to calculate a truncated singular value decomposition (SVD) of \(A\), and to project the columns of \(A\) onto the left singular vectors to generate a new representation of the documents that has a much lower dimensionality. The output of the SVD statement is a truncated singular value decomposition of \(A\), for which the parameter \(k\) defines how many singular values and singular vectors to compute. Singular value decomposition reduces the dimension of the term-by-document matrix and reveals themes that are present in the document collection.

In general, the value of \(k\) must be large enough to capture the meaning of the document collection, yet small enough to ignore the noise. You can specify this value explicitly in the K= option or accept a value that is recommended by the TEXTMINE procedure. A value between 50 and 200 should work well for a document collection that contains thousands of documents.

An important purpose of singular value decomposition is to reduce a high-dimensional term-by-document matrix into a low-dimensional representation that reveals information about the document collection. The columns of the \(A\) form the coordinates of the document space, and the rows form the coordinates of the term space. Each document in the collection is represented as a vector in \(m\)-dimensional space and each term as a vector in \(n\)-dimensional space. The singular value decomposition captures this same information by using a smaller number of basis vectors than would be necessary if you analyzed \(A\) directly.

For example, consider the columns of \(A\), which represent the document space. By construction, the columns of \(U\) also reside in \(m\)-dimensional space. If \(U\) has only one column, the line between that vector and the origin would form the best fit line, in a least squares sense, to the original document space. If \(U\) has two columns, then these columns would form the best fit plane to the original document space. In general, the first \(k\) columns of \(U\) form the best fit \(k\)-dimensional subspace for the document space. Thus, you can project the columns of \(A\) onto the first \(k\) columns of \(U\) in order to optimally reduce the dimension of the document space from \(m\) to \(k\).

The projection of a document \(d\) (one column of \(A\)) onto \(U\) results in \(k\) real numbers that are defined by the inner product \(d\) with each column of \(U\). That is, \(p_i = d^T u_i\). With this representation, each document forms
a $k$-dimensional vector that can be considered a theme in the document collection. You can then calculate the Euclidean distance between each document and each column of $U$ to determine the documents that are described by this theme.

In a similar fashion, you can repeat the previous process by using the rows of $A$ and the first $k$ columns of $V$. This generates a best fit $k$-dimensional subspace for the term space. This representation is used to group terms into similar clusters. These clusters also represent concepts that are prevalent in the document collection. Thus, singular value decomposition can be used to cluster both the terms and the documents into meaningful representations of the entire document collection.

**Computation**

The computation of the singular vector decomposition is fully parallelized in PROC TEXTMINE via multithreading and distributed computing. Computing singular value decomposition is an iterative process that involves considerable communication among the computer nodes in a distributed computing environment. Therefore, adding more computer nodes for computing singular value decomposition might not always improve efficiency. Conversely, when the data size is not large enough, adding too many computer nodes for computation might lead to a noticeable increase in communication time and sometimes might even slow down the overall computation.

**SVD-Only Mode**

If you run PROC TEXTMINE without a PARSE statement (called SVD-only mode), PROC TEXTMINE directly takes the term-by-document matrix as input and computes singular value decomposition (SVD). This functionality enables you to parse documents and compute the SVD separately in two procedure calls. This approach is useful when you want to try different parameters for SVD computation after document parsing. When you run PROC TEXTMINE in SVD-only mode, the DATA= option in the PROC TEXTMINE statement names the data table that contains the term-by-document matrix.

**Topic Discovery**

You can use the TEXTMINE procedure to discover topics that exist in your collection. In PROC TEXTMINE, topics are calculated as a “rotation” of the SVD dimensions in order to maximize the sum of squares of the term loadings in the $V$ matrix. This rotation preserves the spatial information that the SVD provides, but it also allows the newly rotated SVD dimensions to become semantically interpretable. Topics are characterized by a set of weighted terms. Documents that contain many of these weighted terms are highly associated with the topic, and documents that contain few of them are less associated with the topic. The term scores are found in the $U$ matrix that has been rotated to maximize the explanatory power of each topic. The columns of the $V$ matrix characterize the strength of the association of each document with each topic. Finally, the TEXTMINE procedure can output a topic table that contains the best set of descriptor terms for each topic.

Because topic discovery is derived from the $U$ matrix of SVD (each column of the $U$ matrix is rotated and corresponds to a topic), topic discovery options are specified in the SVD statement.
Output Data Tables

This section describes the output data tables that PROC TEXTMINE produces when you specify the corresponding option.

The OUTCHILD= Data Table

The OUTCHILD= option in the PARSE statement specifies the data table to contain a compressed representation of the sparse term-by-document matrix, which is usually very sparse. To save space, this matrix is stored in COO format.

If you do not specify the SHOWDROPPEDTERMS option in the PARSE statement, this data table saves only the kept terms.\(^1\)

The child frequencies are not attributed to their corresponding parent (as they are in the data table specified in the OUTPARENT= option). Using the example in the previous section, the data table that is generated by the OUTCHILD= option will have two entries:

\[
\begin{align*}
\text{t1} & \quad \text{d1} & \quad 8 \\
\text{t2} & \quad \text{d1} & \quad 1
\end{align*}
\]

The term count of “said” in \textit{d1} is not attributed to its parent, “say.” The data table that is specified in the OUTCHILD= option can be combined with the data table that is specified in the OUTTERMS= option to construct the data table that is specified in the OUTPARENT= option.

When you specify the SHOWDROPPEDTERMS option in the PARSE statement, the data table saves all the terms that appear in the data table that is specified in the OUTTERMS= option in the PARSE statement.

The OUTCONFIG= Data Table

The OUTCONFIG= option in the PARSE statement specifies a SAS data table to contain the configuration that PROC TEXTMINE uses in the current run. The primary purpose of this data table is to relay the configuration information from the TEXTMINE procedure to the TMSCORE procedure so that the TMSCORE procedure can use options that are consistent with the TEXTMINE procedure during scoring.

Table 24.9 shows the configuration information that is contained in this data table.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Indicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Source language of the documents</td>
</tr>
<tr>
<td>Stemming</td>
<td>Whether stemming is used: “Y” indicates that stemming is used, and “N” indicates that it is not used</td>
</tr>
<tr>
<td>Tagging</td>
<td>Whether tagging is used: “Y” indicates that tagging is used, and “N” indicates that it is not used</td>
</tr>
<tr>
<td>NG</td>
<td>Whether noun grouping is used: “Y” indicates that noun grouping is used, and “N” indicates that it is not used</td>
</tr>
</tbody>
</table>

\(^1\)Kept terms are terms that are marked as kept in the data table specified in the OUTTERMS= option in the PARSE statement.
Table 24.9  continued

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entities</td>
<td>Whether entities should be extracted: “STD” indicates that entities should be extracted, and “N” indicates that entities should not be extracted. When the SELECT statement is specified, “K” indicates that entities are kept, and “D” indicates that entities are ignored.</td>
</tr>
<tr>
<td>Multiterm</td>
<td>The name of the multiterm SAS data table</td>
</tr>
<tr>
<td>Cellwgt</td>
<td>How the cells of the term-by-document matrix are weighted</td>
</tr>
</tbody>
</table>

The contents of this data table are case-sensitive.

The OUTDOCPRO= Data Table

The OUTDOCPRO= option in the SVD statement specifies a SAS data table to contain the projections of the columns of the term-by-document matrix onto the columns of U. Because each column of the term-by-document matrix corresponds to a document, the output forms a new representation of the input documents in a space that has much lower dimensionality. If the K= option in the SVD statement is set to k and the input data table contains n documents, the output will have n rows and k + 1 columns. Each row of the output corresponds to a document. The first column of the output contains the ID of the documents, and the name of the column is the same as the variable that is specified in the DOC_ID statement. The remaining k columns are the projections and are named “COL1” to “COLk.”

The OUTPARENT= Data Table

The OUTPARENT= option in the PARSE statement specifies a SAS data table to contain a compressed representation of the sparse term-by-document matrix. The term-by-document matrix is usually very sparse. To save space, this matrix is stored in COO format.

This data table contains three columns: _TERMNUM_, _DOCUMENT_, and _COUNT_. The _TERMNUM_ column contains the ID of the terms (which corresponds to the “Key” column of the data table that is generated by the OUTTERMS= option), the _DOCUMENT_ column contains the ID of the documents, and the _COUNT_ column contains the term counts. For example, (t1 d1 k) means that term t1 appears k times in document d1.

The term counts can be weighted, if requested. The data table saves only the terms that are marked as kept in the data table that is specified in the OUTTERMS= option in the PARSE statement. In the data table, the child frequencies are attributed to the corresponding parent. For example, assume that “said” has term ID t1 and appears eight times in document d1, “say” has term ID t2 and appears one time in document d1, “say” is the parent of “said”, and neither cell weighting nor term weighting is applied. Then the data table that is specified in the OUTPARENT= option will contain the following entry:

\[
t2 \quad d1 \quad 9
\]

The term count of “said” in d1 is attributed to its parent, “say.”

---

2Many elements of the matrix are 0.
The OUTPOS= Data Table

The OUTPOS= option in the PARSE statement specifies a SAS data table to contain the position information about the child terms’ occurrences in the document collection. Table 24.10 shows the variables in this data table.

Table 24.10  Variables in the OUTPOS= Data Table

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term</td>
<td>A lowercase version of the term</td>
</tr>
<tr>
<td>Role</td>
<td>The term’s part of speech (this variable is empty if the NOTAGGING option is specified in the PARSE statement)</td>
</tr>
<tr>
<td>Parent</td>
<td>A lowercase version of the parent term</td>
</tr>
<tr>
<td><em>Start</em></td>
<td>The starting position of the term’s occurrence (the first position is 0)</td>
</tr>
<tr>
<td><em>End</em></td>
<td>The ending position of the term’s occurrence</td>
</tr>
<tr>
<td>Sentence</td>
<td>The sentence where the occurrence appears</td>
</tr>
<tr>
<td>Paragraph</td>
<td>The paragraph where the occurrence appears (this has not been implemented in the current release, and the value is always set to 0)</td>
</tr>
<tr>
<td>Document</td>
<td>The ID of the document where the occurrence appears</td>
</tr>
<tr>
<td>Target</td>
<td>The value of the target variable that is associated with the document ID if a variable is specified in the TARGET statement</td>
</tr>
</tbody>
</table>

If you exclude terms by specifying the IGNORE option in the SELECT statement, then those terms are excluded from the OUTPOS= data table. No synonym lists, start lists, or stop lists are used when generating the OUTPOS= data table.

The OUTTERMS= Data Table

The OUTTERMS= option in the PARSE statement specifies a SAS data table to contain the summary information about the terms in the document collection. Table 24.11 shows the variables in this data table.

Table 24.11  Variables in the OUTTERMS= Data Table

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term</td>
<td>A lowercase version of the term</td>
</tr>
<tr>
<td>Role</td>
<td>The term’s part of speech (this variable is empty if the NOTAGGING option is specified in the PARSE statement)</td>
</tr>
<tr>
<td>Attribute</td>
<td>An indication of the characters that compose the term. Possible attributes are as follows:</td>
</tr>
<tr>
<td></td>
<td>Alpha       only alphabetic characters</td>
</tr>
<tr>
<td></td>
<td>Mixed       a combination of attributes</td>
</tr>
<tr>
<td></td>
<td>Num         only numbers</td>
</tr>
<tr>
<td></td>
<td>Punct       punctuation characters</td>
</tr>
<tr>
<td></td>
<td>Entity      an identified entity</td>
</tr>
</tbody>
</table>
Table 24.11  continued

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freq</td>
<td>The frequency of a term in the entire document collection</td>
</tr>
<tr>
<td>Numdocs</td>
<td>The number of documents that contain the term</td>
</tr>
<tr>
<td>_keep</td>
<td>The keep status of the term: “Y” indicates that the term is kept for analysis, and “N” indicates that the term should be dropped in later stages of analysis. To ensure that the OUTTERMS= data table is of a reasonable size, only terms that have _keep=Y are kept in the OUTTERMS= data table by default.</td>
</tr>
<tr>
<td>Key</td>
<td>The assigned term number (each unique term in the parsed documents and each unique parent term has a unique Key value)</td>
</tr>
<tr>
<td>Parent</td>
<td>The Key value of the term’s parent or a “.” (period):</td>
</tr>
<tr>
<td>Parent_id</td>
<td>Another description of the term’s parent: Parent contains the parent’s term number if a term is a child, but Parent_id contains this value for all terms.</td>
</tr>
<tr>
<td>_ispar</td>
<td>An indication of term’s status as a parent, child, or neither:</td>
</tr>
<tr>
<td></td>
<td>• A “+” (plus sign) indicates that the term is a parent.</td>
</tr>
<tr>
<td></td>
<td>• A “.” (period) indicates that the term is a child.</td>
</tr>
<tr>
<td></td>
<td>• A missing value indicates that the term is neither a parent nor a child.</td>
</tr>
<tr>
<td>Weight</td>
<td>The weights of the terms</td>
</tr>
</tbody>
</table>

If you do not specify the SHOWDROPPEDTERMS option in the PARSE statement, this data table saves only the terms that have _keep=Y. This helps ensure that the OUTTERMS= data table is of a reasonable size. When you specify the SHOWDROPPEDTERMS option, the data table also saves terms that have _keep=N.
**The OUTTOPICS= Data Table**

The OUTTOPICS= option specifies the data table for storing the topics that have been discovered. This data table contains three columns: _topicid, _termCutoff, and _name. If the K= option in the SVD statement is set to \( k \), the _topicid column contains the topic index, which is an integer from 1 to \( k \). The _termCutoff column contains the cutoff value that is recommended in order to determine which terms actually belong to the topic. The weights for the terms and topics are contained in V matrix, which is stored in the data table that is specified in the SVDV= option in the SVD statement. The _name column contains the generated topic name, which is the descriptive label for each topic and provides a synopsis of the discovered topics. The generated topic name contains the terms that have the highest term loadings after the rotation has been performed. The number of terms that are used in the generated name is determined by the NUMLABELS= option in the SVD statement.

---

**Examples: TEXTMINE Procedure**

---

**Example 24.1: Parsing with No Options Turned On**

This example parses five documents, which are in a generated data table. The following DATA step generates the five documents:

```plaintext
/* 1) create data table */

data mycas.CarNominations;
infile datalines delimiter='|' missover;
length text $70 ;
input text$ i;
datalines;
The Ford Taurus is the World Car of the Year. |1
Hyundai won the award last year. |2
Toyota sold the Toyota Tacoma in bright green. |3
The Ford Taurus is sold in all colors except for lime green. |4
The Honda Insight was World Car of the Year in 2008. |5
;
run;
```

The following statements run PROC TEXTMINE to parse the documents.

```plaintext
/* 2) starting code */
proc textmine data=mycas.CarNominations;
doc_id i;
var text;
parse
  nostemming notagging nonoungroups
termwgt   = none
cellwgt   = none
reducef   = 1
entities  = none
outparent = mycas.outparent
outterms  = mycas.outterms
```
outchild = mycas.outchild
outconfig = mycas.outconfig;
run;

/* 3) print outterms data table */
data outterms; set mycas.outterms; run;
proc print data=outterms; run;

Output 24.1.1 shows the content of the mycas.outterms data table. In this example, stemming, part-of-speech tagging, and noun group extraction are suppressed and NONE is specified for entity identification, term and cell weighting, and term filtering. No synonym list, multiterm list, or stop list is specified. As a result of this configuration, there is no child term in the mycas.outterms data table. Also, the mycas.outparent data table and the mycas.outchild data table are exactly the same. The TEXTMINE procedure automatically drops punctuation and numbers.

Output 24.1.1 The mycas.outterms Data Table

<table>
<thead>
<tr>
<th>Obs</th>
<th>Term</th>
<th>Role</th>
<th>Attribute</th>
<th>Freq</th>
<th>numdocs</th>
<th>_keep</th>
<th>Key</th>
<th>Parent</th>
<th>Parent_id</th>
<th>_ispar</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>all</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>toyota</td>
<td>Alpha</td>
<td>2</td>
<td>1</td>
<td>Y</td>
<td>2</td>
<td>.</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ford</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>3</td>
<td>.</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>tacoma</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>4</td>
<td>.</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>year</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>5</td>
<td>.</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>taurus</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>6</td>
<td>.</td>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>won</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>honda</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>8</td>
<td>.</td>
<td>8</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>bright</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>9</td>
<td>.</td>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>sold</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>10</td>
<td>.</td>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>colors</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>11</td>
<td>.</td>
<td>11</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>lime</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>12</td>
<td>.</td>
<td>12</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>except</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>hyundai</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>in</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>is</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>for</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>17</td>
<td>.</td>
<td>17</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>world</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>green</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>19</td>
<td>.</td>
<td>19</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>the</td>
<td>Alpha</td>
<td>8</td>
<td>5</td>
<td>Y</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>of</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>21</td>
<td>.</td>
<td>21</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>award</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>was</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>car</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>insight</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>25</td>
<td>.</td>
<td>25</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>last</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>26</td>
<td>.</td>
<td>26</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 24.2: Parsing with Stemming

This example uses the data table that is generated in Example 24.1. The following statements run PROC TEXTMINE to parse the documents. Because the NOSTEMMING option is not specified in the PARSE statement, words are stemmed (the default).

```sas
/* create data table */
data mycas.CarNominations;
infile datalines delimiter='|' missover;
length text $70 ;
input text$ i;
datalines;
The Ford Taurus is the World Car of the Year. |1
Hyundai won the award last year. |2
Toyota sold the Toyota Tacoma in bright green. |3
The Ford Taurus is sold in all colors except for lime green. |4
The Honda Insight was World Car of the Year in 2008. |5
;
run;

proc textmine data=mycas.CarNominations;
doc_id i;
var text;
parse
   notagging nonoungroups
   termwgt = none
   cellwgt = none
   reducef = 1
   entities = none
   outparent= mycas.outparent
   outterms = mycas.outterms
   outchild = mycas.outchild
   outconfig= mycas.outconfig
;
run;
data outterms; set mycas.outterms; run;
proc print data = outterms; run;
```

Output 24.2.1 shows the content of the mycas.outterms data table. In this example, words are stemmed. You can see that the term “sold” now stems to the parent term “sell.” Also, the mycas.outparent data table and the mycas.outchild data table are different. The parent term “sell” shows up in mycas.outparent (key=11), but not the child term “sold” (key=27). Only “sold” appears in the mycas.outchild data table, and “sell” does not appear.
### Output 24.2.1 The mycas.outputms Data Table with Stemming

<table>
<thead>
<tr>
<th>Obs</th>
<th>Term</th>
<th>Role</th>
<th>Attribute</th>
<th>Freq numdocs</th>
<th>_keep</th>
<th>Key</th>
<th>Parent</th>
<th>Parent_id</th>
<th>_ispars</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>all</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>win</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>2</td>
<td>.</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>toyota</td>
<td>Alpha</td>
<td>2</td>
<td>1 Y</td>
<td>3</td>
<td>.</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ford</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>4</td>
<td>.</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>tacoma</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>5</td>
<td>.</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>year</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>6</td>
<td>.</td>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>taurus</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>won</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>26</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>honda</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>8</td>
<td>.</td>
<td>8</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>bright</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>9</td>
<td>.</td>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>be</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>10</td>
<td>.</td>
<td>10</td>
<td>+ 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>sold</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>27</td>
<td>11</td>
<td>11</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>sell</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>11</td>
<td>.</td>
<td>11</td>
<td>+ 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>colors</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>28</td>
<td>23</td>
<td>23</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>lime</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>12</td>
<td>.</td>
<td>12</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>except</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>hyundai</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>in</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>is</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>29</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>for</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>world</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>17</td>
<td>.</td>
<td>17</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>green</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>the</td>
<td>Alpha</td>
<td>8</td>
<td>5 Y</td>
<td>19</td>
<td>.</td>
<td>19</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>of</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>award</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>21</td>
<td>.</td>
<td>21</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>was</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>car</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>color</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td>+ 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>insight</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>last</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>25</td>
<td>.</td>
<td>25</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 24.3: Adding Entities and Noun Groups

