# Contents

Chapter 1. What’s New in SAS Visual Data Mining and Machine Learning 8.3 Procedures  
Chapter 2. Introduction  
Chapter 3. Shared Concepts  
Chapter 4. The ASTORE Procedure  
Chapter 5. The BNET Procedure  
Chapter 6. The BOOLRULE Procedure  
Chapter 7. The FACTMAC Procedure  
Chapter 8. The FASTKNN Procedure  
Chapter 9. The FISM Procedure  
Chapter 10. The FOREST Procedure  
Chapter 11. The GMM Procedure  
Chapter 12. The GRADBOOST Procedure  
Chapter 13. The GVARCLUS Procedure  
Chapter 14. The MBANALYSIS Procedure  
Chapter 15. The MTLEARN Procedure  
Chapter 16. The MWPCA Procedure  
Chapter 17. The NNET Procedure  
Chapter 18. The RPCA Procedure  
Chapter 19. The SEMISUPLEARN Procedure  
Chapter 20. The SVDD Procedure  
Chapter 21. The SVMACHINE Procedure  
Chapter 22. The TEXTMINE Procedure  
Chapter 23. The TMSCORE Procedure  
Chapter 24. The TSNE Procedure  

Subjects Index  
Syntax Index
Chapter 1
What’s New in SAS Visual Data Mining and Machine Learning 8.3 Procedures

Contents

Overview: What’s New ......................................................... 1
New Procedures ............................................................... 2
  The GMM Procedure ....................................................... 2
  The MTLEARN Procedure ............................................... 2
  The SEMISUPLEARN Procedure ...................................... 2
  The TSNE Procedure .................................................... 3
Enhancements to Procedures ............................................... 3
  The ASTORE Procedure ................................................ 3
  AUTOTUNING ............................................................... 3
  The BNET Procedure .................................................... 4
  The FISM Procedure ..................................................... 4
  The FOREST Procedure ................................................ 4
  The GRADBOOST Procedure ............................................ 5
  The GVARCLUS Procedure ............................................ 5
  The MBANALYSIS Procedure ......................................... 5
  The MWPCA Procedure ................................................ 5
  The RPCA Procedure ................................................... 6
  The SVDD Procedure ................................................... 6
References ........................................................................... 6

Overview: What’s New

SAS Visual Data Mining and Machine Learning 8.3 contains several new procedures and several enhancements to procedures.
New Procedures

The GMM Procedure

The GMM procedure performs clustering—a common step in data exploration—on quantitative data. 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.

The GMM procedure 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. As a benefit of probabilistic modeling, PROC GMM performs “soft clustering,” which provides not only the predicted cluster score but also the probability distribution over all clusters for each observation.

PROC GMM learns the proper number of clusters during the clustering process, which is supported by the Dirichlet process. This is a nonparametric Bayesian machine learning method.

The procedure uses variational Bayes (VB) as the model inference method. VB is an inference method for probabilistic models that approximates the (intractable) posterior distribution, then iteratively updates the parameters until it reaches convergence.

The MTLEARN Procedure

The MTLEARN procedure implements the multitask learning technique for least squares loss with $\ell_1$ (LASSO) and graph structure penalizations. It solves multiple related sparse linear regression problems simultaneously. You use a graph table to encode the relationships between the problems. The MTLEARN procedure shares the data and model parameters among different regression problems, and it solves them simultaneously in order to produce a more robust and accurate predictive model.

Classical machine learning systems treat each task independently and do not share data or model information among different tasks. Multitask learning systems assume that tasks are related. These systems share data and model information among tasks; the sharing is guided by the relationship information. In PROC MTLEARN, the relationship information is contained in a graph table, and you can solve multiple linear regressions at the same time. Each regression task can have a different number of observations, but all regression tasks must have the same dimension (same input variables).

When used with the independent graph table, PROC MTLEARN also serves as an elastic net regularization solver, which is widely used in feature selection.

The SEMISUPLEARN Procedure

The SEMISUPLEARN procedure implements the graph-based semisupervised learning algorithm that relies on label spreading 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 searches in parallel by making full use of multicore computers or a distributed computing environment.

PROC SEMISUPLEARN operates on a labeled data table and an unlabeled data table. For each observation in the query data table, the procedure returns the predicted labels for the observations in both the unlabeled data table and the labeled data table. It uses the radial basis function (RBF) kernel metric to compute distances between the pairs of 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.

The TSNE Procedure

The TSNE procedure implements the t-distributed stochastic neighbor embedding (t-SNE) algorithm for dimension reduction and visualization of high-dimensional data. The procedure minimizes the Kullback-Leibler divergence between high-dimensional and low-dimensional joint probabilities in a way that leverages the heavy-tailed Student’s t-distribution for solving the crowding problem, which is pervasive in dimension reduction. The lower-dimensional representations, or embeddings, that PROC TSNE produces have the advantage of emphasizing natural groupings in the data, making subsequent clustering easier and enabling the user to readily visualize and interpret the relationships between similar observations.

Enhancements to Procedures

The ASTORE Procedure

- DS2CODE is added as an alias for the EPCODE option in the SCORE statement.
- PROC ASTORE now supports models that are stored by the GMM and MTLEARN procedures and by the glm, genmod, logistic, and gampl actions.
- PROC ASTORE now supports BY-group processing if the underlying model supports it. Specifically, models that are stored by the GAM, GENSELECT, LOGSELECT, and REGSELECT procedures all support BY-group processing.

AUTOTUNING

Autotuning enhancements have been made in all procedures that support autotuning: BNET, FACTMAC, FOREST, GRADBOOST, NNET, and SVM.
The new value GRID for the SEARCHMETHOD= option enables you to request that autotuning use a grid search method to find the best hyperparameters.

The new MAXBAYES= option enables you to specify the maximum Bayesian model size for autotuning to use when you specify SEARCHMETHOD=BAYESIAN.

The new MAXTRAINTIME= option enables you to specify the maximum time allowed for training each individual model.

The new TRAINFRACTION= option enables you to specify the fraction of the data to use for training during autotuning.

The BNET Procedure

- The new AUTOTUNE statement in PROC BNET enables you to optimize a set of parameters that is larger than the set of parameters that are searched when you specify the BESTMODEL option.

- Two columns are added to the ODS table ValidInfo that show the misclassification rate and the number of observations for different parameter settings that are searched when you specify the BESTMODEL option.

- The ODS table FitStatistics has two additional rows, which show the validation misclassification rate and the test misclassification rate for partitioned data.

The FISM Procedure

- The new NFIS_RANGE option in the PROC FISM statement enables you to specify the range of the number of items in frequent item sets.

- The maximum limit of the ITEMS= option is set to 1,000.

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

The FOREST Procedure

- The new ISOLATION option in the PROC FOREST statement enables you to generate an isolation forest for anomaly detection.

- The new VIICODE statement enables you to request that SAS DATA step code be created to generate new interaction variables.

- The CROSSVALIDATION statement has new options, and the associated ODS table has more details.
The GRADBOOST Procedure

- The new TRANSFERLEARN statement enables you to request that the procedure use transfer learning to supplement training data with auxiliary data.
- The new VIICODE statement enables you to request that SAS DATA step code be created to generate new interaction variables.
- The CROSSVALIDATION statement has new options, and the associated ODS table has more details.

The GVARCLUS Procedure

- The new SELECT=PENALIZED option enables selection of the optimal clustering step to be based on minimization of the penalized log likelihood.

The MBANALYSIS Procedure

- The new NLHS_RANGE option in the PROC MBANALYSIS statement enables you to specify the range of number of items in the left-hand side (LHS) of a rule.
- The new NRHS_RANGE option in the PROC MBANALYSIS statement enables you to specify the range of number of items in the right-hand side (RHS) of a rule.
- The new ANTECEDENTLIST= option in the PROC MBANALYSIS statement enables you to specify the regular expression strings to match in the antecedent (left-hand side) of a rule.
- The new CONSEQUENTLIST= option in the PROC MBANALYSIS statement enables you to specify the regular expression strings to match in the consequent (right-hand side) of a rule.
- The new SEPARATOR= option in the PROC MBANALYSIS statement enables you to specify the separator character in the antecedent (left-hand side) or consequent (right-hand side) of a rule.
- The maximum limit of the ITEMS= option is set to 1,000.
- The maximum number of rules that are generated per thread on each node is one million.

The MWPCA Procedure

- The new CUMEIGPCTTOL= option in the PROC MWPCA statement specifies the significance level of eigenvalues that determine the rank of the low-rank matrix.
- The new EIGEN_VALUE column is added to the PROC MWPCA output table.
The RPCA Procedure

- The new CUMEIGPCTTOL= option in the PROC RPCA statement specifies the significance level of eigenvalues that determine the rank of the low-rank matrix.

- The new PROJECTIONTYPE= option, which specifies how scoring observations are projected, is added to the CODE statement.

- The RSTORE= option in the SAVESTATE statement can now be used to project new observations on both the principal component space and the low-rank space.

The SVDD Procedure

- The BW=MEAN2 option in the KERNEL statement enables you to request automatic selection of the Gaussian bandwidth parameter by using the modified mean criterion (Liao et al. 2018).

- You can specify multiple Gaussian bandwidth parameter values for the BW= option in the KERNEL statement.

- The MAXTIME= option in the SOLVER statement enables you to specify an upper time limit for solving the SVDD optimization problem.

References

Chapter 2
Introduction

Contents

Overview of SAS Visual Data Mining and Machine Learning Procedures ............... 7
Experimental Software ...................................................................................... 7
About This Book ............................................................................................... 8
Chapter Organization ....................................................................................... 8
Typographical Conventions ............................................................................. 9
Options Used in Examples ............................................................................... 9
Where to Turn for More Information .............................................................. 10
Online Documentation ..................................................................................... 10
SAS Technical Support Services ...................................................................... 10

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 Guide.

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 any production release. 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** 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 in an ODS destination statement as follows:

```sas
ods html style=HTMLBlue;
... 
ods html close;

each pdf style=PearlJ;
...
each pdf close;
```

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

```sas
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>11</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>11</td>
</tr>
<tr>
<td>Loading a SAS Data Set onto a CAS Server</td>
<td>12</td>
</tr>
<tr>
<td>Syntax Common to SAS Visual Data Mining and Machine Learning Procedures</td>
<td>14</td>
</tr>
<tr>
<td>AUTOTUNE Statement</td>
<td>14</td>
</tr>
<tr>
<td>CODE Statement</td>
<td>19</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>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 14 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:

```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 statement and the LIBNAME statement, see *SAS Cloud Analytic Services: Language Reference*. 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:
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:

    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: Language Reference."  

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:

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

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

    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

\texttt{AUTOTUNE < options > ;}

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

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{EVALHISTORY=}</td>
<td>Specifies how to report the evaluation history of the tuner</td>
</tr>
<tr>
<td>\texttt{FRACTION=}</td>
<td>Specifies the fraction of observations to use for validation</td>
</tr>
<tr>
<td>\texttt{KFOLD=}</td>
<td>Specifies the number of folds for $k$-fold cross validation</td>
</tr>
<tr>
<td>\texttt{MAXBAYES=}</td>
<td>Specifies the maximum number of points in the kriging model</td>
</tr>
<tr>
<td>\texttt{MAXEVALS=}</td>
<td>Specifies the maximum number of evaluations</td>
</tr>
<tr>
<td>\texttt{MAXITER=}</td>
<td>Specifies the maximum number of iterations when \texttt{SEARCHMETHOD=}GA or \texttt{SEARCHMETHOD=}BAYESIAN</td>
</tr>
<tr>
<td>\texttt{MAXTIME=}</td>
<td>Specifies the maximum time for all iterations</td>
</tr>
<tr>
<td>\texttt{MAXTRAINTIME=}</td>
<td>Specifies the maximum time for a model train</td>
</tr>
<tr>
<td>\texttt{NPARALLEL=}</td>
<td>Specifies the number of parallel sessions</td>
</tr>
<tr>
<td>\texttt{NSUBSESSIONWORKERS=}</td>
<td>Specifies the number of workers in parallel sessions</td>
</tr>
<tr>
<td>\texttt{OBJECTIVE=}</td>
<td>Specifies the objective function</td>
</tr>
<tr>
<td>\texttt{POPSIZE=}</td>
<td>Specifies the population size when \texttt{SEARCHMETHOD=}GA or \texttt{SEARCHMETHOD=}BAYESIAN</td>
</tr>
<tr>
<td>\texttt{SAMPLESIZE=}</td>
<td>Specifies the sample size when \texttt{SEARCHMETHOD=}LHS or \texttt{SEARCHMETHOD=}RANDOM</td>
</tr>
<tr>
<td>\texttt{SEARCHMETHOD=}</td>
<td>Specifies the search method that the optimizer uses</td>
</tr>
<tr>
<td>\texttt{TARGETEVENT=}</td>
<td>Specifies the target event for ROC-based calculations</td>
</tr>
<tr>
<td>\texttt{TRAINFRACTION=}</td>
<td>Specifies the fraction of observations to use for training</td>
</tr>
<tr>
<td>\texttt{TUNINGPARAMETERS=}</td>
<td>Specifies the custom tuning parameters</td>
</tr>
<tr>
<td>\texttt{USEPARAMETERS=}</td>
<td>Specifies how to handle the \texttt{TUNINGPARAMETERS=} option</td>
</tr>
</tbody>
</table>
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.

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.

MAXBAYES=number

specifies the maximum number of points in the Kriging model. This option is only honored when SEARCHMETHOD=BAYESIAN, and 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.

**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 `POPSIZE=` 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=\texttt{function}

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

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

- \texttt{ASE} uses average squared error as the objective function.
- \texttt{AUC} uses area under the curve as the objective function (nominal type only).
- \texttt{F05} uses the F0.5 coefficient as the objective function (nominal type only).
- \texttt{F1} uses the F1 coefficient as the objective function (nominal type only).
- \texttt{GAMMA} uses the gamma coefficient as the objective function (nominal type only).
- \texttt{GINI} uses the Gini coefficient as the objective function (nominal type only).
- \texttt{KS} uses the Kolmogorov-Smirnov coefficient as the objective function (nominal type only).
- \texttt{MAE} uses the mean absolute error as the objective function (interval type only).
- \texttt{MCE} uses the misclassification rate as the objective function (nominal type only).
- \texttt{MCLL} uses the multiclass log loss as the objective function (nominal type only).
- \texttt{MISC} uses the misclassification error percentage as the objective function (nominal type only).
- \texttt{MSE} uses the mean squared error as the objective function (interval type only).
- \texttt{MSLE} uses the mean squared logarithmic error as the objective function (interval type only).
- \texttt{RASE} uses the root average squared error as the objective function.
- \texttt{RMAE} uses the root mean absolute error as the objective function (interval type only).
- \texttt{RMSLE} uses the root mean squared logarithmic error as the objective function (interval type only).
- \texttt{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=\texttt{number}

specifies the maximum population size, where population is the number of configuration evaluations in one iteration (population). The \texttt{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 \texttt{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=\texttt{number}

specifies the total number of evaluations, where \texttt{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.

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.

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 >)
TUNEPMRS=(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>
<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=\textit{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.

\textbf{INDENTSIZE=}\textit{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.

\textbf{LABELID=}\textit{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.

\textbf{LINESIZE=}\textit{value}

\textbf{LS=}\textit{value}

specifies the line size for the generated code. You can specify a value in the range 64 to 254; the default is 120.

\textbf{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.

\textbf{OUT=}\textit{CAS-libref.data-table}

creates a CAS table that contains the score code. \textit{CAS-libref.data-table} is a two-level name, where \textit{CAS-libref} refers to the \textit{caslib} and session identifier, and \textit{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 11. You can specify this CAS table in the \texttt{runCodeTable} action in the \texttt{dataStep} action set to score another data table; for more information, see \textit{SAS Viya: System Programming Guide}. This option cannot be specified with the FILE= option.

\textbf{PARTITION Statement}

\texttt{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 \textit{partition-options}:

\texttt{FRACTION(< TEST=}\textit{fraction} > \texttt{< VALIDATE=}\textit{fraction} > \texttt{< SEED=}\textit{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.

\[
\text{ROLE}=\text{variable} (\text{< TEST}='\text{value'} \text{>} \text{< TRAIN}='\text{value'} \text{>} \text{< VALIDATE}='\text{value'} \text{>})
\]

\[
\text{ROLEVAR}=\text{variable} (\text{< TEST}='\text{value'} \text{>} \text{< TRAIN}='\text{value'} \text{>} \text{< VALIDATE}='\text{value'} \text{>})
\]

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:
Chapter 3: Shared Concepts

```sas
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.

```sas
proc forest data=mycas.inData;
    partition roleVar=Group(test='Group 1' train='Group 2')
    ...
run;
```

When you have reserved observations for training, validation, and testing, a model that is fit on the training data is scored on the validation and test data, and statistics are computed separately for each of these subsets.

---

**Multithreading**

This section applies to the following procedures: `ASTORE`, `BOOLRULE`, `FACTMAC`, `FOREST`, `GRADBOOST`, `NNET`, `SVMACHINE`, `TEXTMINE`, and `TMSCORE`.

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
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, NNET, and SVMACHINE.

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 16, “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 $2W/n$).

References

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.
**Chapter 4: The ASTORE Procedure**

**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:

```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
proc options option=(CASHOST CASPORT);
run;
```
cas mysess terminate;

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 11 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&lt;br&gt;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&lt;br&gt;'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.
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 21, “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);
rune;

**Figure 4.1** Scoring with PROC ASTORE

<table>
<thead>
<tr>
<th>Obs</th>
<th>id</th>
<th><em>P</em></th>
<th><em>P_BAD1</em></th>
<th><em>P_BAD0</em></th>
<th><em>I_BAD</em></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>0</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>M</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1.00000</td>
<td>0.00000</td>
<td>1.00000</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax: ASTORE Procedure**

The following statements are available in the ASTORE procedure:

```
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**

```
PROC ASTORE;
```

The PROC ASTORE statement invokes the procedure and does not require any options.
Chapter 4: The ASTORE Procedure

DESCRIBE Statement

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.

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

SCORE STATEMENT

SCORE DATA=CAS-libref.data-table OUT=CAS-libref.data-table RSTORE=CAS-libref.data-table
< . . . > < RSTORE=CAS-libref.data-table > < COPYVARS=(variables) > < EPCODE= == > 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.

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:
**COPYVAR=**variable
**COPYVARS=(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.

**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 4.2  SETOPTION Statement name value Pairs

<table>
<thead>
<tr>
<th>Action</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bnet action</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>dllExportModel action</td>
<td>USEGPU</td>
<td>usegpu</td>
</tr>
<tr>
<td>dnnExportModel action</td>
<td>USEGPU</td>
<td>usegpu</td>
</tr>
<tr>
<td>factmac action</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>filterDesign action</td>
<td>USETRANSFERFUNCTION</td>
<td>func</td>
</tr>
<tr>
<td>forestTrain action</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>gampl action</td>
<td>ALPHA</td>
<td>alpha</td>
</tr>
<tr>
<td>gampl action</td>
<td>COMPUTE_COMPONENTWISE_STATS</td>
<td>compstat</td>
</tr>
<tr>
<td>gampl action</td>
<td>COMPUTE_CONFIDENCE_LIMIT</td>
<td>conflimit</td>
</tr>
<tr>
<td>gampl action</td>
<td>COMPUTE_INTO</td>
<td>into</td>
</tr>
<tr>
<td>gampl action</td>
<td>COMPUTE_LINEAR_PREDICTOR</td>
<td>linpred</td>
</tr>
<tr>
<td>gampl action</td>
<td>COMPUTE_STANDARD_ERROR</td>
<td>stderr</td>
</tr>
<tr>
<td>gampl action</td>
<td>SET_INTO_CUTPOINT</td>
<td>cutpoint</td>
</tr>
<tr>
<td>gbtreeTrain action</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>genmod action</td>
<td>SET_INTO_CUTPOINT</td>
<td>cutpoint</td>
</tr>
<tr>
<td>glm action</td>
<td>ALPHA</td>
<td>alpha</td>
</tr>
<tr>
<td>glm action</td>
<td>COMPUTE_CONFIDENCE_LIMIT</td>
<td>conflimit</td>
</tr>
<tr>
<td>gmm action</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>graphMultiReg action</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>logistic action</td>
<td>SET_INTO_CUTPOINT</td>
<td>cutpoint</td>
</tr>
<tr>
<td>mbAnalysis action</td>
<td>ARM_SCORING_ALGO</td>
<td>algo</td>
</tr>
<tr>
<td>mbAnalysis action</td>
<td>NORM</td>
<td>norm</td>
</tr>
<tr>
<td>robustpca action</td>
<td>RPCA_PROJECTION_TYPE</td>
<td>projtype</td>
</tr>
<tr>
<td>smCalib action</td>
<td>MODELID</td>
<td>modelid</td>
</tr>
<tr>
<td>smCalib action</td>
<td>PROJECTID</td>
<td>projectid</td>
</tr>
<tr>
<td>svddTrain action</td>
<td>BWCOL</td>
<td>bwcol</td>
</tr>
<tr>
<td>svddTrain action</td>
<td>MODELID</td>
<td>modelid</td>
</tr>
<tr>
<td>svmTrain action</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>tmAstore action</td>
<td>None</td>
<td>None</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.
**Chapter 4: The ASTORE Procedure**

**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, it produces additional statistics for each observation, including the confidence intervals for generalized linear models; for generalized additive models, it computes Bayesian confidence bands. 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.

**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.

**MODELID** `modelid`
specifies the ID of the model that is used for scoring, if multiple bandwiths 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 bandwiths that is used to train the models for SVDD. The default value is 0, which scores by using all models for SVDD.

For the `smCalib` action, it 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 scores by using the internal model for the `smCalib` action.

**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 value that is saved from the `smCalib` action. The project IDs are not necessarily contiguous. The default value is 0.

**RPCA_PROJECTION_TYPE** `projtype`
specifies the type of projection, where `projtype` must be either 0 or 1. If set to 1, it projects the scoring observations onto the low-rank subspace; otherwise, it projects the scoring observations onto the principal component space. The default value is 0. If any number other than 0 or 1 is specified, the default value is used.
**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.

**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 SAS Event Stream Processing 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:

```plaintext
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>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>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>
</tbody>
</table>
### Table 4.3 continued

<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>Timing</td>
<td>Timing details</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NOTE: Analytic stores can become very large; breaking down the different stages of scoring a data table in CAS is important and informative.</td>
<td>SCORE</td>
</tr>
</tbody>
</table>

### 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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;
```
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`.

```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 4.1.1 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><em>BAD</em></th>
<th>WARN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>1.00000 6.2878E-8</td>
<td>1.00000</td>
<td>0</td>
<td>M</td>
<td></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.

```r
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.

```r
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:

```r
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:

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

```
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";
   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'FC7E82345DF91C75364F7BF0D177BF93AA9129A26');
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;
# Chapter 5
## The BNET Procedure

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: BNET Procedure</td>
<td>48</td>
</tr>
<tr>
<td>PROC BNET Features</td>
<td>48</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>48</td>
</tr>
<tr>
<td>Getting Started: BNET Procedure</td>
<td>49</td>
</tr>
<tr>
<td>Structure Learning</td>
<td>50</td>
</tr>
<tr>
<td>Probability Table</td>
<td>51</td>
</tr>
<tr>
<td>Variable Selection</td>
<td>51</td>
</tr>
<tr>
<td>Syntax: BNET Procedure</td>
<td>52</td>
</tr>
<tr>
<td>PROC BNET Statement</td>
<td>52</td>
</tr>
<tr>
<td>AUTOTUNE Statement</td>
<td>57</td>
</tr>
<tr>
<td>CODE Statement</td>
<td>63</td>
</tr>
<tr>
<td>FREQ Statement</td>
<td>64</td>
</tr>
<tr>
<td>ID Statement</td>
<td>64</td>
</tr>
<tr>
<td>INPUT Statement</td>
<td>64</td>
</tr>
<tr>
<td>OUTPUT Statement</td>
<td>64</td>
</tr>
<tr>
<td>PARTITION Statement</td>
<td>65</td>
</tr>
<tr>
<td>SAVESTATE Statement</td>
<td>66</td>
</tr>
<tr>
<td>TARGET Statement</td>
<td>66</td>
</tr>
<tr>
<td>Details: BNET Procedure</td>
<td>66</td>
</tr>
<tr>
<td>Independence Tests</td>
<td>66</td>
</tr>
<tr>
<td>Variable Selection</td>
<td>68</td>
</tr>
<tr>
<td>Structure Learning</td>
<td>68</td>
</tr>
<tr>
<td>Parameter Learning</td>
<td>69</td>
</tr>
<tr>
<td>Displayed Output</td>
<td>70</td>
</tr>
<tr>
<td>ODS Table Names</td>
<td>71</td>
</tr>
<tr>
<td>Examples: BNET Procedure</td>
<td>72</td>
</tr>
<tr>
<td>Example 5.1: Naive Bayesian Network</td>
<td>72</td>
</tr>
<tr>
<td>Example 5.2: Tree-Augmented Naive Bayesian Network</td>
<td>72</td>
</tr>
<tr>
<td>Example 5.3: Parent-Child Bayesian Network</td>
<td>73</td>
</tr>
<tr>
<td>Example 5.4: Markov Blanket</td>
<td>74</td>
</tr>
<tr>
<td>Example 5.5: Bayesian Network-Augmented Naive Bayesian Network</td>
<td>74</td>
</tr>
<tr>
<td>Example 5.6: Model Selection</td>
<td>75</td>
</tr>
<tr>
<td>References</td>
<td>79</td>
</tr>
</tbody>
</table>
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 conditional dependency of the random 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, 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 computation 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
- 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:

\[
\text{cas mysess;}
\]
\[
\text{libname mycas cas sessref=mysess;}
\]

The CAS statement creates the CAS session named \text{mysess}, and the LIBNAME statement creates the \text{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 \text{mysess} session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

\[
\text{cas mysess terminate;}
\]

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 11 in Chapter 3, “Shared Concepts.”

---

**Getting Started: BNET 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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 \text{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
```
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 naming your CAS engine libref in the first statement of the following DATA step:

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

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

---

**Structure Learning**

The following statements use the BNET procedure to learn a Bayesian network with Treatment and Sex as nominal variables and Age and Duration as interval variables. The two interval variables are binned into three equal-width levels.

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

The following statements produce Figure 5.1, which shows the network structure that PROC BNET has learned. There are three variables in the network: Treatment is the parent of Pain, and Pain is the parent of Age. From the structure, you can infer that Pain is dependent on Treatment and is also dependent on Age, but is (conditionally) independent of Sex or Duration.

```
proc print data=mycas.network noobs label;
  var _parentnode_ _childnode_
  where _type_="STRUCTURE";
run;
```
Figure 5.1 Network Structure

<table>
<thead>
<tr>
<th>Parent Node</th>
<th>Child Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Treatment</td>
<td>Pain</td>
</tr>
<tr>
<td>Pain</td>
<td>Age</td>
</tr>
</tbody>
</table>

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.

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

Figure 5.2 Probability Table

<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</td>
<td>A</td>
<td>Pain</td>
<td>Yes</td>
<td>0.33333</td>
</tr>
<tr>
<td>Treatment</td>
<td>B</td>
<td>Pain</td>
<td>Yes</td>
<td>0.33333</td>
</tr>
<tr>
<td>Treatment</td>
<td>P</td>
<td>Pain</td>
<td>Yes</td>
<td>0.33333</td>
</tr>
<tr>
<td>Treatment</td>
<td>A</td>
<td>Pain</td>
<td>No</td>
<td>0.72727</td>
</tr>
<tr>
<td>Treatment</td>
<td>A</td>
<td>Pain</td>
<td>Yes</td>
<td>0.27273</td>
</tr>
<tr>
<td>Treatment</td>
<td>B</td>
<td>Pain</td>
<td>No</td>
<td>0.72727</td>
</tr>
<tr>
<td>Treatment</td>
<td>B</td>
<td>Pain</td>
<td>Yes</td>
<td>0.72727</td>
</tr>
<tr>
<td>Treatment</td>
<td>P</td>
<td>Pain</td>
<td>Yes</td>
<td>0.72727</td>
</tr>
<tr>
<td>Treatment</td>
<td>P</td>
<td>Pain</td>
<td>No</td>
<td>0.72727</td>
</tr>
<tr>
<td>Pain</td>
<td>Yes</td>
<td>Age</td>
<td>&lt;67</td>
<td>0.17857</td>
</tr>
<tr>
<td>Pain</td>
<td>Yes</td>
<td>Age</td>
<td>&lt;75</td>
<td>0.35714</td>
</tr>
<tr>
<td>Pain</td>
<td>Yes</td>
<td>Age</td>
<td>&gt;=75</td>
<td>0.46429</td>
</tr>
<tr>
<td>Pain</td>
<td>No</td>
<td>Age</td>
<td>&lt;67</td>
<td>0.31579</td>
</tr>
<tr>
<td>Pain</td>
<td>No</td>
<td>Age</td>
<td>&lt;75</td>
<td>0.60526</td>
</tr>
<tr>
<td>Pain</td>
<td>No</td>
<td>Age</td>
<td>&gt;=75</td>
<td>0.07895</td>
</tr>
</tbody>
</table>

Variable Selection

The network in Figure 5.1 does not include variables Sex or Duration, because PROC BNET automatically selects those variables by using independence tests. PROC BNET produces Figure 5.3, which shows the variable selection results. 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 for the ALPHA= option, which is not specified). Sex is conditionally independent of Pain given Treatment; therefore, PROC BNET also removes it from the network.
Figure 5.3  Variable Selection

The BNET Procedure

<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>

Syntax: BNET Procedure

The following statements are available in the BNET procedure:

```plaintext
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

```plaintext
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>DATA=</td>
<td>Specifies the input data set</td>
</tr>
<tr>
<td>NUMBIN=</td>
<td>Specifies the binning number 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>
</tbody>
</table>
Table 5.1  continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Independence Test Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEPTEST=</td>
<td>Specifies the methods for independence tests</td>
</tr>
<tr>
<td>ALPHA=</td>
<td>Specifies the significance level for independence tests by using chi-square or G-square statistics</td>
</tr>
<tr>
<td>MIALPHA=</td>
<td>Specifies the significant level for independence tests by using mutual information</td>
</tr>
</tbody>
</table>

**Structure Learning Options**

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

**Model Selection Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BESTMODEL</td>
<td>Requests that the best model be selected</td>
</tr>
</tbody>
</table>

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**

requests that the best model be selected 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 on the validation data to automatically decide the best set of parameter values among these options.

By default, a best model is not selected.

**NOTE:** If you specify BESTMODEL in PROC statement, then AUTOTUNE statement will be 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 48.

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

INDEPTEST=ALL | CHIGSQUARE | CHISQUARE | GSQUARE | MI
specifies the method 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=CHIGSQUARE.

MAXPARENTS=integer
specifies the maximum number of parents that is 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, testing data and any data 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

specifies the number of binning levels for all interval variables. PROC BNET bins each interval variable into integer equal-width levels. The valid range of integer is 2 to 1024, inclusive.

By default, NUMBIN=5.

OUTNETWORK=CAS-libref.data-table
OUTNET=CAS-libref.data-table
names the CAS data table to contain the network structure and the probability distributions. 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 48.

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 PRESCREENING=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.

By default, PRESCREENING=1.

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.

STRUCTURE=MB | NAIVE | PC | TAN

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 the following values:

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 VARSELECT= 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 MAXPARENTS 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 PC.

TAN  learns the tree-augmented naive Bayesian network structure. The TAN structure includes a direct link from the target to each input variable plus a tree structure among the input variables.

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:
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 conditionally dependent on 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 conditionally dependent on 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.

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 rolls back 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= but you do not specify the BESTMODEL option, PROC BNET uses the first specified value, and ignores the remaining values.

By default, VARSELECT=1.

**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 14 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>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>KFOLD=</td>
<td>Specifies the number of folds for $k$-fold cross validation</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 train</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>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.