This example uses the data table that is generated in Example 24.1. The following statements run PROC TEXTMINE to parse the documents. Because the NONOUNGROUPS option is not specified in the PARSE statement, noun groups are extracted, and because the ENTITIES=STD option is specified, entities are identified.

```sas
/* create data table */
data mycas.CarNominations;
infile datalines delimiter='|' missover;
length text $70 ;
input text$ i;
datalines;
  The Ford Taurus is the World Car of the Year. |1
  Hyundai won the award last year. |2
  Toyota sold the Toyota Tacoma in bright green. |3
  The Ford Taurus is sold in all colors except for lime green. |4
  The Honda Insight was World Car of the Year in 2008. |5
; run;

proc textmine data=mycas.CarNominations;
doc_id i;
var text i;
PARSE
  notagging
  termwgt = none
  cellwgt = none
  reducef = 1
  entities = std
  outparent = mycas.outparent
  outterms = mycas.outterms
  outchild = mycas.outchild
  outconfig = mycas.outconfig
; run;
data outterms; set mycas.outterms; run;
proc print data=outterms; run;
```

Output 24.3.1 shows the content of the mycas.outterms data table. Compared to Output 24.2.1, the mycas.outterms data table is longer, because it contains entities and noun groups.
### Output 24.3.1 The mycas.outterms Data Table with Noun Group Extraction and Entity Identification

<table>
<thead>
<tr>
<th>Obs</th>
<th>Term</th>
<th>Role</th>
<th>Attribute</th>
<th>Freq</th>
<th>numdocs_keep</th>
<th>Key</th>
<th>Parent</th>
<th>Parent_id_ispar</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>all</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>win</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>2</td>
<td>.</td>
<td>2 +</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>tacoma</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>3</td>
<td>.</td>
<td>3</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>year</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>4</td>
<td>.</td>
<td>4</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>taurus</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>5</td>
<td>.</td>
<td>5</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>lime green</td>
<td>nlpNounGroup</td>
<td>Alpha</td>
<td>1 Y</td>
<td>6</td>
<td>.</td>
<td>6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>won</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>30</td>
<td>2</td>
<td>2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>honda</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>bright</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>8</td>
<td>.</td>
<td>8</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>in 2008</td>
<td>nlpDate</td>
<td>Entity</td>
<td>1 Y</td>
<td>9</td>
<td>.</td>
<td>9</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>be</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>10</td>
<td>.</td>
<td>10 +</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>sold</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>31</td>
<td>12</td>
<td>12</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>bright green</td>
<td>nlpNounGroup</td>
<td>Alpha</td>
<td>1 Y</td>
<td>11</td>
<td>.</td>
<td>11</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>sell</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>12</td>
<td>.</td>
<td>12 +</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>colors</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>32</td>
<td>27</td>
<td>27</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>lime</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>hyundai</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>1 Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>except</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>in</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>is</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>33</td>
<td>10</td>
<td>10</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>toyota</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2 Y</td>
<td>17</td>
<td>.</td>
<td>17</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>last year</td>
<td>nlpDate</td>
<td>Entity</td>
<td>1 Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>ford</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2 Y</td>
<td>19</td>
<td>.</td>
<td>19</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>for</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>world</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>21</td>
<td>.</td>
<td>21</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>green</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>the</td>
<td>Alpha</td>
<td>8</td>
<td>5 Y</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>of</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>award</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>25</td>
<td>.</td>
<td>25</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>was</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>34</td>
<td>10</td>
<td>10</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>car</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>26</td>
<td>.</td>
<td>26</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>color</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>27</td>
<td>.</td>
<td>27 +</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>insight</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>28</td>
<td>.</td>
<td>28</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>last</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>29</td>
<td>.</td>
<td>29</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Example 24.4: Adding Part-of-Speech Tagging

This example uses the data table that is generated in Example 24.1. The following statements run PROC TEXTMINE to parse the documents. Because the NOTAGGING option is not specified in the PARSE statement, PROC TEXTMINE uses context clues to determine a term’s part of speech.

```sas
/* create data table */
data mycas.CarNominations;
  infile datalines delimiter='|' missover;
  length text $70 ;
  input text$ i;
datalines;
  The Ford Taurus is the World Car of the Year. |1
  Hyundai won the award last year. |2
  Toyota sold the Toyota Tacoma in bright green. |3
  The Ford Taurus is sold in all colors except for lime green. |4
  The Honda Insight was World Car of the Year in 2008. |5
; run;

proc textmine data=mycas.CarNominations;
  doc_id i;
  var text;
  parse
    termwgt = none
    cellwgt = none
    reducef = 1
    entities = std
    outparent = mycas.outparent
    outterms = mycas.outterms
    outchild = mycas.outchild
    outconfig = mycas.outconfig
  ; run;
data outterms; set mycas.outterms; run;
proc print data= outterms; run;
```

Output 24.4.1 shows the content of the mycas.outterms data table. Compared to Output 24.3.1, the mycas.outterms data table also contains the part-of-speech tag for the terms.
## Output 24.4.1 The mycas.outterms Data Table with Part-of-Speech Tagging

<table>
<thead>
<tr>
<th>Obs</th>
<th>Term</th>
<th>Role</th>
<th>Attribute</th>
<th>Freq</th>
<th>numdocs</th>
<th><em>keep</em></th>
<th>Key</th>
<th>Parent</th>
<th>Parent_id</th>
<th><em>ispar</em></th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>is</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>30</td>
<td>26</td>
<td>26</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>was</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>31</td>
<td>26</td>
<td>26</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>bright</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>taurus</td>
<td>N</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>2</td>
<td>.</td>
<td>2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>won</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>32</td>
<td>17</td>
<td>17</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>for</td>
<td>PPOS</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>3</td>
<td>.</td>
<td>3</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>lime green</td>
<td>nlpNounGroup</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>4</td>
<td>.</td>
<td>4</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>lime</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>5</td>
<td>.</td>
<td>5</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>in 2008</td>
<td>nlpDate</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>6</td>
<td>.</td>
<td>6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>the</td>
<td>DET</td>
<td>Alpha</td>
<td>8</td>
<td>5</td>
<td>Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>except</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>8</td>
<td>.</td>
<td>8</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>bright green</td>
<td>nlpNounGroup</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>9</td>
<td>.</td>
<td>9</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>color</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>10</td>
<td>.</td>
<td>10</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>in</td>
<td>PPOS</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>11</td>
<td>.</td>
<td>11</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>hyundai</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>12</td>
<td>.</td>
<td>12</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>sold</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>28</td>
<td>33</td>
<td>28</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>toyota</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>1</td>
<td>Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>last year</td>
<td>nlpDate</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>ford</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>all</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>win</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>17</td>
<td>.</td>
<td>17</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>car</td>
<td>PN</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>colors</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>34</td>
<td>10</td>
<td>10</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>award</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>19</td>
<td>.</td>
<td>19</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>insight</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>of</td>
<td>PPOS</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>21</td>
<td>.</td>
<td>21</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>honda</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>world</td>
<td>PN</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>last</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>green</td>
<td>N</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>25</td>
<td>.</td>
<td>25</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>be</td>
<td>V</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>26</td>
<td>.</td>
<td>26</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>tacoma</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>27</td>
<td>.</td>
<td>27</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>sell</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>28</td>
<td>.</td>
<td>28</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>year</td>
<td>N</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>29</td>
<td>.</td>
<td>29</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Example 24.5: Adding Synonyms

This example uses the data table that is generated in Example 24.1. So far, by looking at the mycas.outterms data tables that are generated by Example 24.1 to Example 24.4, you can see that the data are very “vehicle focused.” If what is important to you is whether or not a car is mentioned in the text, and not the particular model, then you can use a synonym list to map each vehicle model to the broader term “car”. The following DATA step generates the synonym list, and the following statements show this mapping:

```sas
/* create data table */
data mycas.CarNominations;
infile datalines delimiter='|' missover;
length text $70 ;
input text$ i;
datalines;
The Ford Taurus is the World Car of the Year. |1
Hyundai won the award last year. |2
Toyota sold the Toyota Tacoma in bright green. |3
The Ford Taurus is sold in all colors except for lime green. |4
The Honda Insight was World Car of the Year in 2008. |5
;
run;

/* create synonym list */
data mycas.synds;
infile datalines delimiter=',';
length Term $13;
input Term $ TermRole $ Parent $ ParentRole$;
datalines;
   insight, PN, car, N,
   taurus, N, car, N,
   tacoma, PN, car, N,
;
run;

proc textmine data=mycas.CarNominations;
doc_id i;
var text;
parse
   termwgt = none
cellwgt = none
reducef = 1
entities = std
synonym = mycas.synds
outparent = mycas.outparent
outterms = mycas.outterms
outchild = mycas.outchild
outconfig = mycas.outconfig
;
RUN;

data outterms; set mycas.outterms; run;
proc print data= outterms; run;
```
Output 24.5.1 shows the content of the mycas.outterms data table. You can see that the term “insight” is assigned the parent term “car”. Only the term “car” appears in the mycas.outparent data table.

**Output 24.5.1 The mycas.outterms Data Table with Synonym Mapping**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Term</th>
<th>Role</th>
<th>Attribute</th>
<th>Freq</th>
<th>numdocs</th>
<th>_keep</th>
<th>Key</th>
<th>Parent</th>
<th>Parent_id</th>
<th>_ispar</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>is</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>28</td>
<td>25</td>
<td>25</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>was</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>29</td>
<td>25</td>
<td>25</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>bright</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>taurus</td>
<td>N</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>30</td>
<td>2</td>
<td>2</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>car</td>
<td>N</td>
<td>Alpha</td>
<td>4</td>
<td>4</td>
<td>Y</td>
<td>2</td>
<td>.</td>
<td>2 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>won</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>31</td>
<td>17</td>
<td>17</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>for</td>
<td>PPOS</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>5</td>
<td>.</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>lime green</td>
<td>nlpNounGroup</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>4</td>
<td>.</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>lime</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>5</td>
<td>.</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>in 2008</td>
<td>nlpDate</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>6</td>
<td>.</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>the</td>
<td>DET</td>
<td>Alpha</td>
<td>8</td>
<td>5</td>
<td>Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>except</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>8</td>
<td>.</td>
<td>8</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>bright green</td>
<td>nlpNounGroup</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>9</td>
<td>.</td>
<td>9</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>color</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>10</td>
<td>.</td>
<td>10 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>in</td>
<td>PPOS</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>11</td>
<td>.</td>
<td>11</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>hyundai</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>12</td>
<td>.</td>
<td>12</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>sold</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>32</td>
<td>26</td>
<td>26</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>toyota</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>1</td>
<td>Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>last year</td>
<td>nlpDate</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>ford</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>all</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>win</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>17</td>
<td>.</td>
<td>17 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>car</td>
<td>PN</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>colors</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>33</td>
<td>10</td>
<td>10 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>award</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>19</td>
<td>.</td>
<td>19</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>insight</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>34</td>
<td>2</td>
<td>2 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>of</td>
<td>PPOS</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>honda</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>21</td>
<td>.</td>
<td>21</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>world</td>
<td>PN</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>last</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>green</td>
<td>N</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>be</td>
<td>V</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>25</td>
<td>.</td>
<td>25 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>tacoma</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>35</td>
<td>2</td>
<td>2 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>sell</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>26</td>
<td>.</td>
<td>26 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>year</td>
<td>N</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>27</td>
<td>.</td>
<td>27</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Example 24.6: Adding a Custom Stop List

This example uses the data table that is generated in Example 24.1 and uses a stop list to drop the term “car” functioning as a proper noun.

```sas
/* create data table */
data mycas.CarNominations;
infile datalines delimiter='|' missover;
length text $70;
input text$ i;
datalines;
The Ford Taurus is the World Car of the Year. |1
Hyundai won the award last year. |2
Toyota sold the Toyota Tacoma in bright green. |3
The Ford Taurus is sold in all colors except for lime green. |4
The Honda Insight was World Car of the Year in 2008. |5
;
run;

data mycas.newStopList;
length Term $16 TermRole $16;
infile datalines delimiter=',';
input Term $ TermRole $;
datalines;
   car, PN,
;
run;

proc textmine data=mycas.CarNominations;
doc_id i;
var text;
parse
termwgt = none
cellwgt = none
reducef = 1
entities = std
stop = mycas.newStopList
outparent = mycas.outparent
outterms = mycas.outterms
outchild = mycas.outchild
outconfig = mycas.outconfig
;
run;

data outterms; set mycas.outterms; run;
proc print data= outterms; run;
```

Output 24.6.1 shows the content of the mycas.outterms data table. You can see that the term “car, PN” is not in the mycas.outterms data table because that term and role were added to the custom stop list.
## Output 24.6.1 The mycas.outterms Data Table Filtered Using Stop List

<table>
<thead>
<tr>
<th>Obs</th>
<th>Term</th>
<th>Role</th>
<th>Attribute</th>
<th>Freq</th>
<th>numdocs_keep</th>
<th>Key</th>
<th>Parent</th>
<th>Parent_id_ispar</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>is</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>29</td>
<td>25</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>was</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>30</td>
<td>25</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>bright</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>taurus</td>
<td>N</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>2</td>
<td>.</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>won</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>31</td>
<td>17</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>for</td>
<td></td>
<td>PPOS</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>3</td>
<td>.</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>lime green</td>
<td>nlpNounGroup</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>4</td>
<td>.</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>lime</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>5</td>
<td>.</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>in 2008</td>
<td></td>
<td>nlpDate</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>6</td>
<td>.</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>the</td>
<td>DET</td>
<td>Alpha</td>
<td>8</td>
<td>5 Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>except</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>8</td>
<td>.</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>bright green</td>
<td>nlpNounGroup</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>9</td>
<td>.</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>color</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>10</td>
<td>.</td>
<td>10 +</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>in</td>
<td>PPOS</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>11</td>
<td>.</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>hyundai</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>12</td>
<td>.</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>sold</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>32</td>
<td>27</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>toyota</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>1 Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>last year</td>
<td>nlpDate</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>ford</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>2 Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>all</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>win</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>17</td>
<td>.</td>
<td>17 +</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>colors</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>33</td>
<td>10</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>award</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>insight</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>19</td>
<td>.</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>of</td>
<td>PPOS</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>honda</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>21</td>
<td>.</td>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>world</td>
<td>PN</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>last</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>green</td>
<td>N</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>be</td>
<td>V</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>25</td>
<td>.</td>
<td>25 +</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>tacoma</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>26</td>
<td>.</td>
<td>26</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>sell</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>27</td>
<td>.</td>
<td>27 +</td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>year</td>
<td>N</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>28</td>
<td>.</td>
<td>28</td>
<td>1</td>
</tr>
</tbody>
</table>
Example 24.7: Adding a Multiterm List

You can specify a multiterm list to define terms that consist of multiple words. This example uses the data table that is generated in Example 24.1 to show how to use the MULTITERM= option. The following DATA steps generate and uses a multiterm list:

```plaintext
/* create data table */
data mycas.CarNominations;
infile datalines delimiter='|' missover;
  length text $70 ;
  input text$ i;
  datalines;
    The Ford Taurus is the World Car of the Year. |1
    Hyundai won the award last year. |2
    Toyota sold the Toyota Tacoma in bright green. |3
    The Ford Taurus is sold in all colors except for lime green. |4
    The Honda Insight was World Car of the Year in 2008. |5
  ;
run;

/* create multiterm list */
data mycas.multiterms;
infile datalines delimiter='|';
  length multiterm $64;
  input multiterm$;
  datalines;
    except for :3:Prep
  ;
run;

proc textmine data=mycas.CarNominations;
doc_id i;
var text;
parse
termwgt = none
cellwgt = none
reducef = 1
entities = std
multiterm = mycas.multiterms
outparent = mycas.outparent
outterms = mycas.outterms
outchild = mycas.outchild
outconfig = mycas.outconfig
;
run;

data outterms; set mycas.outterms; run;
proc print data=outterms; run;
```

Output 24.7.1 shows the content of the mycas.outterms data table. In the preceding statements, “except for” is defined as an individual term in the third DATA step. In the mycas.outterms data table, you can see that the two terms “except” and “for” have become one term, “except for.”
### Output 24.7.1 The mycas.outterms Data Table Using a Multiterm List

<table>
<thead>
<tr>
<th>Obs</th>
<th>Term</th>
<th>Role</th>
<th>Attribute</th>
<th>Freq</th>
<th>numdocs</th>
<th>keep</th>
<th>Key</th>
<th>Parent</th>
<th>Parent_id</th>
<th>_ispar</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>is</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>29</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>was</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>30</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>bright</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>taurus</td>
<td>N</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>2</td>
<td></td>
<td>2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>won</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>31</td>
<td>16</td>
<td>16</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>lime green</td>
<td>nlpNounGroup</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>except for</td>
<td>PPOS</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>lime</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>5</td>
<td></td>
<td>5</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>in 2008</td>
<td>nlpDate</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>the</td>
<td>DET</td>
<td>Alpha</td>
<td>8</td>
<td>5</td>
<td>Y</td>
<td>7</td>
<td></td>
<td>7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>bright green</td>
<td>nlpNounGroup</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>color</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>9</td>
<td></td>
<td>9</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>in</td>
<td>PPOS</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>10</td>
<td></td>
<td>10</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>hyundai</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>11</td>
<td>11</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>sold</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>32</td>
<td>27</td>
<td>27</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>toyota</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>1</td>
<td>Y</td>
<td>12</td>
<td>12</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>last year</td>
<td>nlpDate</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>13</td>
<td>13</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>ford</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>14</td>
<td>14</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>all</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>15</td>
<td></td>
<td>15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>win</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>16</td>
<td></td>
<td>16</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>car</td>
<td>PN</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>17</td>
<td></td>
<td>17</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>colors</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>33</td>
<td>9</td>
<td>9</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>award</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>18</td>
<td></td>
<td>18</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>insight</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>19</td>
<td></td>
<td>19</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>of</td>
<td>PPOS</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>20</td>
<td></td>
<td>20</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>honda</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>21</td>
<td></td>
<td>21</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>world</td>
<td>PN</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>22</td>
<td></td>
<td>22</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>last</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>23</td>
<td></td>
<td>23</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>green</td>
<td>N</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>24</td>
<td></td>
<td>24</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>be</td>
<td>V</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>25</td>
<td></td>
<td>25</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>tacoma</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>26</td>
<td>26</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>sell</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>27</td>
<td></td>
<td>27</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>year</td>
<td>N</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>28</td>
<td></td>
<td>28</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Example 24.8: Selecting Parts of Speech and Entities to Ignore

This example uses the data table that is generated in Example 24.1. If you want to eliminate prepositions, determiners, and proper nouns from your analysis, you can add a SELECT statement that lists these part-of-speech labels. If you also want to eliminate entities that are labeled “nlpDate,” you can add another SELECT statement that includes “nlpDate” in the label list.

```sas
/* create data table */
data mycas.CarNominations;
infile datalines delimiter='|' missover;
length text $70 ;
input text$ i;
datalines;
The Ford Taurus is the World Car of the Year. |1
Hyundai won the award last year. |2
Toyota sold the Toyota Tacoma in bright green. |3
The Ford Taurus is sold in all colors except for lime green. |4
The Honda Insight was World Car of the Year in 2008. |5
;
run;

proc textmine data=mycas.CarNominations;
doc_id i;
var text;
parse
   termwgt = none
   cellwgt = none
   reducef = 1
   entities = std
   outparent = mycas.outparent
   outterms = mycas.outterms
   outchild = mycas.outchild
   outconfig = mycas.outconfig
   ;
select "PPOS" "DET" "PN"/ignore;
select "nlpDate"/group="entities" ignore;
run;

data outterms; set mycas.outterms; run;
proc print data= outterms; run;
```

Output 24.8.1 shows the content of the mycas.outterms data table. You can see that prepositions, determiners, and proper nouns are excluded. Terms that are labeled “nlpDate” are also excluded.
Output 24.8.1  The mycas.outterms Data Table Ignoring Specified Parts of Speech and Entities

<table>
<thead>
<tr>
<th>Obs</th>
<th>Term</th>
<th>Role</th>
<th>Attribute</th>
<th>Freq</th>
<th>numdocs_keep</th>
<th>Key Parent Parent_id_ispar</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>is</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y 19 16 16 .</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>was</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 20 16 16 .</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>bright</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 1 1 1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>taurus</td>
<td>N</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y 2 2 2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>won</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 21 12 12 .</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>lime green</td>
<td>nlpNounGroup</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 3 3 3</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>lime</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 4 4 4</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>except</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 5 5 5</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>bright green</td>
<td>nlpNounGroup</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 6 6 6</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>color</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 7 7 7 +</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>hyundai</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y 8 8 8</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>sold</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y 22 17 17 .</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>toyota</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>1</td>
<td>Y 9 9 9</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>ford</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y 10 10 10</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>all</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 11 11 11</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>win</td>
<td>V</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 12 12 +</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>colors</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 23 7 7 .</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>award</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 13 13 13</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>last</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y 14 14 14</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>green</td>
<td>N</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y 15 15 15</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>be</td>
<td>V</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y 16 16 +</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>sell</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y 17 17 +</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>year</td>
<td>N</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y 18 18 18</td>
<td>1</td>
</tr>
</tbody>
</table>
Overview: TMSCORE Procedure

The TMSCORE procedure scores textual data in SAS Viya. In text mining, scoring is the process of applying parsing and singular value decomposition (SVD) projections to new textual data. The TMSCORE procedure performs this scoring of new documents, and its primary outputs are the Outparent data table (which holds the parsing results of the term-by-document matrix) and the Outdocpro data table (which holds the reduced-dimensional representation of the score collection). PROC TMSCORE uses some of the output data tables of the TEXTMINE procedure as input data to ensure consistency between scoring and training. During scoring, the new textual data must be parsed using the same settings that the training data were parsed with, indexed using only the subset of terms that were used during training, and projected onto the reduced-dimensional subspace of the singular value decomposition that was derived from the training data. To facilitate this process, you specify the CONFIG=, TERMS=, and SVDU= options in PROC TEXTMINE to create three data tables (Outconfig, Outterms, and Svdu, respectively), and then you specify those three data tables as inputs to PROC TMSCORE. For more information about these data tables, see the CONFIG=, TERMS=, and SVDU= options, respectively, in the section “PROC TMSCORE Statement” on page 620.
PROC TMSCORE Features

The TMSCORE procedure processes large-scale textual data in parallel to achieve efficiency and scalability. The following list summarizes the basic features of PROC TMSCORE:

- Functionalities that are related to document parsing, term-by-document matrix creation, and dimension reduction are integrated into one procedure to process data more efficiently.
- Parsing and term-by-document matrix creation are performed in parallel.
- Computation of document projection is performed in parallel.
- All phases of processing use a high degree of multithreading.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```sas
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```sas
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```sas
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Getting Started: TMSCORE Procedure

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

The following DATA steps generate two data tables: the mycas.getstart data table contains 36 observations, and the mycas.getstart_score data table contains 31 observations. Both data tables have two variables: the text variable contains the input documents, and the did variable contains the ID of the documents. Each row in each data table represents a “document” for analysis.

data mycas.getstart;
  infile datalines delimiter='|' missover;
  length text $150;
  input text$ did;
  datalines;
  High-performance analytics hold the key to unlocking the unprecedented business value of big data. Organizations looking for optimal ways to gain insights from big data in shorter reporting windows are turning to SAS. As the gold-standard leader in business analytics for more than 36 years, SAS frees enterprises from the limitations of traditional computing and enables them to draw instant benefits from big data. Faster Time to Insight. From banking to retail to health care to insurance, SAS is helping industries glean insights from data that once took days or weeks in just hours, minutes or seconds. It's all about getting to and analyzing relevant data faster. Revealing previously unseen patterns, sentiments and relationships. Identifying unknown risks. And speeding the time to insights. High-Performance Analytics from SAS Combining industry-leading analytics software with high-performance computing technologies produces fast and precise answers to unsolvable problems and enables our customers to gain greater competitive advantage. SAS In-Memory Analytics eliminate the need for disk-based processing allowing for much faster analysis. SAS In-Database executes analytic logic into the database itself for improved agility and governance. SAS Grid Computing creates a centrally managed, shared environment for processing large jobs and supporting a growing number of users efficiently. Together, the components of this integrated, supercharged platform are changing the decision-making landscape and redefining how the world solves big data business problems. Big data is a popular term used to describe the exponential growth and availability and use of information, both structured and unstructured. Much has been written on the big data trend and how it can...
serve as the basis for innovation, differentiation and growth. | 36

run;

data mycas.getstart_score;
  infile datalines delimiter='|' missover;
  length text $150;
  input text$ did;
  datalines;
  Big data according to SAS| 1
  At SAS, consider two other dimensions| 2
  when thinking about big data:| 3
  Variability. In addition to the| 4
  increasing velocities and varieties of data, data| 5
  flows can be highly inconsistent with periodic peaks.| 6
  Is something big trending in the social media?| 7
  Perhaps there is a high-profile IPO looming.| 8
  Maybe swimming with pigs in the Bahamas is suddenly| 9
  the must-do vacation activity. Daily, seasonal and|10
  event-triggered peak data loads can be challenging|11
  to manage - especially with social media involved.|12
  Complexity. When you deal with huge volumes of data,|13
  it comes from multiple sources. It is quite an|14
  undertaking to link, match, cleanse and|15
  transform data across systems. However,|16
  it is necessary to connect and correlate|17
  relationships, hierarchies and multiple data|18
  linkages or your data can quickly spiral out of|19
  control. Data governance can help you determine|20
  how disparate data relates to common definitions|21
  and how to systematically integrate structured|22
  and unstructured data assets to produce|23
  high-quality information that is useful,|24
  appropriate and up-to-date.|25
  Ultimately, regardless of the factors involved,|26
  I believe that the term big data is relative|27
  it applies (per Gartner's assessment)|28
  whenever an organization's ability|29
  to handle, store and analyze data|30
  exceeds its current capacity.|31
run;

The following statements use PROC TEXTMINE for processing the input text data table mycas.getstart and create three data tables (mycas.outconfig, mycas.outterms, and mycas.svdu), which can be used in PROC TMSCORE for scoring:

```plaintext
proc textmine data = mycas.getstart;
doc_id did;
variables text;
parse
  outterms = mycas.outterms
  outconfig = mycas.outconfig
  reducef = 2;
svd
  k = 5
```
svdu = mycas.svdu;
run;

The following statements then use PROC TMSCORE to score the input text data table mycas.getstart_score. The statements take the three data tables that are generated by PROC TEXTMINE as input and create a data table named mycas.docpro, which contains the projection of the documents in the input data table mycas.getstart_score.

```
proc tmscore
   data = mycas.getstart_score
   terms = mycas.outterms
   config = mycas.outconfig
   svdu = mycas.svdu
   svddocpro = mycas.docpro;
   doc_id did;
   variables text;
run;
```

The output from this analysis is presented in Figure 25.1.

The following statements use PROC PRINT to show the content of the first 10 rows of the sorted mycas.docpro data table, which is generated by the TMSCORE procedure:

```
data docpro;
   set mycas.docpro;
run;
proc sort data=docpro;
   by did;
run;
proc print data = docpro (obs=10);
run;
```

Figure 25.1 shows the output of PROC PRINT.

```
Obs did  COL1  COL2  COL3  COL4  COL5
 1  1  0.8460041362 -0.022725647  0.1330595299  0.5146460484  0.0345709829
 2  2  0.3312354984  0.573779031  0.0066225814  0.7472313995  0.0515950108
 3  3  0.8520340979 -0.358672789  0.1873407858  0.3187325661 -0.093299024
 4  4  0.64928804  0.2747636778  0.4454014167 -0.316447556 -0.2447128
 5  5  0.9430684788 -0.185746085  0.0903397136  0.0816038571  0.2475879297
 6  6  0.8325586063 -0.174210986 -0.353242685  0.388738294  0.0345709829
 7  7  0.901438766  0.0115370594  0.3626555424  0.1689222234  0.1649887393
 8  8  0.6826827301 -0.004113157 -0.213214441  0.3301557457  0.6160066215
 9  9  0.8548352509  0.2464171755  0.3749249411  0.1417168616  0.218821592
10 10  0.6727152395  0.1569493092  0.0507091334 -0.653034877  0.306259946
```
Syntax: TMSCORE Procedure

The following statements are available in the TMSCORE procedure:

```
PROC TMSCORE DATA=CAS-libref.data-table < options > ;
   VARIABLES variable ;
   DOC_ID variable ;
```

PROC TMSCORE Statement

```
PROC TMSCORE DATA=CAS-libref.data-table < options > ;
```

The PROC TMSCORE statement invokes the procedure. Table 25.1 summarizes the `options` in the statement by function. The `options` are then described fully in alphabetical order.

<table>
<thead>
<tr>
<th><code>option</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic Options</strong></td>
<td></td>
</tr>
<tr>
<td>CONFIG=</td>
<td>Specifies the data table that contains the configuration information</td>
</tr>
<tr>
<td>DATA</td>
<td>DOC=</td>
</tr>
<tr>
<td>SVDU=</td>
<td>Specifies the data table that contains the U matrix whose columns are the left singular vectors</td>
</tr>
<tr>
<td>TERMS=</td>
<td>Specifies the data table that contains the terms to be used for scoring</td>
</tr>
<tr>
<td><strong>Output Options</strong></td>
<td></td>
</tr>
<tr>
<td>OUTPARENT=</td>
<td>Specifies the data table that contains the term-by-document frequency matrix that is used to model the document collection. In this matrix, the child terms are not represented and child terms’ frequencies are attributed to their corresponding parents.</td>
</tr>
<tr>
<td>SVDDOCPRO=</td>
<td>Specifies the data table that contains the projections of the documents</td>
</tr>
</tbody>
</table>

You must specify the following option:

```
DATA=CAS-libref.data-table
DOC=CAS-libref.data-table
```

names the input data table for PROC TMSCORE to use. `CAS-libref.data-table` is a two-level name, where

- `CAS-libref` refers to a collection of information that is defined in the LIBNAME statement and includes the `caslib`, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 616.
PROC TMSCORE Statement ♦ 621

data-table specifies the name of the input data table.

The input data table contains documents for PROC TMSCORE to score. Each row of the input data table must contain one text variable and one ID variable, which correspond to the text and the unique ID of a document, respectively.

You can also specify the following options:

**CONFIG=** CAS-libref.data-table

specifies the input data table that contains configuration information for PROC TMSCORE. **CAS-libref.data-table** is a two-level name, where **CAS-libref** refers to the caslib and session identifier, and **data-table** specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 616. Specify the table that was generated by the **OUTCONFIG=** option in the PARSE statement of the TEXTMINE procedure during training. For more information about this data table, see the section “The OUTCONFIG= Data Table” on page 595 of Chapter 24, “The TEXTMINE Procedure.”

**OUTPARENT=** CAS-libref.data-table

specifies the output data table to contain a compressed representation of the sparse term-by-document frequency matrix. **CAS-libref.data-table** is a two-level name, where **CAS-libref** refers to the caslib and session identifier, and **data-table** specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 616. The data table contains only the kept representative terms, and the child frequencies are attributed to the corresponding parent. For more information about the compressed representation of the sparse term-by-document frequency matrix, see the section “The OUTPARENT= Data Table” on page 596 of Chapter 24, “The TEXTMINE Procedure.”

**SVDDOCPRO=** CAS-libref.data-table

specifies the output data table to contain the reduced dimensional projections for each document. **CAS-libref.data-table** is a two-level name, where **CAS-libref** refers to the caslib and session identifier, and **data-table** specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 616. The contents of this data table are formed by multiplying the term-by-document frequency matrix by the input data table that is specified in the **SVDU=** option and then normalizing the result.

**SVDU=** CAS-libref.data-table

specifies the input data table that contains the U matrix, which is created during training by PROC TEXTMINE. **CAS-libref.data-table** is a two-level name, where **CAS-libref** refers to the caslib and session identifier, and **data-table** specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 616. The data table contains the information that is needed to project each document into the reduced dimensional space. For more information about the contents of this data table, see the **SVDDU=** option in Chapter 24, “The TEXTMINE Procedure.”

**TERMS=** CAS-libref.data-table

specifies the input data table of terms to be used by PROC TMSCORE. **CAS-libref.data-table** is a two-level name, where **CAS-libref** refers to the caslib and session identifier, and **data-table** specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 616. Specify the table that was generated by the **OUTTERMS=** option in the PARSE statement of the TEXTMINE
procedure during training. This data table conveys to PROC TMSCORE which terms should be used in the analysis and whether they should be mapped to a parent. The data table also assigns to each term a key that corresponds to the key that is used in the input data table that is specified by the SVDU= option. For more information about this data table, see the section “The OUTTERMS= Data Table” on page 597 of Chapter 24, “The TEXTMINE Procedure.”

**DOC_ID Statement**

```plaintext
DOC_ID variable ;
```

This statement specifies the `variable` that contains the ID of each document. The ID of each document must be unique; it can be either a number or a string of characters.

**VARIABLES Statement**

```plaintext
VARIABLES variable ;

VAR variable ;
```

This statement specifies the `variable` that contains the text to be processed.

**Details: TMSCORE Procedure**

For information about the techniques that are used for nature language processing, term processing, and singular value decomposition, see the section “Details: TEXTMINE Procedure” on page 588 of Chapter 24, “The TEXTMINE Procedure.”

**System Configuration**

**Prerequisites for Running PROC TMSCORE**

To use the TMSCORE procedure, the language binary files that are provided under that license must be available on the grid for parsing text.
Chapter 26
The TSNE Procedure

Overview: TSNE Procedure

The TSNE procedure implements the \textit{t}-distributed stochastic neighbor embedding (\textit{t}-SNE) dimension reduction method in SAS Viya. The \textit{t}-SNE method is well suited for visualization of high-dimensional data, as well as for feature engineering and preprocessing for subsequent clustering and modeling. PROC TSNE 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, \textit{t}-SNE is appropriate for nonlinear data and emphasizes existing groupings in the data. The method is named \textit{t}-SNE because it models the pairwise distances in low dimensions according to Student’s \textit{t}-distribution. The \textit{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 \textit{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. You can use PROC TSNE to read and write data in distributed form and to perform computation in parallel by making full use of multicore computers or distributed computing environments.

The TSNE procedure operates on an input data table. For each observation in the input data table, the procedure 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 statement. To speed up computations, PROC TSNE uses the Barnes-Hut approximation to the perplexity loss.

**PROC TSNE Features**

PROC TSNE enables you to use parallel execution in a distributed computing environment or on a single-machine. The procedure has the following basic features:

- is highly distributed and multithreaded
- returns low-dimensional embeddings that are based on a parallel implementation of the t-SNE algorithm

**Using CAS Sessions and CAS Engine Librefs**

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```sas
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```sas
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```sas
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 9 in Chapter 3, “Shared Concepts.”
Getting Started: TSNE Procedure

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 10 and “Loading a SAS Data Set onto a CAS Server” on page 10 in Chapter 3, “Shared Concepts.”

This example shows how to use the TSNE procedure to obtain an embedding from observations in a 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. PROC TSNE 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 Sashelp.Iris 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.

```sas
data mycas.iris;
set sashelp.iris;
id=_n_;
run;
```

The following statements run PROC TSNE and output the results to ODS tables:

```sas
proc tsne
data = mycas.iris
nDimensions = 2
perplexity = 5
learningRate = 100
maxIters = 500;
input SepalLength SepalWidth PetalLength PetalWidth;
output out = mycas.tsne_out copyvars=(id species);
run;
```

The NDIMENSIONS=2 option requests that the model return two embedding dimensions; the PERPLEXITY=5 option specifies the perplexity value; the LEARNINGRATE=100 option specifies the learning rate for the optimization; and the INPUT statement specifies that the SepalLength, SepalWidth, PetalLength, and PetalWidth variables be used as inputs. The OUTPUT statement requests that the embedding be written to the data table mycas.tsne_out, and the COPYVARS= option requests that the ID and Species variables be copied to the output.

The following statements download the data table mycas.tsne_out from CAS to the local SAS data set tsne_out:

```sas
data tsne_out;
set mycas.tsne_out;
run;
```

The following PROC SGPLOT statements plot embedding dimension _DIM_2_ against embedding dimension _DIM_1_:

```sas
```
proc sgplot data=tsne_out;
  title "Iris embedding";
  title1 "Scatter plot of iris embedding";
  scatter x=_DIM_1_ y=_DIM_2_ / group=species
    markerattrs=(symbol=CircleFilled);
run;

Output 26.1 shows the results. The colors in the scatter plot indicate three distinct clusters, which correspond to the three iris species.

Figure 26.1 Scatter Plot of Iris Data Embedding

Syntax: TSNE Procedure

The following statements are available in the TSNE procedure:

PROC TSNE <options> ;
  DISPLAY <table-list> </options> ;
  DISPLAYOUT table-spec-list </options> ;
  INPUT variables ;
  OUTPUT OUT=CAS-libref.data-table <options> ;
  AUTOTUNE <options> ;

The PROC TSNE statement and at least one INPUT statement are required. You can specify multiple INPUT statements.

The following sections describe the PROC TSNE statement and then describe the other statements in alphabetical order.
The PROC TSNE statement invokes the procedure. Table 26.1 summarizes the options available in the PROC TSNE statement.