**INDEPTEST (VALUES=**`value-list` **INIT=**`value` **EXCLUDE)**
specifies information about the method to use for independence tests. For more information, see the INDEPTEST= 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 INDEPTEST 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.
Chapter 5: The BNET Procedure

**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.
Chapter 5: The BNET Procedure

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=\textit{value-list} INIT=\textit{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=\textit{value-list}**
  
  specifies a list of structures, where \textit{value-list} is a space-separated list of MB, NAIVE, PC, and TAN.

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

- **INIT=\textit{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**

**CODE FILE=\textit{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. PROC BNET does not support duplicate input variables. If the INPUT statement contains a duplicate variable, PROC BNET returns an error message and then exits. 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 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 48.

*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**

\[
\text{PARTITION \ partition-option ;}
\]

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\(<\text{TEST}=\text{fraction}> <\text{VALIDATE}=\text{fraction}> <\text{SEED}=\text{number}>\)**

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 48.

**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.

**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).

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

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

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, $X \leftarrow Y \leftarrow Z$, $X \rightarrow Y \rightarrow Z$, 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), and Markov blanket (MB). 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

$$BIC(X, T) = \max(BIC(X \rightarrow T), BIC(T \rightarrow X))$$

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

### 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:

$$\hat{p}(x_i | \pi(X_i)) = \frac{\text{counts}_{\text{of}}(x_i, p_i(X_i)) + \alpha}{\text{counts}_{\text{of}}(p_i(X_i)) + \alpha \times n_i}$$

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 number of observations and the number of observations used.

**Variable Level**

The “Variable Level” table contains the details of each level of the variables. The columns include the observed target, predicted event, predicted nonevent, and total numbers of events or nonevents for the training data.

**Variable Order**

The “Variable Order” table contains the order of the input variables based on the BIC score with the target.

**Variable Information**

The “Variable Information” table contains the variable information such as number of levels, number of missing values, and so on.
Iteration Report

The “Iteration Report” table contains the variable selection results.

Validation Information

The “Validation Information” table contains the validation results. If the PARTITION statement is specified, then the misclassification errors mean misclassification errors in the validation data; if not, then the misclassification errors mean misclassification errors in the training data.

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</td>
<td>PROC BNET</td>
<td>Default</td>
</tr>
</tbody>
</table>
Chapter 5: The BNET Procedure

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.

```
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:

```
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.

```
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:

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

```sas
proc bnet data=mycas.iris numbin=3 structure=PC
    prescreening=0 vareselect=0
    outnetwork=mycas.network;
    target Species;
    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.

```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>SepalWidth</td>
<td>SepalLength</td>
</tr>
<tr>
<td>Species</td>
<td>SepalWidth</td>
</tr>
<tr>
<td>PetalWidth</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=TAN` results in a parent-child Bayesian network:

```sas
proc bnet data=mycas.iris numbin=3 structure=TAN
    prescreening=0 vareselect=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;
```
Chapter 5: The BNET Procedure

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

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:

```sas
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.

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

Output 5.4.1 Markov Blanket 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>NINQ</td>
<td>DELINQ</td>
</tr>
<tr>
<td>BAD</td>
<td>DEROG</td>
</tr>
<tr>
<td>DELINQ</td>
<td>DEROG</td>
</tr>
<tr>
<td>NINQ</td>
<td>DEROG</td>
</tr>
</tbody>
</table>

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:
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.

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: 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.

data dmagecr_part;
  set sampsio.dmagecr;
  seed=12345;
  if ranuni(seed) < 0.7 then partind=1;
  else partind=0;
  id=_N_;
Chapter 5: The BNET Procedure

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 Exisctcr Job Depends Telephon
       Foreign/level=nominal;
   input Age Amount Duration/level=interval;
   partition rolevar= PartInd (TRAIN='1' VALIDATE='0' TEST='2');
   ods output nobsp=nobs fitstatistics=fit validinfo=validinfo;
run;

Output 5.6.1 shows the information of number of observations: 695 observations are used for training and 305 are used for validation.

Output 5.6.1  Model Selection: Number of Observations

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

Output 5.6.2 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 5.6.2  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 5.6.3 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 MAXPARENTS is greater than or equal to 2. The TAN structure has 77 misclassification errors when VARSELECT=0 and MAXPARENTS=2. The naive Bayesian network has 81 misclassification errors when VARSELECT=0 and MAXPARENTS is greater than or equal to 2. The MB Bayesian network structure has 113 misclassification errors.

<table>
<thead>
<tr>
<th>Best Model</th>
<th>Misclassification Rate</th>
<th>Misclassification Errors</th>
<th>N Observations for Assessment</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</td>
<td>PC</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.23934</td>
<td>73</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>0</td>
<td>PC</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.23934</td>
<td>73</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>0</td>
<td>PC</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.23934</td>
<td>73</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>0</td>
<td>PC</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.25246</td>
<td>77</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>0</td>
<td>TAN</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.26557</td>
<td>81</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>0</td>
<td>Naive</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.26557</td>
<td>81</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>0</td>
<td>Naive</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.26557</td>
<td>81</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>0</td>
<td>Naive</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.26557</td>
<td>81</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>0</td>
<td>Naive</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.26885</td>
<td>82</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>1</td>
<td>PC</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.26885</td>
<td>82</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>1</td>
<td>PC</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.26885</td>
<td>82</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>1</td>
<td>PC</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.26885</td>
<td>82</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>1</td>
<td>PC</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.28197</td>
<td>86</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>0</td>
<td>PC</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.28197</td>
<td>86</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>0</td>
<td>Naive</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.34754</td>
<td>106</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>1</td>
<td>PC</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.34754</td>
<td>106</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>1</td>
<td>Naive</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.34754</td>
<td>106</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>1</td>
<td>Naive</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.34754</td>
<td>106</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>1</td>
<td>Naive</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.34754</td>
<td>106</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>1</td>
<td>Naive</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.35738</td>
<td>109</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>1</td>
<td>TAN</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.37049</td>
<td>113</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>3</td>
<td>MB</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.37049</td>
<td>113</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>3</td>
<td>MB</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.37049</td>
<td>113</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>3</td>
<td>MB</td>
<td>BestSet</td>
</tr>
<tr>
<td></td>
<td>0.37049</td>
<td>113</td>
<td>305</td>
<td>0.05</td>
<td>1</td>
<td>3</td>
<td>MB</td>
<td>BestSet</td>
</tr>
</tbody>
</table>
Chapter 5: The BNET Procedure

Output 5.6.3 continued

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

The following statements produce Output 5.6.4, 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.

```plaintext
proc print data=mycas.network noobs label;
  var _parentnode_ _childnode_;  
  where _type_="STRUCTURE";
run;
```
Output 5.6.4 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**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: BOOLRULE Procedure</td>
<td>82</td>
</tr>
<tr>
<td>PROC BOOLRULE Features</td>
<td>82</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>83</td>
</tr>
<tr>
<td>Getting Started: BOOLRULE Procedure</td>
<td>83</td>
</tr>
<tr>
<td>Syntax: BOOLRULE Procedure</td>
<td>87</td>
</tr>
<tr>
<td>PROC BOOLRULE Statement</td>
<td>87</td>
</tr>
<tr>
<td>DOCINFO Statement</td>
<td>90</td>
</tr>
<tr>
<td>OUTPUT Statement</td>
<td>91</td>
</tr>
<tr>
<td>SCORE Statement</td>
<td>92</td>
</tr>
<tr>
<td>TERMIINFO Statement</td>
<td>92</td>
</tr>
<tr>
<td>Details: BOOLRULE Procedure</td>
<td>93</td>
</tr>
<tr>
<td>BOOLLEAR for Boolean Rule Extraction</td>
<td>93</td>
</tr>
<tr>
<td>Term Ensemble Process</td>
<td>94</td>
</tr>
<tr>
<td>Rule Ensemble Process</td>
<td>95</td>
</tr>
<tr>
<td>Measurements Used in BOOLLEAR</td>
<td>96</td>
</tr>
<tr>
<td>Precision, Recall, and the F1 Score</td>
<td>96</td>
</tr>
<tr>
<td>$g$-Score</td>
<td>97</td>
</tr>
<tr>
<td>Estimated Precision</td>
<td>97</td>
</tr>
<tr>
<td>Improvability Test</td>
<td>98</td>
</tr>
<tr>
<td>Shrinking the Search Space</td>
<td>98</td>
</tr>
<tr>
<td>Feature Selection</td>
<td>98</td>
</tr>
<tr>
<td>Significance Testing</td>
<td>98</td>
</tr>
<tr>
<td>$k$-Best Search</td>
<td>99</td>
</tr>
<tr>
<td>Improvability Test</td>
<td>99</td>
</tr>
<tr>
<td>Early Stop Based on the F1 Score</td>
<td>99</td>
</tr>
<tr>
<td>Output Data Sets</td>
<td>99</td>
</tr>
<tr>
<td>CANDIDATETERMS= Data Table</td>
<td>99</td>
</tr>
<tr>
<td>RULES= Data Table</td>
<td>100</td>
</tr>
<tr>
<td>RULETERMS= Data Table</td>
<td>101</td>
</tr>
<tr>
<td>Scoring Data Set</td>
<td>101</td>
</tr>
<tr>
<td>OUTMATCH= Data Table</td>
<td>101</td>
</tr>
<tr>
<td>Examples: BOOLRULE Procedure</td>
<td>102</td>
</tr>
<tr>
<td>Example 6.1: Rule Extraction for Binary Targets</td>
<td>102</td>
</tr>
<tr>
<td>Example 6.2: Rule Extraction for a Multiclass Target</td>
<td>104</td>
</tr>
<tr>
<td>Example 6.3: Using Events in Rule Extraction</td>
<td>106</td>
</tr>
<tr>
<td>Example 6.4: Scoring</td>
<td>107</td>
</tr>
<tr>
<td>References</td>
<td>110</td>
</tr>
</tbody>
</table>
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:

\[
\text{(cut} \land \text{rate} \land \text{bank} \land \text{percent} \land \sim \text{sell}) \lor \\
\text{(market} \land \text{money} \land \sim \text{year} \land \text{percent} \land \sim \text{sale}) \lor \\
\text{(repurchase} \land \text{fee}) \lor \\
\text{(rate} \land \text{prime rate}) \lor \\
\text{(federal} \land \text{rate} \land \text{maturity})
\]

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:

```
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 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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.
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.

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:

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

```sas
proc 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 100.

### Figure 6.2 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>apple_fruit</td>
<td>1</td>
<td>be &amp; apple</td>
<td>0.93333</td>
<td>1</td>
<td>0.875</td>
</tr>
<tr>
<td>2</td>
<td>apple_fruit</td>
<td>2</td>
<td>produce</td>
<td>1.00000</td>
<td>1</td>
<td>1.000</td>
</tr>
</tbody>
</table>

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

```sas
proc boolrule
data = mycas.bow;
docid = _document_
  termin = _terminum_;
score
  ruleterms = mycas.ruleterms
  outmatch = mycas.matches;
run;
proc 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 101.
Syntax: BOOLRULE Procedure

The following statements are available in the BOOLRULE procedure:

```
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.

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 6.1 PROC BOOLRULE Statement Options

<table>
<thead>
<tr>
<th>option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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</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 83.

- **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 83.

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 97.

**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`

 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`.

**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 99. By default, `MAXTRIESIN=150`.

**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 99. 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 TERMMINFO= option is not specified, variable is also used as the label of terms.

TERMMINFO=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 83.

Each row of the input data table must contain one variable for the term ID. If you specify this option, you must use the TERMMINFO 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 83.

   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 99.

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 83.

   For more information about this data table, see the section “Output Data Sets” on page 99.
Chapter 6: The BOOLRULE Procedure

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

specifies a data table to contain the terms in each rule that is generated by the BOOLRULE procedure. \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 83.

For more information about this data table, see the section “Output Data Sets” on page 99.

SCORE Statement

\texttt{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 \texttt{options}:

\textbf{OUTMATCH=\texttt{CAS-libref.data-table}}

specifies a data table to contain the rule-matching results (that is, whether a document satisfies a rule). \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 83.

For more information about this data table, see the section “Scoring Data Set” on page 101.

\textbf{RULETERMS=\texttt{CAS-libref.data-table}}

specifies a data table that contains the terms in each rule that the BOOLRULE procedure generates. \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 83.

For more information about this data table, see the section “RULETERMS= Data Table” on page 101.

TERMININFO Statement

\texttt{TERMININFO < options> ;}

The TERMININFO statement specifies information about the data table that is specified in the \texttt{TERMININFO=} option in the \texttt{PROC BOOLRULE} statement. If you specify the \texttt{TERMININFO=} data table in the \texttt{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 \texttt{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 TERMID= 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 TERMID= 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.

**Figure 6.4** Term Ensemble Process for Creating a Rule

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](image)

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 $k_{out} - 1$ additional rules, where
Chapter 6: The BOOLRULE Procedure

$k_{\text{out}}$ 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 96.

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.

\[\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}\]
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

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

where \( P \) is the number of positive documents; \( N \) is the number of negative documents; \( O(\text{TP}), O(\text{FP}), O(\text{TN}), \) and \( O(\text{FN}) \) refer to the observed TP, FP, TN, and FN of a term; and \( E(\text{TP}), E(\text{FP}), E(\text{TN}), \) and \( E(\text{FN}) \) 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 (\( \text{err}_i \)) to favor shorter rules over longer rules:

\[
\text{precision}_i^m(t) = \frac{\text{TP}_{i,t}}{\text{TP}_{i,t} + \frac{P}{N+P} \times m} - \text{err}_{i-1}
\]

\[
\text{err}_i = \frac{\text{TP}_{i,t}}{\text{TP}_{i,t} + \text{FP}_{i,t}} - \frac{\text{TP}_{i,t} + \frac{P}{N+P} \times m}{\text{TP}_{i,t} + \text{FP}_{i,t} + m} + \text{err}_{i-1}
\]

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. \( \text{TP}_{i,t} \) and \( \text{FP}_{i,t} \) 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, BOOLCLEAR applies the g-test to a perfect confusion table that is defined as:

<table>
<thead>
<tr>
<th>TP</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>FP</td>
</tr>
</tbody>
</table>

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, BOOLCLEAR 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, BOOLCLEAR 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

BOOLCLEAR 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, BOOLCLEAR 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 BOOLCLEAR truncate its search path and avoid generating overly specific rules.

- **Minimum positive document coverage**: BOOLCLEAR 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**: BOOLCLEAR 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.
Chapter 6: The BOOLRULE Procedure

Table 6.2  Fields in the CANDIDATETERMS= 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</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>starts from 1)</td>
</tr>
<tr>
<td>Term</td>
<td>A lowercase version of the term</td>
</tr>
<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</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>added to the rule set</td>
</tr>
<tr>
<td>rTP</td>
<td>The number of positive documents that are satisfied by the rule set when</td>
</tr>
<tr>
<td></td>
<td>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 set when</td>
</tr>
<tr>
<td></td>
<td>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 set when the rule is</td>
</tr>
<tr>
<td></td>
<td>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>
<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.

```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:
Example 6.1: Rule Extraction for Binary Targets

```
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. By default, TARGETTYPE=BINARY. 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 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.
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 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
  I 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>1</td>
<td>phone</td>
<td>0.80000</td>
<td>1.00</td>
<td>0.66667</td>
</tr>
<tr>
<td>2</td>
<td>electronics</td>
<td>2</td>
<td>resolution</td>
<td>0.85714</td>
<td>0.75</td>
<td>1.00000</td>
</tr>
<tr>
<td>3</td>
<td>movies</td>
<td>3</td>
<td>movie</td>
<td>0.85714</td>
<td>0.75</td>
<td>1.00000</td>
</tr>
<tr>
<td>4</td>
<td>books</td>
<td>4</td>
<td>book</td>
<td>1.00000</td>
<td>1.00</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 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.

```sas
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
  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:

```sas
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.

```latex
\begin{verbatim}
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;
\end{verbatim}

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.

```latex
\begin{verbatim}
data mycas.reviews_test;
  infile datalines delimiter='|' missover;
  length text $300;
  input text$ did;
  datalines;
  love it! a great phone, even better than advertised|1
  I like the book, GREATEST in this genre|2
;
run;
\end{verbatim}

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:

```latex
\begin{verbatim}
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;
\end{verbatim}
```
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
   mpos = 1
gpos = 1;
docinfo
   id = did
targettype = binary
targets = (positive);
terminfo
   id = key
   label = term;
output
   ruleterms = mycas.ruleterms
   rules = mycas.rules;
run;
```

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>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**References**

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 in an output data table, which is produced by the OUTMODEL statement. 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:

```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 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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
   nfactors=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.
114 F Chapter 7: The FACTMAC Procedure

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>NFACTORS=</td>
<td>Specifies the number of factors to estimate for the model</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations</td>
</tr>
<tr>
<td>SEED=</td>
<td>Specifies the seed to be used for pseudorandom number generation</td>
</tr>
<tr>
<td>LEARNSTEP=</td>
<td>Specifies the learning step size for the SGD algorithm</td>
</tr>
<tr>
<td>NTHREADS=</td>
<td>Specifies the number of threads to use on each computation node</td>
</tr>
<tr>
<td>NONNEGATIVE</td>
<td>Requests nonnegative factorization</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 112.

\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 \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 112.
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 14 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>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>KFOLD=</td>
<td>Specifies the number of folds for k-fold cross validation</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 train</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>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:

```sas
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 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 125. 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 125. 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 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 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=CAS-libref.data-table**

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 112.
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=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 112.

TARGET Statement

**TARGET variable < LEVEL=INTERVAL > ;**

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

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2}$$

over the training set, subject to the max-norm regularization constraint

$$\|v_j\|_\infty < B, j = 1, \ldots, p$$

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 125</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 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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. This example uses the MovieLens 100K version.

There are four columns in the MovieLens 100K data set: user ID, item ID (each item is a movie), timestamp, and rating. This example predicts the rating for a specified user ID and an item ID. The data set is very sparse because most combinations of users and movies are not rated.

You can download the compressed archive file from the website at http://files.grouplens.org/datasets/movielens/ml-100k.zip and use any third-party unzip tool to extract all the files in the archive to the destination directory of your choice. The file that contains the ratings is u.data. Assuming the destination directory is /data, the following DATA step loads the data table from the directory into your CAS session:

---

1Disclaimer: 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

proc casutil;
  load file="~/data/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:

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;

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.

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>0.00000</td>
<td>0.00000</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>-1.06146</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.

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>3</td>
<td>4.09834</td>
</tr>
<tr>
<td>2</td>
<td>186</td>
<td>302</td>
<td>3</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. 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:

```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 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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;

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
  thresdist = 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:
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>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>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 130.

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

K=number
specifies the number of neighbors to return, where number is a positive real number.
By default, K=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 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. 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 130.

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.

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 130.

THRESHDIST=number
specifies the distance threshold for the model, where number is an integer greater than 0. PROC FASTKNN excludes from the computation distances greater than number.
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
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 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

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

names the output data table for PROC FASTKNN to use. You must specify this option before any other options. \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 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 \texttt{CAS-libref}, see the section “Using CAS Sessions and CAS Engine Librefs” on page 130.

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

You can also specify the following option:

\texttt{COPYVAR=variable}

\texttt{COPYVARS=(variables)}

lists one or more \texttt{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.

---

**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, \(k\), and the value of the distance threshold specified in the THRESHDIST= option in the PROC FASTKNN statement.
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, 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 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 &quot;OutputCasTables Table&quot; on page 137</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>
<thead>
<tr>
<th>Data Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN_OUT</td>
<td>Lists the nearest neighbors</td>
</tr>
</tbody>
</table>
### Table 8.3  continued

<table>
<thead>
<tr>
<th>Data table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIST_OUT</td>
<td>Lists the distances from query 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 143.

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
Chapter 9: The FISM Procedure

- 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 11 in Chapter 3, “Shared Concepts.”

Getting Started: 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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:

```plaintext
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 140, 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:

```plaintext
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:

```plaintext
PROC FISM options ;
  CUSTOMER variables ;
  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**

```plaintext
PROC FISM options ;
```

The PROC FISM statement invokes the procedure.

You must specify the following `option`: 
DATA=\texttt{CAS-libref.data-table}

names the input data table for PROC FISM to use. \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 140. \texttt{data-table} specifies the name of the input data table.

You must specify either of the following options:

- \texttt{SUPPORT=} \texttt{number}
  
  specifies the minimum level of support (minimum frequency of an item) for a rule, where \texttt{number} must be an integer greater than or equal to 0. This option overrides the specification of the PCTSUP= option.

- \texttt{PCTSUP=} \texttt{number}

- \texttt{SUPPORT=} \texttt{number}

- \texttt{SUPPCT=} \texttt{number}

- \texttt{PCTSUPPORT=} \texttt{number}

  specifies the minimum level of support for a rule as a percentage of the number of baskets in the input data table, where \texttt{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:

- \texttt{ITEMS=} \texttt{number}

  specifies the number of items in a frequent item set, where \texttt{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.

- \texttt{NFIS_RANGE=} \texttt{(number, number)}

  specifies the range of number of items in the frequent item sets, where \texttt{number} must be an integer between 1 and 1,000.

  By default, NFIS_RANGE=(1,1000).

\textbf{CUSTOMER Statement}

\texttt{CUSTOMER variable ;}

The CUSTOMER statement groups one or more specified target variables into a basket. You can also specify this statement as ID variables.
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 140.

**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 140.

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 \( \text{mine(CFPT | node)} \), 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>OUTFREQ=</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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 in Chapter 3, “Shared Concepts.”
This example identifies frequent item sets in the *sampsio.assoc* 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.assoc* data set into your CAS session by specifying your CAS engine libref in the first statement in the following DATA step:

```plaintext
data mycas.assoc;
  set sampsio.assoc;
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.assoc data table:

```plaintext
proc fism data=mycas.assoc 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

Overview: FOREST Procedure .......................................................... 148
PROC FOREST Features ................................................................. 149
Using CAS Sessions and CAS Engine Librefs ................................. 149
Getting Started: FOREST Procedure ............................................. 149
Syntax: FOREST Procedure .......................................................... 154
  PROC FOREST Statement ......................................................... 155
  AUTOTUNE Statement ............................................................... 160
  CODE Statement .................................................................. 163
  CROSSVALIDATION Statement .............................................. 164
  GROW Statement .................................................................. 164
  ID Statement ..................................................................... 165
  INPUT Statement .................................................................. 165
  OUTPUT Statement ................................................................ 166
  PARTITION Statement ............................................................. 167
  SAVESTATE Statement ............................................................ 167
  TARGET Statement ................................................................ 168
  VIICODE Statement ................................................................. 168
  WEIGHT Statement ................................................................ 169
Details: FOREST Procedure .......................................................... 169
  Bagging the Data .................................................................. 169
  Training a Decision Tree ......................................................... 169
  Loh Method ....................................................................... 170
  Predicting an Observation ....................................................... 171
  Measuring Prediction Error ..................................................... 171
  Handling Missing Values ........................................................ 172
    Strategies ....................................................................... 172
    Specifics ......................................................................... 172
  Handling Values That Are Absent from Training Data ............... 173
  Measuring Variable Importance ............................................. 173
    Residual Sum of Squares Importance Method .................. 173
    Random Branch Assignment Importance Method ........... 176
  Isolation Forests ................................................................. 176
  Hyperparameter Tuning ........................................................... 176
  k-fold Cross Validation .......................................................... 177
    Determining the Number of Parallel Evaluations ............. 177
  Displayed Output .................................................................. 178
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 11 in Chapter 3, “Shared Concepts.”

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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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>
<tbody>
<tr>
<td>Bad</td>
<td>Response</td>
<td>Binary</td>
<td>$1 = \text{applicant defaulted on the loan or is seriously delinquent}$ $0 = \text{applicant paid off the loan}$</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 $'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.

```plaintext
/* 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;

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>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Trees</td>
</tr>
<tr>
<td>Number of Variables Per Split</td>
</tr>
<tr>
<td>Seed</td>
</tr>
<tr>
<td>Bootstrap Percentage</td>
</tr>
<tr>
<td>Number of Bins</td>
</tr>
<tr>
<td>Number of Input Variables</td>
</tr>
<tr>
<td>Maximum Number of Tree Nodes</td>
</tr>
<tr>
<td>Minimum Number of Tree Nodes</td>
</tr>
<tr>
<td>Maximum Number of Branches</td>
</tr>
<tr>
<td>Minimum Number of Branches</td>
</tr>
<tr>
<td>Maximum Depth</td>
</tr>
<tr>
<td>Minimum Depth</td>
</tr>
<tr>
<td>Maximum Number of Leaves</td>
</tr>
<tr>
<td>Minimum Number of Leaves</td>
</tr>
<tr>
<td>Maximum Leaf Size</td>
</tr>
<tr>
<td>Minimum Leaf Size</td>
</tr>
<tr>
<td>OOB Misclassification Rate</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.

<table>
<thead>
<tr>
<th>Number of Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
</tr>
<tr>
<td>Number of Observations Read</td>
</tr>
<tr>
<td>Number of Observations Used</td>
</tr>
</tbody>
</table>

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>324.39</td>
<td>105.04</td>
<td>1.0000</td>
</tr>
<tr>
<td>Delinq</td>
<td>78.3280</td>
<td>11.2333</td>
<td>0.2415</td>
</tr>
<tr>
<td>Value</td>
<td>70.9231</td>
<td>21.9801</td>
<td>0.2186</td>
</tr>
<tr>
<td>Derog</td>
<td>43.5352</td>
<td>5.7456</td>
<td>0.1342</td>
</tr>
<tr>
<td>nlnq</td>
<td>37.6304</td>
<td>2.9902</td>
<td>0.1160</td>
</tr>
<tr>
<td>CLNo</td>
<td>33.6453</td>
<td>3.5898</td>
<td>0.1037</td>
</tr>
<tr>
<td>CLAge</td>
<td>33.2247</td>
<td>5.2678</td>
<td>0.1024</td>
</tr>
<tr>
<td>Job</td>
<td>25.9289</td>
<td>2.4354</td>
<td>0.0799</td>
</tr>
<tr>
<td>Loan</td>
<td>25.1847</td>
<td>5.4856</td>
<td>0.0776</td>
</tr>
<tr>
<td>MortDue</td>
<td>21.4545</td>
<td>2.9830</td>
<td>0.0661</td>
</tr>
<tr>
<td>YoJ</td>
<td>19.9675</td>
<td>2.2958</td>
<td>0.0616</td>
</tr>
<tr>
<td>Reason</td>
<td>5.0139</td>
<td>1.7765</td>
<td>0.0155</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 169. 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.
The following statements are available in the FOREST procedure:

```plaintext
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.

### Figure 10.5 Fit Statistics

<table>
<thead>
<tr>
<th>Number of Trees</th>
<th>OOB Square Error</th>
<th>Training Average 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>0.1064</td>
<td>0.0831</td>
<td>0.139</td>
<td>0.1109</td>
<td>1.082</td>
<td>0.674</td>
</tr>
<tr>
<td>2</td>
<td>0.0995</td>
<td>0.0716</td>
<td>0.131</td>
<td>0.0980</td>
<td>0.809</td>
<td>0.313</td>
</tr>
<tr>
<td>3</td>
<td>0.0955</td>
<td>0.0682</td>
<td>0.125</td>
<td>0.0926</td>
<td>0.617</td>
<td>0.250</td>
</tr>
<tr>
<td>4</td>
<td>0.0894</td>
<td>0.0644</td>
<td>0.116</td>
<td>0.0841</td>
<td>0.539</td>
<td>0.225</td>
</tr>
<tr>
<td>5</td>
<td>0.0879</td>
<td>0.0639</td>
<td>0.115</td>
<td>0.0864</td>
<td>0.465</td>
<td>0.221</td>
</tr>
<tr>
<td>6</td>
<td>0.0856</td>
<td>0.0627</td>
<td>0.116</td>
<td>0.0841</td>
<td>0.421</td>
<td>0.215</td>
</tr>
<tr>
<td>7</td>
<td>0.0841</td>
<td>0.0629</td>
<td>0.115</td>
<td>0.0859</td>
<td>0.360</td>
<td>0.217</td>
</tr>
<tr>
<td>8</td>
<td>0.0825</td>
<td>0.0621</td>
<td>0.111</td>
<td>0.0861</td>
<td>0.329</td>
<td>0.215</td>
</tr>
<tr>
<td>9</td>
<td>0.0815</td>
<td>0.0613</td>
<td>0.109</td>
<td>0.0827</td>
<td>0.318</td>
<td>0.212</td>
</tr>
<tr>
<td>10</td>
<td>0.0810</td>
<td>0.0611</td>
<td>0.107</td>
<td>0.0841</td>
<td>0.308</td>
<td>0.211</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>0.0752</td>
<td>0.0607</td>
<td>0.101</td>
<td>0.0794</td>
<td>0.254</td>
<td>0.212</td>
</tr>
<tr>
<td>92</td>
<td>0.0752</td>
<td>0.0607</td>
<td>0.101</td>
<td>0.0797</td>
<td>0.254</td>
<td>0.212</td>
</tr>
<tr>
<td>93</td>
<td>0.0752</td>
<td>0.0607</td>
<td>0.101</td>
<td>0.0797</td>
<td>0.254</td>
<td>0.212</td>
</tr>
<tr>
<td>94</td>
<td>0.0752</td>
<td>0.0607</td>
<td>0.101</td>
<td>0.0797</td>
<td>0.254</td>
<td>0.212</td>
</tr>
<tr>
<td>95</td>
<td>0.0752</td>
<td>0.0607</td>
<td>0.102</td>
<td>0.0794</td>
<td>0.254</td>
<td>0.212</td>
</tr>
<tr>
<td>96</td>
<td>0.0752</td>
<td>0.0607</td>
<td>0.101</td>
<td>0.0794</td>
<td>0.254</td>
<td>0.212</td>
</tr>
<tr>
<td>97</td>
<td>0.0751</td>
<td>0.0607</td>
<td>0.101</td>
<td>0.0789</td>
<td>0.254</td>
<td>0.212</td>
</tr>
<tr>
<td>98</td>
<td>0.0752</td>
<td>0.0608</td>
<td>0.102</td>
<td>0.0795</td>
<td>0.254</td>
<td>0.212</td>
</tr>
<tr>
<td>99</td>
<td>0.0752</td>
<td>0.0607</td>
<td>0.101</td>
<td>0.0795</td>
<td>0.254</td>
<td>0.212</td>
</tr>
<tr>
<td>100</td>
<td>0.0751</td>
<td>0.0607</td>
<td>0.101</td>
<td>0.0785</td>
<td>0.254</td>
<td>0.212</td>
</tr>
</tbody>
</table>
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>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>
<tr>
<td><strong>Splitting Options</strong></td>
<td></td>
</tr>
<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 missing interval inputs to the leftmost branch of the split, and unknown and missing nominal levels to the largest branch in the split.
- **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=BUCKET.

**DATA=CAS-libref.data-table** names 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 149.

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

INBAGFRACTION=number specifies the fraction of the random bootstrap sample of the training data to be used for growing each tree in the forest, where number is a value between 0 and 1.

By default, INBAGFRACTION=0.6. This value can be tuned with the AUTOTUNE statement.

INMODEL=<<CAS-libref.>>data-table specifies the data table that you have previously saved as a forest model by using the OUTMODEL= option in a previous run of PROC FOREST. 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 149.

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 OUTMODEL= option, or the FOREST procedure will error.

ISOLATION < (SAMPLEN=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 176.

You can specify the following option:

SAMPLEN=number specifies the 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 SAVESTATE statements are ignored and the BINMETHOD, INBAGFRACTION, RBAIMP, VARS_TO_TRY, and VII options in the PROC FOREST statement are ignored.

LOH=L specifies a number of variables (L) that are preselected to consider for candidate splits for each node. The variables are selected using the Loh method.

If L is less than the value of the VARS_TO_TRY= option (m), then the variables are selected from among the m variables. If L is greater than or equal to m, or if no 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.

By default, MINLEAFSIZE=5.

**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.

**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=20.

**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 149.

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 173.

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 175. 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=, 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 14 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>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>KFOLD=</td>
<td>Specifies the number of folds for k-fold cross validation</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 train</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>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:
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.

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.

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.

**INIT=** *number*

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

By default, **INIT** is the square root of the number of the input variables, rounded up.

**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**

```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><strong>COMMENT</strong></td>
<td>Adds comments to the generated code</td>
</tr>
<tr>
<td><strong>FILE=</strong></td>
<td>Names the file in which to save the generated code</td>
</tr>
<tr>
<td><strong>FORMATWIDTH=</strong></td>
<td>Specifies the numeric format width for the regression coefficients</td>
</tr>
<tr>
<td><strong>INDENTSIZE=</strong></td>
<td>Specifies the number of spaces to indent the generated code</td>
</tr>
<tr>
<td><strong>LABELID=</strong></td>
<td>Specifies a number used to construct names and labels</td>
</tr>
<tr>
<td><strong>LINESIZE=</strong></td>
<td>Specifies the line size for the generated code</td>
</tr>
<tr>
<td><strong>NOTRIM</strong></td>
<td>Compares formatted values, including blank padding</td>
</tr>
<tr>
<td><strong>OUT=</strong></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 19 in Chapter 3, “Shared Concepts.”