### Table 26.1 PROC TSNE Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Data Table Option</strong></td>
<td></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the input data table</td>
</tr>
<tr>
<td><strong>t-SNE Options</strong></td>
<td></td>
</tr>
<tr>
<td>LEARNINGRATE=</td>
<td>Specifies the learning rate for the optimization</td>
</tr>
<tr>
<td>MAXITERS=</td>
<td>Specifies the maximum number of iterations</td>
</tr>
<tr>
<td>NDIMENSIONS=</td>
<td>Specifies the number of dimensions to return for the embedding</td>
</tr>
<tr>
<td>NOPRINT</td>
<td>Suppresses ODS output</td>
</tr>
<tr>
<td>NTHREADS=</td>
<td>Specifies the number of threads to use on each computation node</td>
</tr>
<tr>
<td>PERPLEXITY=</td>
<td>Specifies the perplexity value</td>
</tr>
<tr>
<td>SEED=</td>
<td>Specifies the seed to be used for pseudorandom number generation</td>
</tr>
</tbody>
</table>

You can specify the following options:

- **DATA=CAS-libref.data-table**
  
  names the input data table for PROC TSNE to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

  - **CAS-libref** refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 624.

  - **data-table** specifies the name of the input data table.

- **LEARNINGRATE=number**
  
  specifies the learning rate for the gradient descent optimization, where *number* is a positive real number. The learning rate controls the amount by which the embeddings are updated at each iteration.

  By default, LEARNINGRATE=100. This value can be tuned with the AUTOTUNE statement.

- **MAXITERS=number**
  
  specifies the maximum number of iterations for the algorithm to perform, where *number* is an integer greater than or equal to 1.

  By default, MAXITERS=1000. This value can be tuned with the AUTOTUNE statement.
**NDIMENSIONS=number**
specifies the number of dimensions to return, where number is either 2 or 3.

By default, NDIMENSIONS=2.

**NOPRINT**
suppresses ODS output.

**NTHREADS=number-of-threads**
specifies the number of threads to use for the computation, where number-of-threads is an integer. The default value is the maximum number of available threads per computer.

**PERPLEXITY=number**
specifies the perplexity value to use for the computation, where number is a real number greater than zero. For more information, see the section “Details: TSNE Procedure” on page 634.

By default, PERPLEXITY=30. This value can be tuned with the AUTOTUNE statement.

**SEED=random-seed**
specifies an integer that is used to start the pseudorandom number generator. This option enables you to reproduce the same sample output. If you do not specify a seed or you specify a value less than or equal to 0, the seed is generated from reading the time of day from the computer’s clock.

By default, SEED=0.

---

### AUTOTUNE Statement

**AUTOTUNE < options > ;**

The AUTOTUNE statement searches for the best combination of values of the LEARNINGRATE=, MAX-ITERS=, and PERPLEXITY= options in the PROC TSNE statement.

Table 26.2 summarizes the options that you can specify in the AUTOTUNE statement. For more information about all options except the TUNINGPARAMETERS= option, see the option’s description in the section “AUTOTUNE Statement” on page 12 in Chapter 3, “Shared Concepts.” The TUNINGPARAMETERS= option is described following Table 26.2.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDLOOKUP</td>
<td>Specifies that the table specified in the HISTORYTABLE= option contain the rows from the table specified in the LOOKUPTABLE= option</td>
</tr>
<tr>
<td>EVALHISTORY=</td>
<td>Specifies how to report the evaluation history of the tuner</td>
</tr>
<tr>
<td>FRACTION=</td>
<td>Specifies the fraction of observations to use for validation</td>
</tr>
<tr>
<td>HISTORYTABLE=</td>
<td>Specifies the CAS table that contains the evaluation history</td>
</tr>
<tr>
<td>KFOLD=</td>
<td>Specifies the number of folds for k-fold cross validation</td>
</tr>
<tr>
<td>LIVEUPDATE</td>
<td>Specifies that the table specified in the HISTORYTABLE= option be updated at every evaluation</td>
</tr>
<tr>
<td>LOCALLSEARCH</td>
<td>Enables local search optimization</td>
</tr>
</tbody>
</table>
Table 26.2  continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOKUPTABLE=</td>
<td>Specifies the CAS table to use for evaluation lookup</td>
</tr>
<tr>
<td>MAXBAYES=</td>
<td>Specifies the maximum number of points in the kriging model</td>
</tr>
<tr>
<td>MAXEVALS=</td>
<td>Specifies the maximum number of evaluations</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>MAXTIME=</td>
<td>Specifies the maximum time for all iterations</td>
</tr>
<tr>
<td>MAXTRAINETIME=</td>
<td>Specifies the maximum time for a model training</td>
</tr>
<tr>
<td>NCONVITER=</td>
<td>Specifies the number of convergence iterations</td>
</tr>
<tr>
<td>NOGRIDSHUFFLE</td>
<td>Requests that the grid points not be shuffled</td>
</tr>
<tr>
<td>NPARALLEL=</td>
<td>Specifies the number of parallel sessions</td>
</tr>
<tr>
<td>NSUBSESSIONWORKERS=</td>
<td>Specifies the number of workers in parallel sessions</td>
</tr>
<tr>
<td>OBJECTIVE=</td>
<td>Specifies the objective function</td>
</tr>
<tr>
<td>POPSIZE=</td>
<td>Specifies the population size when SEARCHMETHOD=GA or SEARCHMETHOD=BAYESIAN</td>
</tr>
<tr>
<td>SAMPLESIZE=</td>
<td>Specifies the sample size when SEARCHMETHOD=LHS or SEARCHMETHOD=RANDOM</td>
</tr>
<tr>
<td>SEARCHMETHOD=</td>
<td>Specifies the search method that the optimizer uses</td>
</tr>
<tr>
<td>SECONDOBJECTIVE=</td>
<td>Specifies the second objective to use for tuning</td>
</tr>
<tr>
<td>SELECTINITPOINT</td>
<td>Specifies that the tuner select the best evaluation from the lookup table</td>
</tr>
<tr>
<td>TARGETEVENT=</td>
<td>Specifies the target event for ROC-based calculations</td>
</tr>
<tr>
<td>TRAINFRACTION=</td>
<td>Specifies the fraction of observations to use for training</td>
</tr>
<tr>
<td>TUNINGPARAMETERS=</td>
<td>Specifies the custom tuning parameters</td>
</tr>
<tr>
<td>USEPARAMETERS=</td>
<td>Specifies how to handle the TUNINGPARAMETERS= option</td>
</tr>
</tbody>
</table>

TUNINGPARAMETERS=(suboption | . . . | < suboption >)
TUNEPARMS=(suboption | . . . | < suboption >)

specifies which parameters to tune and which ranges to tune over. If USEPARAMETERS=STANDARD, this option is ignored.

You can specify one or more of the following suboptions:

**LEARNINGRATE (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)**
specifies information about the learning rate to use for tuning the t-SNE model. For more information, see the **LEARNINGRATE=** option in the PROC TSNE statement.

You can specify the following additional suboptions:

**LB=number**
specifies the minimum learning rate to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=10.
**UB=** *number*

specifies the maximum learning rate to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=2000.

**VALUES=** *value-list*

specifies a list of values to consider for the learning rate during tuning, where *value-list* is a space-separated list of integer numbers greater than or equal to 1. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=** *number*

specifies the initial learning rate for the tuner to use.

By default, INIT=1000.

**EXCLUDE**

excludes the learning rate from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

**MAXITERS (LB=** *number* **UB=** *number* **VALUES=** *value-list* **INIT=** *number* **EXCLUDE)**

specifies information about the maximum number of iterations to use for tuning the t-SNE model. For more information, see the MAXITERS= option in the PROC TSNE statement.

You can specify the following additional suboptions:

**LB=** *number*

specifies the minimum number of iterations to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, LB=250.

**UB=** *number*

specifies the maximum number of iterations to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.

By default, UB=5000.

**VALUES=** *value-list*

specifies a list of numbers of iterations to consider during tuning, where *value-list* is a space-separated list of positive integers. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

**INIT=** *number*

specifies the initial number of iterations for the tuner to use.

By default, INIT=300.

**EXCLUDE**

excludes the number of iterations from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.
PERPLEXITY (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies information about the perplexity to use for tuning the t-SNE model. For more informa-
tion, see the PERPLEXITY= option in the PROC TSNE statement.
You can specify the following additional suboptions:

LB=number
specifies the minimum perplexity to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.
By default, LB=5.

UB=number
specifies the maximum perplexity to consider during tuning. If you specify this suboption, you cannot specify the VALUES= suboption.
By default, UB=50.

VALUES=value-list
specifies a list of perplexities to consider during tuning, where value-list is a space-separated list of numbers greater than 0. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

INIT=number
specifies the initial perplexity for the tuner to use.
By default, INIT=30.

EXCLUDE
excludes the perplexity from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored.

DISPLAY Statement

DISPLAY <table-list> < / options> ;
The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.
If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see SAS Output Delivery System: Procedures Guide.
You can specify the table-list as a list of table names, paths, partial pathnames, and regular expressions.
The table names that you can specify are listed in the section “ODS Table Names” on page 635. A path is a table name that is prefixed with dot-separated grouping information. For example, a
SelectionSummary table that a procedure produces during a selection routine might have the path `Bygroup1.Summary.SelectionSummary`. A partial pathname does not include all groups; for example, `SelectionSummary` and `Summary.SelectionSummary` are partial pathnames for `Bygroup1.Summary.SelectionSummary`.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both `SelectionSummary` and `Summary.SelectionSummary` select `Bygroup1.Summary.SelectionSummary`.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the `Bygroup1.Summary.SelectionSummary` table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the `Bygroup1.Summary.SelectionSummary` table is not selected.

You can specify the following options after a slash (/):

- **CASESENSITIVE**
  performs a case-sensitive comparison of table names in the `table-list` to display table names when tables are subsetted for display. To preserve case, you must enclose table names in the `table-list` in quotation marks.

- **EXCLUDE**
  displays all display tables except those that you specify in the `table-list`.

- **EXCLUDEALL**
  suppresses display of all tables. This option takes precedence over the other options.

- **TRACE**
  displays the display table names, labels, and paths.

**DISPLAYOUT Statement**

```
DISPLAYOUT table-spec-list < / options > ;
```

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see SAS Output Delivery System: Procedures Guide.

The `table-spec-list` specifies a list of CAS output tables to create. Each entry in the list has either a `key=value` format or a `key` format:

- `key=value` specifies `key` as the ODS table name, path, or partial pathname, and specifies `value` as the CAS output table name.

- `key` specifies `key` as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “ODS Table Names” on page 635. You cannot specify the ODS table named OutputCasTables in the `table-spec-list`.

Table names and partial pathnames are discussed under the DISPLAY statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following options after a slash (/):
INCLUDEALL
creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the `table-spec-list` specification is ignored.

NOREPLACE
does not replace any existing CAS output table of the same name.

REPEATED
replicates all CAS output tables on all nodes.

INPUT Statement

```
INPUT variables;
```

The INPUT statement specifies the `variables` to use in the t-SNE analysis. The `variables` must be of interval type. You must specify at least one input `variable`.

OUTPUT Statement

```
OUTPUT OUT=CAS-libref.data-table <options> ;
```

The OUTPUT statement creates an embedding data table to contain the results of the procedure run.

You must specify the following `option`:

```
OUT=CAS-libref.data-table
```

names the output data table for PROC TSNE to use. You must specify this option before any other options. `CAS-libref.data-table` is a two-level name, where

- `CAS-libref` refers to a collection of information that is defined in the LIBNAME statement and includes the `caslib`, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about `CAS-libref`, see the section “Using CAS Sessions and CAS Engine Librefs” on page 624.

- `data-table` specifies the name of the output data table.

You can also specify the following `option`:

```
COPYVAR=variable
COPYVARS=(variables)
```

lists one or more `variables` from the input data table to be transferred to the output data table.
Details: TSNE Procedure

The t-SNE method for dimensionality reduction was introduced by Van der Maaten and Hinton (2008). The method estimates the embeddings \( y_i \) in two or three dimensions by minimizing the Kullback-Leibler loss, which is defined as

\[
C = \text{KL}(P \| Q) = \sum_{i=1}^{N} \sum_{j=1}^{N} p_{ij} \log \frac{p_{ij}}{q_{ij}}
\]

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

\[
q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|y_k - y_i\|^2)^{-1}}
\]

The conditional probability \( p_{i|j} \) follows the expression

\[
p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2)}{2\sigma_i^2}
\]

where \( \sigma_i \) is a bandwidth parameter that varies for each observation \( x_i \). PROC TSNE sets \( \sigma_i \) in such a way that the perplexity

\[
\text{Perp}_i = 2^{-\sum_j p_{j|i} \log_2 p_{j|i}}
\]

matches the fixed PERPLEXITY= option value that the user specifies. In general, lower perplexity yields embeddings that have smaller, more separated groupings.

To avoid the \( O(N^2) \) computational complexity of the similarity computations, PROC TSNE employs the Barnes-Hut approximation that is proposed by Van der Maaten (2014).

The TSNE procedure 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= options.
Displayed Output

The TSNE procedure displays various tables that are related to the t-SNE algorithm. 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 t-SNE algorithm. This information includes the perplexity along with the seed that is specified in the SEED= option in the PROC TSNE statement.

Number of Observations

The “Number of Observations” table displays the number of observations that are read from the input data table and used.

Iteration History

The “Iteration History” table displays the iteration history, which consists of the iteration number and the loss for every 100 iterations.

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. Because this table is not a typical ODS table that contains analytical results, you cannot include it in the table-spec-list in the DISPLAYOUT statement.

ODS Table Names

Each table that the TSNE procedure creates has a name associated with it, and you must use this name to refer to the table when you use ODS statements. The name of each table and a short description of its contents are listed in Table 26.3.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>PROC TSNE</td>
<td>Default</td>
</tr>
<tr>
<td>NObs</td>
<td>Number of observations</td>
<td>PROC TSNE</td>
<td>Default</td>
</tr>
<tr>
<td>OptIterHistory</td>
<td>Iteration history</td>
<td>PROC TSNE</td>
<td>Default</td>
</tr>
<tr>
<td>OutputCASTables</td>
<td>See the section “OutputCasTables Table” on page 635</td>
<td>PROC TSNE</td>
<td>Default</td>
</tr>
</tbody>
</table>
Output Data Tables

The TSNE procedure creates a data table to which it writes the embeddings. You specify the name of this data table in the OUTPUT statement. Details about the data table are shown in Table 26.4.

<table>
<thead>
<tr>
<th>Data Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSNE_OUT</td>
<td>Lists the estimated embeddings</td>
</tr>
</tbody>
</table>

References


Subject Index

k-fold cross validation
   FOREST procedure, 195
   GRADBOOST procedure, 270
   NNET procedure, 408

OutputCasTables table
   FACTMAC procedure, 136
   FASTKNN procedure, 150
   GRADBOOST procedure, 272
   MTLearn procedure, 360
   TSNE procedure, 635

active-set solver
   SVDD procedure, 500

ASTORE procedure
   computational method, 22
   hyperparameter tuning, 23
   input data tables, 33, 34
   multithreading, 22
   output data tables, 34
   test data, 21
   validation, 21

autoencoder
   NNET procedure, 409

AUTOTUNE statement
   syntax (Shared Concepts), 12

bagging the data
   FOREST procedure, 187

bandwidth relative scale
   SVDD procedure, 497

best configuration
   FOREST procedure, 197
   GRADBOOST procedure, 272
   NNET procedure, 411

binning
   BNET procedure, 60

BNET procedure, 49
   binning, 60
   displayed output, 77
   fit statistics table, 77
   independence tests, 58, 59
   INNETWORK= option, 55
   input data tables, 58
   missing values, 60
   model information table, 77
   model selection, 58
   number of observations table, 77
   ODS table names, 78
   output data tables, 71
   output network data set, 60
   PROC BNET features, 50
   PROC BNET syntax, 57
   structure learning, 59, 60
   validation information table, 77
   variable information table, 77
   variable level table, 77
   variable order table, 77
   variable selection, 61, 62
   variable selection report table, 77

BOOLRULE procedure
   computational method, 22
   hyperparameter tuning, 23
   multithreading, 22
   test data, 21
   validation, 21

boosting
   GRADBOOST procedure, 262

cluster summary
   GVARCLUS procedure, 295

CODE statement
   syntax (Shared Concepts), 18

comparison with existing methods
   SVDD procedure, 499

computational method
   ASTORE procedure, 22
   BOOLRULE procedure, 22
   FACTMAC procedure, 22
   FOREST procedure, 22
   GRADBOOST procedure, 22
   NNET procedure, 22, 407
   SVMACHINE procedure, 22
   TEXTMINE procedure, 22
   TMSCORE procedure, 22

convergence status
   NNET procedure, 410

cross validation results
   NNET procedure, 411

cross-validation fit statistics
   FOREST procedure, 197, 272

descriptive statistics
   GMM procedure, 225

details
   KPCA procedure, 317
   MWPCA procedure, 375
   RPCA procedure, 436
Subject Index

displayed output
BNET procedure, 77
FACTMAC procedure, 136
FASTKNN procedure, 149
FISM procedure, 159
FOREST procedure, 196
GMM procedure, 224
GRADBOOST procedure, 271
GVARCLUS procedure, 295
KPCA procedure, 319
MBANALYSIS procedure, 335
MTLEARN procedure, 360
NNET procedure, 410
RPCA procedure, 439
SEMISUPLEARN procedure, 456
SPARSEML procedure, 469
SVMACHINE procedure, 561
TSNE procedure, 635
early stopping
GRADBOOST procedure, 263
evaluation history
FOREST procedure, 198
GRADBOOST procedure, 272
NNET procedure, 410
FACTMAC procedure, 121
OutputCasTables table, 136
computational method, 22
displayed output, 136
final exact loss, 136
hyperparameter tuning, 23
input data tables, 127
interval variables, 136
iteration history, 136
model information, 136
multithreading, 22
number of observations, 136
ODS table names, 137
output data tables, 134, 137
test data, 21
validation, 21
fast incremental solver
SVDD procedure, 501
FASTKNN procedure, 141
OutputCasTables table, 150
displayed output, 149
input data tables, 145
model information, 149
number of observations, 150
ODS table names, 150
output data tables, 148, 151
final exact loss
FACTMAC procedure, 136
FISM procedure, 158
displayed output, 159
FP-growth algorithm, 158
input data tables, 156
limitations, 160
fit statistics
FOREST procedure, 197
GRADBOOST procedure, 271
fit statistics table
BNET procedure, 77
SVMACHINE procedure, 561
FitStat table
NNET procedure, 411
FOREST procedure, 164
k-fold cross validation, 195
bagging the data, 187
best configuration, 197
computational method, 22
cross-validation fit statistics, 197, 272
displayed output, 196
evaluation history, 198
fit statistics, 197
handling missing values, 190
handling values that are absent from training data, 191
hyperparameter tuning, 23, 195
input data tables, 172
isolation forests, 194
measuring variable importance, 191
model information, 196
multithreading, 22
number of observations, 196
ODS table names, 198
output data tables, 184
OutputCasTables table, 198
predicted probability names, 198
preselection, 188
splitting criteria, 182
test data, 21
training a decision tree, 187
tuner information, 197
tuner results, 197
tuner summary, 197
tuner timing, 197
validation, 21
variable importance, 196, 197
FP-growth algorithm
FISM procedure, 158
MBANALYSIS procedure, 333
fraction outlier parameter
SVDD procedure, 497
GMM procedure, 207
cluster summary output data table, 216
descriptive statistics, 225
displayed output, 224
input data tables, 216
missing values, 222
model information, 225
number of observations, 225
ODS table names, 225
output data tables, 220
GRADBOOST procedure, 234
k-fold cross validation, 270
OutputCasTables table, 272
best configuration, 272
boosting, 262
computational method, 22
displayed output, 271
early stopping, 263
evaluation history, 272
fit statistics, 271
handling missing values, 265
hyperparameter tuning, 23, 269
input data tables, 242
measuring prediction error, 264
measuring variable importance, 266
model information, 271
monotonicity constraints, 262
multithreading, 22
number of observations, 271
ODS table names, 273
output data tables, 256
predicted probability names, 272
splitting a nominal variable, 261
subsampling the data, 260
test data, 21
training a decision tree, 261
transfer learning, 269
tuner information, 271
tuner results, 272
tuner summary, 271
tuner timing, 272
validation, 21
variable importance, 271
graphical lasso
GVARCLUS procedure, 293
graphical representation of precision matrix
GVARCLUS procedure, 294
GRAPHTYPE= option examples
MTLEARN procedure, 355
GVARCLUS procedure, 281
cluster summary, 295
displayed output, 295
graphical lasso, 293
graphical representation of precision matrix, 294
input data tables, 287
maximum iterations, 288
maximum members, 288
maximum steps, 288
minimum clusters, 288
missing values, 293
number of observations, 295
ODS table names, 295
preprocessing, 294
rho, 289
select, 289
selection, 295
stop, 289
tolerance, 289
handling missing values
FOREST procedure, 190
GRADBOOST procedure, 265
handling values that are absent from training data
FOREST procedure, 191
hyperparameter importance
NNET procedure, 411
hyperparameter tuning
ASTORE procedure, 23
BOOLRULE procedure, 23
FACTMAC procedure, 23
FOREST procedure, 23, 195
GRADBOOST procedure, 23, 269
NNET procedure, 23
SVMACHINE procedure, 23
TEXTMINE procedure, 23
TMSCORE procedure, 23
independence tests, 73
BNET procedure, 58, 59
INNETWORK= option
BNET procedure, 55
input variables
NNET procedure, 398
interval variables
FACTMAC procedure, 136
MTLEARN procedure, 360
isolation forests
FOREST procedure, 194
iteration history
NNET procedure, 410
iteration history table
SVMACHINE procedure, 561
iteration report table
SVMACHINE procedure, 562
KPCA procedure
k-means clustering information, 319
descriptive statistics, 319
details, 317
displayed output, 319
examples, 321
Subject Index