**CROSSVALIDATION Statement**

```
CROSSVALIDATION < options > ;
```

The CROSSVALIDATION statement performs \( k \)-fold cross validation to find the average estimated validation error. You cannot specify the 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.

**GROW Statement**

```
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 149.
- `data-table` specifies the name of the output data table.

You can also specify the following options:

```
COPYVAR=variable
COPYVARS=(variables)
ROLE< =name >
```

- lists one or more `variables` from the input data table to be transferred to the output data table.
- 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 149.
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 19 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 175.
**WEIGHT Statement**

```plaintext
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: 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 \texttt{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 170. 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 \texttt{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.

\section*{Loh Method}

Specify the \texttt{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 \texttt{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

\[ ASE_{\text{int}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \]

\[ ASE_{\text{cat}} = \frac{1}{JN} \sum_{i=1}^{N} \sum_{j=1}^{J} (\delta_{ij} - \hat{p}_{ij})^2 \]

\[ \text{Misc} = \frac{1}{N} \sum_{i=1}^{N} 1(y_i \neq \hat{y}_i) \]

\[ \text{LogLoss} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{J} \delta_{ij} \log(\tilde{p}_{ij}) \]

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 \( \tilde{p}_{ij} \) is \( \hat{p}_{ij} \) truncated away from 0 and 1:

\[ \tilde{p}_{ij} = \max(\min(p_{ij}, 1 - 10^{-10}), 10^{-10}) \]

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

\[
RSS = \sum_{\lambda} \sum_{i \in \lambda} (y_i - \hat{y}_{\lambda}^T)^2
\]

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

$$\text{RSS} = \sum_{\lambda} \sum_{\Phi} N^\lambda_\Phi \left[ \sum_{\tau \neq \Phi} \left( P^\lambda_\tau \right)^2 + \left( 1 - P^\lambda_\Phi \right)^2 \right]$$

where

- $\Phi$ is the actual response level
- $N^\lambda_\Phi$ is the number of observations on leaf $\lambda$ that have response level $\Phi$
- $P^\lambda_\tau$ 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 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

$$\Delta^v_d = \text{RSS}_d - \sum_i \text{RSS}^d_i$$

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
- $\text{RSS}^d_i$ 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

$$\sum_{d=1}^D \Delta^v_d$$

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

$$
\sum_{d,e} \frac{\Delta_d^v + \Delta_e^w}{\Delta_d^v}
$$

where

- $d$ and $e$ denote nodes, with node $d$ being the parent of node $e$
- $v$ and $w$ are variables, with $v$ being the splitting variable for node $d$ and $w$ being 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, with node $d$ being the parent of node $e$, which is the parent of node $f$
- $v$, $w$, and $x$ are variables, with $v$ being the splitting variable for node $d$, $w$ being the splitting variable for node $e$, and $x$ being 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 only compare a two-way interaction with another two-way interaction, or a three-way interaction 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

$$I_{\text{RBA}}(v) \propto \sum_{i=1}^{n} \text{Loss}(y_i, \hat{y}_i) - \sum_{i=1}^{n} \text{Loss}(y_i, \hat{y}_i)$$

where $\hat{y}_i$ is the modified prediction for observation $i$ and $\hat{y}_i$ is the standard prediction.

For an interval target, PROC 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

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: \( \text{NSUBSESSIONWORKERS} = 1 + \frac{\text{Number of Observations} \times \text{Number of Columns}}{50,000,000} \) (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 \( 2 \times \frac{W}{n} \), where $W$ is the number of workers that are connected to the server and $n$ is the number of workers in the parallel subsessions.
Chapter 10: The FOREST Procedure

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 173. 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 176. 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 171. 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 display 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 10.6 ODS Tables Produced by PROC FOREST

<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>OutCASTbl</td>
<td>See the section “OutputCasTables Table” on page 179</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>
<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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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:

```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.

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`.

```sas
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.
### 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>0.1107</td>
<td>0.1007</td>
<td>0.151</td>
<td>0.1411</td>
<td>0.495</td>
<td>0.397</td>
</tr>
<tr>
<td>2</td>
<td>0.1116</td>
<td>0.0928</td>
<td>0.155</td>
<td>0.1315</td>
<td>0.489</td>
<td>0.312</td>
</tr>
<tr>
<td>3</td>
<td>0.1040</td>
<td>0.0872</td>
<td>0.138</td>
<td>0.1093</td>
<td>0.436</td>
<td>0.290</td>
</tr>
<tr>
<td>4</td>
<td>0.1026</td>
<td>0.0859</td>
<td>0.133</td>
<td>0.1143</td>
<td>0.427</td>
<td>0.288</td>
</tr>
<tr>
<td>5</td>
<td>0.0990</td>
<td>0.0845</td>
<td>0.131</td>
<td>0.1152</td>
<td>0.401</td>
<td>0.283</td>
</tr>
<tr>
<td>6</td>
<td>0.0949</td>
<td>0.0823</td>
<td>0.126</td>
<td>0.1039</td>
<td>0.368</td>
<td>0.278</td>
</tr>
<tr>
<td>7</td>
<td>0.0922</td>
<td>0.0811</td>
<td>0.119</td>
<td>0.1002</td>
<td>0.354</td>
<td>0.277</td>
</tr>
<tr>
<td>8</td>
<td>0.0908</td>
<td>0.0810</td>
<td>0.116</td>
<td>0.0974</td>
<td>0.330</td>
<td>0.278</td>
</tr>
<tr>
<td>9</td>
<td>0.0905</td>
<td>0.0805</td>
<td>0.116</td>
<td>0.0956</td>
<td>0.325</td>
<td>0.277</td>
</tr>
<tr>
<td>10</td>
<td>0.0901</td>
<td>0.0809</td>
<td>0.114</td>
<td>0.0932</td>
<td>0.319</td>
<td>0.277</td>
</tr>
<tr>
<td>91</td>
<td>0.0861</td>
<td>0.0788</td>
<td>0.106</td>
<td>0.0950</td>
<td>0.294</td>
<td>0.273</td>
</tr>
<tr>
<td>92</td>
<td>0.0860</td>
<td>0.0787</td>
<td>0.105</td>
<td>0.0939</td>
<td>0.293</td>
<td>0.273</td>
</tr>
<tr>
<td>93</td>
<td>0.0860</td>
<td>0.0788</td>
<td>0.106</td>
<td>0.0950</td>
<td>0.293</td>
<td>0.273</td>
</tr>
<tr>
<td>94</td>
<td>0.0860</td>
<td>0.0788</td>
<td>0.106</td>
<td>0.0950</td>
<td>0.293</td>
<td>0.273</td>
</tr>
<tr>
<td>95</td>
<td>0.0861</td>
<td>0.0788</td>
<td>0.105</td>
<td>0.0950</td>
<td>0.294</td>
<td>0.273</td>
</tr>
<tr>
<td>96</td>
<td>0.0861</td>
<td>0.0788</td>
<td>0.107</td>
<td>0.0932</td>
<td>0.294</td>
<td>0.273</td>
</tr>
<tr>
<td>97</td>
<td>0.0861</td>
<td>0.0788</td>
<td>0.105</td>
<td>0.0928</td>
<td>0.294</td>
<td>0.273</td>
</tr>
<tr>
<td>98</td>
<td>0.0860</td>
<td>0.0787</td>
<td>0.104</td>
<td>0.0926</td>
<td>0.293</td>
<td>0.273</td>
</tr>
<tr>
<td>99</td>
<td>0.0861</td>
<td>0.0788</td>
<td>0.104</td>
<td>0.0928</td>
<td>0.294</td>
<td>0.273</td>
</tr>
<tr>
<td>100</td>
<td>0.0861</td>
<td>0.0788</td>
<td>0.103</td>
<td>0.0926</td>
<td>0.294</td>
<td>0.273</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**.
Example 10.2: Generating an Isolation Forest

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>0.1007</td>
<td>0.1411</td>
<td>0.397</td>
</tr>
<tr>
<td>2</td>
<td>0.0928</td>
<td>0.1315</td>
<td>0.312</td>
</tr>
<tr>
<td>3</td>
<td>0.0872</td>
<td>0.1093</td>
<td>0.290</td>
</tr>
<tr>
<td>4</td>
<td>0.0859</td>
<td>0.1143</td>
<td>0.288</td>
</tr>
<tr>
<td>5</td>
<td>0.0845</td>
<td>0.1152</td>
<td>0.283</td>
</tr>
<tr>
<td>6</td>
<td>0.0823</td>
<td>0.1039</td>
<td>0.278</td>
</tr>
<tr>
<td>7</td>
<td>0.0811</td>
<td>0.1002</td>
<td>0.277</td>
</tr>
<tr>
<td>8</td>
<td>0.0810</td>
<td>0.0974</td>
<td>0.278</td>
</tr>
<tr>
<td>9</td>
<td>0.0805</td>
<td>0.0956</td>
<td>0.277</td>
</tr>
<tr>
<td>10</td>
<td>0.0809</td>
<td>0.0932</td>
<td>0.277</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>0.0788</td>
<td>0.0950</td>
<td>0.273</td>
</tr>
<tr>
<td>92</td>
<td>0.0787</td>
<td>0.0939</td>
<td>0.273</td>
</tr>
<tr>
<td>93</td>
<td>0.0788</td>
<td>0.0950</td>
<td>0.273</td>
</tr>
<tr>
<td>94</td>
<td>0.0788</td>
<td>0.0950</td>
<td>0.273</td>
</tr>
<tr>
<td>95</td>
<td>0.0788</td>
<td>0.0950</td>
<td>0.273</td>
</tr>
<tr>
<td>96</td>
<td>0.0788</td>
<td>0.0932</td>
<td>0.273</td>
</tr>
<tr>
<td>97</td>
<td>0.0788</td>
<td>0.0928</td>
<td>0.273</td>
</tr>
<tr>
<td>98</td>
<td>0.0787</td>
<td>0.0926</td>
<td>0.273</td>
</tr>
<tr>
<td>99</td>
<td>0.0788</td>
<td>0.0928</td>
<td>0.273</td>
</tr>
<tr>
<td>100</td>
<td>0.0788</td>
<td>0.0926</td>
<td>0.273</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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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, x1 and x2. 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).
data mycas.spiral;
  keep x1 x2 y id;
  nNow = 200;
  sdNow = 0.2;
  four_pi = 4 * constant("pi");
call streaminit(33131333);
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;
      target y /level=nominal;
      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.
Example 10.2: Generating an Isolation Forest

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.

**Output 10.2.1** Spiral Data Colored by Anomaly Score
References


Chapter 11
The GMM Procedure

Contents

Overview: GMM Procedure .............................................. 188
GMM Procedure Features ............................................. 188
GMM Procedure Results .............................................. 188
Using CAS Sessions and CAS Engine Librefs .......................... 189
Getting Started: GMM Procedure ................................... 189
Syntax: GMM Procedure ................................................ 194
   PROC GMM Statement ............................................... 195
   DISPLAY Statement ................................................. 197
   DISPLAYOUT Statement ........................................... 198
   INPUT Statement ................................................... 199
   SAVESTATE Statement ............................................ 199
   SCORE Statement .................................................. 199
Details: GMM Procedure ................................................ 200
   Gaussian Mixture Model ....................................... 200
   Dirichlet Process .................................................. 201
   Missing Values ...................................................... 202
   Random Seed ......................................................... 202
   Variational Bayes Method .................................... 202
   Covariance Matrix Type .................................... 202
   Score Output ....................................................... 203
   ClusterSumOut Output ................................ ...... 203
   ClusterCovOut Output ................................ ...... 203
   Displayed Output ............................................... 204
      Number of Observations ................................... 204
      Descriptive Statistics ...................................... 204
      Model Information ........................................... 204
   ODS Table Names ............................................... 204
Examples: GMM Procedure ............................................ 205
   Example 11.1: Performing Cluster Analysis .......................... 205
   Example 11.2: Performing Cluster Analysis and Storing the Model 208
References .............................................................. 210
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 200.
- 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 201.
- 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 202.

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 203.
- 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 203.
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 203.

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 199.

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 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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;
  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.
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).

```
proc print noobs data=mycas.score(obs=10);
run;
```

**Figure 11.4** Cluster Score Table

**Using PROC GMM for Soft Clustering**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>c</td>
<td>_CLUSTER_1</td>
<td>_CLUSTER_2</td>
<td>_CLUSTER_3</td>
</tr>
<tr>
<td>-0.32659</td>
<td>1.54244</td>
<td>1</td>
<td>0.269141085</td>
<td>0.6053353352</td>
<td>0.1255235798</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>
</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>
</tr>
<tr>
<td>1.67034</td>
<td>-0.47339</td>
<td>1</td>
<td>0.0001843824</td>
<td>0.990062543</td>
<td>0.0097530746</td>
</tr>
<tr>
<td>-1.0178</td>
<td>0.27807</td>
<td>1</td>
<td>0.020096045</td>
<td>0.9741773484</td>
<td>0.0057266066</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>
</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>
</tr>
<tr>
<td>-0.25232</td>
<td>-1.61289</td>
<td>1</td>
<td>0.0006497335</td>
<td>0.999345977</td>
<td>4.6684486-6</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>
</tr>
<tr>
<td>-0.33846</td>
<td>-3.07875</td>
<td>1</td>
<td>0.0000492516</td>
<td>0.9999507405</td>
<td>7.9404955E-9</td>
</tr>
</tbody>
</table>

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:

```sas
proc print noobs data=mycas.clustersum;
run;
```

![Figure 11.6 Cluster Summary Table](image)

**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>1</td>
<td>-1.74786</td>
<td>2.50806</td>
</tr>
<tr>
<td>2</td>
<td>110</td>
<td>0</td>
<td>-0.05153</td>
<td>0.05330</td>
</tr>
<tr>
<td>3</td>
<td>102</td>
<td>1</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:

```sas
proc print noobs data=mycas.clustercov;
run;
```

![Figure 11.7 Cluster Covariance Table](image)

**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:

```sas
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><strong>Input and Output Data Set Option</strong></td>
<td></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the input data table</td>
</tr>
<tr>
<td><strong>Performance Option</strong></td>
<td></td>
</tr>
<tr>
<td>NTHREADS=</td>
<td>Specifies the number of threads to use for the computation</td>
</tr>
<tr>
<td><strong>Clustering Options</strong></td>
<td></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>INFEERENCE=</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 189.

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 203.
CLUSTERSUMOUT=\texttt{CAS-libref.data-table}
creates the output data table that contains the summary of the clustering results. \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 189.

The summary consists of the following items: the cluster ID (_\texttt{CLUSTER\_ID}_), the numbers of observations in the clusters (_\texttt{SIZE}_), the clusters closest to the clusters (_\texttt{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 203.

\texttt{DATA=\texttt{CAS-libref.data-table}}
names the input data table for PROC GMM 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 \texttt{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 \texttt{LIBNAME} statement. For more information about \texttt{CAS-libref}, see the section “Using CAS Sessions and CAS Engine Librefs” on page 189.

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

\texttt{INFEERENCE=VB < suboptions>}
performs the inference by using the variational Bayesian (VB) algorithm; for more information see the section “Variational Bayes Method” on page 202.

You can specify one or more of the following \texttt{suboptions}:

\texttt{COVARIANCE=keyword} specifies the type of covariance matrices in the Gaussian mixture model. For more information, see the section “Covariance Matrix Type” on page 202. You can specify the following \texttt{keywords}:

\begin{itemize}
\item \texttt{DIAGONAL} specifies the covariance matrices of the Gaussian distributions as diagonal matrices.
\item \texttt{FULL} specifies the covariance matrices of the Gaussian distributions as full matrices.
\end{itemize}

By default, \texttt{COVARIANCE=DIAGONAL}.

\texttt{MAXVBITER=number} specifies the maximum number of VB iterations. The VB iterations stop either when the number of iterations reaches the \texttt{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 \texttt{THRESHOLD=number}. When the VB iterations stop with the threshold criterion satisfied, the VB reaches its convergence. By default, \texttt{MAXVBITER=100}.

\texttt{THRESHOLD=number} specifies the threshold to define the convergence of the VB algorithm. By default, \texttt{THRESHOLD=0.01}.
**DISPLAY Statement**

**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 204. 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 (/):

**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 202.
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 204. 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 <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

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 189.

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 203.

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 189.

`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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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.
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.

```sas
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**

<table>
<thead>
<tr>
<th>The GMM Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
</tr>
<tr>
<td>Number of Observations Used</td>
</tr>
</tbody>
</table>

**Output 11.1.2** Descriptive Statistics

<table>
<thead>
<tr>
<th>Input Variable Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</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 203. The following PROC SGPLOT 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
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 203. 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.

```
proc print noobs data=mycas.clustersum;
run;
```

Output 11.1.5  Cluster Summary Table

<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>2</td>
<td>2.41892</td>
<td>-0.10389</td>
</tr>
<tr>
<td>2</td>
<td>54</td>
<td>0</td>
<td>4.79783</td>
<td>2.38858</td>
</tr>
<tr>
<td>3</td>
<td>51</td>
<td>0</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 203. In this table, you can see that PROC GMM successfully captures the variation in the shapes of the clusters.

```
proc print noobs data=mycas.clustercov;
run;
```

Output 11.1.6  Cluster Covariance Table

<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.
Example 11.2: Performing Cluster Analysis and Storing the Model

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:

```plaintext
proc astore;
  score data=mycas.fish_test out=mycas.newscore
    copyvars=(_ALL_) rstore=mycas.astore;
run;
```
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.

```
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>_CLUSTER_1</th>
<th>_CLUSTER_2</th>
<th>_CLUSTER_3</th>
<th>_CLUSTER_4</th>
<th>_CLUSTER_5</th>
<th>_CLUSTER_6</th>
<th>_CLUSTER_7</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>_CLUSTER_8</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 .......................................................... 212
Using CAS Sessions and CAS Engine Librefs ........................................ 212
Getting Started: GRADBOOST Procedure .............................................. 213
Syntax: GRADBOOST Procedure .............................................................. 217
PROC GRADBOOST Statement ............................................................. 218
AUTOTUNE Statement ..................................................................... 223
CODE Statement ........................................................................... 228
CROSSVALIDATION Statement ......................................................... 228
ID Statement .................................................................................. 229
INPUT Statement ............................................................................. 229
OUTPUT Statement ......................................................................... 230
PARTITION Statement ..................................................................... 231
SAVESTATE Statement ..................................................................... 231
TARGET Statement ........................................................................... 232
TRANSFERLEARN Statement ............................................................. 232
VIICODE Statement ........................................................................... 232
WEIGHT Statement ........................................................................... 233
Details: GRADBOOST Procedure .......................................................... 233
Subsampling the Data ...................................................................... 233
Training a Decision Tree .................................................................... 234
Boosting .......................................................................................... 234
Early Stopping .................................................................................. 235
Measuring Prediction Error ................................................................. 235
Handling Missing Values .................................................................... 236
Strategies .......................................................................................... 236
Specifics ............................................................................................ 237
Measuring Variable Importance ............................................................ 237
Residual Sum of Squares Importance Method ........................................ 238
Random Branch Assignment Importance Method .................................. 240
Transfer Learning .............................................................................. 240
Hyperparameter Tuning ..................................................................... 241
k-fold Cross Validation ....................................................................... 241
Displayed Output ............................................................................... 241
Model Information ............................................................................ 241
Number of Observations .................................................................... 241
Variable Importance .......................................................................... 242
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 234.

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:

RBA Variable Importance ........................................ 242
Fit Statistics ...................................................... 242
Tuner Information .................................................. 242
Tuner Summary ..................................................... 242
Tuner Timing ........................................................ 242
Best Configuration .................................................. 242
Tuner Results ....................................................... 242
Cross-Validation Fit Statistics ..................................... 242
Evaluation History .................................................. 243
OutputCasTables Table ............................................ 243
Predicted Probability Names ...................................... 243
ODS Table Names ................................................... 243
Example: GRADBOOST Procedure .......................... 244
Example 12.1: Scoring New Data by Using a Previous Boosting Model .... 245
Example 12.2: Transfer Learning ............................... 248
References .......................................................... 251
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
proc options option=(CASHOST CASPORT);
run;
```

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 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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.

```plaintext
data mycas.hmeq;
    length Bad Loan MortDue Value 8 Reason Job $7 YoJ Derog Delinq CLAge nInq CLNo DebtInc 8;
    set sampsio.hmeq;
run;
```

```plaintext
proc print data=mycas.hmeq(obs=10); run;
```

Output 12.1 shows the first 10 observations of mycas.hmeq.

Figure 12.1  Partial Listing of the mycas.hmeq Data

<table>
<thead>
<tr>
<th>Obs</th>
<th>Bad</th>
<th>Loan</th>
<th>MortDue</th>
<th>Value</th>
<th>Reason</th>
<th>Job</th>
<th>YoJ</th>
<th>Derog</th>
<th>Delinq</th>
<th>CLAge</th>
<th>nInq</th>
<th>CLNo</th>
<th>DebtInc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1100</td>
<td>25860</td>
<td>39025</td>
<td>Homelmp Other</td>
<td></td>
<td>10.5</td>
<td>0</td>
<td>0</td>
<td>94.367</td>
<td>1</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1500</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td></td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1800</td>
<td>48649</td>
<td>57037</td>
<td>Homelmp Other</td>
<td></td>
<td>5.0</td>
<td>3</td>
<td>2</td>
<td>77.100</td>
<td>1</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2000</td>
<td>62250</td>
<td>73995</td>
<td>Homelmp Sales</td>
<td></td>
<td>16.0</td>
<td>0</td>
<td>0</td>
<td>115.800</td>
<td>0</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2000</td>
<td>45000</td>
<td>55000</td>
<td>Homelmp Other</td>
<td></td>
<td>3.0</td>
<td>0</td>
<td>0</td>
<td>86.067</td>
<td>2</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2200</td>
<td>24280</td>
<td>34687</td>
<td>Homelmp Other</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1300.867</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2300</td>
<td>28192</td>
<td>40150</td>
<td>Homelmp Other</td>
<td></td>
<td>4.5</td>
<td>0</td>
<td>0</td>
<td>54.600</td>
<td>1</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2400</td>
<td>50000</td>
<td>73995</td>
<td>Homelmp ProfExe</td>
<td></td>
<td>5.0</td>
<td>1</td>
<td>0</td>
<td>.</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2400</td>
<td>17180</td>
<td>.</td>
<td>Homelmp Other</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>14.567</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>2500</td>
<td>15000</td>
<td>20200</td>
<td>Homelmp</td>
<td></td>
<td>18.0</td>
<td>0</td>
<td>0</td>
<td>136.067</td>
<td>1</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>
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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Trees</td>
<td>100</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.1</td>
</tr>
<tr>
<td>Subsampling Rate</td>
<td>0.5</td>
</tr>
<tr>
<td>Number of Variables Per Split</td>
<td>12</td>
</tr>
<tr>
<td>Number of Bins</td>
<td>20</td>
</tr>
<tr>
<td>Number of Input Variables</td>
<td>12</td>
</tr>
<tr>
<td>Maximum Number of Tree Nodes</td>
<td>63</td>
</tr>
<tr>
<td>Minimum Number of Tree Nodes</td>
<td>27</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>5</td>
</tr>
<tr>
<td>Minimum Depth</td>
<td>5</td>
</tr>
<tr>
<td>Maximum Number of Leaves</td>
<td>32</td>
</tr>
<tr>
<td>Minimum Number of Leaves</td>
<td>14</td>
</tr>
<tr>
<td>Maximum Leaf Size</td>
<td>2628</td>
</tr>
<tr>
<td>Minimum Leaf Size</td>
<td>5</td>
</tr>
<tr>
<td>Seed</td>
<td>12345</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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
<td>5960</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>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 DebtInc is the most important predictor of loan default.

Figure 12.4  Variable Importance

<table>
<thead>
<tr>
<th>Variable</th>
<th>Variable Importance</th>
<th>Std Dev Importance</th>
<th>Relative Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>DebtInc</td>
<td>26.4984</td>
<td>50.2784</td>
<td>1.0000</td>
</tr>
<tr>
<td>Delinq</td>
<td>7.0530</td>
<td>3.8573</td>
<td>0.2662</td>
</tr>
<tr>
<td>nlnq</td>
<td>6.4986</td>
<td>1.6383</td>
<td>0.2452</td>
</tr>
<tr>
<td>CLAge</td>
<td>5.3236</td>
<td>2.9120</td>
<td>0.2009</td>
</tr>
<tr>
<td>Derog</td>
<td>5.2378</td>
<td>3.4850</td>
<td>0.1977</td>
</tr>
<tr>
<td>CLNo</td>
<td>5.1225</td>
<td>1.5044</td>
<td>0.1933</td>
</tr>
<tr>
<td>Value</td>
<td>4.9268</td>
<td>4.1444</td>
<td>0.1859</td>
</tr>
<tr>
<td>Job</td>
<td>4.9220</td>
<td>1.7925</td>
<td>0.1857</td>
</tr>
<tr>
<td>YoJ</td>
<td>4.2145</td>
<td>1.4583</td>
<td>0.1590</td>
</tr>
<tr>
<td>MortDue</td>
<td>4.2129</td>
<td>1.8211</td>
<td>0.1590</td>
</tr>
<tr>
<td>Loan</td>
<td>3.1382</td>
<td>1.5031</td>
<td>0.1184</td>
</tr>
<tr>
<td>Reason</td>
<td>0.9067</td>
<td>0.7821</td>
<td>0.0342</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 ;
```

---

**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>0.1442</td>
<td>0.1995</td>
<td>0.455</td>
</tr>
<tr>
<td>2</td>
<td>0.1325</td>
<td>0.1995</td>
<td>0.424</td>
</tr>
<tr>
<td>3</td>
<td>0.1224</td>
<td>0.1993</td>
<td>0.399</td>
</tr>
<tr>
<td>4</td>
<td>0.1146</td>
<td>0.1990</td>
<td>0.379</td>
</tr>
<tr>
<td>5</td>
<td>0.1083</td>
<td>0.1780</td>
<td>0.363</td>
</tr>
<tr>
<td>6</td>
<td>0.1025</td>
<td>0.1522</td>
<td>0.348</td>
</tr>
<tr>
<td>7</td>
<td>0.0977</td>
<td>0.1319</td>
<td>0.336</td>
</tr>
<tr>
<td>8</td>
<td>0.0942</td>
<td>0.1183</td>
<td>0.326</td>
</tr>
<tr>
<td>9</td>
<td>0.0902</td>
<td>0.1119</td>
<td>0.314</td>
</tr>
<tr>
<td>10</td>
<td>0.0872</td>
<td>0.1059</td>
<td>0.306</td>
</tr>
<tr>
<td>91</td>
<td>0.0362</td>
<td>0.0492</td>
<td>0.134</td>
</tr>
<tr>
<td>92</td>
<td>0.0358</td>
<td>0.0478</td>
<td>0.133</td>
</tr>
<tr>
<td>93</td>
<td>0.0353</td>
<td>0.0461</td>
<td>0.132</td>
</tr>
<tr>
<td>94</td>
<td>0.0350</td>
<td>0.0455</td>
<td>0.130</td>
</tr>
<tr>
<td>95</td>
<td>0.0347</td>
<td>0.0450</td>
<td>0.130</td>
</tr>
<tr>
<td>96</td>
<td>0.0344</td>
<td>0.0448</td>
<td>0.128</td>
</tr>
<tr>
<td>97</td>
<td>0.0340</td>
<td>0.0446</td>
<td>0.128</td>
</tr>
<tr>
<td>98</td>
<td>0.0335</td>
<td>0.0441</td>
<td>0.126</td>
</tr>
<tr>
<td>99</td>
<td>0.0333</td>
<td>0.0438</td>
<td>0.125</td>
</tr>
<tr>
<td>100</td>
<td>0.0331</td>
<td>0.0430</td>
<td>0.124</td>
</tr>
</tbody>
</table>
```
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

**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>Add 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>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>NTREES=</td>
<td>Specifies the number of trees to grow in the boosting 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 which the gradient boosting model is to be saved</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>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>

<table>
<thead>
<tr>
<th><strong>Splitting Options</strong></th>
<th></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 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 234. During model scoring, observations are assigned to a node in a tree based upon 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 leftmost branch of the split, and assigns observations that have unknown and missing nominal levels to the largest branch in the split.

**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=BUCKET.

**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 212.

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

**EARLYSTOP**(suboption …)

specifies options for stopping the gradient boosting model training early. For more information, see the section “Early Stopping” on page 235. You can specify one or more of the following suboptions:

**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 no early stopping is performed. If you want to use autotuning by specifying AUTOTUNE statement, but you do not want autotuning to use early stopping, then specify STAGNATION=0.

By default, STAGNATION=0. When the AUTOTUNE statement is specified, STAGNATION=4 by default.

**TOLERANCE=**number

specifies the *number* to be used as the tolerance for 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 212.

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.

**LASSO=number**

**L1=number**

specifies the L1 norm regularization parameter, where *number* must be nonnegative.

By default, LASSO=0. This value can be tuned with the AUTOTUNE statement.

**LEARNINGRATE=number**

specifies the learning rate for the gradient boosting algorithm, where *number* must be between 0 and 1, inclusive.

By default, LEARNINGRATE=0.1. This value can be tuned with the AUTOTUNE statement.

**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.

**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=5.

**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.

By default, MINLEAFSIZE=5.

**MINUSERINSEARCH=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 GRADBOOST uses the USEINSEARCH policy to handle missing values for that variable.

By default, MINUSERINSEARCH=1.

**NOPRINT**

suppresses ODS output.
Chapter 12: The GRADBOOST Procedure

**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=20.

**OUTMODEL=<CAS-libref.>data-table**
specifies the data table to which to save the gradient boosting 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 212.

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 OUT= table from the OUTPUT statement. For a continuous response, PROC GRADBOOST generates an output table named PredName, which indicates the name of the predicted value column. For a categorical response, PROC GRADBOOST 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 237.

**RIDGE=number**

**L2=number**
specifies the L2 norm regularization parameter on prediction. The number must be nonnegative.

By default, RIDGE=0. This value can be tuned with the AUTOTUNE statement.

**SAMPLINGRATE=number**
specifies the fraction of the training data to be used for growing each tree in the boosting model.

By default, SAMPLINGRATE=0.5. This value can be tuned with the AUTOTUNE statement.

**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 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 “Interaction Importance” on page 239. You can specify the following values:

\[ 2 \quad \text{calculates the importance of all two-way variable interactions.} \]

\[ 3 \quad \text{calculates the importance of all three-way and all two-way variable interactions.} \]

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=, NTREES=, 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 \textit{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 14 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>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>KFOLD=</td>
<td>Specifies the number of folds for k-fold cross validation</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 train</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>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:

**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.

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.

**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**

```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 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>

For more information about the syntax of the CODE statement, see the section “CODE Statement” on page 19 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 the this statement if you specify either the AUTOTUNE statement or the PARTITION statement.
You can specify the following options:

**K FOLD=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.

By default, no monotonic constraints are placed on model with respect to the variables.

**OUTPUT Statement**

```r
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 212.

`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>
<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 212.
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 240.

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 downweighting any observation in the auxiliary data.
  By default, BURN=0.

SHRINKAGE=number
  specifies the number to apply as the weighting factor for downweighting 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 downweighted, where number must be greater than 0 less than or equal to 1/2.
  By default, TRIMMING=0.01.

VIIICODE Statement

VIIICODE < 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 19 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 239.

---

**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.

---

**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 234. 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

\[
L(F) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)
\]

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.

The GRADBOOST procedure performs early stopping by considering a number of consecutive decreases. Two suboptions of the EARLYSTOP option in the PROC GRADBOOST statement dictate how the early stopping is performed: STAGNATION=number and TOLERANCE=$\tau$ ($\tau$ can be any number greater than or equal to 0).

If the relative change in error from iteration $i-1$ to iteration $i$ is less than $\tau$,

$$\left| \frac{Err_{i-1} - Err_i}{Err_{i-1}} \right| < \tau,$$

for number iterations, then the boosting process stops and returns the model from the last iteration in which the relative error difference was greater than or equal to $\tau$. If $\tau = 0$, then the stopping criterion is determined by consecutive increases in the error.

**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 ASE_{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

$$RSS = \sum_\lambda \sum_{i \in \lambda} \left( y_i - \hat{y}_i^\lambda \right)^2$$

where

- $i$ is an observation on leaf $\lambda$
- $y_i$ is the predicted value of the response variable of observation $i$
- $\hat{y}_i^\lambda$ is the actual value of the response variable on leaf $\lambda$

The residual sum of squares (RSS) for classification trees is defined as

$$RSS = \sum_\lambda \sum_{\Phi} N_\Phi^\lambda \left[ \sum_{\tau \neq \Phi} \left( P_\tau^\lambda \right)^2 + \left( 1 - P_\Phi^\lambda \right)^2 \right]$$

where

- $\Phi$ is the actual response level
- $N_\Phi^\lambda$ is the number of observations on leaf $\lambda$ that have response level $\Phi$
- $P_\tau^\lambda$ 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 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

$$\Delta_d = RSS_d - \sum_i RSS_i^d$$

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

$$\sum_{d=1}^D \Delta_\lambda^v$$

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 `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

\[
\sum_{d,e} \frac{\Delta^v_d + \Delta^w_e}{\Delta^v_d}
\]

where

- \( d \) and \( e \) denote nodes, with node \( d \) being the parent of node \( e \)
- \( v \) and \( w \) are variables, with \( v \) being the splitting variable for node \( d \) and \( w \) being 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

\[
\sum_{d,e,f} \frac{\Delta^v_d + \Delta^w_e + \Delta^x_f}{\Delta^v_d + \Delta^w_e}
\]

where

- \( d \), \( e \), and \( f \) denote nodes, with node \( d \) being the parent of node \( e \), which is the parent of node \( f \)
- \( v \), \( w \), and \( x \) are variables, with \( v \) being the splitting variable for node \( d \), \( w \) being the splitting variable for node \( e \), and \( x \) being 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 only compare a two-way interaction with another two-way interaction, or a three-way interaction 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.
Chapter 12: The GRADBOOST Procedure

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 towards the auxiliary data, the GRADBOOST procedure downweights auxiliary observations that are dissimilar from the training observations. The factor that is applied to downweight 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 either in a low quantile or 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 downweighted.

The GRADBOOST procedure begins downweighting 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

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
- NTREES= option for the number of trees
- 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 237. 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 240. 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 235. 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.

ODS Table Names

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>
<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 243</td>
<td>PROC GRADBOOST / OUTMODEL=</td>
<td>OUTPUT</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>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>
<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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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:

```
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 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`.

```
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.
### Output 12.1.1  Fit Statistics, Fit at Run Time

<table>
<thead>
<tr>
<th>Number of Trees</th>
<th>Training Square Error</th>
<th>Training Misclassification Rate</th>
<th>Training Log Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2152</td>
<td>0.3940</td>
<td>0.621</td>
</tr>
<tr>
<td>2</td>
<td>0.1957</td>
<td>0.2515</td>
<td>0.581</td>
</tr>
<tr>
<td>3</td>
<td>0.1797</td>
<td>0.2271</td>
<td>0.547</td>
</tr>
<tr>
<td>4</td>
<td>0.1673</td>
<td>0.1960</td>
<td>0.520</td>
</tr>
<tr>
<td>5</td>
<td>0.1566</td>
<td>0.1524</td>
<td>0.497</td>
</tr>
<tr>
<td>6</td>
<td>0.1475</td>
<td>0.1491</td>
<td>0.476</td>
</tr>
<tr>
<td>7</td>
<td>0.1400</td>
<td>0.1500</td>
<td>0.458</td>
</tr>
<tr>
<td>8</td>
<td>0.1333</td>
<td>0.1361</td>
<td>0.442</td>
</tr>
<tr>
<td>9</td>
<td>0.1273</td>
<td>0.1363</td>
<td>0.427</td>
</tr>
<tr>
<td>10</td>
<td>0.1226</td>
<td>0.1317</td>
<td>0.415</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>0.0565</td>
<td>0.0702</td>
<td>0.198</td>
</tr>
<tr>
<td>92</td>
<td>0.0564</td>
<td>0.0702</td>
<td>0.198</td>
</tr>
<tr>
<td>93</td>
<td>0.0563</td>
<td>0.0709</td>
<td>0.197</td>
</tr>
<tr>
<td>94</td>
<td>0.0561</td>
<td>0.0702</td>
<td>0.197</td>
</tr>
<tr>
<td>95</td>
<td>0.0558</td>
<td>0.0696</td>
<td>0.196</td>
</tr>
<tr>
<td>96</td>
<td>0.0557</td>
<td>0.0696</td>
<td>0.195</td>
</tr>
<tr>
<td>97</td>
<td>0.0555</td>
<td>0.0689</td>
<td>0.195</td>
</tr>
<tr>
<td>98</td>
<td>0.0553</td>
<td>0.0691</td>
<td>0.194</td>
</tr>
<tr>
<td>99</td>
<td>0.0551</td>
<td>0.0689</td>
<td>0.193</td>
</tr>
<tr>
<td>100</td>
<td>0.0548</td>
<td>0.0678</td>
<td>0.192</td>
</tr>
</tbody>
</table>
The following statements use a previously saved model to score new data:

```plaintext
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>Fit Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Square Error</td>
<td>0.2152</td>
</tr>
<tr>
<td>Misclassification Rate</td>
<td>0.3940</td>
</tr>
<tr>
<td>Log Loss</td>
<td>0.621</td>
</tr>
<tr>
<td>91</td>
<td>0.0565</td>
</tr>
<tr>
<td>92</td>
<td>0.0564</td>
</tr>
<tr>
<td>93</td>
<td>0.0563</td>
</tr>
<tr>
<td>94</td>
<td>0.0561</td>
</tr>
<tr>
<td>95</td>
<td>0.0558</td>
</tr>
<tr>
<td>96</td>
<td>0.0557</td>
</tr>
<tr>
<td>97</td>
<td>0.0555</td>
</tr>
<tr>
<td>98</td>
<td>0.0553</td>
</tr>
<tr>
<td>99</td>
<td>0.0551</td>
</tr>
<tr>
<td>100</td>
<td>0.0548</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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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 comes from a target population from which data is 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;
```
Example 12.2: Transfer Learning

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.

```latex
%macro runProc(data=, auxVar=, outfit=);
  proc gradboost data=&data. outmodel=mycas.model seed=3331333;
    input x: /level=interval;
    target y /level=nominal;
    transferLearn &auxVar. / burn=10;
  run;
  proc gradboost data=mycas.test inmodel=mycas.model;
    output out=mycas.score;
    ods output FitStatistics=&outfit. ;
  run;
%mend;

%runProc(data=mycas.train, auxVar=datarole, outfit=fit_train);
%runProc(data=mycas.train, auxVar=constantZero, outfit=fit_noTransfer);
%runProc(data=mycas.noAlien, auxVar=constantZero, outfit=fit_noAlien);

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;
run;

proc sgrender data=result template=transferLearning;
run;
```

Plot 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 if for the first 10 trees because down-weighting does not begin in until tree eleven in this example.

**Output 12.2.1** ASE versus Number of Trees for Three Models
References


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
Chapter 13: The GVARCLUS Procedure

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
- exploits all the available cores and concurrent threads. For information about how PROC GVARCLUS
uses threads, see the section “Multithreading” on page 22 in Chapter 3, “Shared Concepts.”
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:

```
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 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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;
  end;
output;
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

The GVARCLUS Procedure

<table>
<thead>
<tr>
<th>Number of Observations Read</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Used</td>
<td>100</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.

<table>
<thead>
<tr>
<th></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>

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.

```sas
proc template;
  define statgraph dendrogram;
  begingroup;
    layout overlay;
```
**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 INPUT 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 13.1 PROC GVARCLUS 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>DATA=</td>
<td>Specifies the input data table</td>
</tr>
<tr>
<td><strong>Options Related to Variable Clustering</strong></td>
<td></td>
</tr>
<tr>
<td>EXACT</td>
<td>Performs graphical variable clustering without preprocessing</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations for each estimation step</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><strong>Options for Output Tables</strong></td>
<td></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`

names 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 255.

- **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 255.

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 255.
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 \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 255.

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 \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 255.

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=\texttt{NONE} | \texttt{PENALIZED}
selects the optimal clustering on the basis of the specified criterion. You can specify the following values:

- \texttt{NONE} performs no selection, and the output tables of the edges and vertices of the network reflect the clustering from the last step.
- \texttt{PENALIZED} performs selection through minimizing the penalized log likelihood, as shown in the section “Selection” on page 266.

By default, SELECT=NONE.

STOP=\texttt{n}
requests that the action stop if the clustering results do not change in the previous \texttt{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**

\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 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 266. 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 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:
**FREQ Statement**

```
FREQ variable;
```

The `variable` in the FREQ statement identifies a numeric variable in the data set 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:

$$\min \left\{ \frac{1}{2} \| W_{11}^{1/2} \beta - b \|^2 + \lambda \| \beta \|_1 \right\}$$

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 $X_k$ through 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 $\Theta^\lambda$ denote the solution at $\lambda$. The element $(i, j)$ in the matrix is 1 when
$\hat{\Theta}_{ij}^\lambda \neq 0$ 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 $(\hat{\theta}_{ij})_{1 \leq i, j \leq p}$ 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 13.2  ODS TABLES PRODUCED BY PROC GVARCLUS

<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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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. \( \text{RHO}=0.8 \) 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 downloads 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;
```

```sas
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 ($x_1$–$x_2$) and nominal variables ($a$, $c_1$, and $c_2$):

```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;
```
Chapter 13: The GVARCLUS Procedure

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:

```plaintext
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><em>VAR</em></th>
<th><em>vID</em></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/FREQ</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 _vID_ column contains $n$ markers, $v1$ to $vn$, where $v_{i}$ denotes the $i$th 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 $4 \times 4$, 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 ($x1$–$x2$) and a categorical variable ($a$):
Example 13.3: Output the Covariance Matrix in LIL Format

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:

    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


Overview: MBANALYSIS Procedure

The MBANALYSIS (market basket analysis) procedure performs association rule mining on a transaction database. Association rules are in the form of \textit{if} A \textit{then} B, where A and B are items in the transaction database. Association rules helps 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 280.
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:

```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 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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:

```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 274, 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:

```sas
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:

```
PROC MBANALYSIS options ;
  CUSTOMER variables ;
  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 274.
  - `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 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=TRUE | FALSE
specifies whether to normalize the values of the items in the output tables that are specified in the OUTPUT statement.
By default, NORM=FALSE.

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;

CUSTOMER statement groups one or more specified target variables into a basket. You can also specify this statement as ID variables.

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 274.

OUTPUT Statement

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 274.

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 274.
OUTRULE=\textit{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. \textit{CAS-libref.data-table} is a
two-level name, where \textit{CAS-libref} refers to the caslib and session identifier, and \textit{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 274.

SAVESTATE Statement

\textbf{SAVESTATE RSTORE=\textit{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:

\textbf{RSTORE=\textit{CAS-libref.data-table}}

specifies a data table in which to save the analytic store for the model. \textit{CAS-libref.data-table} is a
two-level name, where \textit{CAS-libref} refers to the caslib and session identifier, and \textit{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 274.

TARGET Statement

\textbf{TARG\textit{ET variable ;}}

The TARGET statement specifies a single nominal \textit{variable} to use as the target variable. If this \textit{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:

\[ \text{butter, bread} \Rightarrow \text{Milk} \]

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

\[ A \Rightarrow B : P(A, B) = \frac{X}{Y} \]

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

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 \( \text{mine}(	ext{CFPT}|\text{node}) \), 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 14.1.

<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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 in Chapter 3, “Shared Concepts.”

**Example 14.1: 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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;
```
Chapter 14: The MBANALYSIS Procedure

The following statements run the association analysis on the mycas.assocs data table:

```
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 15

The MTLEARN Procedure

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:

```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 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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.
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 \(Y_2\) and \(Y_3\) are connected, meaning that the regression weights for \(Y_2\) and \(Y_3\) are expected to be similar. This data table contains four variables: id is the row ID variable, and \(Y_1\), \(Y_2\), and \(Y_3\) 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:

```sas
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;
```

`GRAPHTYPE=CUSTOM` indicates the use of a custom graph table that is specified in the `GRAPHTABLE=` option; `REGL1=0.5` specifies the value of the \(\ell_1\) penalization weight; `REGL2=1.0` specifies the value of the \(\ell_2\) penalization weight; `MAXITERS=20` specifies the maximum number of iterations; `TOLERANCE=1E–2` specifies the optimization tolerance as a stopping criterion; `SEED=123` specifies the seed to use for pseudorandom number generation; `GRAPHTABLE=MYCAS.TOYR` specifies the user-defined graph table;
MODELOUT=MYCAS.MTL_OUTW writes the estimated multitask regression weights to the data table mycas.mtl_outW; GRAPHOUT=MYCAS.MTL_OUTR writes the graph table to the data table mycas.mtl_outR. The INPUT statement specifies that the X1, X2, and X3 variables be used as inputs. The TARGET statement specifies that the Y1, Y2, and Y3 variables be used as targets. The OUTPUT statement writes the scored results to the data table mycas.mtl_out, and the COPYVARS= option copies the id, X1, X2, and X3 variables to the output. The SAVESTATE statement with the RSTORE= option stores the estimated model in the data table mycas.mtl_ast for future scoring.

Figure 15.1 shows the values of the parameters that are used in multitask learning.

**Figure 15.1 Model Information**

The MTLEARN Procedure

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seed</td>
</tr>
<tr>
<td>L1 Regularization</td>
</tr>
<tr>
<td>L2 Regularization</td>
</tr>
<tr>
<td>Tolerance</td>
</tr>
<tr>
<td>Maximum Iterations</td>
</tr>
</tbody>
</table>

Figure 15.2 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 15.2 Number of Observations**

| Number of Observations Read | 10 |
| Number of Observations Used | 10 |

Figure 15.3 shows statistics for each interval variable in the INPUT and TARGET statements, including the mean and standard deviation.

**Figure 15.3 Interval Variable Statistics**

<table>
<thead>
<tr>
<th>Interval Variables</th>
<th>Variable</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X1</td>
<td>0.486773</td>
<td>0.863056</td>
</tr>
<tr>
<td></td>
<td>X2</td>
<td>0.198444</td>
<td>1.215561</td>
</tr>
<tr>
<td></td>
<td>X3</td>
<td>-0.149870</td>
<td>1.083567</td>
</tr>
<tr>
<td></td>
<td>Y1</td>
<td>4.013309</td>
<td>9.723587</td>
</tr>
<tr>
<td></td>
<td>Y2</td>
<td>3.856637</td>
<td>9.545267</td>
</tr>
<tr>
<td></td>
<td>Y3</td>
<td>4.929953</td>
<td>8.636507</td>
</tr>
</tbody>
</table>

Figure 15.4 shows the iteration history of the objective function.
The following statements use the PRINT procedure to extract the first 10 observations from the output score table. The results are shown in Figure 15.5.

```sas
proc print noobs data=mycas.mtl_out(obs=10);
run;
```

### Figure 15.5 Multitask Learning Score Table

<table>
<thead>
<tr>
<th></th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>P_Y1</th>
<th>P_Y2</th>
<th>P_Y3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.91262</td>
<td>1.28876</td>
<td>1.56804</td>
<td>2.8956</td>
<td>2.6659</td>
<td>2.6911</td>
</tr>
<tr>
<td>3</td>
<td>0.29152</td>
<td>-0.17617</td>
<td>-0.17539</td>
<td>3.8039</td>
<td>3.7416</td>
<td>3.7899</td>
</tr>
<tr>
<td>6</td>
<td>0.42693</td>
<td>1.31982</td>
<td>-0.02247</td>
<td>-2.1305</td>
<td>-2.2776</td>
<td>-2.3144</td>
</tr>
<tr>
<td>9</td>
<td>1.56877</td>
<td>-0.90052</td>
<td>0.33429</td>
<td>20.2176</td>
<td>19.9016</td>
<td>20.1579</td>
</tr>
<tr>
<td>1</td>
<td>-0.69684</td>
<td>-1.48781</td>
<td>1.38306</td>
<td>0.2040</td>
<td>0.4411</td>
<td>0.4553</td>
</tr>
<tr>
<td>4</td>
<td>0.72148</td>
<td>0.47267</td>
<td>-0.52269</td>
<td>4.9828</td>
<td>4.7928</td>
<td>4.8505</td>
</tr>
<tr>
<td>7</td>
<td>-1.04654</td>
<td>-0.19002</td>
<td>-1.75929</td>
<td>-9.6041</td>
<td>-9.3888</td>
<td>-9.5053</td>
</tr>
<tr>
<td>2</td>
<td>0.97006</td>
<td>2.40560</td>
<td>-1.63855</td>
<td>-1.9203</td>
<td>-2.2583</td>
<td>-2.3010</td>
</tr>
<tr>
<td>5</td>
<td>1.56378</td>
<td>-1.04760</td>
<td>-0.29873</td>
<td>20.8952</td>
<td>20.5755</td>
<td>20.8414</td>
</tr>
<tr>
<td>8</td>
<td>0.15596</td>
<td>0.29972</td>
<td>-0.36697</td>
<td>0.1176</td>
<td>0.0649</td>
<td>0.0640</td>
</tr>
</tbody>
</table>

The following PROC PRINT statements display the estimated regression weights table, which is shown in Figure 15.6:

```sas
proc print noobs data=mycas.mtl_outW;
run;
```

### Figure 15.6 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 15.7.

```sas
proc print noobs data=mycas.mtl_outR;
run;
```

![Figure 15.7 Output Graph Table](image)

### Syntax: MTLEARN Procedure

The following statements are available in the MTLEARN procedure:

```sas
PROC MTLEARN <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

```sas
PROC MTLEARN <options>;
```

The PROC MTLEARN statement invokes the procedure. Table 15.1 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></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the input data table</td>
</tr>
<tr>
<td>GRAPHTABLE=</td>
<td>Specifies the user-defined graph table</td>
</tr>
<tr>
<td><strong>Multitask Learning Options</strong></td>
<td></td>
</tr>
<tr>
<td>GRAPHTYPE=</td>
<td>Specifies the type of graph table</td>
</tr>
<tr>
<td>MAXITERS=</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>REGL1=</td>
<td>Specifies the $\ell_1$ (LASSO) penalization weight</td>
</tr>
</tbody>
</table>
Table 15.1 continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGL2=</td>
<td>Specifies the $\ell_2$ graph penalization weight</td>
</tr>
<tr>
<td>SEED=</td>
<td>Specifies the seed to use for pseudorandom number generation</td>
</tr>
<tr>
<td>TOLERANCE=</td>
<td>Specifies the optimization tolerance (absolute $\ell_2$ difference of solution) as a stopping criterion</td>
</tr>
</tbody>
</table>

**Output Data Table Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAPHOUT=</td>
<td>Specifies the output data table in which to save the graph table</td>
</tr>
<tr>
<td>MODELOUT=</td>
<td>Specifies the output data table in which to save the estimated multitask 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 288.

*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 288.

**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 288.

**GRAPHTYPE=** CLUSTER | CUSTOM | FUSE | INDEP

specifies the type of graph table.

You can specify one of the following values:
Chapter 15: The MTLEARN Procedure

**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.

**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 288.

**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.

**REGL2=** *number*

specifies the $\ell_2$ graph penalization weight, where *number* is greater than or equal to 0. The default value is 0.01.

**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.
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 304. 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 304. 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**

```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:

**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 288.

- **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**

```plaintext
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 288.

---

**TARGET Statement**

```plaintext
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,

$$\hat{W} = \arg \min_W \frac{1}{2} \sum_{i=1}^{K} \|X_i W_i - y_i\|_2^2 + \frac{1}{2} \rho \ell_2 \|R^T W\|_F^2 + \rho \ell_1 \|W\|_1$$

where $R$ is the graph table (matrix) that represents the relationships between $K$ tasks. Note that the specific form of $R$ can be generated in various ways. One simple way is to model the relationships among tasks by using an undirected graph, where each task is a node, and the two nodes are connected if the two tasks are related. To encode the connections, let

$$\mathcal{E} = \{ e_k | k = 1, \ldots, n_e \}$$

be the set of edges in the graph, where $n_e = |\mathcal{E}|$ is the total number of edges in the graph. Define a matrix $R \in \mathbb{R}^{n_e \times K}$

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 15.8 shows an example of a cluster graph structure of five tasks, whose corresponding graph table matrix is

\[
R = \begin{bmatrix}
1 & -\frac{1}{3} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\
-\frac{1}{3} & 1 & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\
-\frac{1}{4} & -\frac{1}{4} & 1 & -\frac{1}{4} & -\frac{1}{4} \\
-\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & 1 & -\frac{1}{4} \\
-\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & 1 \\
\end{bmatrix}
\]

The cluster graph type is best used when all tasks are similarly related to one another.

**Figure 15.8** 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 15.9 shows an
example of an independent graph structure of five tasks, whose corresponding graph table matrix is

\[ R = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
\end{bmatrix} \]

**Figure 15.9** Independent Graph Structure with Five Targets (Tasks 1 to 5)

The independent graph type is best used when all tasks are independent of one another. It can also be used for elastic net regression problems.

- When **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 15.10** shows an example of a fuse graph structure of five tasks, whose corresponding graph table matrix is

\[ R = \begin{bmatrix}
1 & -1 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 1 & -1 \\
\end{bmatrix} \]
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 15.11 shows an example of a custom graph. Using the definition of $R$ near the beginning of the section “Details: MTLEARN Procedure” on page 298, the corresponding graph table matrix is

$$ R = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix} $$
Chapter 15: The MTLEARN Procedure

Figure 15.11 Example of a Custom Graph Structure with Five Targets (Tasks 1 to 5)

The following DATA step creates the custom graph table shown in Figure 15.8. This data table contains six variables: id is the row ID variable, and Task1, Task2, Task3, Task4, and Task5 are the target variables.

```
data customR;
  input id Task1 Task2 Task3 Task4 Task5;
cards;
  0  1 -1  0  0  0  
  1  0  0  1 -1  0  
  2  0  0  1  0 -1  
  3  0  0  0  1 -1  ;
run;
```

The following statements display the generated graph table, which is shown in Output 15.12.

```
proc print noobs data=customR;
  run;
```

Figure 15.12 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, \( \ell_1 \) penalization weight, \( \ell_2 \) 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 \textit{table-spec-list} in the \texttt{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 15.2.

<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 variables</td>
<td>INPUT and TARGET LEVEL=INTERVAL</td>
<td></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 303</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.
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

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 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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:

```sas
data turbines;
  input time Turbine1 Turbine2 Turbine3 Turbine4;
datalines;
  1     3241.250445 4079.490559 3505.854227 3392.565663
  2     2395.881828 3410.130913 2783.411447 2694.114311
  3     3628.878452 4543.677282 3818.729737 3701.380754
  4     2823.608204 3806.751908 3221.480613 3029.188791
  5     3743.196614 4632.571843 3859.732421 3817.242149
  6     1807.899427 2536.174173 1698.306145 1619.159414
  7     2240.006055 3271.723656 2477.616826 2210.066151
... more lines ...
```

The following statements plot the amount of energy that each turbine produces over time. Output 16.1 shows the resulting plot.

```sas
proc sgplot data =turbines;
  series x= time y= Turbine1/legendlabel = "Turbine1" lineattrs = (thickness = 0.1);
  series x= time y= Turbine2/legendlabel = "Turbine2" lineattrs = (thickness = 0.1);
  series x= time y= Turbine3/legendlabel = "Turbine3" lineattrs = (thickness = 0.1);
  series x= time y= Turbine4/legendlabel = "Turbine4" lineattrs = (thickness = 0.1);
  yaxis label="Turbine";
run;
```
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.

```latex
data mycas.turbines;
set turbines;
run;
```

The following statements run PROC MWPCA:

```latex
proc mw pca data=mycas.turbines windowsize=200 stepsize=1;
id time;
output out=mycas.windowpcs standardpc;
run;
```

Note that the STANDARDPC option is used in the OUTPUT statement. Using this option is recommended.

Figure 16.2 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 procedure starts with the first 200 observations in the data table, calculates the principal components, moves
ahead one observation and uses observations 2–201 to calculate the principal components, and continues in this manner until the entire data table has been used.

In the “Dimensions” table, you can see the number of observations and variables in the input table. The MWPCA 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 RPCA/PCA algorithm ran successfully for all windows.

![Figure 16.2 ODS Tables](image)

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 16.3 shows the resulting plot.

```
proc sgplot data =mycas.windowpcs;
  series x= window_id y= Turbine1/legendlabel = "Turbine1" lineattrs = (thickness = 1);
  series x= window_id y= Turbine2/legendlabel = "Turbine2" lineattrs = (thickness = 1);
  series x= window_id y= Turbine3/legendlabel = "Turbine3" lineattrs = (thickness = 1);
  series x= window_id y= Turbine4/legendlabel = "Turbine4" lineattrs = (thickness = 1);
  yaxis label="First Principal Component";
  xaxis label="Window";
run;
```

As you can see in Output 16.3, 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.
Chapter 16: The MWPCA Procedure

**Figure 16.3** First Principal Component

The following statements are available in the MWPCA procedure:

```plaintext
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

PROC MWPCA STEPSIZE=number WINDOWSIZE=number < options > ;

The PROC MWPCA statement invokes the procedure. Table 16.1 summarizes the options available in the PROC MWPCA statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Data Table Option</td>
<td>DATA= Specifies the input data table</td>
</tr>
<tr>
<td>Other Options</td>
<td>CENTER Centers the input data</td>
</tr>
<tr>
<td></td>
<td>CUMEIGPCTTOL= Specifies the significance level of eigenvalues that determine the rank of the low-rank matrix in each window</td>
</tr>
<tr>
<td></td>
<td>NTHREADS= Specifies the maximum number of threads to use on each computation node</td>
</tr>
<tr>
<td></td>
<td>ROBUST Uses robust principal component analysis on each window</td>
</tr>
<tr>
<td></td>
<td>STEPSIZE= Specifies the step size</td>
</tr>
<tr>
<td></td>
<td>WINDOWSIZE= 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 317.

**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
Chapter 16: The MWPCA Procedure

statement. For more information about CAS libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 306.

Data Table specifies the name of the input data table.

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.

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 18, “The RPCA Procedure.” If you do not specify this option, the regular PCA method is used on each window.

DISPLAY Statement

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 318. 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, 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 (/):

DISPLAY < table-list > < / options > ;
**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**

```sql
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 318. 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 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:

```
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 306.
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.

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 $P^S_i$ is

$$P^S_{ij} = \frac{P_{ij} \times \sigma_i}{(WTW)_{jj}}$$

where $\sigma_i$ 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 306.

### 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.

LAMBDALAMBA=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).

LAMBDWEIGHT=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 18, “The RPCA Procedure.”

By default, LAMBDWEIGHT=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 368. 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}.

SVD Statement

    SVD <options> ;

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 | RANDOM
    specifies the type of the SVD solver. You can specify the following values:

    EIGEN       uses the eigenvalue decomposition method.
    RANDOM      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.
Chapter 16: The MWPCA Procedure

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 Not Successful in One or More Windows.”

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 16.2.

<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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 in Chapter 3, “Shared Concepts.”

Example 16.1: 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.
Example 16.1: Anomaly Detection of Parking Lot Floodlights

```plaintext
data mycas.parkinglotlights;
  input time e49548 e49549 e49551 e49552 e49553 e49554;
datalines;
  ... more lines ...
```

Figure 16.4 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 16.4**  Energy Consumption (Kilowatts) of Floodlights, First 16 Days

![Energy Consumption Graph](image)

Figure 16.5 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.

![Energy Consumption Graph](image)
The following code uses PROC MWPCA to illustrate how moving-windows principal component analysis can help determine when the lights begin to fail:

```
proc MWPCA data=mycas.parkinglotlights windowsize = 240 stepsize = 1;
  id time;
  input e49548 e49549 e49551 e49552 e49553 e49554;
  output out=mycas.output standardpc;
run;
```

Figure 16.6 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 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 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 due to a light malfunction. Figure 16.6 shows how PROC MWPCA detects malfunctions in each light circuit by tracking the first principal components of their energy consumption.
Figure 16.6  First Principal Components over the Sliding Windows

References

Chapter 17
The NNET Procedure

Contents

Overview: NNET Procedure .................................................. 324
PROC NNET Features .......................................................... 324
Using CAS Sessions and CAS Engine Librefs .......................... 325
Getting Started: NNET Procedure ......................................... 325
Syntax: NNET Procedure ...................................................... 327
PROC NNET Statement ......................................................... 327
ARCHITECTURE Statement .................................................. 329
AUTOTUNE Statement ......................................................... 330
CODE Statement ............................................................... 334
CROSSVALIDATION Statement .............................................. 335
HIDDEN Statement ............................................................. 335
INPUT Statement ............................................................... 336
OPTIMIZATION Statement ................................................... 336
OUTPUT Statement ............................................................. 338
PARTITION Statement ......................................................... 339
TARGET Statement ............................................................. 340
TRAIN Statement ............................................................... 341
WEIGHT Statement ............................................................. 343
Details: NNET Procedure ..................................................... 344
Computational Method ......................................................... 344
Hyperparameter Tuning ....................................................... 344
k-fold Cross Validation ...................................................... 345
Determining the Number of Parallel Evaluations ..................... 345
Autoencoder ................................................................. 346
Displayed Output ............................................................. 346
Iteration History ............................................................... 346
Convergence Status .......................................................... 346
Model Information ............................................................ 346
Score Information ............................................................. 346
Tuner Information ............................................................. 347
Tuner Results ................................................................. 347
Evaluation History ........................................................... 347
Best Configuration ........................................................... 347
Tuner Summary ............................................................... 347
TunerTiming ................................................................. 347
ROCInfo ................................................................. 347
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 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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 17.1 shows the model information for the neural network.

![Figure 17.1 Model Information](image)

**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 17.2 shows the misclassification rate for the training sample.

![Figure 17.2 Score Information for Training Data](image)

<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 17.3 shows the misclassification rate for the validation sample.

### Figure 17.3 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>44</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>44</td>
</tr>
<tr>
<td>Misclassification Rate</td>
<td>0.0909</td>
</tr>
</tbody>
</table>

## Syntax: NNET Procedure

The following statements are available in the NNET procedure:

```plaintext
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

```plaintext
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 325.