input data tables, 311
model information, 319
number of observations, 319
ODS table names, 320
output data tables, 316, 320
task timing, 319

limitations
FISM procedure, 160
MBANALYSIS procedure, 335

linear kernels
SPARSEML procedure, 468
SVDD procedure, 491
SVMACHINE procedure, 558

mathematical formulation
SVDD procedure, 491
MBANALYSIS procedure, 332
displayed output, 335
FP-growth algorithm, 333
input data tables, 328
limitations, 335
scoring with PROC ASTORE, 334

mean criteria
SVDD procedure, 496
measuring prediction error
GRADBOOST procedure, 264
measuring variable importance
FOREST procedure, 191
GRADBOOST procedure, 266

misclassification matrix table
SVMACHINE procedure, 561

missing values
BNET procedure, 60
GMM procedure, 222
GVARCLUS procedure, 293
MTLEARN procedure, 360

model information
FOREST procedure, 196
GMM procedure, 225
GRADBOOST procedure, 271
NNET procedure, 410

model information table
BNET procedure, 77
SVMACHINE procedure, 561

model selection
BNET procedure, 58
modified mean criteria
SVDD procedure, 496
monotonicity constraints
GRADBOOST procedure, 262
MTLEARN procedure, 339
OutputCasTables table, 360
displayed output, 360

GRAPHTYPE= option examples, 355
input data tables, 346
interval variables, 360
iteration history, 360
missing values, 360
model information, 360
number of observations, 360
ODS table names, 361
output data tables, 353, 361

multithreading
ASTORE procedure, 22
BOOLRULE procedure, 22
FACTMAC procedure, 22
FOREST procedure, 22
GRADBOOST procedure, 22
NNET procedure, 22
SVMACHINE procedure, 22
TEXTMINE procedure, 22
TMSCORE procedure, 22

MWPCA procedure
details, 375
dimensions, 376
displayed output, 376
equations, 377
input data tables, 369
model information, 376
ODS table names, 377
output data tables, 372
results summary, 376

NNET procedure, 384
k-fold cross validation, 408
autoencoder, 409
best configuration, 411
computational method, 22, 407
convergence status, 410
cross validation results, 411
displayed output, 410
evaluation history, 410
FitStat table, 411
hyperparameter importance, 411
hyperparameter tuning, 23
input data tables, 388
iteration history, 410
model information, 410
multithreading, 22
ODS table names, 412
output data tables, 402
parameter tuning, 408
PredIntoName, 411
PredName, 412
PredProbName, 411
ROC curve, 411
score information, 410
test data, 21
tuner information, 410
tuner results, 410
tuner summary, 411
tuner timing, 411
validation, 21
nonlinear kernels
SPARSEML procedure, 468
SVDD procedure, 491
SVMACHINE procedure, 558
number of hidden neurons
FOREST procedure, 397
number of observations
FOREST procedure, 196
GMM procedure, 225
GRADBOOST procedure, 271
GVARCLUS procedure, 295
number of observations table
BNET procedure, 77
SVMACHINE procedure, 561
optimization
SVMACHINE procedure, 560
options summary
PARSE statement, 578
PROC FACTMAC statement, 126
PROC FASTKNN statement, 144
PROC FOREST statement, 171
PROC GMM statement, 215
PROC GRADBOOST statement, 240
PROC GVARCLUS statement, 287
PROC KPCA statement, 310
PROC MTLEARN statement, 345
PROC MWPCA statement, 369
PROC RPCA statement, 426
PROC SEMISUPLEARN statement, 449
PROC TEXTMINE statement, 576
PROC TMSCORE statement, 620
PROC TSNE statement, 627
SELECT statement, 583
SVD statement, 584
output data tables
FACTMAC procedure, 137
FASTKNN procedure, 151
KPCA procedure, 320
MTLEARN procedure, 361
RPCA procedure, 440
SEMISUPLEARN procedure, 457
TSNE procedure, 636
output network data set
BNET procedure, 60
OutputCasTables table
FOREST procedure, 198
parameter learning, 53, 76
parameter tuning
SVMACHINE procedure, 560
PARTITION statement
syntax (Shared Concepts), 20
predicted probability names
FOREST procedure, 198
GRADBOOST procedure, 272
predicted probability variables table
SVMACHINE procedure, 562
predicted target variable table
SVMACHINE procedure, 562
PredIntoName
SVMACHINE procedure, 411
PredName
SVMACHINE procedure, 412
PredProbName
SVMACHINE procedure, 411
preprocessing
GVARCLUS procedure, 294
preselection
FOREST procedure, 188
PROC ASTORE, 27
PROC ASTORE features, 28
PROC BNET features
BNET procedure, 50
PROC BNET syntax
BNET procedure, 57
PROC FISM, 153
PROC FISM features, 154
PROC KPCA, 303
PROC KPCA features, 304
PROC MBANALYSIS, 325
PROC MBANALYSIS features, 326
PROC MWPCA, 363
PROC MWPCA features, 364
PROC RPCA, 419
PROC RPCA features, 420
PROC SPARSEML, 461
PROC SPARSEML features, 462
PROC SVMACHINE, 545
PROC SVMACHINE features, 546
radial basis function bandwidth parameter
SVDD procedure, 494
rba variable importance
variable importance, 271
ROC curve
SVMACHINE procedure, 411
RPCA procedure
details, 436
dimensions, 439
displayed output, 439
examples, 441
input data tables, 427
model information, 439
ODS table names, 440
output data tables, 440
results summary, 439

score information
  NNET procedure, 410

scoring process
  SVMACHINE procedure, 560
scoring with PROC ASTORE
  MBANALYSIS procedure, 334
selection
  GVARCLUS procedure, 295
SEMIUSLEARN procedure, 445
  displayed output, 456
  input data tables, 450
  loss, 456
  model information, 456
  ODS table names, 456
  output data tables, 454, 457

Shared Concepts
  AUTOTUNE statement, 12
  CODE statement, 18
  PARTITION statement, 20
s solvers
  SVDD procedure, 500
sparse matrix
  TEXTMINE procedure, 596
SPARSEML procedure
  displayed output, 469
  input data tables, 465
  linear kernels, 468
  nonlinear kernels, 468
  ODS table names, 470
  output data tables, 467
splitting a nominal variable
  GRADBOOST procedure, 261
splitting criteria
  FOREST procedure, 182
stochastic subset solver
  SVDD procedure, 500
structure learning, 53, 75
  BNET procedure, 59, 60
subsampling the data
  GRADBOOST procedure, 260
SVDD formulation with weight variable
  SVDD procedure, 498
SVDD procedure
  active-set solver, 500
  bandwidth relative scale, 497
  comparison with existing methods, 499
  fast incremental solver, 501
  fraction outlier parameter, 497
  input data tables, 485
linear kernels, 491
mathematical formulation, 491
mean criteria, 496
modified mean criteria, 496
nonlinear kernels, 491
ODS table names, 502
radial basis function bandwidth parameter, 494
s solvers, 500
stochastic subset solver, 500
SVDD formulation with weight variable, 498
trace criteria, 497

SVMACHINE procedure
  computational method, 22
  displayed output, 561
  fit statistics table, 561
  hyperparameter tuning, 23
  input data tables, 550
  iteration history table, 561
  iteration report table, 562
  linear kernels, 558
  misclassification matrix table, 561
  model information table, 561
  multithreading, 22
  nonlinear kernels, 558
  number of observations table, 561
  ODS table names, 562
  optimization, 560
  output data tables, 555
  predicted probability variables table, 562
  predicted target variable table, 562
  scoring process, 560
  test data, 21
  training results table, 561
  validation, 21
target variables
  NNET procedure, 403
test data
  ASTORE procedure, 21
  BOOLRULE procedure, 21
  FACTMAC procedure, 21
  FOREST procedure, 21
  GRADBOOST procedure, 21
  NNET procedure, 21
  SVMACHINE procedure, 21
  TEXTMINE procedure, 21
  TMSCORE procedure, 21
TEXTMINE procedure, 570
  cell weight, 579
  computational method, 22
  coordinate list (COO) format, 593
  entity, 579
  filtering term by frequency, 581
  hyperparameter tuning, 23
input data tables, 577
language used by input data tables, 577
multiterm words list, 579
multithreading, 22
noun groups, 579
number of threads, 577
show dropped terms, 581
sparse format, 592
sparse matrix, 596
start list, 581
stemming, 580
stop list, 581
SVD, singular value decomposition, 593
synonym list, 581
tagging, 580
term weight, 582
test data, 21
transactional style, 596
validation, 21
variable name style, 577
TMSCORE procedure, 615
computational method, 22
hyperparameter tuning, 23
input data tables, 620
multithreading, 22
system configuration, 622
test data, 21
validation, 21
TMSCORE procedure, system configuration prerequisite, 622
trace criteria
SVDD procedure, 497
training a decision tree
FOREST procedure, 187
GRADBOOST procedure, 261
training results table
SVMACHINE procedure, 561
transactional style
TEXTMINE procedure, 596
transfer learning
GRADBOOST procedure, 269
TSNE procedure, 623
OutputCasTables table, 635
displayed output, 635
input data tables, 627
iteration history, 635
model information, 635
number of observations, 635
ODS table names, 635
output data tables, 633, 636
tuner information
FOREST procedure, 197
GRADBOOST procedure, 271
NNET procedure, 410
tuner results
FOREST procedure, 197
GRADBOOST procedure, 272
NNET procedure, 411
tuner summary
FOREST procedure, 197
GRADBOOST procedure, 271
NNET procedure, 411
tuner timing
FOREST procedure, 197
GRADBOOST procedure, 272
NNET procedure, 411
validation
ASTORE procedure, 21
BOOLRULE procedure, 21
FACTMAC procedure, 21
FOREST procedure, 21
GRADBOOST procedure, 21
NNET procedure, 21
SVMACHINE procedure, 21
TEXTMINE procedure, 21
TMSCORE procedure, 21
validation information table
BNET procedure, 77
variable importance
FOREST procedure, 196, 197
GRADBOOST procedure, 271
rba variable importance, 271
variable information table
BNET procedure, 77
variable level table
BNET procedure, 77
variable order table
BNET procedure, 61, 62
variable selection
BNET procedure, 77
variable selection report table
BNET procedure, 77
weight variable
NNET procedure, 407
Syntax Index

ACT= option
  HIDDEN statement, 397
  TARGET statement, 404
ACTSET option
  SOLVER statement, 488
ADD option
  VIICODE statement (FOREST), 186
  VIICODE statement (GRADBOOST), 259
ADDTREES option
  PROC GRADBOOST statement, 241
ALGORITHM= option
  OPTIMIZATION statement, 399
ALPHA= option
  PROC GMM statement, 216
ANNEALINGRATE= option
  OPTIMIZATION statement, 399, 400
ANOMALYDETECTION statement
  RPCA procedure, 430
ANTECEDENTLIST= option
  PROC MBANALYSIS statement, 329
APPENDLOOKUP option
  AUTOTUNE statement, 13
ARCHITECTURE statement
  NNET procedure, 390
ASSIGNMISSING= option
  PROC ASTORE statement, 32
ASTORE procedure
  PROC ASTORE statement, 32
ASTORE procedure, CODE statement
  COMMENT option, 19
  FILE= option, 19
  FORMATWIDTH= option, 19
  INDENTSIZE= option, 19
  LABELID= option, 19
  LINESIZE= option, 19
  NOTRIM option, 20
  OUT= option, 20
ASTORE procedure, DESCRIBE statement, 32
  COPYVARS= option, 33
  EPCODE= option, 33
  RSTORE= option, 32
  STORE= option, 32
ASTORE procedure, DOWNLOAD statement, 33
  RSTORE= option, 33
  STORE= option, 34
ASTORE procedure, PROC ASTORE statement, 32
  DATA= option, 33, 34
ASTORE procedure, SCORE statement, 34
  COPYVARS= option, 35
  OUT= option, 34
ASTORE procedure, SETOPTION statement, 35
ASTORE procedure, syntax, 32
ASTORE procedure, UPLOAD statement, 41
  RSTORE= option, 41
  STORE= option, 41
AUTOTUNE statement
  ASTORE procedure, 12
  BNET procedure, 63
  BOOLRULE procedure, 12
  FACTMAC procedure, 12
  FOREST procedure, 12, 176
  GRADBOOST procedure, 12, 128, 247
  MTLERARN procedure, 347
  NNET procedure, 12, 391
  SEMISUPLEARN procedure, 450
  SVMACHINE procedure, 12, 551
  TEXTMINE procedure, 12
  TMSCORE procedure, 12
  TSNE procedure, 628
BINMETHOD = option
  PROC FOREST statement, 172
  PROC GRADBOOST statement, 242
BNET procedure, 57
  CODE statement, 69
  FREQ statement, 70
  ID statement, 70
  INPUT statement, 70
  OUTPUT statement, 71
  PROC BNET statement, 57
  SAVESTATE statement, 72
  syntax, 57
  TARGET statement, 72
BNET procedure, AUTOTUNE statement, 63
  TUNINGPARAMETERS= option, 64
BNET procedure, OUTPUT statement
  COPYVARS= option, 71
  OUT= option, 71
  ROLE= option, 71
BNET procedure, PARTITION statement
  FRACTION option, 72
  ROLEVAR= option, 72
BNET procedure, PROC BNET statement
  DATA= option, 58
  NTHREADS= option, 60
BNET procedure, SAVESTATE statement
RSTORE= option, 72
BOOLRULE procedure, 95
DOCINFO statement, 98
OUTPUT statement, 99
PROC BOOLRULE statement, 96
SCORE statement, 100
syntax, 95
TERMINFO statement, 101
BOOLRULE procedure, CODE statement
COMMENT option, 19
FILE= option, 19
FORMATWIDTH= option, 19
INDENTSIZE= option, 19
LABELID= option, 19
LINESIZE= option, 19
NOTRIM option, 20
OUT= option, 20
BOOLRULE procedure, DOCINFO statement, 98
EVENTS= option, 98
ID= option, 99
TARGET= option, 99
TARGETTYPE= option, 99
BOOLRULE procedure, OUTPUT statement, 99
CANDIDATETERMS= option, 99
RULES= option, 100
RULETERMS= option, 100
BOOLRULE procedure, PROC BOOLRULE statement, 96
DATA= option, 96
DOC= option, 96
DOCID= option, 97
DOCINFO= option, 97
GNEG= option, 97
GPOS= option, 97
MAXCANDIDATES= option, 97
MAXCANDS= option, 97
MAXTRIESIN= option, 97
MAXTRIESOUT= option, 98
MINSUPPORTS= option, 98
MNEG= option, 98
MPOS= option, 98
TERMID= option, 98
TERMINFO= option, 98
BOOLRULE procedure, SCORE statement, 100
OUTMATCH= option, 100
RULETERMS= option, 100
BOOLRULE procedure, TERMINFO statement, 101
ID= option, 101
LABEL= option, 101
BURN= option
TRANSFERLEARN statement, 259
C= option

PROC SPARSEML statement, 465
PROC SVMACHINE statement, 549
CANDIDATETERMS= option
OUTPUT statement, 99
CASESENSITIVE option
DISPLAY statement (FACTMAC), 132
DISPLAY statement (FASTKNN), 147
DISPLAY statement (GMM), 218
DISPLAY statement (GVARCLUS), 290
DISPLAY statement (KPCA), 313
DISPLAY statement (MTLEARN), 352
DISPLAY statement (MWPCA), 371
DISPLAY statement (RPCA), 432
DISPLAY statement (TSNE), 632
CCRITERION= option
LRAPPROXIMATION statement, 315
CELLWGT= option
PARSE statement, 579
CENTER option
PROC KPCA statement, 311
PROC MWPCA statement, 369
PROC RPCA statement, 427
CENTROIDS= option
PROC RPCA statement, 311
CHAID option
GROW statement (FOREST), 182
CHISQUARE option
GROW statement (FOREST), 182
CLUSMETHOD= option
LRAPPROXIMATION statement, 315
CLUSTERCOVOUT= option
PROC GMM statement, 216
CLUSTERSUMOUT= option
PROC GMM statement, 216
CODE statement
ASTORE procedure, 18, 32
BNET procedure, 69
BOOLRULE procedure, 18
FACTMAC procedure, 18, 131
FOREST procedure, 18, 131
GRADBOOST procedure, 18, 254
NNET procedure, 18, 396
RPCA procedure, 431
SVDD procedure, 486
SVMACHINE procedure, 18, 554
TEXTMINE procedure, 18
TMSCORE procedure, 18
COL= option
SVD statement, 585
COLSTATISTICS= option
PROC RPCA statement, 427
COMB= option
HIDDEN statement, 398
TARGET statement, 404
COMMENT option
  CODE statement (ASTORE), 19
  CODE statement (BOOLRULE), 19
  CODE statement (FACTMAC), 19
  CODE statement (FOREST), 19
  CODE statement (GRADBOOST), 19
  CODE statement (NNET), 19
  CODE statement (SVMACHINE), 19
  CODE statement (TEXTMINE), 19
  CODE statement (TMSCORE), 19

COMMFREQ=option
  OPTIMIZATION statement, 399, 400

CONF= option
  PROC MBANALYSIS statement, 329

CONFIG= option
  TMSCORE statement, 621

CONSEQUENTLIST= option
  PROC MBANALYSIS statement, 329

COPYVARS= option
  OUTPUT statement, 71, 134, 149, 184, 220, 257, 354, 402, 454, 467, 556, 633
  SCORE statement (ASTORE), 35

DISPLAYOUT statement
  FACTMAC procedure, 132
  FASTKNN procedure, 147
  GMM procedure, 218
  GVARCLUS procedure, 291
  KPCA procedure, 313
  MTLEARN procedure, 352
  MWPCA procedure, 371
  RPCA procedure, 432
  TSNE procedure, 632

DATA= option
  HIERARCHY statement, 330
  PROC ASTORE statement, 33, 34
  PROC BNET statement, 58
  PROC BOOLRULE statement, 96
  PROC FACTMAC statement, 127
  PROC FASTKNN statement, 145
  PROC FISM statement, 156
  PROC FOREST statement, 172
  PROC GMM statement, 216
  PROC GRADBOOST statement, 242
  PROC GVARCLUS statement, 287
  PROC KPCA statement, 311
  PROC MBANALYSIS statement, 328
  PROC MTLEARN statement, 346
  PROC MWPCA statement, 369
  PROC NNET statement, 388

DECOMP= option
  PROC RPCA statement, 427

DISPLAY statement
  FACTMAC procedure, 131
  FASTKNN procedure, 146
  GMM procedure, 217
  GVARCLUS procedure, 290
  KPCA procedure, 312
  MTLEARN procedure, 351
  MWPCA procedure, 370
  RPCA procedure, 431
  TSNE procedure, 631

DISTRIBUTION option
  PROC GRADBOOST statement, 243

DOC= option
  PROC BOOLRULE statement, 96
  PROC TMSCORE statement, 620

DOC_ID statement
  TEXTMINE procedure, 577
  TMSCORE procedure, 622

DOWNLOAD statement
  PROC GRADBOOST statement, 243

EARLYSTOP option
  PROC GRADBOOST statement, 243
  PROC SVMACHINE statement, 550
EIGENVEC= option
   PROC KPCA statement, 311
ENTITIES= option
   PARSE statement, 579
ENTROPY option
   GROW statement (FOREST), 182
ENTRY= option
   SVD statement, 585
EPCODE= option
   DESCRIBE statement, 33
ERROR= option
   TARGET statement, 404
EVALHISTORY= option
   AUTOTUNE statement, 13
EVENTS= option
   DOCINFO statement, 98
EXACT option
   PROC GVARCLUS statement, 288
EXACTSCORE= option
   SAVESTATE statement, 316
EXACTWEIGHT option
   SVD statement, 585
EXCLUDE option
   DISPLAY statement (FACTMAC), 132
   DISPLAY statement (FASTKNN), 147
   DISPLAY statement (GMM), 218
   DISPLAY statement (GVARCLUS), 290
   DISPLAY statement (KPCA), 313
   DISPLAY statement (MTLEARN), 352
   DISPLAY statement (MWPCA), 371
   DISPLAY statement (RPCA), 432
   DISPLAY statement (TSNE), 632
EXCLUDEALL option
   DISPLAY statement (FACTMAC), 132
   DISPLAY statement (FASTKNN), 147
   DISPLAY statement (GVARCLUS), 290
   DISPLAY statement (KPCA), 313
   DISPLAY statement (MTLEARN), 352
   DISPLAY statement (MWPCA), 371
   DISPLAY statement (RPCA), 432
   DISPLAY statement (TSNE), 632
FACTMAC procedure, PROC FACTMAC statement, 126
   DATA= option, 127
   FILE= option, 131
   LEARNSTEP= option, 127
   LEVEL= option, 133
   MAXITER= option, 127
   NFACTORS= option, 127
   NONNEGATIVE option, 127
   NOPRINT, 127
   NTHREADS= option, 127
   OUTMODEL, 127
   SEED= option, 128
FACTMAC procedure, SAVESTATE statement, 134
   RSTORE= option, 134
FACTMAC procedure, syntax, 126
FACTMAC procedure, TARGET statement, 135
FASTINC option
   PROC SVDD statement, 489
FASTKNN procedure, DISPLAY statement
   CASESENSITIVE option, 147
   EXCLUDE option, 147
   EXCLUDEALL option, 147
   TRACE option, 147
FASTKNN procedure, DISPLAYOUT statement
   INCLUDEALL option, 147
   NOREPLACE option, 147
   REPEATED option, 147
FASTKNN procedure, ID statement, 148
FASTKNN procedure, INPUT statement, 148
FASTKNN procedure, OUTPUT statement, 148
   COPYVARS= option, 149
   OUT= option, 148
FASTKNN procedure, PROC FASTKNN statement, 144
   DATA= option, 145
   K= option, 145
   METHOD= option, 145
   NOPRINT option, 145
   NTHREADS= option, 145
OUTDIST= option, 145
PARALLELIZATION= option, 145
QUERY= option, 146
SEED= option, 146
THRESHDIST= option, 146
FASTKNN procedure, syntax, 144
FILE= option
   CODE statement, 396, 431, 486, 554
   CODE statement (ASTORE), 19
   CODE statement (BOOLRULE), 19
   CODE statement (FACTMAC), 19
   CODE statement (FOREST), 19
   CODE statement (GRADBOOST), 19
   CODE statement (NNET), 19
   CODE statement (SVMACHINE), 19
   CODE statement (TMSCORE), 19
PROC FACTMAC statement, 131
FISM procedure, CUSTOMER statement, 157
FISM procedure, OUTPUT statement, 157
   OUT= option, 157
   OUTFREQ= option, 157
FISM procedure, PROC FISM statement, 156
   DATA= option, 156
   ITEMS= option, 157
   NFIS_RANGE= option, 157
   PCTSUP= option, 156
   SUPPORT= option, 156
FISM procedure, syntax, 156
FISM procedure, TARGET statement, 158
FIXEDMU option
   RPCA statement, 374
   PROC RPCA statement, 428
FOREST procedure, AUTOTUNE statement, 176
   TUNINGPARAMETERS= option, 177
FOREST procedure, CODE statement, 181
   COMMENT option, 19
   FILE= option, 19
   FORMATWIDTH= option, 19
   INDENTSIZE= option, 19
   LABELID= option, 19
   LINESIZE= option, 19
   NOTRIM option, 20
   OUT= option, 20
FOREST procedure, CROSSVALIDATION statement, 181
   KFOLD= option, 182
   NOPARALLEL option, 182
   NSUBSESSIONWORKERS= option, 182
FOREST procedure, GROW statement, 182
   CHAID option, 182
   CHISQUARE option, 182
   ENTROPY option, 182
   FTEST option, 183
   GINI option, 183
   IGR option, 183
   RSS option, 183
FOREST procedure, ID statement, 183
FOREST procedure, INPUT statement, 183
   LEVEL= option, 183
FOREST procedure, OUTPUT statement, 184
   COPYVARS= option, 184
   OUT= option, 184
   ROLE option, 184
FOREST procedure, PARTITION statement
   FRACTION option, 20, 185
   ROLEVAR= option, 20, 185
FOREST procedure, PROC FOREST statement, 171
   ASSIGNMISSING= option, 171
   BINMETHOD = option, 172
   DATA= option, 172
   INBAGFRACTION= option, 173
   INMODEL= option, 173
   ISOLATION option, 173
   LOH= option, 173
   MAXBRANCH= option, 174
   MAXDEPTH= option, 174
   MINLEAFSIZE= option, 174
   MINUSEINSEARCH= option, 174
   NOPRINT option, 174
   NTHREADS= option, 174
   NTREES= option, 174
   NUMBIN= option, 174
   OUTMODEL= option, 174
   PRINTTARGET option, 175
   RBAIMP option, 175
   SEED= option, 175
   VARS_TO_TRY= option, 175
   VII= option, 175
   VOTE= option, 175
FOREST procedure, SAVESTATE statement, 185
   RSTORE= option, 185
FOREST procedure, syntax, 170
FOREST procedure, TARGET statement, 186
   LEVEL= option, 186
FOREST procedure, VIICODE statement, 186
   ADD option, 186
   LIMIT= option, 186
   MISS option, 186
   MULTIPLY option, 186
   SUBTRACT option, 186
   THRESHOLD= option, 186
FORMATWIDTH= option
   CODE statement (ASTORE), 19
   CODE statement (BOOLRULE), 19
   CODE statement (FACTMAC), 19
   CODE statement (FOREST), 19
   CODE statement (GRADBOOST), 19
CODE statement (NNET), 19
CODE statement (SVMACHINE), 19
CODE statement (TEXTMINE), 19
CODE statement (TMSCORE), 19
FRACTION option
  PARTITION statement, 403
  PARTITION statement (BNET), 72
  PARTITION statement (FOREST), 20, 185
  PARTITION statement (GRADBOOST), 20, 257
  PARTITION statement (SVMACHINE), 556
FRACTION= option
  AUTOTUNE statement, 13
FREQ statement
  BNET procedure, 70
  GVARCLUS procedure, 292
FTEST option
  GROW statement (FOREST), 183
GAMMA= option
  PROC SEMISUPLEARN statement, 450
GINI option
  GROW statement (FOREST), 183
GLIM option
  ARCHITECTURE statement, 390
GMM procedure, DISPLAY statement
  CASESENSITIVE option, 218
  EXCLUDE option, 218
  EXCLUDEALL option, 218
  TRACE option, 218
GMM procedure, DISPLAYOUT statement
  INCLUDEALL option, 219
  NOREPLACE option, 219
  REPEATED option, 219
GMM procedure, INPUT statement, 219
GMM procedure, OUTPUT statement
  COPYVARS= option, 220
  OUT= option, 220
GMM procedure, PROC GMM statement, 215
  ALPHA= option, 216
  CLUSTERCOVOUT= option, 216
  CLUSTERSUMOUT= option, 216
  DATA= option, 216
  MAXCLUSTERS= option, 217
  NTHREADS, 217
  SEED= option, 217
GMM procedure, SAVESTATE statement, 219
  RSTORE= option, 219
GMM procedure, SCORE statement, 220
GMM procedure, syntax, 215
GNEG= option
  PROC BOOLRULE statement, 97
GPOS= option
  PROC BOOLRULE statement, 97
GRADBOOST procedure, AUTOTUNE statement, 247
  TUNINGPARAMETERS= option, 129, 248, 451
GRADBOOST procedure, CODE statement, 254
  COMMENT option, 19
  FILE= option, 19
  FORMATWIDTH= option, 19
  INDENTSIZE= option, 19
  LABELID= option, 19
  LINESIZE= option, 19
  NOTRIM option, 20
  OUT= option, 20
GRADBOOST procedure, CROSSVALIDATION statement, 255
  KFOLD= option, 255
  NOPARALLEL option, 255
  NSUBSESSIONWORKERS= option, 255
GRADBOOST procedure, ID statement, 255
GRADBOOST procedure, INPUT statement, 255
  LEVEL= option, 256
  MONOTONIC = option, 256
GRADBOOST procedure, OUTPUT statement, 256
  COPYVARS= option, 257
  OUT= option, 256
  ROLE option, 257
GRADBOOST procedure, PARTITION statement
  FRACTION option, 20, 257
  ROLEVAR= option, 20, 258
GRADBOOST procedure, PROC GRADBOOST statement, 240
  ADDTREES option, 241
  ASSIGNMISSING= option, 241
  BINMETHOD = option, 242
  DATA= option, 242
  DISTRIBUTION option, 243
  EARLYSTOP option, 243
  INMODEL= option, 243
  LASSO= option, 244
  LEARNINGRATE= option, 244
  MAXBRANCH= option, 244
  MAXDEPTH= option, 244
  MINLEAFSIZE= option, 244
  MINUSEINSEARCH= option, 244
  NOMSEARCH option, 245
  NOPRINT option, 245
  NTHREADS= option, 245
  NTHREADS= option, 245
  NTREES= option, 245
  NUMBIN= option, 245
  OFFSET option, 245
  OUTMODEL option, 246
  PRINTTARGET option, 246
  RBAIMP option, 246
  RIDGE= option, 246
  SAMPLINGRATE option, 246
SEED= option, 246
VARS_TO_TRY= option, 246
VII= option, 247
GRADBOOST procedure, SAVESTATE statement, 258
RSTORE= option, 258
GRADBOOST procedure, syntax, 240
GRADBOOST procedure, TARGET statement, 258
LEVEL= option, 258
GRADBOOST procedure, TRANSFERLEARN statement, 259
BURN = option, 259
SHRINKAGE = option, 259
TRIMMING = option, 259
GRADBOOST procedure, VIICODE statement, 259
ADD option, 259
LIMIT= option, 260
MISS option, 260
MULTIPLY option, 260
SUBTRACT option, 260
THRESHOLD= option, 260
GRAPHOUT= option
PROC MTLEARN statement, 346
GRAPHTABLE= option
PROC MTLEARN statement, 346
GRAPHTYPE= option
PROC MTLEARN statement, 346
GROUP= option
SELECT statement, 583
GROW statement
FOREST procedure, 182
GVARCLUS procedure, DISPLAY statement
CASESENSITIVE option, 290
EXCLUDE option, 290
EXCLUDEALL option, 290
TRACE option, 291
GVARCLUS procedure, DISPLAYOUT statement
INCLUDEALL option, 291
NOREPLACE option, 291
REPEATED option, 291
GVARCLUS procedure, INPUT statement, 292
LEVEL= option, 292
GVARCLUS procedure, PROC GVARCLUS statement, 287
DATA= option, 287
EXACT option, 288
MAXITER= option, 288
MAXMEMBER= option, 288
MAXSTEP= option, 288
MINCLUSTER= option, 288
OUTCP= option, 288
OUTEDGE= option, 288
OUTTREE= option, 289
OUTVERT= option, 289
RHO= option, 289
SELECT= option, 289
STOP= option, 289
TOL= option, 289
TOLERANCE= option, 289
GVARCLUS procedure, syntax, 286
HIDDEN statement
NNET procedure, 397
HIERARCHY statement
MBANALYSIS procedure, 330
HISTORYTABLE= option
AUTOTUNE statement, 14
ID statement
BNET procedure, 70
FACTMAC procedure, 133
FASTKNN procedure, 148
FOREST procedure, 183
GRADBOOST procedure, 255
KPCA procedure, 314
MWPCA procedure, 372
RPCA procedure, 433
SVDD procedure, 486
SVMACHINE procedure, 554
ID= option
DOCINFO statement, 99
TERMININFO statement, 101
IGNORE option
SELECT statement, 583
IGR option
GROW statement (FOREST), 183
IN_TERMS= option
SVD statement, 585
INBAGFRACTION= option
PROC FOREST statement, 173
INCLUDEALL option
DISPLAYOUT statement (FACTMAC), 133
DISPLAYOUT statement (FASTKNN), 147
DISPLAYOUT statement (GMM), 219
DISPLAYOUT statement (GVARCLUS), 291
DISPLAYOUT statement (KPCA), 313
DISPLAYOUT statement (MTLEARN), 353
DISPLAYOUT statement (MWPCA), 371
DISPLAYOUT statement (RPCA), 433
DISPLAYOUT statement (TSNE), 633
INDENTSIZE= option
CODE statement (ASTORE), 19
CODE statement (BOOLRULE), 19
CODE statement (FACTMAC), 19
CODE statement (FOREST), 19
CODE statement (GRADBOOST), 19
CODE statement (NNET), 19
CODE statement (SVMACHINE), 19
CODE statement (TEXTMINE), 19
CODE statement (TMSCORE), 19
INMODEL= option
  PROC FOREST statement, 173
  PROC GRADBOOST statement, 243
PROC NNET statement, 389
INPUT statement
  BNET procedure, 70
  FACTMAC procedure, 133
  FASTKNN procedure, 148
  FOREST procedure, 183
  GMM procedure, 219
  GRADBOOST procedure, 255
  GVARCLUS procedure, 292
  KPCA procedure, 314
  MTLREAR procedure, 353
  MWPCA procedure, 372
  NNET procedure, 398
  RPCA procedure, 433
  SEMISUPLEARN procedure, 453
  SPARSEML procedure, 467
  SVDD procedure, 487
  SVMACHINE procedure, 554
  TSNE procedure, 633
ISOLATION option
  PROC FOREST statement, 173
ITEMS= option
  PROC FISM statement, 157
  PROC MBANALYSIS statement, 329
ITERATIONREPORT option
  PROC SVMACHINE statement, 550
K= option
  PROC FASTKNN statement, 145
  SVD statement, 585
KEEP option
  SELECT statement, 583
KEEPVARS, KEEPVARIABLES
  SVD statement, 586
KERNEL statement
  KPCA procedure, 314
  SEMISUPLEARN procedure, 453
  SVDD procedure, 487
  SVMACHINE procedure, 555
KFOLD= option
  AUTOTUNE statement, 14
  CROSSVALIDATION statement, 182, 255, 397
KNN option
  PROC SEMISUPLEARN statement, 453
KPCA procedure, DISPLAY statement
  CASESENSITIVE option, 313
  EXCLUDE option, 313
  EXCLUDEALL option, 313
  TRACE option, 313
KPCA procedure, DISPLAYOUT statement
  INCLUDEALL option, 313
  NOREPLACE option, 313
  REPEATED option, 314
KPCA procedure, ID statement, 314
KPCA procedure, INPUT statement, 314
KPCA procedure, KERNEL statement, 314
  LINEAR option, 314
  POLYNOMIAL option, 314
  RBF option, 314
KPCA procedure, LRAPPROXIMATION statement, 315
  CCRITERION option, 315
  CLUSMETHOD= option, 315
  MAXCLUS option, 315
  MAXITER option, 315
KPCA procedure, OUTPUT statement, 316
  NPC= option, 316
  OUT= option, 316
KPCA procedure, PROC KPCA statement, 310
  CENTER option, 311
  CENTROIDS= option, 311
  DATA= option, 311
  EIGENVAL= option, 311
  EIGENVEC= option, 311
  METHOD= option, 311
  NTHREADS= option, 312
  RANKTHRESHOLD= option, 312
  SCALE option, 312
KPCA procedure, SAVESTATE statement, 316
  EXACTSCORE=, 316
  RSTORE=, 316
KPCA procedure, syntax, 310
LABEL= option
  PROC SEMISUPLEARN statement, 450
  TERMTINFO statement, 101
LABELID= option
  AUTO SELECT statement, 583
LABELS option
  SELECT statement, 583
LAMBDA= option
  RPCA statement, 374
  PROC RPCA statement, 428
LAMBDAWEIGHT= option
  RPCA statement, 374
**Syntax Index**