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

**Note:** The data set options WHERE, TEMPNAMES, and TEMPEXPRESS are not supported.

INMODEL=\textit{CAS-libref.data-table} specifies a model for stand-alone scoring or coding. \textit{CAS-libref.data-table} is a two-level name, where \textit{CAS-libref} refers to the caslib and session identifier, and \textit{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 325.

MISSING=\textit{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:

- **MAX** substitutes missing values with the maximum value of the corresponding interval input variable.
- **MEAN** substitutes missing values with the mean value of the corresponding interval input variable.
- **MIN** substitutes missing values with the minimum value of the corresponding interval input variable.

By default, observations that have missing values are excluded from the analysis.

NTHREADS=\textit{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 \textit{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 17.3. 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.
**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 346):

**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.

---

**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.
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, 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 17.1 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 14 in Chapter 3, “Shared Concepts.” The TUNINGPARAMETERS= option is described in detail following Table 17.1.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>KFOLD=</td>
<td>Specifies the number of folds for k-fold cross validation</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 train</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>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 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 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 NUNITSi suboption.

NUNITSi (LB=number UB=number VALUES=value-list INIT=number EXCLUDE)
specifies tuning information for neurons in the ith hidden layer, where i is any integer 1–5, inclusive. By default, up to two hidden layers are tried during tuning. An NUNITSi 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\textsuperscript{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\textsuperscript{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 (3n, 100), where *n* is the number of model inputs.

**VALUES=value-list**
specifies a list of candidate numbers of neurons in the i\textsuperscript{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\textsuperscript{th} hidden layer.

By default, INIT=1.

**EXCLUDE**
excludes the number of neurons in the i\textsuperscript{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\textsubscript{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**

```sas
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**

omits 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**

```plaintext
CROSSVALIDATION < options > ;
```

The CROSSVALIDATION statement performs $k$-fold cross validation to find the average estimated validation error. You cannot specify the this statement if you specify either the AUTOTUNE statement or the PARTITION statement.

You can specify the following options:

**KFOILD=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**

```plaintext
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.

You can specify the following measures:

**EXP**

specifies the exponential function.

**IDENTITY**

specifies the identity function.

**LOGISTIC**

specifies the logistic function.
**Chapter 17: The NNET Procedure**

**RECTIFIER** specifies the rectifier activation function.
**SIN** specifies the sine function.
**TANH** specifies the hyperbolic tangent function.

By default, ACT=TANH.

**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**

```
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**

```
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:

**ALGORITHM=LBFGS | SGD \(<\text{sgd-options}>\) | HF**

**ALG=LBFGS | SGD \(<\text{sgd-options}>\) | HF**
specifies the optimization algorithm to use during training. You can specify the following optimization algorithms:

- **HF (Experimental)** specifies 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** specifies the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm.
- **SGD** specifies the stochastic gradient descent algorithm.

When **ALGORITHM=SGD**, 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 SGD progresses, causing smaller steps as SGD 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 SGD 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 SGD. New iterates for SGD are found by using
\[ w_{k+1} = w_k - \eta \left\| I_k \right\| \sum_{(x_i, y_i) \in I_k} \nabla L(w_k; x_i, y_i) \]
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 with SGD, especially with small data set, it is likely that the learning rate is set too large.

By default, **LEARNINGRATE=0.001**. The **number** must be a nonnegative double.
Chapter 17: The NNET Procedure

**MINIBATCHSIZE=number**

specifies the size of the minibatches used in SGD.

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 SGD algorithm. If `number` is less than or equal to 0 or not specified, a random seed from the computer clock is used.

**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. For LBFGS, the algorithm stops after MAXITER= iterations if convergence has not been achieved. For SGD, `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 > ;
```
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:

**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* 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 325.

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=**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 17.2 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= suboption 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=frac TEST=frac < 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 346.

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 325.

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 ALGORITHM=SGD and 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 ALGORITHM=SGD and 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 and CAS Engine Librefs” on page 325.

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.

WSEED=random-seed
SEED=random-seed
specifies the seed for generating initial random weights. If you do not specify a seed or you specify a value less than or equal to 0, the seed is generated from the computer clock. This option enables you to reproduce the same sample output.

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.
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), and 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
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: \( \text{NSUBSESSIONWORKERS} = 1 + \text{Number of Observations} \times \text{Number of Columns} / 50,000,000 \) (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 \(2 \times W/n\), 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.
The “Tuner Information” table contains the values of options used by tuner.

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.

The “Evaluation History” table contains values of all tuning parameters and the objective function for all model configurations evaluated by the tuner.

The “Best Configuration” table contains the values of hyperparameters and objective function for the best configuration from tuner.

The “Tuner Summary” table contains statistics for the tuning process.

The “TunerTiming” table contains the run time break down of different tasks during tuning.

The “ROCInfo” table contains the ROC curve information for the specified event in the nominal target. For more information, see Chapter 22, “The ASSESS Procedure” (SAS Visual Statistics: Procedures Guide).


The “Cross Validation Results” table contains the average error rate (misclassification error or average square error) of k-fold cross validation.

The “PredProbName” table contains the name mapping between the score variables and the corresponding events in the nominal target.

The “PredIntoName” table contains the name mapping between the predicted target variable and the original target variable when the target is nominal.
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 17.3.

<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>CrossRefValidateMLFitStat</td>
<td>Error metrics for ( k )-fold cross validation</td>
<td>CROSSVALIDATION</td>
<td>Default</td>
</tr>
<tr>
<td>CrossRefValidationResults</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>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>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>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>
</tbody>
</table>
### Table 17.3  continued

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<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>
<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>FitStat</td>
<td>Error metrics for supervised learning models</td>
<td>AUTOTUNE, TARGET</td>
<td>Default</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>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</td>
<td>PRINTTARGET, LEVEL=NOMINAL</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>
</tbody>
</table>

---

### Examples: 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 in Chapter 3, “Shared Concepts.”
Example 17.1: 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:

```plaintext
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.

```plaintext
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 17.1.1 shows the model information for the neural network.
Output 17.1.1 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 17.1.2 shows the misclassification rate of the training data.

Output 17.1.2 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 17.1.3 shows the misclassification rate of the validation data.

Output 17.1.3 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 17.1.4 shows the misclassification rate of the testing data.

Output 17.1.4 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 17.2: 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 325. 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.

```
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.

```
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 17.2.1 shows the setup values used by the tuner.

Output 17.2.1 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>Maximum Evaluations</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 17.2.2 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 17.2.2 Tuner Results

<table>
<thead>
<tr>
<th>Tuner Results</th>
<th>Default and Best Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation</td>
<td>Hidden Layers</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
</tr>
<tr>
<td>39</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>2</td>
</tr>
<tr>
<td>46</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
</tr>
</tbody>
</table>

Output 17.2.3 shows the best values of the tuning parameters from the tuning process.

### Output 17.2.3 Best Configuration

<table>
<thead>
<tr>
<th>Best Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation</td>
</tr>
<tr>
<td>Hidden Layers</td>
</tr>
<tr>
<td>Neurons in Hidden Layer 1</td>
</tr>
<tr>
<td>Neurons in Hidden Layer 2</td>
</tr>
<tr>
<td>L1 Regularization</td>
</tr>
<tr>
<td>L2 Regularization</td>
</tr>
<tr>
<td>Misclassification</td>
</tr>
</tbody>
</table>

Output 17.2.4 shows the tuner summary.

### Output 17.2.4 Tuner Summary

<table>
<thead>
<tr>
<th>Tuner Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Configuration Objective Value</td>
</tr>
<tr>
<td>Best Configuration Objective Value</td>
</tr>
<tr>
<td>Worst Configuration Objective Value</td>
</tr>
<tr>
<td>Initial Configuration Evaluation Time in Seconds</td>
</tr>
<tr>
<td>Best Configuration Evaluation Time in Seconds</td>
</tr>
<tr>
<td>Number of Improved Configurations</td>
</tr>
<tr>
<td>Number of Evaluated Configurations</td>
</tr>
<tr>
<td>Total Tuning Time in Seconds</td>
</tr>
<tr>
<td>Parallel Tuning Speedup</td>
</tr>
</tbody>
</table>
Output 17.2.5 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>Tuner Task Timing</th>
<th>tuner task timing</th>
<th>tuner task timing</th>
<th>tuner task timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuner</td>
<td>59.27</td>
<td>38.55</td>
<td></td>
</tr>
<tr>
<td>Total Objective Evaluations</td>
<td>78.11</td>
<td>50.81</td>
<td></td>
</tr>
<tr>
<td>Tuner</td>
<td>137.39</td>
<td>89.36</td>
<td></td>
</tr>
<tr>
<td>Total CPU Time</td>
<td>16.35</td>
<td>10.64</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tuner Task Timing</th>
<th>Tuner Task Timing</th>
<th>Tuner Task Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total CPU Time</td>
<td>153.74</td>
<td>100.00</td>
</tr>
</tbody>
</table>

**References**


Chapter 18
The RPCA Procedure

Contents

Overview: RPCA Procedure .......................................................... 355
PROC RPCA Features ................................................................. 356
Using CAS Sessions and CAS Engine Librefs ............................... 356
Getting Started: RPCA Procedure .............................................. 357
Syntax: RPCA Procedure ............................................................. 360
PROC RPCA Statement ............................................................. 360
CODE Statement ................................................................. 364
DISPLAY Statement ............................................................... 364
DISPLAYOUT Statement ......................................................... 365
ID Statement ................................................................. 366
INPUT Statement ............................................................... 366
OUTDECOMP Statement ......................................................... 366
SAVSTATE Statement ............................................................ 367
SVD Statement ............................................................... 368
Details: RPCA Procedure .......................................................... 368
Augmented Lagrange Multiplier Method ..................................... 369
Accelerated Proximal Gradient Method ..................................... 369
Scaling and Centering ............................................................. 370
Cumulative Eigenvalue Percentage Tolerance ............................. 370
Displayed Output ............................................................... 370
ODS Table Names ............................................................... 371
Output Data Tables ............................................................... 371
Examples: RPCA Procedure ...................................................... 372
Example 18.1: Analyzing the Energy Output of the SAS Solar Farm 372
References ............................................................... 377

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

**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:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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;
datalines;
  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
```
You can load rpcaData into your CAS session by naming your CAS engine libref in the first statement of the following DATA step:

```sas
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:

```sas
proc rpca data=mycas.rpcaData
decomp=svd
   outlowrank=mycas.lowrankmat
   outsparse=mycas.sparsemat;
   id index;
   outdecomp svdleft=mycas.svdleft
;  
run;
```
Getting Started: RPCA Procedure

svddiag=mycas.svddiag
svdright=mycas.svddiag;
run;

These statements produce the mycas.lowrankmat and mycas.sparsemat tables, which are the decompositions of mycas.rpcaData. They also produce the mycas.svleft, mycas.svddiag, and mycas.svdrdht tables, which are the decompositions of the mycas.lowrankmat table. The DECOMP= option produces the SVD decomposition of the low-rank matrix.

Figure 18.1 ODS Tables

<table>
<thead>
<tr>
<th>Data Source</th>
<th>RPCADATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPCA Method</td>
<td>Augmented Lagrange Multiplier</td>
</tr>
<tr>
<td>SVD Method</td>
<td>Eigenvalue Decomposition</td>
</tr>
<tr>
<td>Lambda</td>
<td>0.1414214</td>
</tr>
<tr>
<td>Lambda Weight</td>
<td>1</td>
</tr>
<tr>
<td>Cumulative Eigenvalue Percent Tolerance Specified</td>
<td>1</td>
</tr>
<tr>
<td>Cumulative Eigenvalue Percent Tolerance Applied</td>
<td>1</td>
</tr>
</tbody>
</table>

| Number of Observations | 50 |
| Number of Variables | 2 |
| Number of Observations Used | 50 |
| Number of Observations with Missing Values | 0 |

<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>
</tbody>
</table>

Figure 18.1 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 18.2 shows that the sparsity of the sparse matrix increases to 0.97 and the rank of the low-rank matrix increases to 2.

```
proc rpca data=mycas.rpcaData
   lambda=3.5
   sparse=mycas.sparsemat2;
   id index;
```
run;

proc print data=mycas.sparsemat2;
run;

Figure 18.2  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>
</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).

Syntax: RPCA Procedure

The following statements are available in the RPCA procedure:

```plaintext
PROC RPCA <options> ;
   ID variables ;
   INPUT variables ;
   SVD <options> ;
   OUTDECOMP <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 18.1 summarizes the options available in the PROC RPCA 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>
</tbody>
</table>
Table 18.1 continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>PREFIX=</td>
<td>Specifies a prefix for naming the principal components</td>
</tr>
<tr>
<td>SCALE</td>
<td>Scales the input data</td>
</tr>
<tr>
<td>TOLERANCE=</td>
<td>Specifies the convergence tolerance</td>
</tr>
<tr>
<td><strong>Decomposition Output Table Options</strong></td>
<td></td>
</tr>
<tr>
<td>OUTERROR=</td>
<td>Specifies the name of the error output table</td>
</tr>
<tr>
<td>OUTLOWRANK=</td>
<td>Specifies the name of the low-rank output table</td>
</tr>
<tr>
<td>OUTSPARSE=</td>
<td>Specifies the name of the sparse output table</td>
</tr>
<tr>
<td><strong>Simple Statistics Table Option</strong></td>
<td></td>
</tr>
<tr>
<td>COLSTATISTICS=</td>
<td>Specifies the name of the table that contains simple statistics for the variables in the input data set</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 370.

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

specifies the name of the output table to contain simple statistics for the variables of the input data table.

*caslib* is a two-level name, where *caslib* 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 356.

**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 370.
**Chapter 18: The RPCA Procedure**

**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* 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 356.

**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 \( \lambda_{weight} \) 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 \( \lambda_{weight} \) 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 368.

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.
PROC RPCA Statement

**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**.

**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 356.

**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 356.

**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 356.

**PREFIX=name**
specifies a prefix for naming the principal components. By default, the components are named Prin1, Prin2, . . . , Prinn. If you specify **PREFIX=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 **VALIDVARNAMES**= system option. For more information about the **VALIDVARNAMES**= 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 370.
TOLERANCE=number
    specifies the convergence criterion for the RPCA algorithm. The default value is $10^{-7}$.

---

**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:

**FILE=filename**
    specifies the name of the file to write the SAS score code to.

You can also specify the following option:

**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 371. 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 371. 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 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:

**SVDDIA< =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 356.
**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 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 356.
You can use the ASTORE procedure to specify the projection type. Specify the option RPCA_PROJECTION_TYPE in the SETOPTION statement of PROC ASTORE: the value 0 projects the scoring observations onto the principal component space, and the value 1 projects the scoring observations onto the low-rank subspace.

SVD Statement

```sas
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

```
m = \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:

\[
\begin{align*}
\text{minimize} \quad & l(L, S, Y) = ||L||_* + \lambda ||S||_1 + < Y, M - L - S > + \frac{\mu}{2} ||M - L - S||_F^2 \\
\end{align*}
\]

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).

### 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:

\[
\begin{align*}
\text{minimize} \quad & \mu ||L||_* + \mu \lambda ||S||_1 + \frac{1}{2} ||M - L - S||_F^2 \\
\end{align*}
\]

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.

---

1 The 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.

2 The singular value thresholding operator is based on the singular value decomposition.
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:

$$L(i, j) = L(i, j) * (\sigma_j) + \mu_j$$

$$S(i, j) = S(i, j) * (\sigma_j)$$

$$N(i, j) = N(i, j) * (\sigma_j)$$

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.

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.

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 18.2.

<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 18.3 lists details about these data tables.
Table 18.3 Output Data Tables Produced by PROC RPCA

<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>OUTERERROR</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>
<tr>
<td>SVDDIAI</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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 in Chapter 3, “Shared Concepts.”

Example 18.1: 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 goals of the example are to predict the daytime hourly energy output (KWH_Delivered) from the solar farm by using robust principal components and to detect observations that are acting as outliers by using the sparse matrix.

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.
Example 18.1: Analyzing the Energy Output of the SAS Solar Farm

```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

... 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 sampct=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:

```sas
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;
```
Chapter 18: The RPCA Procedure

```sas
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:

```sas
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';
run;

data mycas.EnergyforecastOnTestDataRPCA;
set mycas.rpcaprojectedtest;
    %inc 'rpcareg.sas';
run;
```

You can repeat the same steps by using standard PCA (with the PCA procedure) as well:

```sas
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:

```sas
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;
```

```sas
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:
Example 18.1: Analyzing the Energy Output of the SAS Solar Farm

```sas
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:

```sas
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.

Next, you can use the following statements to run PROC RPCA on the original solar farm data. Using the resulting sparse matrix, you can identify outliers in this high-dimensional data table.

```sas
proc rpca data=mycas.solarfarm outsparse=mycas.sparse method = APG scale center;
    id date hour;
    input &inputvars.;
run;
```
In the sparse matrix (not shown), there are many nonzero values. But not every nonzero value indicates an outlier. To identify observations that act as outliers, you can set any element of the sparse matrix that is within three standard deviations of its related column to zero, and count the number of nonzero columns for each observation in the resulting matrix. Note that the decision of how to extract the significant values from the sparse matrix is a judgment call.

The following code modifies the sparse matrix and obtains the frequency of nonzero variables for each row of the matrix:

```/*find the standard deviation of the columns of the sparse matrix*/
proc sql noprint;
    select "'"||STRIP(NAME)||"' " into:quotedinputvars separated by ', ' from varlist where NAME not in ("date","hour","KWH_Delivered");
quit;
proc cas;
    simple.summary /
        inputs={&quotedinputvars.}
        subset={"std"}
        table={name="sparse"}
        casout={name="sparse_summary",replace=true};
    run;
quit;
proc transpose data=mycas.sparse_summary out=mycas.std_values prefix=std_;
    id _column_;
    var _Std_;
run;
data _null_;
    call symput(
        'std_vars',
        strip(cat('std_',tranwrd("&inputvars.",',', std_'))));
run;
/*add a dummy variable to the sparse table to be able to merge with the sparse_summary table*/
data mycas.sparse;
    _name_='_Std_';
    set mycas.sparse;
run;
data mycas.sparse_modified;
    merge mycas.sparse mycas.std_values;
    by _name_;
    array inputvars[*] &inputvars.;
    array std_vars[*] &std_vars.;
    count=0;
    do v=1 to dim(inputvars);
        if (std_vars[v]) then do;
            if abs(inputvars[v]/std_vars[v]) <3 then inputvars[v]=0;
        end;
        count + (abs(inputvars[v]) > 0);
    end;
    drop _name_ &std_vars. v;
run;
proc sql;
    create table counts as
        select count as non_zero_var_counts,
            count(count) as frequency from mycas.sparse_modified
In Output 18.1.1, you can see that there are four observations that have 19 nonzero variables in the modified sparse matrix. These four observations are related to November 11–16, 2015. The solar farm maintenance records show that a row in one of the solar farm’s arrays was not tracking the sun. This anomaly was identified by using the sparse matrix.

You can use the PRINT procedure as follows to print the Counts data table:

```sas
proc print data=counts;
run;
```

**Output 18.1.1**  Frequency of Nonzero Variables in the Modified Sparse Matrix

<table>
<thead>
<tr>
<th>Obs</th>
<th>non_zero_var_counts</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2092</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>891</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>572</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>293</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>176</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>155</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>89</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>50</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>47</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>24</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>4</td>
</tr>
</tbody>
</table>

**References**


Chapter 19
The SEMISUPLEARN Procedure

Overview: SEMISUPLEARN Procedure

The SEMISUPLEARN procedure implements the graph-based semisupervised learning algorithm that relies on label spreading 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 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 11 in Chapter 3, “Shared Concepts.”
Getting Started: 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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);
   run;

   data mycas.label;
   set hmeq(obs=200);
   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;
   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 19.1 shows the number of unlabeled observations, number of labeled observations, number of levels for target variable, gamma value, and maximum number of iterations.
Syntax: SEMISUPLEARN Procedure

The following statements are available in the SEMISUPLEARN procedure:

```plaintext
PROC SEMISUPLEARN <options> ;
  DISPLAY <table-list> </options> ;
  DISPLAYOUT table-spec-list </options> ;
  INPUT variables / <LEVEL=INTERVAL> ;
  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 19.1 summarizes the options available in the PROC SEMISUPLEARN statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unlabeled Data Table Options</strong></td>
<td></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the unlabeled data table</td>
</tr>
<tr>
<td><strong>Labeled Data Table Options</strong></td>
<td></td>
</tr>
<tr>
<td>LABEL=</td>
<td>Specifies the labeled data table</td>
</tr>
<tr>
<td><strong>Semisupervised Learning Algorithm Options</strong></td>
<td></td>
</tr>
<tr>
<td>GAMMA=</td>
<td>Specifies the inverse of the variable when computing the similarity measure by using the Gaussian kernel</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>Specifies the maximum number of iterations</td>
</tr>
</tbody>
</table>
You can specify the following options:

**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 380.

- **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 380.

**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 an integer from 1 to 64, inclusive. The default value is the maximum number of available threads per computer.
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 387. 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 387. 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 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.

OUTPUT Statement

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 380.

define data-table

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**

**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.

---

**Details: SEMISUPLEARN Procedure**

The distance metric in the SEMISUPLEARN procedure is defined as

\[
D_{p,q} = \exp(-\gamma \sum_{k=1}^{K} (p_k - q_k)^2)
\]

where \(D\) is the Gaussian kernel distance metric, \(K\) 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 \(p\) and \(q\) represent the index of the data in the combined input from both the unlabeled data and the labeled data.
Because in most cases the labeled data are relatively small, the 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, and maximum number of iterations in the PROC SEMISUPLEARN statement.

**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 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 names of each table and a short description of its 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>ModelInfo</td>
<td>Model information</td>
<td>PROC</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEMISUPLEARN</td>
<td></td>
</tr>
<tr>
<td>OutputCASTables</td>
<td>See the section “OutputCasTables Table” on page 387</td>
<td>PROC</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEMISUPLEARN</td>
<td></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 19.3.

<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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 in Chapter 3, “Shared Concepts.”

Example 19.1: 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 Existcr.

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);
   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:
Example 19.1: German Credit Benchmark Data

```
proc semisuplearn data=mycas.dmagecrunlabel label=mycas.dmagecrlabel;
  input duration amount installp resident existcr;
  output out = mycas.out;
  target good_bad;
run;
```

Output 19.1.1 shows the number of unlabeled observations, number of labeled observations, number of levels for the target variable, gamma value, and maximum number of iterations.

**Output 19.1.1** 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>
</tbody>
</table>
## Chapter 20

### The SVDD Procedure

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: SVDD Procedure</td>
<td>392</td>
</tr>
<tr>
<td>PROC SVDD Features</td>
<td>392</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>393</td>
</tr>
<tr>
<td>Getting Started: SVDD Procedure</td>
<td>393</td>
</tr>
<tr>
<td>Syntax: SVDD Procedure</td>
<td>398</td>
</tr>
<tr>
<td>PROC SVDD Statement</td>
<td>398</td>
</tr>
<tr>
<td>CODE Statement</td>
<td>400</td>
</tr>
<tr>
<td>ID Statement</td>
<td>400</td>
</tr>
<tr>
<td>INPUT Statement</td>
<td>400</td>
</tr>
<tr>
<td>KERNEL Statement</td>
<td>401</td>
</tr>
<tr>
<td>SAVESTATE Statement</td>
<td>402</td>
</tr>
<tr>
<td>SOLVER Statement</td>
<td>402</td>
</tr>
<tr>
<td>WEIGHT Statement</td>
<td>403</td>
</tr>
<tr>
<td>Details: SVDD Procedure</td>
<td>403</td>
</tr>
<tr>
<td>Mathematical Formulation</td>
<td>404</td>
</tr>
<tr>
<td>Radial Basis Function (RBF) Bandwidth Parameter</td>
<td>407</td>
</tr>
<tr>
<td>Fraction Outlier Parameter</td>
<td>410</td>
</tr>
<tr>
<td>SVDD Formulation with a Weight Variable</td>
<td>411</td>
</tr>
<tr>
<td>Comparison with One-Class Support Vector Machines</td>
<td>411</td>
</tr>
<tr>
<td>SVDD Solvers</td>
<td>412</td>
</tr>
<tr>
<td>Processing Mode and SVDD Results</td>
<td>413</td>
</tr>
<tr>
<td>Scoring Multiple SVDD Models with the ASTORE Procedure</td>
<td>413</td>
</tr>
<tr>
<td>ODS Table Names</td>
<td>415</td>
</tr>
<tr>
<td>Examples: SVDD Procedure</td>
<td>415</td>
</tr>
<tr>
<td>Example 20.1: Analysis of Aircraft Engine Degradation</td>
<td>415</td>
</tr>
<tr>
<td>Example 20.2: Automatic RBF Kernel Bandwidth Parameter Value Selection</td>
<td>420</td>
</tr>
<tr>
<td>Example 20.3: Stochastic Subset Solver</td>
<td>428</td>
</tr>
<tr>
<td>Example 20.4: Analysis of Tennessee Eastman Process Data Using Multiple Bandwidth Values and Consensus-Based Classification</td>
<td>435</td>
</tr>
<tr>
<td>References</td>
<td>441</td>
</tr>
</tbody>
</table>
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 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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 y5 (inlet water temperature 1) and y4 (pressure drop 6) to perform an SVDD training. Variables y4 and y5 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
    ... more lines ...

The following code standardizes variables y4 and y5:

```sas
proc sql noprint;
    select min(y4), min(y5), max(y4), max(y5)
    into :miny4, :miny5, :maxy4, :maxy5
    from Refrigerant;
quit;

```sas
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 20.1 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 393, 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 20.2 through Figure 20.5.

```plaintext
proc svdd data=mycas.Refrigerant;
  id y4 y5;
  input 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 20.2 summarizes the key options and the input data variables.
The “Training Results” table in Figure 20.3 shows the number of support vectors and the $R^2$ threshold value.

**Figure 20.3** 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 20.4 displays the number of observations in the training data set.

**Figure 20.4** Refrigerant Analysis Training Observations

| Number of Observations Read | 360 |
| Number of Observations Used | 360 |

The “Optimization Summary” table in Figure 20.5 shows whether the solution is optimal, the number of iterations that were required, and the objective function value.

**Figure 20.5** 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 $200 \times 200$ 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 20.6 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 20.1 and the training data in Figure 20.6 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 20.6 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.
Figure 20.6 Scoring Results Scatter Plot

Syntax: SVDD Procedure

The following statements are available in the SVDD procedure:

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.

PROC SVDD Statement

PROC SVDD < options > ;

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 393.

*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**

**FRACTION=number**

**FRAC=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=CAS-libref.data-table**
**SVDD Procedure**

---

**SV=** 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. **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 393.

---

**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.

**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 401.

---

**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 (DELTA=number)
RBF / BW=MEAN2

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 409.
- 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 409.
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 393.

SOLVER Statement

```
SOLVER solver-type /options ;
```

The SOLVER statement specifies the type of optimization solver for SVDD training. You can specify the following `solver-types`:

```
ACTSET
```

uses the active-set solver for SVDD training. For more information about this solver, see the section “Active-Set Solver” on page 412.

```
STOCHS
```

uses a stochastic subset solver for SVDD training. For more information about this solver, see the section “Stochastic Subset Solver” on page 413.

You can specify the following `options` after a slash (/) for either `solver-type`:

```
MAXITER=number
```

specifies the maximum number of iterations 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, `MAXITER=25` for the active-set solver and `MAXITER=200` for the stochastic subset solver.

```
MAXTIME=t
```

specifies an upper limit of `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 `t` can be any positive number; the default value is 36,000 seconds, or 10 hours.

```
STOL=number
```

specifies the tolerance value for the solver, where `number` must be a real number greater than 0. To obtain a solution more quickly, specify a higher value for `number`. By default, `STOL=1E–4`.

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.

MAXSV=number
specifies the maximum number of support vectors, where number must be an integer greater than 0. To obtain a solution more quickly, specify a lower value for number. By default, MAXSV=3500.

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 405). 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

$$\|x_i - a\|^2 \leq R^2 + \xi_i \text{ for all } i = 1, \ldots, n$$

$$\xi_i \geq 0 \text{ for all } i = 1, \ldots, n$$

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

$$\max \ \sum_{i=1}^{n} \alpha_i (x_i \cdot x_j) - \sum_{i,j} \alpha_i \alpha_j (x_i \cdot x_j)$$

subject to

$$\sum_{i=1}^{n} \alpha_i = 1$$

$$0 \leq \alpha_i \leq C \text{ for all } i = 1, \ldots, n$$

where $\alpha_i \in \mathbb{R}$ are the Lagrange multipliers and $C = \frac{1}{nf}$ 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 404) 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 405, 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 20.7 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 20.8 illustrates the effect of $s$ on the 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), or the modified mean criterion method (Liao et al. 2018) 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 and the modified mean criterion method by specifying BW=MEAN2, 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

$$\sqrt{\frac{\tilde{D}^2}{\ln \frac{n-1}{\delta^2}}}$$

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

$$\sqrt{\frac{\tilde{D}^2}{\ln \frac{n-1}{\delta^2}}}$$

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).
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 404, 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 20.9 illustrates the effect of $f$ on the data boundary.

**Figure 20.9  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

$$K(x, y) = \exp \left(-\frac{\|x - y\|^2}{2s^2}\right)$$

where $x$ and $y$ are any two observations from the training data and $s$ is the bandwidth parameter; $\alpha_i \in \mathbb{R}$ are the Lagrange multipliers, for all $i = 1, \ldots, n$; and $w_i$ are the weight values, for all $i = 1, \ldots, n$.

Using a weight variable in SVDD formulation results in observations with higher weights having higher upper bounds on the value of the Lagrange multipliers. Such observations are more likely to end up with a positive nonzero value of their Lagrange multipliers and hence are designated as support vectors.

The WEIGHT statement has no impact on the results when the value of the fraction outlier parameter $f$ is less than $\frac{\min w_i}{\sum_{i=1}^{n} w_i}$ for all $i = 1, \ldots, n$.

---

**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

$$\sum_{i=1}^{n} \alpha_i = 1$$
$$0 \leq \alpha_i \leq \frac{1}{v n} \text{ for all } i = 1, \ldots, n$$

where \( \alpha_i \in \mathbb{R} \) are the Lagrange multipliers, \( v \) 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 405 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. (2018).

**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:

```sas
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.

```sas
proc astore;
   score data=mycas.scoreds
      out=mycas.score_results
      rstore=mycas.state;
quit;
```

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.

```sas
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.

```sas
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 20.1.

<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>ModellInfo</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>
<tr>
<td>OptSummary</td>
<td>Statistics related to optimization solver</td>
<td>PROC SVDD</td>
<td>Default</td>
</tr>
<tr>
<td>StatusTable</td>
<td>Status of each training model</td>
<td>PROC SVDD</td>
<td>Default</td>
</tr>
<tr>
<td>TrainingResults</td>
<td>Displays the training results</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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 in Chapter 3, “Shared Concepts.”

Example 20.1: 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:

```sas
data eid(keep=engine);
  set engine;
  by engine;
  if first.engine then output;
rung;

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;
rung;

%macro a;
data _null_;
  set reid;
  call symput('e'||left(_n_), engine);
rung;
%do i=1 %to 5;
  %if &i=1 %then %do;
    %let st=&&e&i;
  %end;
  %else %do;
    %let st=&st &&e&i;
  %end;
%end;
%end;
```
Example 20.1: Analysis of Aircraft Engine Degradation

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;
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;
%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 393, 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 20.1.1 through Figure 20.1.2.
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 SAVE STATE statement creates an analytic store for the model and saves it as a binary object in the mycas.state data table. You can use the analytic store later in the ASTORE procedure to score new data. The “Model Information” table in Output 20.1.1 summarizes the key options and input data variables.

**Output 20.1.1**  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 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 20.1.2 shows the number of support vectors and the threshold $R^2$ value.