- **PROC RPCA statement, 428**
- **LANGUAGE= option**
- **PROC TEXTMINE statement, 577**
- **LASSO= option**
- **PROC GRADBOOST statement, 244**
- **LEARNINGRATE= option**
- **PROC GRADBOOST statement, 244**
- **PROC TSNE statement, 627**
- **LEARNINGRATE= option**
- **OPTIMIZATION statement, 399, 400**
- **LEARNSTEP= option**
- **PROC FACTMAC statement, 127**
- **LEVEL= option**
- **INPUT statement, 183, 256, 292, 398, 487, 554**
- **OUTPUT statement, 135**
- **PROC FACTMAC statement, 133**
- **PROC MTLEARN statement, 353**
- **TARGET statement, 186, 258, 354, 405, 557**
- **LIFT= option**
- **PROC MBANALYSIS statement, 329**
- **LIMIT= option**
- **VIICODE statement (FOREST), 186**
- **VIICODE statement (GRADBOOST), 260**
- **LINEAR option**
- **PROC SVMACHINE statement, 555**
- **LINESIZE= option**
- **CODE statement (ASTORE), 19**
- **CODE statement (BOOLRULE), 19**
- **CODE statement (FACTMAC), 19**
- **CODE statement (FOREST), 19**
- **CODE statement (GRADBOOST), 19**
- **CODE statement (NNET), 19**
- **CODE statement (SVMACHINE), 19**
- **CODE statement (TEXTMINE), 19**
- **CODE statement (TMSCORE), 19**
- **LISTNODE= option**
- **PROC NNET statement, 389**
- **LIVEUPDATE option**
- **AUTOTUNE statement, 14**
- **LOCALSEARCH option**
- **AUTOTUNE statement, 14**
- **LOH= option**
- **PROC FOREST statement, 173**
- **LOOKUPTABLE= option**
- **AUTOTUNE statement, 14**
- **LRAPPROXIMATION statement**
- **KPCA procedure, 315**
- **MAX_K= option**
- **SVD statement, 585**
- **MAXBAYES= option**
- **AUTOTUNE statement, 14**
- **MAXBRANCH= option**
- **PROC FOREST statement, 174**
- **PROC GRADBOOST statement, 244**
- **MAXBSKTSZ= option**
- **PROC MBANALYSIS statement, 329**
- **MAXCANDIDATES= option**
- **PROC BOOLRULE statement, 97**
- **MAXCANDS= option**
- **PROC BOOLRULE statement, 97**
- **MAXCLUS= option**
- **LRAPPROXIMATION statement, 315**
- **MAXCLUSTERS= option**
- **PROC GMM statement, 217**
- **MAXDEPTH= option**
- **PROC FOREST statement, 174**
- **PROC GRADBOOST statement, 244**
- **MAXEVALS= option**
- **AUTOTUNE statement, 14**
- **MAXITER= option**
- **AUTOTUNE statement, 15**
- **LRAPPROXIMATION statement, 315**
- **OPTIMIZATION statement, 401**
- **PROC FACTMAC statement, 127**
- **PROC GVARCLUS statement, 288**
- **PROC RPCA statement, 428**
- **PROC SEMISUPLEARN statement, 450**
- **PROC SVMACHINE statement, 550**
- **RPCA statement, 374**
- **SOLVER statement, 489**
- **MAXITERS= option**
- **PROC MTLEARN statement, 347**
- **PROC SPARSEML statement, 465**
- **PROC TSNE statement, 627**
- **MAXMEMBER= option**
- **PROC GVARCLUS statement, 288**
- **MAXRANK= option**
- **SVD statement, 375, 435**
- **MAXSV= option**
- **SOLVER statement, 489**
- **MAXTIME= option**
- **AUTOTUNE statement, 15**
- **OPTIMIZATION statement, 401**
- **MAXTRAINTIME= option**
- **AUTOTUNE statement, 15**
- **MAXTRIESIN= option**
- **PROC BOOLRULE statement, 97**
- **MAXTRIESOUT= option**
- **PROC BOOLRULE statement, 98**
- **MBANALYSIS procedure, CUSTOMER statement, 330**
- **MBANALYSIS procedure, HIERARCHY statement, 330**
- **DATA= option, 330**
- **MBANALYSIS procedure, OUTPUT statement, 331**
- **OUT= option, 331**
- **OUTFREQ= option, 331**
OUTRULE= option, 331
MBANALYSIS procedure, PROC MBANALYSIS statement, 328
ANTECEDENTLIST= option, 329
CONF= option, 329
CONSEQUENTLIST= option, 329
DATA= option, 328
ITEMS= option, 329
LIFT= option, 329
MAXBSKTSZ= option, 329
MINBSKTSZ= option, 329
NLHS_RANGE= option, 329
NORM, 330
NRHS_RANGE= option, 330
PCTSUP= option, 330
SEPARATOR= option, 330
SUP_LIFT= option, 330
SUPPORT= option, 328
MBANALYSIS procedure, SAVESTATE statement, 331
RSTORE= option, 331
MBANALYSIS procedure, syntax, 328
MBANALYSIS procedure, TARGET statement, 332
METHOD= option
ANOMALYDETECTION statement, 430
PROC FASTKNN statement, 145
PROC KPCA statement, 311
PROC RPCA statement, 428
RPCA statement, 374
SVD statement, 375, 435
MINBSKTSZ= option
PROC MBANALYSIS statement, 329
MINCLUSTER= option
PROC GV ARCLUS statement, 288
MINIBATCHSIZE= option
OPTIMIZATION statement, 399, 401
MINLEAFSIZE= option
PROC FOREST statement, 174
PROC GRADBOOST statement, 244
MINSUPPORTS= option
PROC BOOLRULE statement, 98
PROC NNET statement, 389
MLP DIRECT option
ARCHITECTURE statement, 391
MLP option
ARCHITECTURE statement, 390
MNEG= option
PROC BOOLRULE statement, 98
MODELOUT= option
PROC MTLEARN statement, 347
MOMENTUM= option
OPTIMIZATION statement, 399, 401
MONOTONIC = option
INPUT statement, 256
MPOS= option
PROC BOOLRULE statement, 98
MTLEARN procedure, AUTOTUNE statement, 347
TUNINGPARAMETERS= option, 348
MTLEARN procedure, DISPLAY statement
CASESENSITIVE option, 352
EXCLUDE option, 352
EXCLUDEALL option, 352
TRACE option, 352
MTLEARN procedure, DISPLAYOUT statement
INCLUDEALL option, 353
NOREPLACE option, 353
REPEATED option, 353
MTLEARN procedure, INPUT statement, 353
MTLEARN procedure, OUTPUT statement, 353
COPYVARS= option, 354
OUT= option, 353
MTLEARN procedure, PROC MTLEARN statement, 345
DATA= option, 346
GRAPHOUT= option, 346
GRAPHTABLE= option, 346
GRAPHTYPE= option, 346
LEVEL= option, 353
MAXITERS= option, 347
MODELOUT= option, 347
NTHREADS= option, 347
SEED= option, 347
TOLERANCE= option, 347
MTLEARN procedure, SAVESTATE statement, 354
RSTORE= option, 354
MTLEARN procedure, syntax, 345
MTLEARN procedure, TARGET statement, 354
LEVEL= option, 354
MU= option
RPCA statement, 374
PROC RPCA statement, 429
MULTIPLY option
VIICODE statement (FOREST), 186
VIICODE statement (GRADBOOST), 260
MULTITERM= option
PARSE statement, 579
MWPCA procedure, DISPLAY statement
CASESENSITIVE option, 371
EXCLUDE option, 371
EXCLUDEALL option, 371
TRACE option, 371
MWPCA procedure, DISPLAYOUT statement
  INCLUDEALL option, 371
  NOREPLACE option, 371
  REPEATED option, 372
MWPCA procedure, ID statement, 372
MWPCA procedure, INPUT statement, 372
MWPCA procedure, OUTPUT statement, 372
  NPC= option, 373
  OUT= option, 372
  PCANGLES= option, 373
  STANDARDPC= option, 373
MWPCA procedure, PROC MWPCA statement, 369
  CENTER, 369
  CUMEIGPCTTOL= option, 369
  DATA= option, 369
  NTHREADS= option, 370
  ROBUST= option, 370
  STEPSIZE= option, 369
  WINDOWSIZEx option, 369
MWPCA procedure, RPCA statement, 374
  FIXEDMU option, 374
  LAMBDA= option, 374
  LAMBDAWEIGHT= option, 374
  MAXITER= option, 374
  METHOD= option, 374
  MU= option, 374
  TOLERANCE= option, 374
MWPCA procedure, SVD statement, 375
  MAXRANK= option, 375
MWPCA procedure, syntax, 368
NBINS option
  PROC SVDD statement, 485
NCONVITER= option
  AUTOTUNE statement, 15
NDIMENSIONS= option
  PROC TSNE statement, 628
NEWVARNAMES
  TEXTMINE statement, 577
NFACTORS= option
  PROC FACTMAC statement, 127
NFIS_RANGE= option
  PROC FISM statement, 157
NLHS_RANGE= option
  PROC MBANALYSIS statement, 329
NMATCH = option
  SOLVER statement, 490
NNET procedure, ARCHITECTURE statement, 390
  GLIM option, 390
  MLP DIRECT option, 391
  MLP option, 390
NNET procedure, AUTOTUNE statement, 391
  TUNINGPARAMETERS= option, 392
NNET procedure, CODE statement, 396
  COMMENT option, 19
  FILE= option, 19, 396
  FORMATWIDTH= option, 19
  INDENTSIZE= option, 19
  LABELID= option, 19
  LINESIZE= option, 19
  NOCOMPPGM option, 396
  NOTRIM option, 20
  OUT= option, 20
NNET procedure, CROSSVALIDATION statement, 397
  KFOLD= option, 397
  NOPARALLEL option, 397
  NSUBSESSIONWORKERS= option, 397
NNET procedure, HIDDEN statement, 397
  ACT= option, 397
  COMB= option, 398
NNET procedure, INPUT statement, 398
  LEVEL= option, 398
NNET procedure, OPTIMIZATION statement, 398
  ALGORITHM= option, 399
  ANNEALINGRATE= option, 399, 400
  COMMFRQ= option, 399, 400
  LEARNINGRATE= option, 399, 400
  MAXITER= option, 401
  MAXTIME= option, 401
  MINIBATCHSIZE= option, 399, 401
  MOMENTUM= option, 399, 401
  NUMCORRECTIONS= option, 400
  REGL1= option, 401
  REGL2= option, 401
  SEED= option, 401, 400
  USELOCKING option, 401, 400
NNET procedure, OUTPUT statement, 402
  COPYVARS= option, 402
  OUT= option, 402
  ROLE= option, 402
NNET procedure, PARTITION statement, 403
  FRACTION option, 403
  ROLEVAR= option, 403
NNET procedure, PROC NNET statement, 388
  DATA= option, 388
  INMODEL= option, 389
  LISTNODE= option, 389
  MISSING= option, 389
  NTHREADS= option, 389
  PRINTTARGET, 389
  STANDARDIZE= option, 390
  TARGETSTANDARDIZE= option, 390
NNET procedure, syntax, 388
NNET procedure, TARGET statement, 390
  ACT= option, 404
  COMB= option, 404
  ERROR= option, 404
LEVEL= option, 405
NNET procedure, TRAIN statement, 405
  DROPOUTHIDDEN= ratio, 406
  DROPOUTINPUT= option, 406
  NUMTRIES= option, 406
  OUTMODEL= option, 405
  RESUME option, 406
  STAGNATION= option, 406
  VALIDATION= option, 406
  VALIDGOAL= option, 406
  WSEED= option, 406
NNET procedure, WEIGHT statement, 407
NOCOMPPGM option
  CODE statement, 396
NOCUTOFFS option
  SVD statement, 585
NOGRIDSHUFFLE option
  AUTOTUNE statement, 15
NOMSEARCH option
  PROC GRADBOOST statement, 245
NONG option
  PARSE statement, 579
NONNEGATIVE option
  PROC FACTMAC statement, 127
NONOUNGROUPS option
  PARSE statement, 579
NOPARALLEL option
  CROSSVALIDATION statement, 182, 255, 397
NOPRINT
  PROC FACTMAC statement, 127
NOPRINT option
  PROC FASTKNN statement, 145
  PROC FOREST statement, 174
  PROC GRADBOOST statement, 245
  PROC SVDD statement, 485
  PROC SVMACHINE statement, 550
  PROC TSNE statement, 628
NOREPLACE option
  DISPLAYOUT statement (FACTMAC), 133
  DISPLAYOUT statement (FASTKNN), 148
  DISPLAYOUT statement (GMM), 219
  DISPLAYOUT statement (GVARCLUS), 291
  DISPLAYOUT statement (KPCA), 313
  DISPLAYOUT statement (MTLEARN), 353
  DISPLAYOUT statement (MWPCA), 371
  DISPLAYOUT statement (RPCA), 433
  DISPLAYOUT statement (TSNE), 633
NORM
  PROC MBANALYSIS statement, 330
NOSCALE option
  PROC SVMACHINE statement, 550
NOSTEMMING option
  PARSE statement, 580
NOTAGGING option
  PARSE statement, 580
PARSE statement, 580
NOTRIM option
  CODE statement (ASTORE), 20
  CODE statement (BOOLRULE), 20
  CODE statement (FACTMAC), 20
  CODE statement (FOREST), 20
  CODE statement (GRADBOOST), 20
  CODE statement (NNET), 20
  CODE statement (SVMACHINE), 20
  CODE statement (TEXTMINE), 20
  CODE statement (TMSCORE), 20
NPC= option
  OUTPUT statement, 316, 373
NRHS_RANGE= option
  PROC MBANALYSIS statement, 330
NSAMP= option
  SOLVER statement, 490
NSUBSESSIONWORKERS= option
  AUTOTUNE statement, 15
  CROSSVALIDATION statement, 182, 255, 397
NTHREADS= option
  PROC FACTMAC statement, 127
  PROC FASTKNN statement, 145
  PROC FOREST statement, 174
  PROC GRADBOOST statement, 245
  PROC KPCA statement, 312
  PROC MTLEARN statement, 347
  PROC MWPCA statement, 370
  PROC NNET statement, 389
  PROC PCA statement, 429
  PROC SEMISUPLEARN statement, 450
  PROC SPARSEML statement, 465
  PROC SVDD statement, 485
  PROC SVMACHINE statement, 550
  PROC TEXTMINE statement, 577
  PROC TSNE statement, 628
NTREES= option
  PROC FOREST statement, 174
  PROC GRADBOOST statement, 245
NUMBIN= option
  PROC FOREST statement, 174
  PROC GRADBOOST statement, 245
NUMCORRECTIONS= option
  OPTIMIZATION statement, 400
NUMLABELS= option
  SVD statement, 586
NUMSIGVARS= option
  ANOMALYDETECTION statement, 430
NUMTRIES= option
  TRAIN statement, 406
OBJECTIVE= option
  AUTOTUNE statement, 16
OFFSET option
PROC GRADBOOST statement, 245

OPTIMIZATION statement
NNET procedure, 398

OTHRESH= option
SOLVER statement, 489

OUT= option
CODE statement (ASTORE), 20
CODE statement (BOOLRULE), 20
CODE statement (FACTMAC), 20
CODE statement (FOREST), 20
CODE statement (GRADBOOST), 20
CODE statement (NNET), 20
CODE statement (SVMACHINE), 20
CODE statement (TEXTMINE), 20
CODE statement (TMSCORE), 20
OUTPUT statement, 157, 331
OUTPUT statement (BNET), 71
OUTPUT statement (FACTMAC), 134
OUTPUT statement (FASTKNN), 148
OUTPUT statement (FOREST), 184
OUTPUT statement (GMM), 220
OUTPUT statement (GRADBOOST), 256
OUTPUT statement (KPCA), 316
OUTPUT statement (MBANALYSIS), 331
OUTPUT statement (MTLEARN), 353
OUTPUT statement (MWPCA), 372
OUTPUT statement (NNET), 402
OUTPUT statement (SEMISUPLEARN), 454
OUTPUT statement (SPARSEML), 467
OUTPUT statement (SVMACHINE), 555
OUTPUT statement (TSNE), 633
SCORE statement (ASTORE), 34
OUTCHILD= option
PARSE statement, 580
OUTCONFIG= option
PARSE statement, 580
OUTCP= option
PROC GVARCLUS statement, 288
OUTDECOMP statement
RPCA procedure, 433
OUTDIST= option
PROC FASTKNN statement, 145
OUTDOCPRO= option
SVD statement, 586
OUTEDGE= option
PROC GVARCLUS statement, 288
OUTERROR= option
PROC RPCA statement, 429
OUTFREQ= option
OUTPUT statement, 157, 331
OUTLIER_FRACTION= option
PROC SVDD statement, 485
OUTLOWRANK= option
PROC RPCA statement, 429
OUTMATCH= option
SCORE statement, 100
OUTMODEL
PROC FACTMAC statement, 127
OUTMODEL option
PROC GRADBOOST statement, 246
OUTMODEL= option
PROC FASTKNN statement, 174
TRAIN statement, 405
OUTPARENT= option
PARSE statement, 580
PROC TMSCORE statement, 621
OUTPOS= option
PARSE statement, 580
OUTPUT statement
BNET procedure, 71
BOOLRULE procedure, 99
FACTMAC procedure, 134
FASTKNN procedure, 148
FISM procedure, 157
FOREST procedure, 184
GRADBOOST procedure, 256
KPCA procedure, 316
MBANALYSIS procedure, 331
MTLEARN procedure, 353
MWPCA procedure, 372
NNET procedure, 402
SEMISUPLEARN procedure, 454
SPARSEML procedure, 467
SVMACHINE procedure, 555
TSNE procedure, 633
OUTRULE= option
OUTPUT statement, 331
OUTSPARSE= option
PROC KPCA statement, 311
PROC RPCA statement, 429
OUTSV= option
PROC SVDD statement, 486
OUTTERMS= option
PARSE statement, 581
OUTTOPICS= option
SVD statement, 586
OUTTREE= option
PROC KVARCLUS statement, 289
OUTVERT= option
PROC GVARCLUS statement, 289
PARALLELIZATION= option
PROC FASTKNN statement, 145
PARSE statement
TEXTMINE procedure, 578
PARTITION statement
BNET procedure, 71
FOREST procedure, 20, 185
PROC FACTMAC statement, see FACTMAC procedure
PROC FASTKNN statement, see FASTKNN procedure
PROC FOREST statement, see FOREST procedure
PROC GMM statement, see GMM procedure
PROC GRADBOOST statement, see GRADBOOST procedure
PROC GVARCLUS statement, see GVARCLUS procedure
PROC KPCA statement, see KPCA procedure
PROC MTLEARN statement, see MTLEARN procedure
PROC MWPCA statement, see MWPCA procedure
PROC NNET statement, see NNET procedure
PROC PCA statement, see PCA procedure
PROC SEMISUPLEARN statement, see SEMISUPLEARN procedure
PROC SPARSEML statement, see SPARSEML procedure
PROC SVDD statement, see SVDD procedure
PROC SVMACHINE statement, see SVMACHINE procedure
PROC TEXTMINE statement
TEXTMINE procedure, 576
PROC TMSCORE statement
TMSCORE procedure, 620
PROC TSNE statement, see TSNE procedure
PROJECTI0NTYPE= option
CODE statement, 431
QUERY= option
PROC FASTKNN statement, 146
RBAIMP option
PROC FOREST statement, 175
PROC GRADBOOST statement, 246
RBF option
KERNEL statement, 314, 487
PROC SEMISUPLEARN statement, 453
REDUCEF= option
PARSE statement, 581
REGL1= option
PROC MTLEARN statement, 347
PROC SPARSEML statement, 466
REGL2= option
OPTIMIZATION statement, 401
REPEATED option
DISPLAYOUT statement (FACTMAC), 133
DISPLAYOUT statement (FASTKNN), 148
DISPLAYOUT statement (GMM), 219
DISPLAYOUT statement (GVARCLUS), 291
DISPLAYOUT statement (KPCA), 314
DISPLAYOUT statement (MTLEARN), 353
DISPLAYOUT statement (MWPCA), 372
DISPLAYOUT statement (RPCA), 433
DISPLAYOUT statement (TSNE), 633
RES= option
SVD statement, 586
RESOLUTION= option
SVD statement, 586
RESUME option
TRAIN statement, 406
RHO= option
PROC GVARCLUS statement, 289
RIDGE= option
PROC GRADBOOST statement, 246
ROBUST= option
PROC MWPCA statement, 370
ROLE option
OUTPUT statement, 184, 257
ROLE= option
OUTPUT statement, 71, 402, 556
ROLEVAR= option
PARTITION statement, 403
PARTITION statement (BNET), 72
PARTITION statement (FOREST), 20, 185
PARTITION statement (GRADBOOST), 20, 258
PARTITION statement (SVMACHINE), 557
ROTATION= option
SVD statement, 587
ROW= option
SVD statement, 587
RPCA procedure, ANOMALYDETECTION statement, 430
METHOD= option, 430
NUMSIGVARS= option, 430
SIGMACOEF= option, 430
USEMATRIX= option, 430
RPCA procedure, CODE statement, 431
FILE= option, 431
PROJECTTYPE= option, 431
RPCA procedure, DISPLAY statement
CASESENSITIVE option, 432
EXCLUDE option, 432
EXCLUDEALL option, 432
TRACE option, 432
RPCA procedure, DISPLAYOUT statement
INCLUDEALL option, 433
NOREPLACE option, 433
REPEATED option, 433
RPCA procedure, ID statement, 433
RPCA procedure, INPUT statement, 433
RPCA procedure, OUTDECOMP statement, 433
PCLOADINGS= option, 434
PCSCORES= option, 434
SVDDIAG= option, 433
SVDLEFT= option, 434
SVDRIGHT= option, 434
RPCA procedure, PROC RPCA statement, 426
CENTER option, 427
COLSTATISTICS= option, 427
CUMEIGPCTTOL= option, 427
DATA= option, 427
DECOMP= option, 428
FIXEDMU option, 428
LAMBDA= option, 428
LAMBDWEIGHT= option, 428
MAXITER= option, 428
METHOD= option, 428
MU= option, 429
NTHREADS= option, 429
OUTERROR= option, 429
OUTLOWRANK= option, 429
OUTSPARSE= option, 429
PCPREFIX= option, 429
SCALE option, 429
TOLERANCE= option, 429
RPCA procedure, SAVESTATE statement, 434
RSTORE=, 434
RPCA procedure, SVD statement, 435
METHOD= option, 435
RPCA procedure, syntax, 426
RPCA statement
MWPCA procedure, 374
RSS option
GROW statement (FOREST), 183
RSTORE= option
DOWNLOAD statement, 33
PROC SVDD statement, 488
SAVESTATE statement, 72, 134, 185, 219, 258, 316, 331, 354, 434, 468, 557, 582
UPLOAD statement, 41
RSTORE= option
DESCRIBE statement, 32
RTOL= option
SOLVER statement, 490
RULES= option
OUTPUT statement, 100
RULETERMS= option
OUTPUT statement, 100
SCORE statement, 100
SAMPLESIZE= option
AUTOTUNE statement, 17
SAMPLINGRATE option
PROC GRADBOOST statement, 246
SAVESTATE statement
BNET procedure, 72
FACTMAC procedure, 134
FOREST procedure, 185
GMM procedure, 219
GRADBOOST procedure, 258
KPCA procedure, 316
MBANALYSIS procedure, 331
MTLEARN procedure, 354
RPCA procedure, 434
SPARSEML procedure, 468
SVDD procedure, 488
SVMMACHINE procedure, 557, 582
SCALE option
   PROC KPCA statement, 312
   PROC RPCA statement, 429
   PROC SVMACHINE statement, 551
SCORE option
   PROC SVDD statement, 485
SCORE statement
   ASTORE procedure, 34
   BOOLRULE procedure, 100
   GMM procedure, 220
SEARCHMETHOD= option
   AUTOTUNE statement, 17
SECONDOBJECTIVE= option
   AUTOTUNE statement, 17
SEED= option
   PROC FACTMAC statement, 128
   PROC FASTKNN statement, 146
   PROC FOREST statement, 175
   PROC GMM statement, 217
   PROC GRADBOOST statement, 246
   PROC MTLEARN statement, 347
   PROC SEMISUPLEARN statement, 450
   PROC SPARSEML statement, 466
   PROC TSNE statement, 628
   SOLVER statement, 490
   SVD statement, 435
SEED= option
   OPTIMIZATION statement, 400, 401
SELECT statement
   TEXTMINE procedure, 582
SELECT= option
   PROC GVARCLUS statement, 289
SELECTINITPOINT option
   AUTOTUNE statement, 17
SEMISUPLEARN procedure, AUTOTUNE statement, 450
SEMISUPLEARN procedure, INPUT statement, 453
SEMISUPLEARN procedure, KERNEL statement, 453
SEMISUPLEARN procedure, OUTPUT statement, 454
   COPYVARS= option, 454
OUT= option, 454
SEMISUPLEARN procedure, PROC
   SEMISUPLEARN statement, 449
   DATA= option, 450
   GAMMA= option, 450
   KNN option, 453
   LABEL= option, 450
   MAXITER= option, 450
   NTHREADS= option, 450
   RBF option, 453
   SEED= option, 450
SEMISUPLEARN procedure, syntax, 449
SEMISUPLEARN procedure, TARGET statement, 454
SEPARATOR= option
   PROC MBANALYSIS statement, 330
SETOPTION statement
   ASTORE procedure, 35
   SHOWDROPPEDTERMS= option
   PARSE statement, 581
SHRINKAGE= option
   TRANSFERLEARN statement, 259
SIGMACOEFF= option
   ANOMALYDETECTION statement, 430
SOLVER statement
   SVDD procedure, 488
   SPARSEML procedure, INPUT statement, 467
   SPARSEML procedure, OUTPUT statement, 467
   COPYVARS= option, 467
   OUT= option, 467
   SPARSEML procedure, PROC SPARSEML statement, 465
      C= option, 465
      DATA= option, 465
      MAXITERS= option, 465
      NTHREADS= option, 465
      PRINTLEVEL= option, 466
      REGL1= option, 466
      REGL2= option, 466
      SEED= option, 466
      TOLERANCE= option, 466
   SPARSEML procedure, SAVESTATE statement, 468
   RSTORE= option, 468
   SPARSEML procedure, syntax, 465
STAGNATION= option
   TRAIN statement, 406
STANDARDIZE= option
   PROC NNET statement, 390
   STANDARDPC= option
   OUTPUT statement, 373
START= option
   PARSE statement, 581
STD option
   PROC SVDD statement, 485
<table>
<thead>
<tr>
<th>Option/Keyword</th>
<th>Statement/Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>STEPSIZE= option</td>
<td>PROC MWPCA statement, 369</td>
<td></td>
</tr>
<tr>
<td>STOCHS option</td>
<td>PROC SVDD statement, 489</td>
<td></td>
</tr>
<tr>
<td>STOP= option</td>
<td>PARSE statement, 581</td>
<td></td>
</tr>
<tr>
<td>STORE= option</td>
<td>DESCRIBE statement, 32</td>
<td></td>
</tr>
<tr>
<td>SUBTRACT option</td>
<td>VIICODE statement (FOREST), 186</td>
<td></td>
</tr>
<tr>
<td>SUP_LIFT= option</td>
<td>PROC MBANALYSIS statement, 330</td>
<td></td>
</tr>
<tr>
<td>SUPPORT= option</td>
<td>PROC FISM statement, 156</td>
<td></td>
</tr>
<tr>
<td>SVD statement</td>
<td>MWPCA procedure, 375</td>
<td></td>
</tr>
<tr>
<td>SVD statement</td>
<td>RPCA procedure, 435</td>
<td></td>
</tr>
<tr>
<td>SVDD procedure, CODE statement, 486</td>
<td>FILE= option, 486</td>
<td></td>
</tr>
<tr>
<td>SVDD procedure, ID statement, 486</td>
<td>LEVEL= option, 487</td>
<td></td>
</tr>
<tr>
<td>SVDD procedure, INPUT statement, 487</td>
<td>RBF option, 487</td>
<td></td>
</tr>
<tr>
<td>SVDD procedure, KERNEL statement, 487</td>
<td>DATA= option, 485</td>
<td></td>
</tr>
<tr>
<td>SVDD procedure, PROC SVDD statement, 485</td>
<td>NBINS, 485</td>
<td></td>
</tr>
<tr>
<td>SVDD procedure, PROC SVDD statement, 485</td>
<td>NOPRINT, 485</td>
<td></td>
</tr>
<tr>
<td>SVDD procedure, PROC SVDD statement, 485</td>
<td>NTHREADS= option, 485</td>
<td></td>
</tr>
<tr>
<td>SVDD procedure, PROC SVDD statement, 485</td>
<td>OUTLIER_FRACTION= option, 485</td>
<td></td>
</tr>
<tr>
<td>SVDD procedure, PROC SVDD statement, 485</td>
<td>OUTSV=, 486</td>
<td></td>
</tr>
<tr>
<td>SVDD procedure, Kernel statement, 487</td>
<td>RSTORE=, 488</td>
<td></td>
</tr>
<tr>
<td>SVDD procedure, Kernel statement, 487</td>
<td>SCORE, 485</td>
<td></td>
</tr>
<tr>
<td>SVMACHINE procedure, ID statement, 554</td>
<td>LABELID= option, 19</td>
<td></td>
</tr>
<tr>
<td>SVMACHINE procedure, TUNINGPARAMETERS= option, 552</td>
<td>LINESIZE= option, 19</td>
<td></td>
</tr>
<tr>
<td>SVMACHINE procedure, CODE statement, 554</td>
<td>NOTRIM option, 20</td>
<td></td>
</tr>
<tr>
<td>SVMACHINE procedure, OUTPUT statement, 555</td>
<td>OUT= option, 20</td>
<td></td>
</tr>
<tr>
<td>SVMACHINE procedure, PARTITION statement</td>
<td>ROLE= option, 556</td>
<td></td>
</tr>
<tr>
<td>SVMACHINE procedure, PROC SVMACHINE statement, 549</td>
<td>DATA= option, 550</td>
<td></td>
</tr>
<tr>
<td>SVMACHINE procedure, PROC SVMACHINE statement, 549</td>
<td>EARLYSTOP, 550</td>
<td></td>
</tr>
<tr>
<td>SVMACHINE procedure, PROC SVMACHINE statement, 549</td>
<td>ITERATIONREPORT, 550</td>
<td></td>
</tr>
<tr>
<td>SVMACHINE procedure, PROC SVMACHINE statement, 549</td>
<td>LINEAR option, 555</td>
<td></td>
</tr>
<tr>
<td>SVMACHINE procedure, PROC SVMACHINE statement, 549</td>
<td>MAXITER= option, 550</td>
<td></td>
</tr>
<tr>
<td>SVMACHINE procedure, PROC SVMACHINE statement, 549</td>
<td>NOPRINT, 550</td>
<td></td>
</tr>
<tr>
<td>SVMACHINE procedure, PROC SVMACHINE statement, 549</td>
<td>NOSCALE, 550</td>
<td></td>
</tr>
</tbody>
</table>
NTHREADS= option, 550
POLYNOMIAL option, 555
PRINTTARGET option, 550
SCALE option, 551
TOLERANCE option, 551
USEMISS option, 551
SVMACHINE procedure, SAVESTATE statement, 557, 582
RSTORE= option, 557, 582
SVMACHINE procedure, syntax, 549
SVMACHINE procedure, TARGET statement, 557
LEVEL= option, 557
SYNONYM= option
PARSE statement, 581
syntax
  BOOLRULE procedure, 95
  TEXTMINE procedure, 576
  TMSCORE procedure, 620
TARGET statement
  BNET procedure, 72
  FACTMAC procedure, 135
  FISM procedure, 158
  FOREST procedure, 186
  GRADBOOST procedure, 258
  MBANALYSIS procedure, 332
  MTLEARN procedure, 354
  NNET procedure, 403
  SEMISUPLEARN procedure, 454
  SVMACHINE procedure, 557
  TEXTMINE procedure, 588
TARGET= option
  DOCINFO statement, 99
TARGETEVENT= option
  AUTOTUNE statement, 17
TARGETSTANDARDIZE= option
  PROC NNET statement, 390
TARGETTYPE= option
  DOCINFO statement, 99
TERMID= option
  PROC BOOLRULE statement, 98
TERMINFO statement
  BOOLRULE procedure, 101
TERMINFO= option
  PROC BOOLRULE statement, 98
TERMS= option
  PROC TMSCORE statement, 621
TERMWGT= option
  PARSE statement, 582
TEXTMINE procedure, 576
  PARSE statement, 578
  PROC TEXTMINE statement, 576
  SELECT statement, 582
  SVD statement, 584
TEXTMINE procedure, CODE statement
  COMMENT option, 19
  FILE= option, 19
  FORMATWIDTH= option, 19
  INDENTSIZE= option, 19
  LABELID= option, 19
  LINESIZE= option, 19
  NOTRIM option, 20
  OUT= option, 20
TEXTMINE procedure, DOC_ID statement, 577
TEXTMINE procedure, PARSE statement, 578
  CELLWGT= option, 579
  ENTITIES= option, 579
  MULTITERM= option, 579
  NONG option, 579
  NONOUNGROUPS option, 579
  NOSTEMMING option, 580
  NOTAGGING option, 580
  OUTCHILD= option, 580
  OUTCONFIG= option, 580
  OUTPARENT= option, 580
  OUTPOS= option, 580
  OUTTERMS= option, 581
  OUTTOPICS= option, 581
  REDUCEF= option, 581
  SHOWDROPPEDTERMS= option, 581
  START= option, 581
  STOP= option, 581
  SYNONYM= option, 581
  TERMWGT= option, 582
TEXTMINE procedure, PROC TEXTMINE statement, 576
  DATA= option, 577
  LANGUAGE= option, 577
  NEWVARNAMES, 577
  NTHREADS= option, 577
TEXTMINE procedure, SELECT statement, 582
  GROUP= option, 583
  IGNORE option, 583
  KEEP option, 583
  LABELS option, 583
TEXTMINE procedure, SVD statement, 584
  COL= option, 585
  ENTRY= option, 585
  EXACTWEIGHT option, 585
  IN_TERMS= option, 585
  K= option, 585
  KEEPVARS, KEEPVARIABLES, 586
  MAX_K= option, 585
  NOCUTOFFS option, 585
  NUMLABELS= option, 586
  OUTDOCPRO= option, 586
  OUTTOPICS= option, 586
  RES= option, 586
RESOLUTION= option, 586
ROTATION= option, 587
ROW= option, 587
SVDS= option, 587
SVDU= option, 587
SVDV= option, 587
TOL= option, 587
TEXTMINE procedure, TARGET statement, 588
TEXTMINE procedure, VAR statement, 588
TEXTMINE procedure, VARIABLES statement, 588
THRESHDIST= option
PROC FASTKNN statement, 146
THRESHOLD= option
VIICODE statement (FOREST), 186
VIICODE statement (GRADBOOST), 260
TMSCORE procedure, 620
PROC TMSCORE statement, 620
syntax, 620
TMSCORE procedure, CODE statement
COMMENT option, 19
FILE= option, 19
FORMATWIDTH= option, 19
INDENTSIZE= option, 19
LABELID= option, 19
LINESIZE= option, 19
NOTRIM option, 20
OUT= option, 20
TMSCORE procedure, DOC_ID statement, 622
TMSCORE procedure, PROC TMSCORE statement, 620
DATA= option, 620
DOC= option, 620
OUTPARENT= option, 621
SVDDOCPRO= option, 621
SVDU= option, 621
TERMS= option, 621
TMSCORE procedure, TMSCORE statement
CONFIG= option, 621
TMSCORE procedure, VAR statement, 622
TMSCORE procedure, VARIABLES statement, 622
TOL= option
PROC GVARCLUS statement, 289
SVD statement, 587
TOLERANCE option
PROC SVMACHINE statement, 551
TOLERANCE= option
PROC GVARCLUS statement, 289
PROC KPCA statement, 312
PROC MTLEARN statement, 347
PROC RPCA statement, 429
PROC SPARSEML statement, 466
RPCA statement, 374
TRACE option
DISPLAY statement (FACTMAC), 132
DISPLAY statement (FASTKNN), 147
DISPLAY statement (GMM), 218
DISPLAY statement (GVARCLUS), 291
DISPLAY statement (KPCA), 313
DISPLAY statement (MTLEARN), 352
DISPLAY statement (MWPCA), 371
DISPLAY statement (RPCA), 432
DISPLAY statement (TSNE), 632
TRAIN statement
NNET procedure, 405
TRAINFRACTION= option
AUTOTUNE statement, 18
TRANSFERLEARN statement
GRADBOOST procedure, 259
TRIMMING= option
TRANSFERLEARN statement, 259
TSNE procedure, AUTOTUNE statement, 628
TUNINGPARAMETERS= option, 629
TSNE procedure, DISPLAY statement
CASESENSITIVE option, 632
EXCLUDE option, 632
EXCLUDEALL option, 632
TRACE option, 632
TSNE procedure, DISPLAYOUT statement
INCLUDEALL option, 633
NOREPLACE option, 633
REPEATED option, 633
TSNE procedure, INPUT statement, 633
TSNE procedure, OUTPUT statement, 633
COPYVARS= option, 633
OUT= option, 633
TSNE procedure, PROC statement
SEED= option, 628
TSNE procedure, PROC TSNE statement, 627
DATA= option, 627
LEARNINGRATE= option, 627
MAXITERS= option, 627
NDIMENSIONS= option, 628
NPRINT option, 628
NTHREADS= option, 628
PERPLEXITY= option, 628
TSNE procedure, syntax, 626
TUNINGPARAMETERS= option
AUTOTUNE statement, 18, 64, 129, 177, 248, 348, 392, 451, 552, 629
UCC option
PROC SVDD statement, 485
UPLOAD statement
ASTORE procedure, 41
USELOCKING option
OPTIMIZATION statement, 400, 401
USEMATRIX= option
ANOMALYDETECTION statement, 430
USEMISS option
   PROC SVMACHINE statement, 551
USEPARAMETERS= option
   AUTOTUNE statement, 18

VALIDATION= option
   TRAIN statement, 406
VALIDGOAL= option
   TRAIN statement, 406

VAR statement
   TEXTMINE procedure, 588
   TMSCORE procedure, 622

VARIABLES statement
   TEXTMINE procedure, 588
   TMSCORE procedure, 622

VARS_TO.Try= option
   PROC FOREST statement, 175
   PROC GRADBOOST statement, 246

VII= option
   PROC FOREST statement, 175
   PROC GRADBOOST statement, 247

VIICODE statement
   FOREST procedure, 186
   GRADBOOST procedure, 259

VOTE= option
   PROC FOREST statement, 175

WEIGHT statement
   FOREST procedure, 187
   GRADBOOST procedure, 260
   NNET procedure, 407
   SVDD procedure, 490

WINDOWSIZE= option
   PROC MWPCA statement, 369

WINSUMMARY= option
   OUTPUT statement, 373

WSEED= option
   TRAIN statement, 406
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.