**Output 20.1.2**  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 ($C_0$) Value</td>
</tr>
<tr>
<td>Run Time (Seconds)</td>
</tr>
</tbody>
</table>

**Output 20.1.3** displays the number of observations in the training data set.

**Output 20.1.3**  Aircraft Engine Degradation Analysis Training Observations

<table>
<thead>
<tr>
<th>Number of Observations Read</th>
<th>225</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Used</td>
<td>225</td>
</tr>
</tbody>
</table>

The “Optimization Summary” table in Output 20.1.4 shows whether the solution is optimal, the number of iterations that were required and the objective function value.
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.

```sql
proc astore;
  score data=mycas.score
  out=mycas.test_out
  rstore=mycas.state;
quit;
```

Output 20.1.5 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 20.1.5 shows that the $\text{dist}^2(z)$ value serves as a good proxy for engine degradation.

![Output 20.1.5 Scoring Results Scatter Plot](image-url)
Example 20.2: 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. 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 393, 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 mean criterion, 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 y5 (inlet water temperature 1) and y4 (pressure drop 6) to perform an SVDD training. The variables y4 and y5 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):

data refrigerant;
  infile datalines delimiter=',';
  input y4 y5 ;
  label y4="Pressure Drop 6" y5="Inlet Water Temperature 1";
  datalines;
  2.0562,25.0904
  2.04147,25.1676
  2.04922,25.6362
  ...
  ...
... more lines ...

The following code standardizes the variables y4 and y5:

  proc sql noprint;
    select min(y4), min(y5), max(y4), max(y5)
    into :miny4, :miny5, :maxy4, :maxy5
    from Refrigerant;
  quit;

  data Refrigerant;
    set Refrigerant;
    y4=(y4-&miny4)/%sysevalf(&maxy4-&miny4);
    y5=(y5-&miny5)/%sysevalf(&maxy5-&miny5);
  run;

The following PROC SGPLOT statements plot inlet water temperature 1 against pressure drop 6:

  proc sgplot data=Refrigerant;
    title "Refrigerant Analysis";
    title1 "Scatter plot of inlet water temperature against pressure drop";
    scatter x=y4 y=y5/markerattrs=(size=3 symbol=circlefilled);
  run;

Output 20.2.1 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:

``` SAS
   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 393, 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 20.2.2 through Output 20.2.6.

``` SAS
   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 20.2.2 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 20.2.2 Refrigerant Analysis Model Information

<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: 0.1262714382</td>
</tr>
<tr>
<td>Bandwidth Relative Scale: 1</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: 2</td>
</tr>
<tr>
<td>Number of Nominal Variables: 0</td>
</tr>
</tbody>
</table>

The “Training Results” table in Output 20.2.3 shows the number of support vectors and the $R^2$ threshold value.

Output 20.2.3 Refrigerant Analysis Training Results

<table>
<thead>
<tr>
<th>Training Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors: 7</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary: 7</td>
</tr>
<tr>
<td>Number of Outliers: 0</td>
</tr>
<tr>
<td>Number of Dropped Observations: 0</td>
</tr>
<tr>
<td>Threshold R Square Value: 0.76304</td>
</tr>
<tr>
<td>Constant (C_r) Value: 0.23696</td>
</tr>
<tr>
<td>Run Time (Seconds): 0.04022</td>
</tr>
</tbody>
</table>

Output 20.2.4 displays the number of observations in the training data set.

Output 20.2.4 Refrigerant Analysis Training Observations

| Number of Observations Read: 360      |
| Number of Observations Used: 360      |

The “Optimization Summary” table in Output 20.2.5 shows whether the solution is optimal, the number of iterations that were required, and the objective function value.

Output 20.2.5 Refrigerant Analysis Optimization Summary

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations: 1</td>
</tr>
<tr>
<td>Objective Value: 0.2369581441</td>
</tr>
<tr>
<td>Infeasibility: 1.897082E-11</td>
</tr>
<tr>
<td>Optimization Status: Optimal</td>
</tr>
<tr>
<td>Degenerate: No</td>
</tr>
</tbody>
</table>

The “Histogram” table in Output 20.2.6 shows the frequency table of _SVDDDISTANCE_ values of observations in the mycas.Refrigerant training data set.
To evaluate the quality of the training results, scoring is performed on a 200 × 200 data grid. The following DATA step creates a data set named *scoreds* to contain the scoring observations:

```
data scoreds;
  do i=0 to 1 by 0.005;
    do j=0 to 1 by 0.005;
      output;
    end;
  end;
rename i=y5 j=y4;
run;
```

You can load the *scoreds* data set into your CAS session by specifying your CAS engine libref in the first statement in the following DATA step:

```
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:

```
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:

```
data work.score_results;
  set mycas.score_results;
run;
```

The following PROC SGPLOT code plots the scoring results: 

---

**Output 20.2.6** 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>0.725969</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>
Chapter 20: The SVDD Procedure

Proc sgplot data=score_results(where=(_svddscore_ in (-1,1)));
  styleattrs datacontrastcolors=(ligr black) datasymbols=(circlefilled);
  title "Refrigerant Analysis";
  title1 "Scatter plot of inlet water temperature against pressure drop";
  title2 "Scoring Results";
  scatter x=y4 y=y5/group=_svddscore_ markerattrs=(size=3);
run;

Output 20.2.7 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 20.2.7 and the training data in Output 20.2.1 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 393. Despite these shortcomings, the mean criterion provides a reasonable approximation of the training data. The scatter plot of the scoring data in Output 20.2.7 indicates that PROC SVDD was able to correctly identify a majority of the outliers.

Output 20.2.7  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 20.2.8 through Output 20.2.12. The SVDD algorithm uses the bandwidth value that is computed using the modified mean criterion.
Example 20.2: Automatic RBF Kernel Bandwidth Parameter Value Selection

```
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 20.2.8 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 20.2.8  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 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 20.2.9 shows the number of support vectors and the $R^2$ threshold value.

**Output 20.2.9  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>
</tbody>
</table>

Output 20.2.10 displays the number of observations in the training data table.

**Output 20.2.10  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 20.2.11 shows whether the solution is optimal, the number of iterations that were required, and the objective function value.

### Output 20.2.11 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 20.2.12 shows the frequency table of _SVDDDISTANCE_ values of observations in the mycas.Refrigerant training data table.

### Output 20.2.12 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 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.

```sas
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";  
scatter x=y4 y=y5/group=_svddscore_ markerattrs=(size=3);  
run;
```

Output 20.2.13 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 mean criterion was used to compute the RBF kernel bandwidth parameter value. Comparison between Output 20.2.13 and the training data in Output 20.2.1 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 393. Comparing the scoring results in Output 20.2.13 and Output 20.2.7 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 20.2.7 indicates that PROC SVDD was able to correctly identify a majority of the outliers.

**Output 20.2.13** Scoring Results Scatter Plot: Bandwidth Computation Using the Modified Mean Criterion
Example 20.3: 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 413. 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;
        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:

```sas
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 20.3.1 shows the results.
Example 20.3: Stochastic Subset Solver

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:

```plaintext
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 393, 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 20.3.2 through Output 20.3.5:

```plaintext
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 20.3.2 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.
Chapter 20: The SVDD Procedure

Output 20.3.2 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 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 20.3.3 shows the number of support vectors and the $R^2$ threshold value.

Output 20.3.3 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>
</tbody>
</table>

Output 20.3.4 displays the number of observations in the training data set.

Output 20.3.4 Two Donut Training Observations: Active-Set Solver

| Number of Observations Read       | 4000000     |
| Number of Observations Used       | 4000000     |

The “Optimization Summary” table in Output 20.3.5 shows whether the solution is optimal, the number of iterations that were required, and the objective function value.

Output 20.3.5 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 20.3.6 through Output 20.3.9.
Example 20.3: Stochastic Subset Solver

```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 20.3.6 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 20.3.6  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 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 20.3.7 shows the number of support vectors and the $R^2$ threshold value.

```
Output 20.3.7  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 Square Value</td>
</tr>
<tr>
<td>Constant (C_r) Value</td>
</tr>
<tr>
<td>Run Time (Seconds)</td>
</tr>
</tbody>
</table>
```

Output 20.3.8 displays the number of observations in the training data set.
Chapter 20: The SVDD Procedure

Output 20.3.8 Two Donut Training Observations: Stochastic Subset Solver

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
<td>4000000</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>4000000</td>
</tr>
</tbody>
</table>

The “Optimization Summary” table in Output 20.3.9 shows whether the solution is optimal, the number of iterations that were required, and the objective function value.

Output 20.3.9 Two Donut Optimization Summary: Stochastic Subset Solver

<table>
<thead>
<tr>
<th>Optimization Summary</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</td>
<td>200</td>
</tr>
<tr>
<td>Objective Value</td>
<td>0.0915147103</td>
</tr>
<tr>
<td>Infeasibility</td>
<td>2.740319E-11</td>
</tr>
<tr>
<td>Optimization Status</td>
<td>Maximum number of iterations reached.</td>
</tr>
<tr>
<td>Degenerate</td>
<td>No</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 20.3.2 and Output 20.3.6. 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 $200 \times 200$ 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.

```
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:

```sas
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:

```sas
data work.scoreact;
    set mycas.scoreact;
run;
```

The following code plots the scoring results, as shown in Output 20.3.10:

```sas
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;
```

**Output 20.3.10** Scoring Results Scatter Plot: Active-Set Solver

The following code scores the mycas.scoreds data table by using the score analytic store that was created using the STOCHS solver during training:
The following code copies the scorestochs data table into the Work library:

```sql
data Work.scorestochs;
  set mycas.scorestochs;
run;
```

The following code plots the scoring results:

```sql
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;
```

**Output 20.3.11** Scoring Results Scatter Plot: Stochastic Subset Solver

Output 20.3.10 and Output 20.3.11 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 20.3.10 and Output 20.3.11 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 20.4: 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, \( x_1 \text{–} x_{41} \); 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 20.2 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>
<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):

```r
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 393, 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 20.4.1 through Output 20.4.4:

```plaintext
data 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 20.4.1 displays the number of observations in the training data table.

```
Output 20.4.1  TE Process Analysis Training Observations

<table>
<thead>
<tr>
<th>Number of Observations Read</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Used</td>
<td>800</td>
</tr>
</tbody>
</table>
```
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 20.4.2 summarizes the key options and the input data variables for the SVDD model that is trained using the option BW=5.

### Output 20.4.2
#### TE Process Analysis Model Information (MODELID=1, BW=5)

<table>
<thead>
<tr>
<th>Model Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
<td>Active set</td>
</tr>
<tr>
<td>Kernel Type</td>
<td>RBF</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth</td>
<td>5</td>
</tr>
<tr>
<td>Bandwidth Relative Scale</td>
<td>0.7242664168</td>
</tr>
<tr>
<td>Expected Outlier Fraction</td>
<td>1E-6</td>
</tr>
<tr>
<td>Optimization Tolerance</td>
<td>0.0001</td>
</tr>
<tr>
<td>Number of Interval Variables</td>
<td>41</td>
</tr>
<tr>
<td>Number of Nominal Variables</td>
<td>0</td>
</tr>
</tbody>
</table>

The “Training Results” table in Output 20.4.3 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 20.4.3
#### TE Process Analysis Training Results (MODELID=1, BW=5)

<table>
<thead>
<tr>
<th>Training Results</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors</td>
<td>326</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary</td>
<td>326</td>
</tr>
<tr>
<td>Number of Outliers</td>
<td>3</td>
</tr>
<tr>
<td>Number of Dropped Observations</td>
<td>0</td>
</tr>
<tr>
<td>Threshold R Square Value</td>
<td>0.98803</td>
</tr>
<tr>
<td>Constant $(C_r)$ Value</td>
<td>0.01208</td>
</tr>
<tr>
<td>Run Time (Seconds)</td>
<td>0.33465</td>
</tr>
</tbody>
</table>

The “Optimization Summary” table in Output 20.4.4 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 20.4.4
#### TE Process Analysis Optimization Summary (MODELID=1, BW=5)

<table>
<thead>
<tr>
<th>Optimization Summary</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</td>
<td>1</td>
</tr>
<tr>
<td>Objective Value</td>
<td>0.0120807674</td>
</tr>
<tr>
<td>Infeasibility</td>
<td>0.0000963851</td>
</tr>
<tr>
<td>Optimization Status</td>
<td>Optimal</td>
</tr>
<tr>
<td>Degenerate</td>
<td>No</td>
</tr>
</tbody>
</table>

The “Histogram” table in Output 20.4.5 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 20.4.5 TE Process Analysis Histogram (MODELID=1, BW=5)

<table>
<thead>
<tr>
<th>Bin Midpoint Count</th>
<th>Percent (%)</th>
<th>Cumulative (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.983336</td>
<td>3</td>
<td>0.38</td>
</tr>
<tr>
<td>0.983837</td>
<td>5</td>
<td>1.00</td>
</tr>
<tr>
<td>0.984338</td>
<td>14</td>
<td>2.75</td>
</tr>
<tr>
<td>0.984839</td>
<td>31</td>
<td>6.63</td>
</tr>
<tr>
<td>0.985339</td>
<td>42</td>
<td>11.88</td>
</tr>
<tr>
<td>0.985840</td>
<td>63</td>
<td>19.75</td>
</tr>
<tr>
<td>0.986341</td>
<td>62</td>
<td>27.50</td>
</tr>
<tr>
<td>0.986842</td>
<td>96</td>
<td>39.50</td>
</tr>
<tr>
<td>0.987343</td>
<td>91</td>
<td>50.88</td>
</tr>
<tr>
<td>0.987843</td>
<td>393</td>
<td>100.00</td>
</tr>
</tbody>
</table>

The “Model Information” table in Output 20.4.6 summarizes the key options and the input data variables for the SVDD model that is trained using the option BW=20.

Output 20.4.6 TE Process Analysis Model Information (MODELID=16, BW=20)

<table>
<thead>
<tr>
<th>Model Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
<td>Active set</td>
</tr>
<tr>
<td>Kernel Type</td>
<td>RBF</td>
</tr>
<tr>
<td>RBF Kernel Bandwidth</td>
<td>20</td>
</tr>
<tr>
<td>Bandwidth Relative Scale</td>
<td>2.8970656673</td>
</tr>
<tr>
<td>Expected Outlier Fraction</td>
<td>1E-6</td>
</tr>
<tr>
<td>Optimization Tolerance</td>
<td>0.0001</td>
</tr>
<tr>
<td>Number of Interval Variables</td>
<td>41</td>
</tr>
<tr>
<td>Number of Nominal Variables</td>
<td>0</td>
</tr>
</tbody>
</table>

The “Training Results” table in Output 20.4.7 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 20.4.7 TE Process Analysis Training Results (MODELID=16, BW=20)

<table>
<thead>
<tr>
<th>Training Results</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Support Vectors</td>
<td>35</td>
</tr>
<tr>
<td>Number of Support Vectors on Boundary</td>
<td>35</td>
</tr>
<tr>
<td>Number of Outliers</td>
<td>2</td>
</tr>
<tr>
<td>Number of Dropped Observations</td>
<td>0</td>
</tr>
<tr>
<td>Threshold R Square Value</td>
<td>0.88994</td>
</tr>
<tr>
<td>Constant (C_r) Value</td>
<td>0.11006</td>
</tr>
<tr>
<td>Run Time (Seconds)</td>
<td>0.11456</td>
</tr>
</tbody>
</table>

The “Optimization Summary” table in Output 20.4.8 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.
Example 20.4: Analysis of Tennessee Eastman Process Data

Output 20.4.8  TE Process Analysis Optimization Summary (MODELID=16, BW=20)

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</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 20.4.9 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 20.4.9  TE Process Analysis Histogram (MODELID=16, BW=20)

<table>
<thead>
<tr>
<th>Frequency Distribution for SVDD Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin Midpoint</td>
</tr>
<tr>
<td>0.869585</td>
</tr>
<tr>
<td>0.871744</td>
</tr>
<tr>
<td>0.873904</td>
</tr>
<tr>
<td>0.876063</td>
</tr>
<tr>
<td>0.878223</td>
</tr>
<tr>
<td>0.880383</td>
</tr>
<tr>
<td>0.882542</td>
</tr>
<tr>
<td>0.884702</td>
</tr>
<tr>
<td>0.886862</td>
</tr>
<tr>
<td>0.889021</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:

```
proc astore;
  score data=mycas.score
  out=mycas.score_results
  rstore=mycas.scorecat;
```

The following code copies the score_results into the Work library:

```
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 an 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.

```r
    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.

```r
    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.

```r
    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.
References


Chapter 20: The SVDD Procedure


Chapter 21
The SVMACHINE Procedure

Overview: SVMACHINE Procedure

The SVMACHINE procedure implements the support vector machines (SVM) algorithm in SAS Viya. A popular data mining area classification method, the SVM algorithm computes support vector machine learning classifiers for the binary pattern recognition problem; it has 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 the SVM algorithm (applying...
Chapter 21: The SVMACHINE Procedure

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 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 11 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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 in Chapter 3, “Shared Concepts.”

This example trains the model by using German credit benchmark data, which are available in the samp-siо.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 samp-siо.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 samp-siо.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 444, 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 21.1 through Figure 21.3.

```
proc svmachine data=mycas.dmagecr;
   input checking history purpose savings employed marital coapp
        property other job housing telephon foreign/level=nominal;
   input duration amount installp resident existcr depends age/level=interval;
   target good_bad;
run;
```

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 21.1 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.
Chapter 21: The SVMACHINE Procedure

The “Misclassification Matrix” table in Figure 21.2 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 21.3.

### Figure 21.2  German Credit Misclassification Matrix

<table>
<thead>
<tr>
<th>Observed</th>
<th>good</th>
<th>bad</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>good</td>
<td>626</td>
<td>74</td>
<td>700</td>
</tr>
<tr>
<td>bad</td>
<td>142</td>
<td>158</td>
<td>300</td>
</tr>
<tr>
<td>Total</td>
<td>768</td>
<td>232</td>
<td>1000</td>
</tr>
</tbody>
</table>

### Figure 21.3  German Credit Accuracy

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.7840</td>
</tr>
<tr>
<td>Error</td>
<td>0.2160</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>0.8943</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.5267</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 < / option > ;

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.

DATA = CAS-libref.data-table
names the input data table for PROC SVMACHINE 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 444.

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

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.
**NOTE:** Generating the accuracy report for each iteration is computationally expensive and requires a significant amount of time.

**MAXITER=number**
specifies the maximum number of iterations before the process stops, where *number* is a positive integer.

By default, MAXITER=25. In some cases, you can obtain a good model in less than five iterations.

**NOPRINT**
suppresses the generation of ODS outputs. If you specify this option, no ODS tables are generated.

**NOSCALE**
uses the original data during training.

**NTHREADS=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.

**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.

**SCALE**
scales the input variables to between 0 and 1 during training.

By default, all numerical data are scaled before the training.

**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 ≥ 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 21.1 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 14 in Chapter 3, “Shared Concepts.” The TUNINGPARAMETERS option is described following table Table 21.1.

**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.

### Table 21.1  AUTOTUNE Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>KFOLD=</td>
<td>Specifies the number of folds for k-fold cross validation</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 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 train</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 BAYESIAN</td>
</tr>
<tr>
<td>SAMPLESIZE=</td>
<td>Specifies the sample size when SEARCHMETHOD=LHS or RANDOM</td>
</tr>
<tr>
<td>SEARCHMETHOD=</td>
<td>Specifies the search method that the optimizer uses</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.

**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**

```sas
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:

<table>
<thead>
<tr>
<th>FILE=filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>specifies the filename of the file to write the SAS score code to.</td>
</tr>
</tbody>
</table>

The CODE statement is optional.

---

**ID Statement**

```sas
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**

```sas
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 (/):

<table>
<thead>
<tr>
<th>LEVEL=INTERVAL</th>
<th>NOMINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>specifies whether the specified input variables are continuous or categorical. You can specify the following values:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INTERVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>specifies that the input variables are continuous.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOMINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>specifies that the input variables are categorical.</td>
</tr>
</tbody>
</table>

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</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=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 444.

- **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 21.2 shows how this variable is interpreted for each observation.

<table>
<thead>
<tr>
<th>Value</th>
<th>Observation Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.

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 444.

TARGET Statement

```
TARGET variable</options> ;
```

The TARGET statement names the target `variable` whose values PROC SVMACHINE predicts. The target `variable` must be binary and must be different from the variables in the INPUT statement.

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

```
ASC | ASCENDING
```

levelizes the target values in ascending order.

```
DESC | DESCENDING
```

levelizes the target values in descending order.

By default, the order is DESCENDING.

Details: SVMACHINE Procedure

PROC SVMACHINE uses linear or nonlinear kernels to compute support vector machine (SVM) learning classifiers for the binary pattern recognition problem. For more information about the theory and use of SVM learning, see Vapnik (1995); Burges (1998); Cristianini and Shawe-Taylor (2000).

In the linear kernel case, PROC SVMACHINE computes the parameters $w$ and $\beta$ that define the model function,

$$f(x) = w^T x + \beta$$
by solving the following quadratic optimization problem

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} w^T w + C e^T z \\
\text{subject to} & \quad D X w + \beta d \geq e - z \\
& \quad z \geq 0
\end{align*}
\]

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 \( Q = D X X^T D \) 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 452. 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.
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 shows the model information. It includes but is not limited to the inner product of weights, bias, and the number of support vectors.

Iteration History

The “Iteration History” table contains the number of iterations, the complementarity, and the feasibility. The complementarity is controlled by the TOLERANCE= option (which specifies the minimal absolute tolerance at which an iteration stops) and the MAXITER= option (which controls the number of iterations).

Misclassification Matrix

The “Misclassification Matrix” table contains the target information, both observed and predicted. The columns include the observed target, predicted event, predicted nonevent, and total numbers of events or nonevents for the training data.

Fit Statistics

The “Fit Statistics” table contains the model accuracy information, which includes accuracy, error, sensitivity, and specificity. The statistics are calculated from the “Misclassification Matrix” table.

Iteration Report

The “Iteration Report” table contains the model accuracy value for each iteration step. If the PARTITION statement is specified, then the specified validation or testing data accuracy is also reported.

Predicted Probability Variables

The “Predicted Probability Variables” table contains the target variable and the predicted probability variables. The names of the predicted probability variables begin with P_.

Predicted Target Variable

The “Predicted Target Variable” table contains the predicted target variable, whose name begins with I_.

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 21.3.

<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>
</tbody>
</table>

Table 21.3  ODS Tables Produced by PROC SVMACHINE
### Table 21.3 continued

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 in Chapter 3, “Shared Concepts.”

### Example 21.1: 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 21.4 describes the variables in Hmeq.
Table 21.4  Variables in the Home Equity (Hmeq) Data Set

<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 Hmeq data set into your CAS session by specifying your CAS engine libref in the second statement in the following DATA step:

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

The following statements execute the SVM algorithm on the mycas.hmeq data table:

```plaintext
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;
  target bad / desc;
  code file=codefile;
run;
```

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 21.1.1 through Output 21.1.5.

The “Model Information” table in Output 21.1.1 shows that the kernel function is linear, and the penalty parameter value is 1 (both of which are default values).
The observations table in Output 21.1.2 shows that the total number of observations is 5,960 and the number of observations used in the training is 3,364.

Output 21.1.2 Number of Observations

| Number of Observations Read | 5960 |
| Number of Observations Used  | 3364 |

The “Training Results” table in Output 21.1.3 shows the inner product of weights, bias, total slack, and so on.

Output 21.1.3 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 21.1.4 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 21.1.4 Misclassification Matrix

<table>
<thead>
<tr>
<th>Misclassification Matrix</th>
<th>Training Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed</td>
<td>1       0   Total</td>
</tr>
<tr>
<td>1</td>
<td>43      257 300</td>
</tr>
<tr>
<td>0</td>
<td>9       3055 3064</td>
</tr>
<tr>
<td>Total</td>
<td>52      3312 3364</td>
</tr>
</tbody>
</table>
The “Fit Statistics” table in Output 21.1.5 shows information about the accuracy, error, sensitivity, and specificity.

**Output 21.1.5  Fit Statistics**

<table>
<thead>
<tr>
<th>Fit Statistics</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.9209</td>
</tr>
<tr>
<td>Error</td>
<td>0.0791</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>0.1433</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.9971</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 21.2: 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`:

```sas
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`:

```sas
proc svmachine data=mycas.bigdata;
  input x1-x6 / level=interval;
  target y;
run;
```

The “Misclassification Matrix” table in Output 21.2.1 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 21.2.1 Misclassification Matrix

The `SVMACHINE Procedure`

<table>
<thead>
<tr>
<th>Misclassification Matrix</th>
<th>Training Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5164803</td>
</tr>
<tr>
<td>0</td>
<td>248256</td>
</tr>
<tr>
<td>Total</td>
<td>5413059</td>
</tr>
</tbody>
</table>

The “Fit Statistics” table in Output 21.2.2 shows the accuracy (92.85%) and the error (7.15%) of the model.

Output 21.2.2 Fit Statistics

<table>
<thead>
<tr>
<th>Fit Statistics</th>
<th>Statistic</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.9285</td>
<td></td>
</tr>
<tr>
<td>Error</td>
<td>0.0715</td>
<td></td>
</tr>
<tr>
<td>Sensitivity</td>
<td>0.9171</td>
<td></td>
</tr>
<tr>
<td>Specificity</td>
<td>0.9432</td>
<td></td>
</tr>
</tbody>
</table>

References


Chapter 22
The TEXTMINE Procedure

Contents

Overview: TEXTMINE Procedure ........................................... 464
PROC TEXTMINE Features ............................................... 464
Using CAS Sessions and CAS Engine Librefs ....................... 465
Getting Started: TEXTMINE Procedure ............................ 465
Syntax: TEXTMINE Procedure ........................................... 470
PROC TEXTMINE Statement ............................................ 470
DOC_ID Statement ....................................................... 471
PARSE Statement ......................................................... 471
SAVESTATE Statement .................................................. 476
SELECT Statement ........................................................ 476
SVD Statement .............................................................. 477
TARGET Statement ....................................................... 481
VARIABLES Statement .................................................... 481
Details: TEXTMINE Procedure ........................................... 481
Natural Language Processing ............................................ 481
   Stemming ................................................................. 482
   Part-of-Speech Tagging .............................................. 482
   Noun Group Extraction ............................................. 482
   Entity Identification ............................................... 482
   Multiword Terms Handling ....................................... 485
   Language Support .................................................... 485
Term and Cell Weighting ................................................ 485
Sparse Format ............................................................. 486
   Coordinate List (COO) Format .................................... 486
Singular Value Decomposition ....................................... 486
   Applications in Text Mining ..................................... 486
   Computation .......................................................... 487
   SVD-Only Mode ....................................................... 487
   Topic Discovery ...................................................... 487
Output Data Tables ....................................................... 488
   The OUTCHILD= Data Table ..................................... 488
   The OUTCONFIG= Data Table .................................... 488
   The OUTDOCPRO= Data Table ................................... 489
   The OUTPARENT= Data Table .................................... 489
   The OUTPOS= Data Table ......................................... 490
   The OUTTERMS= Data Table ..................................... 490
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:

```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 11 in Chapter 3, “Shared Concepts.”

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 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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.

```sas
proc textmine data=mycas.getstart;
    doc_id did;
    variables text;
    parse outterms = mycas.terms reducef = 1 stop = mycas.en_stoplist;
    svd k = 3 outdocpro = mycas.docpro outtopics = mycas.topics;
    savestate rstore = mycas.aStoreTab;
run;
```

The output from this analysis is presented in Figure 22.2, Figure 22.3 and Figure 22.4.

**Figure 22.1** shows the SAS log that is generated by PROC TEXTMINE; the log provides information about the default configurations used by the procedure and about the input and output files including the number of observations in each of the output tables. The mycas.terms data table lists the discovered terms. The
mycas.docpro data table contains four variables: the first variable is the document ID, and the remaining three variables are obtained by projecting the original document onto the three left-singular vectors that have been rotated with the default orthogonal (varimax) rotation. The mycas.topics data table has 3 variables containing summary information of the discovered topics. Finally, the mycas.astoretab table contains a binary representation of a scoring model.

**Figure 22.1** SAS Log

NOTE: Stemming will be used in parsing.
NOTE: Tagging will be used in parsing.
NOTE: Noun groups will be used in parsing.
NOTE: No TERMWGT option is specified. TERMWGT=ENTROPY will be run by default.
NOTE: No CELLLWGT option is specified. CELLLWGT=LOG will be run by default.
NOTE: No ENTITIES option is specified. ENTITIES=NONE will be run by default.
NOTE: Topics have been requested so the document unit normalization will not occur unless requested.
NOTE: The dense SVD solver was used for this calculation.
NOTE: Wrote 12532 bytes to the savestate file ASTORETAB.
NOTE: The Cloud Analytic Services server processed the request in 1.670414 seconds.
NOTE: The data set MYCAS.TERMS has 134 observations and 11 variables.
NOTE: The data set MYCAS.DOCPRO has 16 observations and 4 variables.
NOTE: The data set MYCAS.TOPICS has 3 observations and 3 variables.
NOTE: The data set MYCAS.ASTORETAB has 1 observations and 2 variables.

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
    data docpro;
        set mycas.docpro;
        run;
    proc sort data=docpro;
        by did;
        run;
    proc print data = docpro (obs=10);
        run;
```

**Figure 22.2** shows the output of PROC PRINT. For information about the output of the OUTDOCPRO= option, see the section “The OUTDOCPRO= Data Table” on page 489.
Figure 22.2 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>0.1111856451</td>
<td>0</td>
<td></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>0.4742893251</td>
<td>0</td>
<td></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:

```r
data topics; set mycas.topics; run;
proc print data = topics;
run;
```

Figure 22.3 shows the output of PROC PRINT. The three discovered topics are listed with four descriptive terms to characterize each topic.

Figure 22.3 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:

```r
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 22.4 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 490.
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 22.5 shows the output of PROC PRINT, which provides the topic score for the documents processed by the ASTORE PROCEDURE.

**Figure 22.4** 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>nlpNounGroup</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>nlpNounGroup</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>nlpNounGroup</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>nlpNounGroup</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>

**Figure 22.5** The mycas.scoreResults Data Table

<table>
<thead>
<tr>
<th>Obs</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 22.1 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
**DOC_ID Statement**

```latex
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.

**PARSE Statement**

```latex
PARSE < parse-options > ;
```

The `PARSE` statement specifies the options for parsing the input documents and creating the term-by-document matrix. Table 22.2 summarizes the `parse-options` in the statement by function. The `parse-options` are then described fully in alphabetical order.
### Table 22.2 PARSE Statement Options

<table>
<thead>
<tr>
<th>parse-option</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`.

**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 485.
- **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 465. 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**
**NONE** 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 465. 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 488.

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 465. 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 488.

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 465. 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 489.

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 465. For more information about this data table, see the section “The OUTPOS= Data Table” on page 490.

OUTTERMS=CAS-libref.data-table
specifies the output data table to contain the summary 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 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 465. For more information about this data table, see the section “Output Data Tables” on page 488.
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=CAS-libref.data-table
specifies the input data table that contains the terms that are to be kept for the analysis. 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 465. 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=CAS-libref.data-table
specifies the input data table that contains the terms to exclude from the analysis. 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 465. 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=CAS-libref.data-table
SYN=CAS-libref.data-table
specifies the input data table that contains user-defined synonyms to be used in the analysis. 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 465. The data table specifies parent-child relationships that enable you to map child terms to a representative parent. The synonym relationship is indicated in the data table that is specified in the OUTTERMS= option and is also reflected in the term-by-document data table that is specified in the OUTPARENT= option. 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 485.

**SAVESTATE Statement**

```plaintext
SAVESTATE RSTORE=\texttt{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:

```plaintext
RSTORE=\texttt{CAS-libref.data-model}
```

specifies a data table in which to save the text mining model. \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 465.

**SELECT Statement**

```plaintext
SELECT label-list /\texttt{GROUP=group-option} > \texttt{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 \texttt{OUTTERMS=} data table. Terms that are excluded by the SELECT statement cannot be included in the \texttt{OUTTERMS=} data table, whereas terms that have \_keep=N can be included in the \texttt{OUTTERMS=} data table if the \texttt{SHOWDROPPEDTERMS} option is specified. Terms excluded by the SELECT statement are excluded from the \texttt{OUTPOS=} data table, but terms that have \_keep=N are included in \texttt{OUTPOS=} data table. Table 22.3 summarizes the options you can specify in the SELECT statement. The options are then described fully in syntactic order.

**Table 22.3 SELECT Statement Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{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>\texttt{GROUP=}</td>
<td>Specifies whether the labels are parts of speech, entities, or attributes</td>
</tr>
<tr>
<td>\texttt{IGNORE}</td>
<td>Ignores terms whose labels are specified in the \texttt{label-list}</td>
</tr>
<tr>
<td>\texttt{KEEP}</td>
<td>Keeps terms whose labels are specified in the \texttt{label-list}</td>
</tr>
</tbody>
</table>

You must specify a \texttt{label-list} and either the \texttt{IGNORE} or \texttt{KEEP} option:
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 22.5 shows all possible part-of-speech tags. Table 22.6 shows all valid English entities. The attribute variable in Table 22.11 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”.

---

**SVD Statement**

```
SVD < 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 22.4 summarizes the svd-options in the statement by function. The svd-options are then described fully in alphabetical order.

<table>
<thead>
<tr>
<th>svd-option</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>
</tbody>
</table>
Table 22.4  continued

<table>
<thead>
<tr>
<th>svd-option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 465. The data table should have the variables that are described in Table 22.11. 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.

\[ \text{K} = k \]

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.

\[ \text{MAX}_K = n \]

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.

\[ \text{NOCUTOFFS} \]

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.

\[ \text{NUMLABELS} = n \]

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.

\[ \text{OUTDOCPRO}=\text{CAS-libref.data-table}<\text{KEEPVARIABLES}=\text{variable-list}><\text{NONORMDOC}> \]

specifies the output data table to contain the projections of the columns of the term-by-document matrix onto the columns of \( U \). \text{CAS-libref.data-table} is a two-level name, where \text{CAS-libref} refers to the 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 465. 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:

\[ \text{KEEPVARIABLES}=\text{variable-list} \]

attaches the content of the variables that are specified in the \text{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 465.

RESOLUTION=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 465.
TARGET Statement

**TARGET statement**

```plaintext
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

```plaintext
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 22.5 shows all possible 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 22.6 shows all valid entities for English. Not all languages support all entities. Table 22.7 and Table 22.8 indicate the languages that are available for each entity.

### Table 22.6  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 22.7  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>
</tbody>
</table>
### Table 22.7  continued

<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>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 22.8  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>
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, 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:

$$w_i = 1 + \sum_j p_{i,j} \log_2 \left( \frac{p_{i,j}}{\log_2(n)} \right)$$

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:

$$w_i = \max_k \left( \log \left( \frac{P(t_i, C_k)}{P(t_i) P(C_k)} \right) \right)$$

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:

<table>
<thead>
<tr>
<th>term</th>
<th>doc</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>d1</td>
<td>8</td>
</tr>
<tr>
<td>t2</td>
<td>d1</td>
<td>1</td>
</tr>
</tbody>
</table>

The term count of “said” in $d1$ 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 22.9 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>
</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 22.9  continued

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<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 22.10 shows the variables in this 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 22.11 shows the variables in this 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>Alpha</td>
<td>only alphabetic characters</td>
</tr>
<tr>
<td>Mixed</td>
<td>a combination of attributes</td>
</tr>
<tr>
<td>Num</td>
<td>only numbers</td>
</tr>
<tr>
<td>Punct</td>
<td>punctuation characters</td>
</tr>
<tr>
<td>Entity</td>
<td>an identified entity</td>
</tr>
</tbody>
</table>
Table 22.11  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></td>
<td>• If a term has a parent, this variable contains the term number of that parent.</td>
</tr>
<tr>
<td></td>
<td>• If a term does not have a parent, this value is a “.” (period).</td>
</tr>
<tr>
<td></td>
<td>• If the values of Key, Parent, and Parent_id are identical, the parent occurs as itself.</td>
</tr>
<tr>
<td></td>
<td>• If the values of Parent and Parent_id are identical but differ from Key, the observation is a child.</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 22.1: 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
;
runc;
```

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 22.1.1 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 22.1.1 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>Y</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></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>2</td>
<td>1</td>
<td></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>3</td>
<td>1</td>
<td></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>4</td>
<td>1</td>
<td></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>5</td>
<td>1</td>
<td></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>6</td>
<td>1</td>
<td></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>7</td>
<td>1</td>
<td></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>8</td>
<td>1</td>
<td></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>9</td>
<td>1</td>
<td></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>10</td>
<td>1</td>
<td></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>11</td>
<td>1</td>
<td></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>12</td>
<td>1</td>
<td></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>13</td>
<td>1</td>
<td></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>14</td>
<td>1</td>
<td></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>15</td>
<td>1</td>
<td></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>16</td>
<td>1</td>
<td></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>17</td>
<td>1</td>
<td></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>18</td>
<td>1</td>
<td></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>19</td>
<td>1</td>
<td></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>20</td>
<td>1</td>
<td></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>21</td>
<td>1</td>
<td></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>22</td>
<td>1</td>
<td></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>23</td>
<td>1</td>
<td></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>24</td>
<td>1</td>
<td></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>25</td>
<td>1</td>
<td></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>26</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 22.2: Parsing with Stemming

This example uses the data table that is generated in Example 22.1. 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 22.2.1 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 22.2.1  The mycas.outterms Data Table with Stemming

<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></td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>win</td>
<td>Alpha</td>
<td></td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>2</td>
<td>.</td>
<td>2 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>toyota</td>
<td>Alpha</td>
<td></td>
<td>2</td>
<td>1</td>
<td>Y</td>
<td>3</td>
<td>.</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ford</td>
<td>Alpha</td>
<td></td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>4</td>
<td>.</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>tacoma</td>
<td>Alpha</td>
<td></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>6</td>
<td>year</td>
<td>Alpha</td>
<td></td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>6</td>
<td>.</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>taurus</td>
<td>Alpha</td>
<td></td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>won</td>
<td>Alpha</td>
<td></td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>26</td>
<td>2</td>
<td>2 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>honda</td>
<td>Alpha</td>
<td></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>10</td>
<td>bright</td>
<td>Alpha</td>
<td></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>11</td>
<td>be</td>
<td>Alpha</td>
<td></td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>10</td>
<td>.</td>
<td>10 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>sold</td>
<td>Alpha</td>
<td></td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>27</td>
<td>11</td>
<td>11 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>sell</td>
<td>Alpha</td>
<td></td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>11</td>
<td>.</td>
<td>11 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>colors</td>
<td>Alpha</td>
<td></td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>28</td>
<td>23</td>
<td>23 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>lime</td>
<td>Alpha</td>
<td></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>16</td>
<td>except</td>
<td>Alpha</td>
<td></td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>hyundai</td>
<td>Alpha</td>
<td></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>18</td>
<td>in</td>
<td>Alpha</td>
<td></td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>is</td>
<td>Alpha</td>
<td></td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>29</td>
<td>10</td>
<td>10 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>for</td>
<td>Alpha</td>
<td></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>21</td>
<td>world</td>
<td>Alpha</td>
<td></td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>17</td>
<td>.</td>
<td>17</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>green</td>
<td>Alpha</td>
<td></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>23</td>
<td>the</td>
<td>Alpha</td>
<td></td>
<td>8</td>
<td>5</td>
<td>Y</td>
<td>19</td>
<td>.</td>
<td>19</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>of</td>
<td>Alpha</td>
<td></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>25</td>
<td>award</td>
<td>Alpha</td>
<td></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>26</td>
<td>was</td>
<td>Alpha</td>
<td></td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>30</td>
<td>10</td>
<td>10 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>car</td>
<td>Alpha</td>
<td></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>28</td>
<td>color</td>
<td>Alpha</td>
<td></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>29</td>
<td>insight</td>
<td>Alpha</td>
<td></td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>last</td>
<td>Alpha</td>
<td></td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>25</td>
<td>.</td>
<td>25</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Example 22.3: Adding Entities and Noun Groups

This example uses the data table that is generated in Example 22.1. 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.

/* 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
   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 22.3.1 shows the content of the mycas.outterms data table. Compared to Output 22.2.1, the mycas.outterms data table is longer, because it contains entities and noun groups.
### Output 22.3.1 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</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</td>
<td>2</td>
<td>.</td>
<td>2 +</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Tacoma</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>.</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>year</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>.</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>taurus</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>.</td>
<td>5</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>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>won</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>30</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>honda</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>bright</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>.</td>
<td>8</td>
<td>1</td>
<td>1</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>9</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>be</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>10</td>
<td>.</td>
<td>10 +</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>sold</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>31</td>
<td>12</td>
<td>12</td>
<td>1</td>
<td>1</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>11</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>sell</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>.</td>
<td>12 +</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>colors</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>32</td>
<td>27</td>
<td>27</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>lime</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>hyundai</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>14</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>except</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>in</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>is</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>33</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>toyota</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>1</td>
<td>Y</td>
<td>17</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>last year</td>
<td>nlpDate</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>18</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>ford</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>19</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>for</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>world</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>21</td>
<td>.</td>
<td>21</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>green</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>the</td>
<td>Alpha</td>
<td>8</td>
<td>5</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>of</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>award</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>25</td>
<td>.</td>
<td>25</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>was</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>34</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>car</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>26</td>
<td>.</td>
<td>26</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>color</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>27</td>
<td>.</td>
<td>27 +</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>insight</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>28</td>
<td>.</td>
<td>28</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>last</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>29</td>
<td>.</td>
<td>29</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example 22.4: Adding Part-of-Speech Tagging

This example uses the data table that is generated in Example 22.1. 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
;
runc;
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
;
runc;
data outterms; set mycas.outterms; run;
proc print data= outterms; run;
```

Output 22.4.1 shows the content of the mycas.outterms data table. Compared to Output 22.3.1, the mycas.outterms data table also contains the part-of-speech tag for the terms.
Output 22.4.1 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_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>30</td>
<td>26</td>
<td>26 .</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>31</td>
<td>26</td>
<td>26 .</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>32</td>
<td>17</td>
<td>17 .</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>for</td>
<td></td>
<td>PPOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>nlpDate</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>6</td>
<td>.</td>
<td>6</td>
<td>1</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></td>
<td>PPPOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>33</td>
<td>28</td>
<td>28 .</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>car</td>
<td>PN</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>colors</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>34</td>
<td>10</td>
<td>10 .</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>award</td>
<td>N</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>insight</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>of</td>
<td>PPOS</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>21</td>
<td>.</td>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>honda</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>world</td>
<td>PN</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>last</td>
<td>A</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>green</td>
<td>N</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>25</td>
<td>.</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>be</td>
<td>V</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>26</td>
<td>.</td>
<td>26 +</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>tacoma</td>
<td>PN</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>27</td>
<td>.</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>sell</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>28</td>
<td>.</td>
<td>28 +</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>year</td>
<td>N</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>29</td>
<td>.</td>
<td>29</td>
<td>1</td>
</tr>
</tbody>
</table>
Example 22.5: Adding Synonyms

This example uses the data table that is generated in Example 22.1. So far, by looking at the mycas.outterms data tables that are generated by Example 22.1 to Example 22.4, 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 22.5.1 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 22.5.1 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>+</td>
<td>1</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>3</td>
<td>.</td>
<td>3</td>
<td>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>+</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>+</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>+</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>.</td>
<td>1</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>+</td>
<td>1</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>.</td>
<td>1</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>+</td>
<td>1</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>.</td>
<td>1</td>
</tr>
</tbody>
</table>
Example 22.6: Adding a Custom Stop List

This example uses the data table that is generated in Example 22.1 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;
```
Example 22.6: Adding a Custom Stop List

Output 22.6.1 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 22.6.1 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</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 Y</td>
<td>29</td>
<td>25</td>
<td>25 .</td>
<td>1</td>
<td></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>
<td></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></td>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>6</td>
<td>for</td>
<td>PPOS</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>3</td>
<td>.</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>lime green</td>
<td>nplnounGroup</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>4</td>
<td>.</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 Y</td>
<td>5</td>
<td>.</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>in 2008</td>
<td>nplDate</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>6</td>
<td>.</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 Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td>1</td>
<td></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>
<td></td>
</tr>
<tr>
<td>12</td>
<td>bright green</td>
<td>nplnounGroup</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>9</td>
<td>.</td>
<td>9</td>
<td>1</td>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>15</td>
<td>hyundai</td>
<td>nplOrganization</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>12</td>
<td>.</td>
<td>12</td>
<td>1</td>
<td></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>
<td></td>
</tr>
<tr>
<td>17</td>
<td>toyota</td>
<td>nplOrganization</td>
<td>Entity</td>
<td>2</td>
<td>1 Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>last year</td>
<td>nplDate</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>ford</td>
<td>nplOrganization</td>
<td>Entity</td>
<td>2</td>
<td>2 Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td>1</td>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
</tbody>
</table>
Example 22.7: 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 22.1 to show how to use the MULTITERM= option. The following DATA steps generate and uses a multiterm 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;
/* 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 22.7.1 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 22.7.1 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>.</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>1</td>
<td></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>.</td>
<td>1</td>
<td></td>
<td>1</td>
<td></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>1</td>
<td></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></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>1</td>
<td></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>.</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>1</td>
<td></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>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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>.</td>
<td>26</td>
<td>1</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>1</td>
<td></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>1</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 22: The TEXTMINE Procedure

Example 22.8: Selecting Parts of Speech and Entities to Ignore

This example uses the data table that is generated in Example 22.1. 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 22.8.1 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 22.8.1** 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</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>19</td>
<td>16</td>
<td>16 .</td>
<td>1</td>
<td></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>20</td>
<td>16</td>
<td>16 .</td>
<td>1</td>
<td></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>1</td>
<td></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>1</td>
<td></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>21</td>
<td>12</td>
<td>12 .</td>
<td>1</td>
<td></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>.</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>lime</td>
<td>A</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>8</td>
<td>except</td>
<td>V</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>9</td>
<td>bright green</td>
<td>nlpNounGroup</td>
<td>Alpha</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>10</td>
<td>color</td>
<td>N</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>
</tr>
<tr>
<td>11</td>
<td>hyundai</td>
<td>nlpOrganization</td>
<td>Entity</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>12</td>
<td>sold</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>22</td>
<td>17</td>
<td>17 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>toyota</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</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>ford</td>
<td>nlpOrganization</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>10</td>
<td>.</td>
<td>10</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>all</td>
<td>A</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>
</tr>
<tr>
<td>16</td>
<td>win</td>
<td>V</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>
</tr>
<tr>
<td>17</td>
<td>colors</td>
<td>N</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>23</td>
<td>7</td>
<td>7 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>award</td>
<td>N</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>
</tr>
<tr>
<td>19</td>
<td>last</td>
<td>A</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>
</tr>
<tr>
<td>20</td>
<td>green</td>
<td>N</td>
<td>Alpha</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>be</td>
<td>V</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>16</td>
<td>.</td>
<td>16 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>sell</td>
<td>V</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>17</td>
<td>.</td>
<td>17 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>year</td>
<td>N</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>1</td>
<td></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 514.

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 11 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
Getting Started: TMSCORE Procedure

name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 11 and “Loading a SAS Data Set onto a CAS Server” on page 12 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 |1
  unlocking the unprecedented business value of big data.|2
  Organizations looking for optimal ways to gain insights|3
  from big data in shorter reporting windows are turning to SAS.|4
  As the gold-standard leader in business analytics |5
  for more than 36 years,|6
  SAS frees enterprises from the limitations of |7
  traditional computing and enables them |8
  to draw instant benefits from big data.|9
  Faster Time to Insight.|10
  From banking to retail to health care to insurance, |11
  SAS is helping industries glean insights from data |12
  that once took days or weeks in just hours, minutes or seconds.|13
  It’s all about getting to and analyzing relevant data faster.|14
  Revealing previously unseen patterns, sentiments and relationships.|15
  Identifying unknown risks.|16
  And speeding the time to insights.|17
  High-Performance Analytics from SAS Combining industry-leading |18
  analytics software with high-performance computing technologies|19
  produces fast and precise answers to unsolvable problems|20
  and enables our customers to gain greater competitive advantage.|21
  SAS In-Memory Analytics eliminate the need for disk-based processing|22
  allowing for much faster analysis.|23
  SAS In-Database executes analytic logic into the database itself |24
  for improved agility and governance.|25
  SAS Grid Computing creates a centrally managed,|26
  shared environment for processing large jobs|27
  and supporting a growing number of users efficiently.|28
  Together, the components of this integrated, |29
  supercharged platform are changing the decision-making landscape|30
  and redefining how the world solves big data business problems.|31
  Big data is a popular term used to describe the exponential growth,|32
  availability and use of information,|33
  both structured and unstructured.|34
  Much has been written on the big data trend and how it can |35
  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:

```
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 23.1.

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 23.1 shows the output of PROC PRINT.

---

**Figure 23.1** The mycas.docpro Data Table

<table>
<thead>
<tr>
<th>Obs</th>
<th>did</th>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
<th>COL4</th>
<th>COL5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.8618192721</td>
<td>-0.167546011</td>
<td>0.037379386</td>
<td>0.4703235489</td>
<td>-0.081206017</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.3275352424</td>
<td>0.5970467719</td>
<td>-0.046820597</td>
<td>0.7262257288</td>
<td>0.081607841</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.8604238893</td>
<td>-0.412231055</td>
<td>0.0599438871</td>
<td>0.2665897873</td>
<td>-0.122771753</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.7741694143</td>
<td>0.4670908635</td>
<td>0.2844247769</td>
<td>-0.207037529</td>
<td>-0.242334173</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0.9576866114</td>
<td>-0.265146311</td>
<td>0.0265294315</td>
<td>0.0244271372</td>
<td>0.1059872269</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0.8078123292</td>
<td>-0.198309231</td>
<td>-0.399949692</td>
<td>0.3182364359</td>
<td>-0.216514445</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0.9183024782</td>
<td>-0.099013401</td>
<td>0.3165905611</td>
<td>0.1756583621</td>
<td>-0.125823135</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0.8321211924</td>
<td>-0.161579007</td>
<td>-0.291096477</td>
<td>0.4392268744</td>
<td>0.0617182277</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>0.8905692292</td>
<td>0.1358612507</td>
<td>0.3702567322</td>
<td>0.1577544227</td>
<td>-0.162639672</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0.6387842303</td>
<td>0.1643248339</td>
<td>0.1598343808</td>
<td>-0.572869995</td>
<td>0.4595922056</td>
</tr>
</tbody>
</table>
Syntax: TMSCORE Procedure

The following statements are available in the TMSCORE procedure:

```plaintext
PROC TMSCORE DATA= <options> ;
    VARIABLES <options> ;
    DOC_ID <options> ;
```

PROC TMSCORE Statement

```plaintext
PROC TMSCORE DATA= <options> ;
```

The PROC TMSCORE statement invokes the procedure. Table 23.1 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>TERMS=</td>
<td></td>
</tr>
<tr>
<td>CONFIG=</td>
<td></td>
</tr>
<tr>
<td>SVDU=</td>
<td></td>
</tr>
</tbody>
</table>

| **Output Options**                         |             |
| OUTPARENT= | | 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. |
| SVDDOCPRO= | | Specifies the data table that contains the projections of the documents |

You must specify the following option:

**DATA=** <options>  
**DOC=** <options>  

names the input data table for PROC TMSCORE to use.  <options> 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 510.

- **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 510. 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 488 of Chapter 22, “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 510. 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 489 of Chapter 22, “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 510. 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 510. 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 SVDU= option in Chapter 22, “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 510. 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 OUT TERMS= Data Table” on page 490 of Chapter 22, “The TEXTMINE Procedure.”

**DOC_ID Statement**

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**

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 481 of Chapter 22, “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.
Overview: TSNE Procedure

The TSNE procedure implements the \( t \)-distributed stochastic neighbor embedding (\( t \)-SNE) dimension reduction method in SAS Viya. The \( t \)-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, \( t \)-SNE is appropriate for nonlinear data and emphasizes existing groupings in the data. The method is named \( t \)-SNE because it models the pairwise distances in low dimensions according to Student’s \( t \)-distribution. The \( t \)-distribution with one degree of freedom has heavier tails than the Gaussian distribution, which means that it assigns higher probability values to large distances. This allows \( t \)-SNE to relax pairwise distances for non-neighboring observations, whereas distances between closely neighboring observations are more exactly preserved. This behavior is desirable because it mitigates the crowding problem in high-dimensional data representation and makes existing groups in the data visually evident. 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 11 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
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 24.1 shows the results. The colors in the scatter plot indicate three distinct clusters, which correspond to the three iris species.

Figure 24.1 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 > ;
```

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.

PROC TSNE Statement

```
PROC TSNE < options > ;
```

The PROC TSNE statement invokes the procedure. Table 24.1 summarizes the options available in the PROC TSNE statement.
Table 24.1  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>
</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 518.
- `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`.

**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`.

**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 524.

By default, **PERPLEXITY=30**.

---

**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 526. 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.
**DISPLAYOUT Statement**

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 526. 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**

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`. 

**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.
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 518.

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

$$ C = KL(P \parallel Q) = \sum_{i=1}^{N} \sum_{j=1}^{N} p_{ij} \log \frac{p_{ij}}{q_{ij}} $$

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 24.2.

<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 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</td>
<td>PROC TSNE</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>“OutputCasTables Table” on page 525</strong></td>
<td></td>
<td></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 24.3.

<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, 177
  GRADBOOST procedure, 241
  NNET procedure, 345

OutputCasTables table
  FACTMAC procedure, 125
  FASTKNN procedure, 137
  GRADBOOST procedure, 243
  MTLEARN procedure, 303
  TSNE procedure, 525

active-set solver
  SVDD procedure, 412

ASTORE procedure
  computational method, 22
  hyperparameter tuning, 23
  input data tables, 33
  multithreading, 22
  output data tables, 33
  test data, 21
  validation, 21

autoencoder
  NNET procedure, 346

AUTOTUNE statement
  syntax (Shared Concepts), 14

bagging the data
  FOREST procedure, 169

bandwidth relative scale
  SVDD procedure, 410

best configuration
  FOREST procedure, 179
  GRADBOOST procedure, 242
  NNET procedure, 347

binning
  BNET procedure, 55

BNET procedure, 47
  binning, 55
  displayed output, 70
  fit statistics table, 70
  independence tests, 53, 54
  input data tables, 53
  iteration report table, 71
  missing values, 54, 55
  model information table, 70
  model selection, 53
  number of observations table, 70
  ODS table names, 71

boosting
  GRADBOOST procedure, 234

cluster summary
  GVARCLUS procedure, 266

CODE statement
  syntax (Shared Concepts), 19

comparison with existing methods
  SVDD procedure, 411

computational method
  ASTORE procedure, 22
  BOOLRULE procedure, 22
  FACTMAC procedure, 22
  FOREST procedure, 22
  GRADBOOST procedure, 22
  NNET procedure, 22, 344
  SVMACHINE procedure, 22
  TEXTMINE procedure, 22
  TMSCORE procedure, 22

convergence status
  NNET procedure, 346

cross validation results
  NNET procedure, 347

cross-validation fit statistics
  FOREST procedure, 179, 242

descriptive statistics
  GMM procedure, 204

details
  MWPCA procedure, 316
  RPCA procedure, 368

displayed output
  BNET procedure, 70
Subject Index

FACTMAC procedure, 124
FASTKNN procedure, 136
FISM procedure, 145
FOREST procedure, 178
GMM procedure, 204
GRADBOOST procedure, 241
GVARCLUS procedure, 266
MBANALYSIS procedure, 282
MTLEARN procedure, 303
NNET procedure, 346
RPCA procedure, 370
SEMISUPLEARN procedure, 387
SVMACHINE procedure, 456
TSNE procedure, 525

early stopping
GRADBOOST procedure, 235
evaluation history
FOREST procedure, 179
GRADBOOST procedure, 243
NNET procedure, 347

FACTMAC procedure, 111
OutputCasTables table, 125
computational method, 22
displayed output, 124
final exact loss, 125
hyperparameter tuning, 23
input data tables, 116
interval variables, 125
iteration history, 124
model information, 124
multithreading, 22
number of observations, 124
ODS table names, 125
output data tables, 122, 126
test data, 21
validation, 21

FASTKNN procedure, 129
OutputCasTables table, 137
displayed output, 136
input data tables, 132
model information, 136
number of observations, 137
ODS table names, 137
output data tables, 136, 137

final exact loss
FACTMAC procedure, 125
FISM procedure, 143
displayed output, 145
FP-growth algorithm, 143
input data tables, 142
limitations, 145

fit statistics
FOREST procedure, 178
GRADBOOST procedure, 242
fit statistics table
BNET procedure, 70
SVMACHINE procedure, 457
FitStat table
NNET procedure, 347

FOREST procedure, 148
k-fold cross validation, 177
bagging the data, 169
best configuration, 179
cross-validation fit statistics, 179, 242
displayed output, 178
evaluation history, 179
fit statistics, 178
handling missing values, 172
handling values that are absent from training data, 173
hyperparameter tuning, 23, 176
input data tables, 156
isolation forests, 176
measuring variable importance, 173
model information, 178
multithreading, 22
number of observations, 178
ODS table names, 179
output data tables, 166
OutputCasTables table, 179
predicted probability names, 179
preselection, 170
splitting criteria, 164
test data, 21
training a decision tree, 169
tuner information, 178
tuner results, 179
tuner summary, 178
tuner timing, 179
validation, 21
variable importance, 178

FP-growth algorithm
FISM procedure, 143
MBANALYSIS procedure, 280
fraction outlier parameter
SVDD procedure, 410

GMM procedure, 187
cluster summary output data table, 195, 196
descriptive statistics, 204
displayed output, 204
input data tables, 196
missing values, 202
model information, 204
number of observations, 204
ODS table names, 204
output data tables, 199
GRADBOOST procedure, 212

- k-fold cross validation, 241
- OutputCasTables table, 243
- best configuration, 242
- boosting, 234
- computational method, 22
- displayed output, 241
- early stopping, 235
- evaluation history, 243
- fit statistics, 242
- handling missing values, 236
- hyperparameter tuning, 23, 241
- input data tables, 220
- measuring prediction error, 235
- measuring variable importance, 237
- model information, 241
- multithreading, 22
- number of observations, 241
- ODS table names, 243
- output data tables, 230
- predicted probability names, 243
- subsampling the data, 233
- test data, 21
- training a decision tree, 234
- transfer learning, 240
- tuner information, 242
- tuner results, 242
- tuner summary, 242
- tuner timing, 242
- validation, 21
- variable importance, 242

Graphical lasso
- GVARNCLUS procedure, 264

Graphical representation of precision matrix
- GVARNCLUS procedure, 265

GRAPHTYPE= option examples
- MTLEARN procedure, 298

GVARNCLUS procedure, 253
- cluster summary, 266
- displayed output, 266
- graphical lasso, 264
- graphical representation of precision matrix, 265
- input data tables, 259
- maximum iterations, 260
- maximum members, 260
- maximum steps, 260
- minimum clusters, 260
- missing values, 264
- number of observations, 266
- ODS table names, 266
- preprocessing, 265
- rho, 261

select, 261
selection, 266
stop, 261
tolerance, 261

Handling missing values
- FOREST procedure, 172
- GRADBOOST procedure, 236

Handling values that are absent from training data
- FOREST procedure, 173

Hyperparameter tuning
- ASTORE procedure, 23
- BOOLRULE procedure, 23
- FACTMAC procedure, 23
- FOREST procedure, 23, 176
- GRADBOOST procedure, 23, 241
- NNET procedure, 23
- SVMACHINE procedure, 23
- TMSCORE procedure, 23

Independence tests, 66
- BNNET procedure, 53, 54

Input variables
- NNET procedure, 336

Interval variables
- FACTMAC procedure, 125
- MTLEARN procedure, 303

Isolation forests
- FOREST procedure, 176

Iteration history
- NNET procedure, 346

Iteration history table
- SVMACHINE procedure, 457

Iteration report table
- BNNET procedure, 71
- SVMACHINE procedure, 457

Limitations
- FISM procedure, 145
- MBANALYSIS procedure, 282

Linear kernels
- SVDD procedure, 403
- SVMACHINE procedure, 454

Mathematical formulation
- SVDD procedure, 404

MBANALYSIS procedure, 279
- displayed output, 282
- FP-growth algorithm, 280
- input data algorithm, 276
- limitations, 282
- scoring with PROC ASTORE, 281

Mean criteria
- SVDD procedure, 409
measuring prediction error
  GRADBOOST procedure, 235
measuring variable importance
  FOREST procedure, 173
  GRADBOOST procedure, 237
misclassification matrix table
  SVMACHINE procedure, 457
missing values
  BNET procedure, 54, 55
  GMM procedure, 202
  GV ARCLUS procedure, 264
  MTLEARN procedure, 303
model information
  FOREST procedure, 178
  GMM procedure, 204
  GRADBOOST procedure, 241
  NNET procedure, 346
model information table
  BNET procedure, 70
  SVMACHINE procedure, 456
model selection
  BNET procedure, 53
modified mean criteria
  SVDD procedure, 409
MTLEARN procedure, 287
  OutputCasTables table, 303
displayed output, 303
  GRAPHTYPE= option examples, 298
input data tables, 293
interval variables, 303
iteration history, 303
missing values, 303
model information, 303
number of observations, 303
ODS table names, 304
output data tables, 297, 304
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, 316
dimensions, 317
displayed output, 317
examples, 318
input data tables, 311
model information, 317
ODS table names, 318
output data tables, 314
results summary, 318

NNET procedure, 324
  k-fold cross validation, 345
  autoencoder, 346
  best configuration, 347
  computational method, 22, 344
  convergence status, 346
  cross validation results, 347
displayed output, 346
evaluation history, 347
FitStat table, 347
hyperparameter tuning, 23
input data tables, 327
iteration history, 346
model information, 346
multithreading, 22
ODS table names, 348
output data tables, 339
parameter tuning, 344
PredIntoName, 347
PredName, 348
PredProbName, 347
ROC curve, 347
score information, 346
test data, 21
tuner information, 346
tuner results, 347
tuner summary, 347
tuner timing, 347
validation, 21
nonlinear kernels
  SVDD procedure, 403
  SVMACHINE procedure, 454
number of hidden neurons
  NNET procedure, 335
number of observations
  FOREST procedure, 178
  GMM procedure, 204
  GRADBOOST procedure, 241
  GV ARCLUS procedure, 266
number of observations table
  BNET procedure, 70
  SVMACHINE procedure, 456
optimization
  SVMACHINE procedure, 455
options summary
  PARSE statement, 471
  PROC FACTMAC statement, 115
  PROC FASTKNN statement, 132
  PROC FOREST statement, 155
  PROC GMM statement, 195
PROC GRADBOOST statement, 218
PROC GVARCLUS statement, 258
PROC MTLEARN statement, 292
PROC MWPCA statement, 311
PROC RPCA statement, 360
PROC SEMISUPLEARN statement, 382
PROC TEXTMINE statement, 470
PROC TMSCORE statement, 514
PROC TSNE statement, 520
SELECT statement, 476
SVD statement, 477
output data tables
   FACTMAC procedure, 126
   FASTKNN procedure, 137
   MTLEARN procedure, 304
   RPCA procedure, 371
   SEMISUPLEARN procedure, 388
   TSNE procedure, 526
output network data set
   BNET procedure, 55
OutputCasTables table
   FOREST procedure, 179
   SEMISUPLEARN procedure, 387
parameter learning, 51, 69
parameter tuning
   NNET procedure, 344
PARTITION statement
   syntax (Shared Concepts), 20
   predicted probability names
   FOREST procedure, 179
   GRADBOOST procedure, 243
   Predicted probability variables table
   SVMACHINE procedure, 457
   Predicted target variable table
   SVMACHINE procedure, 457
PredIntoName
   NNET procedure, 347
PredName
   NNET procedure, 348
PredProbName
   NNET procedure, 347
preprocessing
   GVARCLUS procedure, 265
preselection
   FOREST procedure, 170
PROC ASTORE, 27
PROC ASTORE features, 28
PROC BNET features
   BNET procedure, 48
PROC BNET syntax
   BNET procedure, 52
PROC FISM, 139
PROC FISM features, 139
PROC MBANALYSIS, 273
PROC MBANALYSIS features, 274
PROC MWPCA, 305
PROC MWPCA features, 306
PROC RPCA, 355
PROC RPCA features, 356
PROC SVMACHINE, 443
PROC SVMACHINE features, 444
radial basis function bandwidth parameter
   SVDD procedure, 407
rba variable importance
   variable importance, 242
ROC curve
   NNET procedure, 347
RPCA procedure
   details, 368
   dimensions, 370
   displayed output, 370
   examples, 372
   input data tables, 362
   model information, 370
   ODS table names, 371
   output data tables, 371
   results summary, 371
score information
   NNET procedure, 346
scoring process
   SVMACHINE procedure, 456
scoring with PROC ASTORE
   MBANALYSIS procedure, 281
selection
   GVARCLUS procedure, 266
SEMISUPLEARN procedure, 379
   displayed output, 387
   input data tables, 383
   model information, 387
   ODS table names, 387
   output data tables, 385, 388
   OutputCasTables table, 387
Shared Concepts
   AUTOTUNE statement, 14
   CODE statement, 19
   PARTITION statement, 20
solvers
   SVDD procedure, 412
sparse matrix
   TEXTMINE procedure, 489
splitting criteria
   FOREST procedure, 169
stochastic subset solver
   SVDD procedure, 413
structure learning, 50, 68
BNET procedure, 54, 55
subsample the data
  GRADBOOST procedure, 233
SVDD formulation with weight variable
SVDD procedure, 411
SVDD procedure
  active-set solver, 412
  bandwidth relative scale, 410
  comparison with existing methods, 411
  fraction outlier parameter, 410
  input data tables, 399
  linear kernels, 403
  mathematical formulation, 404
  mean criteria, 409
  modified mean criteria, 409
  nonlinear kernels, 403
  ODS table names, 415
  radial basis function bandwidth parameter, 407
  solvers, 412
  stochastic subset solver, 413
SVDD formulation with weight variable, 411
SVMACHINE procedure
  computational method, 22
  displayed output, 456
  fit statistics table, 457
  hyperparameter tuning, 23
  input data tables, 447
  iteration history table, 457
  iteration report table, 457
  linear kernels, 454
  misclassification matrix table, 457
  model information table, 456
  multithreading, 22
  nonlinear kernels, 454
  number of observations table, 456
  ODS table names, 457
  optimization, 455
  output data tables, 452
  Predicted probability variables table, 457
  Predicted target variable table, 457
  scoring process, 456
  test data, 21
  training results table, 457
  validation, 21

target variables
  NNET procedure, 340
test data
  ASTORE procedure, 21
  BOOLRULE procedure, 21
  FACTMAC procedure, 21
  FOREST procedure, 21
  GRADBOOST procedure, 21
  NNET procedure, 21
model information, 525
number of observations, 525
ODS table names, 526
output data tables, 524, 526

*subject index*

- **tuner information**
  - FOREST procedure, 178
  - GRADBOOST procedure, 242
  - NNET procedure, 347

- **tuner results**
  - FOREST procedure, 179
  - GRADBOOST procedure, 242
  - NNET procedure, 347

- **tuner summary**
  - FOREST procedure, 178
  - GRADBOOST procedure, 242
  - NNET procedure, 347

- **tuner timing**
  - FOREST procedure, 179
  - GRADBOOST procedure, 242
  - NNET procedure, 347

- **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, 71

- **variable importance**
  - FOREST procedure, 178
  - GRADBOOST procedure, 242
  - rba variable importance, 242

- **variable information table**
  - BNET procedure, 70

- **variable level table**
  - BNET procedure, 70

- **variable order table**
  - BNET procedure, 70

- **variable selection, 51, 68**
  - BNET procedure, 55, 56

- **weight variable**
  - NNET procedure, 343
Syntax Index

ACT= option
  HIDDEN statement, 335
  TARGET statement, 340
ACTSET option
  SOLVER statement, 402
ADD option
  VIICODE statement (FOREST), 168
  VIICODE statement (GRADBOOST), 233
ADDTREES option
  PROC GRADBOOST statement, 219
ALGORITHM= option
  OPTIMIZATION statement, 337
ALPHA= option
  PROC GMM statement, 195
ANNEALINGRATE= option
  OPTIMIZATION statement, 337
ANTECEDENTLIST= option
  PROC MBANALYSIS statement, 276
ARCHITECTURE statement
  NNET procedure, 329
ASCENDING option
  TARGET statement, 454
ASSIGNMISSING= option
  PROC FOREST statement, 155
  PROC GRADBOOST statement, 219
ASTORE procedure
  PROC ASTORE statement, 31
ASTORE procedure, CODE statement
  COMMENT option, 19
  FILE= option, 19
  FORMATWIDTH= option, 20
  INDENTSIZE= option, 20
  LABELID= option, 20
  LINESIZE= option, 20
  NOTRIM option, 20
  OUT= option, 20
ASTORE procedure, DESCRIBE statement, 32
  EPCODE= option, 32
  RSTORE= option, 32
  STORE= option, 32
ASTORE procedure, DOWNLOAD statement, 32
  RSTORE= option, 32
  STORE= option, 33
ASTORE procedure, PROC ASTORE statement, 31
  DATA= option, 33
ASTORE procedure, SCORE statement, 33
  COPYV AR= option, 34
  OUT= option, 33
ASTORE procedure, SETOPTION statement, 34
ASTORE procedure, syntax, 31
ASTORE procedure, UPLOAD statement, 37
  RSTORE= option, 37
  STORE= option, 37
AUTOTUNE statement
  PROC ASTORE procedure, 14
  BNET procedure, 57
  BOOLRULE procedure, 14
  FACTMAC procedure, 14
  FOREST procedure, 14, 160
  GRADBOOST procedure, 14, 117, 223
  NNET procedure, 14, 330
  SVMACHINE procedure, 14, 448
  TMSCORE procedure, 14
BINMETHOD = option
  PROC FOREST statement, 156
  PROC GRADBOOST statement, 219
BNET procedure, 52
  CODE statement, 63
  FREQ statement, 64
  ID statement, 64
  INPUT statement, 64
  OUTPUT statement, 64
  PROC BNET statement, 52
  SAVESTATE statement, 66
  syntax, 52
  TARGET statement, 66
BNET procedure, AUTOTUNE statement, 57
  TUNINGPARAMETERS= option, 58
BNET procedure, OUTPUT statement
  COPYV ARS= option, 65
  OUT= option, 65
  ROLE= option, 65
BNET procedure, PARTITION statement
  FRACTION option, 65
  ROLEVAR= option, 66
BNET procedure, PROC BNET statement
  DATA= option, 53
  NTHREADS= option, 55
BNET procedure, SAVESTATE statement
  RSTORE= option, 66
BOOLRULE procedure, 87
  DOCINFO statement, 90
  OUTPUT statement, 91
  PROC BOOLRULE statement, 87
SCORE statement, 92
syntax, 87
TERMINFO statement, 92

BOOLRULE procedure, CODE statement
COMMENT option, 19
FILE= option, 19
FORMATWIDTH= option, 20
INDENTSIZE= option, 20
LABELID= option, 20
LINESIZE= option, 20
NOTRIM option, 20
OUT= option, 20

BOOLRULE procedure, DOCINFO statement, 90
EVENTS= option, 90
ID= option, 90
TARGET= option, 91
TARGETTYPE= option, 91

BOOLRULE procedure, OUTPUT statement, 91
CANDIDATETERMS= option, 91
RULES= option, 91
RULETERMS= option, 92

BOOLRULE procedure, PROC BOOLRULE statement, 87
DATA= option, 88
DOC= option, 88
DOCID= option, 89
DOCINFO= option, 89
GNEG= option, 89
GPOS= option, 89
MAXCANDIDATES= option, 89
MAXCANDS= option, 89
MAXTRIESIN= option, 89
MAXTRIESOUT= option, 89
MINSUPPORTS= option, 89
MNEG= option, 90
MPOS= option, 90
TERMID= option, 90
TERMINFO= option, 90

BOOLRULE procedure, SCORE statement, 92
OUTMATCH= option, 92
RULETERMS= option, 92

BOOLRULE procedure, TERMINFO statement, 92
ID= option, 93
LABEL= option, 93

BURN= option
TRANSFERLEARN statement, 232

C= option
PROC SVMACHINE statement, 447

CANDIDATETERMS= option
OUTPUT statement, 91

CASESENSITIVE option
DISPLAY statement (FACTMAC), 121
DISPLAY statement (FASTKNN), 134
DISPLAY statement (GMM), 198
DISPLAY statement (GVARCLUS), 262
DISPLAY statement (MTLEARN), 295, 384
DISPLAY statement (MWPCA), 313
DISPLAY statement (RPCA), 365
DISPLAY statement (TSNE), 522

CELLWGT= option
PARSE statement, 472

CENTER option
PROC MWPCA statement, 311
PROC RPCA statement, 361

CHAID option
GROW statement (FOREST), 164

CHISQUARE option
GROW statement (FOREST), 165

CLUSTERCOVOUT= option
PROC GMM statement, 195

CLUSTERSUMOUT= option
PROC GMM statement, 196

CODE statement
ASTORE procedure, 19, 32
BNET procedure, 63
BOOLRULE procedure, 19
FACTMAC procedure, 19
FOREST procedure, 19, 163
GRADBOOST procedure, 19, 228
NNET procedure, 19, 334
RPCA procedure, 364
SVDD procedure, 400
SVMACHINE procedure, 19, 451
TEXTMINE procedure, 19
TMSCORE procedure, 19

COL= option
SVD statement, 478

COLSTATISTICS= option
PROC RPCA statement, 361

COMB= option
HIDDEN statement, 336
TARGET statement, 341

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, 337

CONF= option
PROC MBANALYSIS statement, 276

CONFIG= option
TMSCORE statement, 515
CONSEQUENTLIST= option
PROC MBANALYSIS statement, 277
COPYVAR= option
SCORE statement (ASTORE), 34
COPYVARS= option
OUTPUT statement, 65, 123, 136, 166, 200, 230, 297, 339, 386, 453, 524
CROSSVALIDATION statement
FOREST procedure, 164
GRADBOOST procedure, 228
NNET procedure, 335
CTOL= option
SOLVER statement, 403
CUMEIGPCTTOL= option
PROC MWPCA statement, 311
PROC RPCA statement, 361
CUSTOMER statement
FISM procedure, 142
MBANALYSIS procedure, 278
DATA= option
HIERARCHY statement, 278
PROC ASTORE statement, 33
PROC BNET statement, 53
PROC BOORULE statement, 88
PROC FACTMAC statement, 116
PROC FASTKNN statement, 132
PROC FISM statement, 142
PROC FOREST statement, 156
PROC GMM statement, 196
PROC GRADBOOST statement, 220
PROC GVARCLUS statement, 259
PROC MBANALYSIS statement, 276
PROC MTLEARN statement, 293
PROC MWPCA statement, 311
PROC NNET statement, 327
PROC RPCA statement, 362
PROC SEMISUPLEARN statement, 383
PROC SVDD statement, 399
PROC SVMACHINE statement, 447
PROC TEXTMINE statement, 470
PROC TMSCORE statement, 514
PROC TSNE statement, 521
DECOMP= option
PROC RPCA statement, 362
DESCENDING option
TARGET statement, 454
DISPLAY statement
FACTMAC procedure, 120
FASTKNN procedure, 133
GMM procedure, 197
GVARCLUS procedure, 261
MTLEARN procedure, 295, 383
MWPCA procedure, 312
RPCA procedure, 364
TSNE procedure, 522
DISPLAYOUT statement
FACTMAC procedure, 121
FASTKNN procedure, 134
GMM procedure, 198
GVARCLUS procedure, 262
MTLEARN procedure, 296, 384
MWPCA procedure, 313
RPCA procedure, 365
TSNE procedure, 523
DOC= option
PROC BOORULE statement, 88
PROC TMSCORE statement, 514
DOC_ID statement
TEXTMINE procedure, 471
TMSCORE procedure, 516
DOCID= option
PROC BOORULE statement, 89
DOCINFO statement
BOOLRULE procedure, 90
DOCINFO= option
PROC BOORULE statement, 89
DOWNLOAD statement
ASTORE procedure, 32
DROPOUTHIDDEN= option
TRAIN statement, 342
DROPOUTINPUT= option
TRAIN statement, 342
EARLYSTOP option
PROC GRADBOOST statement, 220
ENTITIES= option
PARSE statement, 473
ENTROPY option
GROW statement (FOREST), 165
ENTRY= option
SVD statement, 478
EPCODE= option
DESCRIBE statement, 32
ERROR= option
TARGET statement, 341
EVALHISTORY= option
AUTOTUNE statement, 15
EVENTS= option
DOCINFO statement, 90
EXACT option
PROC GVARCLUS statement, 259
EXACTWEIGHT option
SVD statement, 478
EXCLUDE option
DISPLAY statement (FACTMAC), 121
DISPLAY statement (FASTKNN), 134
**Syntax Index**

DISPLAY statement (GMM), 198
DISPLAY statement (GVARCLUS), 262
DISPLAY statement (MITLEARN), 295, 384
DISPLAY statement (MWPCA), 313
DISPLAY statement (RPCA), 365
DISPLAY statement (TSNE), 523

EXCLUDEALL option
DISPLAY statement (FACTMAC), 121
DISPLAY statement (FASTKNN), 134
DISPLAY statement (GMM), 198
DISPLAY statement (GVARCLUS), 262
DISPLAY statement (MITLEARN), 295, 384
DISPLAY statement (MWPCA), 313
DISPLAY statement (RPCA), 365
DISPLAY statement (TSNE), 523

FACTMAC procedure, AUTOTUNE statement, 117
FACTMAC procedure, CODE statement, 120
   COMMENT option, 19
   FILE= option, 19
   FORMATWIDTH= option, 20
   INDENTSIZE= option, 20
   LABELID= option, 20
   LINESIZE= option, 20
   NOTRIM option, 20
   OUT= option, 20
FACTMAC procedure, DISPLAY statement
   CASESENSITIVE option, 121
   EXCLUDE option, 121
   EXCLUDEALL option, 121
   TRACE option, 121
FACTMAC procedure, DISPLAYOUT statement
   INCLUDEALL option, 121
   NOREPLACE option, 121
   REPEATED option, 121
FACTMAC procedure, ID statement, 122
FACTMAC procedure, INPUT statement, 122
FACTMAC procedure, OUTPUT statement, 122
   COPYVARS= option, 123
   LEVEL= option, 123
   OUT= option, 122
FACTMAC procedure, PROC FACTMAC statement, 115
   DATA= option, 116
   FILE= option, 120
   LEARNSTEP= option, 116
   LEVEL= option, 122
   MAXITER= option, 116
   NFACTORS= option, 116
   NONNEGATIVE option, 116
   NOPRINT, 116
   NTHREADS= option, 116
   OUTMODEL, 116
   SEED= option, 117
FACTMAC procedure, SAVESTATE statement, 123
   RSTORE= option, 123
FACTMAC procedure, syntax, 115
FACTMAC procedure, TARGET statement, 123
FASTKNN procedure, DISPLAY statement
   CASESENSITIVE option, 134
   EXCLUDE option, 134
   EXCLUDEALL option, 134
   TRACE option, 134
FASTKNN procedure, DISPLAYOUT statement
   INCLUDEALL option, 135
   NOREPLACE option, 135
   REPEATED option, 135
FASTKNN procedure, ID statement, 135
FASTKNN procedure, INPUT statement, 135
FASTKNN procedure, OUTPUT statement, 135
   COPYVARS= option, 136
   OUT= option, 136
FASTKNN procedure, PROC FASTKNN statement, 132
   DATA= option, 132
   K= option, 133
   NOPRINT option, 133
   NTHREADS= option, 133
   OUTDIST= option, 133
   PARALLELIZATION= option, 133
   QUERY= option, 133
   THRESHDIST= option, 133
FASTKNN procedure, syntax, 131
   FILE= option
   CODE statement, 335, 364, 400, 451
   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
   PROC FACTMAC statement, 120
FISM procedure, CUSTOMER statement, 142
FISM procedure, OUTPUT statement, 143
   OUT= option, 143
   OUTFREQ= option, 143
FISM procedure, PROC FISM statement, 141
   DATA= option, 142
   ITEMS= option, 142
   NFIS_RANGE= option, 142
   PCTSUP= option, 142
   SUPPORT= option, 142
FISM procedure, syntax, 141
FISM procedure, TARGET statement, 143
FIXEDMU option
RPCA statement, 315
PROC RPCA statement, 362
FOREST procedure, AUTOTUNE statement, 160
TUNINGPARAMETERS= option, 160
FOREST procedure, CODE statement, 163
COMMENT option, 19
FILE= option, 19
FORMATWIDTH= option, 20
INDENTSIZE= option, 20
LABELID= option, 20
LINESIZE= option, 20
NOTRIM option, 20
OUT= option, 20
FOREST procedure, CROSSVALIDATION statement, 164
KFOLD= option, 164
NOPARALLEL option, 164
NSUBSESSIONWORKERS= option, 164
FOREST procedure, GROW statement, 164
CHAID option, 164
CHISQUARE option, 165
ENTROPY option, 165
FTEST option, 165
GINI option, 165
IGR option, 165
RSS option, 165
FOREST procedure, ID statement, 165
FOREST procedure, INPUT statement, 165
LEVEL= option, 165
FOREST procedure, OUTPUT statement, 166
COPYVARS= option, 166
OUT= option, 166
ROLE option, 166
FOREST procedure, PARTITION statement
FRACTION option, 20, 167
ROLEVAR= option, 21, 167
FOREST procedure, PROC FOREST statement, 155
ASSIGNMISSING= option, 155
BINMETHOD = option, 156
DATA= option, 156
INBAGFRACTION= option, 157
INMODEL= option, 157
ISOLATION option, 157
LOH= option, 157
MAXBRANCH= option, 158
MAXDEPTH= option, 158
MINLEAFSIZE= option, 158
MINUSEINSEARCH= option, 158
NOPRINT option, 158
NTREES= option, 158
NUMBIN= option, 158
OUTMODEL= option, 158
PRINTTARGET option, 158
RBAIMP option, 159
SEED= option, 159
VARS_TO_TRY= option, 159
VII= option, 159
VOTE= option, 159
FOREST procedure, SAVESTATE statement, 167
RSTORE= option, 167
FOREST procedure, syntax, 154
FOREST procedure, TARGET statement, 168
LEVEL= option, 168
FOREST procedure, VIICODE statement, 168
ADD option, 168
LIMIT= option, 168
MISS option, 168
MULTIPLY option, 168
SUBTRACT option, 168
THRESHOLD= option, 168
FORMATWIDTH= 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
FRACTION option
PARTITION statement, 340
PARTITION statement (BNET), 65
PARTITION statement (FOREST), 20, 167
PARTITION statement (GRADBOOST), 20, 231
PARTITION statement (SVMACHINE), 453
FRACTION= option
AUTOTUNE statement, 15
FREQ statement
BNET procedure, 64
GVARCLUS procedure, 263
FTEST option
GROW statement (FOREST), 165
GAMMA= option
PROC SEMISUPLEARN statement, 383
GINI option
GROW statement (FOREST), 165
GLIM option
ARCHITECTURE statement, 329
GMM procedure, DISPLAY statement
CASESENSITIVE option, 198
EXCLUDE option, 198
EXCLUDEALL option, 198
TRACE option, 198
GMM procedure, DISPLAYOUT statement
INCLUDEALL option, 198
NOREPLACE option, 198
REPEATED option, 198
GMM procedure, INPUT statement, 199
GMM procedure, OUTPUT statement
COPYVARS= option, 200
OUT= option, 199
GMM procedure, PROC GMM statement, 195
ALPHA= option, 195
CLUSTERCOVOUT= option, 195
CLUSTERSUMOUT= option, 196
DATA= option, 196
MAXCLUSTERS= option, 197
NTHREADS= option, 197
SEED= option, 197
GMM procedure, SAVESTATE statement, 199
RSTORE= option, 199
GMM procedure, SCORE statement, 199
GMM procedure, syntax, 194
GNEG= option
PROC BOOLRULE statement, 89
GPOS= option
PROC BOOLRULE statement, 89
GRADBOOST procedure, AUTOTUNE statement, 223
TUNINGPARAMETERS= option, 118, 224
GRADBOOST procedure, CODE statement, 228
COMMENT option, 19
FILE= option, 19
FORMATWIDTH= option, 20
INDENTSIZE= option, 20
LABELID= option, 20
LINESIZE= option, 20
NOTRIM option, 20
OUT= option, 20
GRADBOOST procedure, CROSSVALIDATION statement, 228
KFOLD= option, 229
NOPARALLEL option, 229
NSUBSESSIONWORKERS= option, 229
GRADBOOST procedure, ID statement, 229
GRADBOOST procedure, INPUT statement, 229
LEVEL= option, 229
MONOTONIC = option, 229
GRADBOOST procedure, OUTPUT statement, 230
COPYVARS= option, 230
OUT= option, 230
ROLE option, 230
GRADBOOST procedure, PARTITION statement
FRACTION option, 20, 231
ROLEVAR= option, 21, 231
GRADBOOST procedure, PROC GRADBOOST statement, 218
ADDTREES option, 219
ASSIGNMISSING= option, 219
BINMETHOD = option, 219
DATA= option, 220
EARLYSTOP option, 220
INMODEL= option, 220
LASSO= option, 221
LEARNINGRATE= option, 221
MAXBRANCH= option, 221
MAXDEPTH= option, 221
MINLEAFSIZE= option, 221
MINUSEINSEARCH= option, 221
NOPRINT option, 221
NTREES= option, 222
NUMBIN= option, 222
OUTMODEL option, 222
PRINTTARGET option, 222
RBAIMP option, 222
RIDGE= option, 222
SAMPLINGRATE= option, 222
SEED= option, 222
VARS_TO_TRY= option, 223
VII= option, 223
GRADBOOST procedure, SAVESTATE statement, 231
RSTORE= option, 231
GRADBOOST procedure, syntax, 217
GRADBOOST procedure, TARGET statement, 232
LEVEL= option, 232
GRADBOOST procedure, TRANSFERLEARN statement, 232
BURN = option, 232
SHRINKAGE = option, 232
TRIMMING = option, 232
GRADBOOST procedure, VIICODE statement, 232
ADD option, 233
LIMIT= option, 233
MISS option, 233
MULTIPLY option, 233
SUBTRACT option, 233
THRESHOLD= option, 233
GRAPHOUT= option
PROC MTLEARN statement, 293
GRAPHTABLE= option
PROC MTLEARN statement, 293
GRAPHTYPE= option
PROC MTLEARN statement, 293
GROUP= option
SELECT statement, 477
GROW statement
FOREST procedure, 164
GVARCLUS procedure, DISPLAY statement
CASESENSITIVE option, 262
EXCLUDE option, 262
EXCLUDEALL option, 262
TRACE option, 262
GVARCLUS procedure, DISPLAYOUT statement
INCLUDEALL option, 263  
NOREPLACE option, 263  
REPEATED option, 263  
GVARCLUS procedure, INPUT statement, 263  
LEVEL= option, 263  
GVARCLUS procedure, PROC GVARCLUS statement, 258  
DATA= option, 259  
EXACT option, 259  
MAXITER= option, 260  
MAXMEMBER= option, 260  
MAXSTEP= option, 260  
MINCLUSTER= option, 260  
OUTCP= option, 260  
OUTEDGE= option, 260  
OUTTRE= option, 261  
OUTVERT= option, 261  
RHO= option, 261  
SELECT= option, 261  
STOP= option, 261  
TOL= option, 261  
TOLERANCE= option, 261  
GVARCLUS procedure, syntax, 258  
HIDDEN statement  
NNET procedure, 335  
HIERARCHY statement  
MBANALYSIS procedure, 278  
ID statement  
BNET procedure, 64  
FACTMAC procedure, 122  
FASTKNN procedure, 135  
FOREST procedure, 165  
GRADBOOST procedure, 229  
MWPCA procedure, 314  
RPCA procedure, 366  
SVD procedure, 400  
SVMACHINE procedure, 451  
ID= option  
DOCINFO statement, 90  
TERMINFO statement, 93  
IGNORE option  
SELECT statement, 477  
IGR option  
GROW statement (FOREST), 165  
IN_TERMS= option  
SVD statement, 478  
INBAGFRACTION= option  
PROC FOREST statement, 157  
INCLUDEALL option  
DISPLAYOUT statement (FACTMAC), 121  
DISPLAYOUT statement (FASTKNN), 135  
DISPLAYOUT statement (GMM), 198  
DISPLAYOUT statement (GVARCLUS), 263  
DISPLAYOUT statement (MTLEARN), 296, 385  
DISPLAYOUT statement (MWPCA), 313  
DISPLAYOUT statement (RPCA), 366  
DISPLAYOUT statement (TSNE), 523  
INDENTSIZE= 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  
INMODEL= option  
PROC FOREST statement, 157  
PROC GRADBOOST statement, 220  
PROC NNET statement, 328  
INPUT statement  
BNET procedure, 64  
FACTMAC procedure, 122  
FASTKNN procedure, 135  
FOREST procedure, 165  
GMM procedure, 199  
GRADBOOST procedure, 229  
GVARCLUS procedure, 263  
MTLEARN procedure, 296  
MWPCA procedure, 314  
NNET procedure, 336  
RPCA procedure, 366  
SEMISUPLEARN procedure, 385  
SVDD procedure, 400  
SVMACHINE procedure, 451  
TSNE procedure, 523  
ISOLATION option  
PROC FOREST statement, 157  
ITEMS= option  
PROC FISM statement, 142  
PROC MBANALYSIS statement, 277  
ITERATIONREPORT option  
PROC SVMACHINE statement, 447  
K= option  
PROC FASTKNN statement, 133  
SVD statement, 479  
KEEP option  
SELECT statement, 477  
KEEPVARS, KEEPVARIABLES  
SVD statement, 479  
KERNEL statement  
SVDDD procedure, 401  
SVMACHINE procedure, 452  
K FOLD= option
AUTOTUNE statement, 15
CROSSVALIDATION statement, 164, 229, 335

LABEL= option
PROC SEMISUPLEARN statement, 383
TERMININFO statement, 93

LABELID= 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

LABELS option
SELECT statement, 477

LAMBDA= option
RPCA statement, 315
PROC RPCA statement, 362

LAMBDAWEIGHT= option
RPCA statement, 315
PROC RPCA statement, 362

LANGUAGE= option
PROC TEXTMINE statement, 471

LASSO= option
PROC GRADBOOST statement, 221

LEANINGRATE= option
PROC GRADBOOST statement, 221
PROC TSNE statement, 521

LEANINGRATE= option
PROC TSNE statement, 521

LEVEL= option
INPUT statement, 165, 229, 263, 336, 401, 451
OUTPUT statement, 123
PROC FACTMAC statement, 122
PROC MTLEARN statement, 296
TARGET statement, 168, 232, 298, 341

LIFT= option
PROC MBANALYSIS statement, 277

LIMIT= option
VIIICODE statement (FOREST), 168
VIIICODE statement (GRADBOOST), 233

LINEAR option
PROC SVMACHINE statement, 452

LINESIZE= option
CODE statement (ASTORE), 20
CODE statement (BOOLRULE), 20
CODE statement (FACTMAC), 20
CODE statement (FOREST), 20
CODE statement (GRADBOOST), 20

LOH= option
PROC FOREST statement, 157

MAX_K= option
SVD statement, 479

MAXBAYES= option
PROC FOREST statement, 158
PROC GRADBOOST statement, 221

MAXBSKTSZ= option
PROC FOREST statement, 158

MAXCANDIDATES= option
PROC BOOLRULE statement, 89

MAXCANDS= option
PROC BOOLRULE statement, 89

MAXCLUSTERS= option
PROC GMM statement, 197

MAXDEPTH= option
PROC FOREST statement, 158
PROC GRADBOOST statement, 221

MAXEVALS= option
PROC MBANALYSIS statement, 277

MAXITER= option
PROC GRADBOOST statement, 221

MAXVARS= option
PROC FACTMAC statement, 116

OPTIMIZATION statement, 337

MAXITERS= option
PROC TBANALYSIS statement, 277

MAXMEMBER= option
PROC SVMACHINE statement, 452

MAXRANK= option
PROC SVMACHINE statement, 452

MAXSV= option
PROC SVMACHINE statement, 452

MAXTIME= option
PROC SVMACHINE statement, 452

MAXTRYIN= option
PROC SVMACHINE statement, 452

MAXTRYOUT= option
PROC SVMACHINE statement, 452
PROC BOOLRULE statement, 89  
MBANALYSIS procedure, CUSTOMER statement, 278  
MBANALYSIS procedure, HIERARCHY statement, 278  
   DATA= option, 278  
MBANALYSIS procedure, OUTPUT statement, 278  
   OUT= option, 278  
   OUTFREQ= option, 278  
   OUTRULE= option, 279  
MBANALYSIS procedure, PROC MBANALYSIS statement, 276  
   ANTECEDENTLIST= option, 276  
   CONF= option, 276  
   CONSEQUENTLIST= option, 277  
   DATA= option, 276  
   ITEMS= option, 277  
   LIFT= option, 277  
   MAXBSKTSZ= option, 277  
   MINBSKTSZ= option, 277  
   NLHS_RANGE= option, 277  
   NORM= option, 277  
   NRHS_RANGE= option, 277  
   PCTSUP= option, 276  
   SEPARATOR= option, 277  
   SUP_LIFT= option, 277  
   SUPPORT= option, 276  
MBANALYSIS procedure, SAVESTATE statement, 279  
   RSTORE= option, 279  
MBANALYSIS procedure, syntax, 275  
MBANALYSIS procedure, TARGET statement, 279  
METHOD= option  
   PROC RPCA statement, 363  
   RPCA statement, 316  
   SVD statement, 316, 368  
MINBSKTSZ= option  
   PROC MBANALYSIS statement, 277  
MINCLUSTER= option  
   PROC GVARCLUS statement, 260  
MINIBATCHSIZE= option  
   OPTIMIZATION statement, 338  
MINLEAFSIZE= option  
   PROC FOREST statement, 158  
   PROC GRADBOOST statement, 221  
MINSUPPORTS= option  
   PROC BOOLRULE statement, 89  
MINUSEINSEARCH= option  
   PROC FOREST statement, 158  
   PROC GRADBOOST statement, 221  
MISS option  
   VIICODE statement (FOREST), 168  
   VIICODE statement (GRADBOOST), 233  
MISSING= option  
   PROC NNET statement, 328  
   MLP DIRECT option  
      ARCHITECTURE statement, 329  
   MLP option  
      ARCHITECTURE statement, 329  
   MNEG= option  
      PROC BOOLRULE statement, 90  
   MODELOUT= option  
      PROC MBANALYSIS statement, 294  
   MOMENTUM= option  
      OPTIMIZATION statement, 338  
   MONOTONIC = option  
      INPUT statement, 229  
MPOS= option  
   PROC BOOLRULE statement, 90  
MTLEARN procedure, DISPLAY statement  
      CASESENSITIVE option, 295, 384  
      EXCLUDE option, 295, 384  
      EXCLUDEALL option, 295, 384  
      TRACE option, 295, 384  
MTLEARN procedure, DISPLAYOUT statement  
      INCLUDEALL option, 296, 385  
      NOREPLACE option, 296, 385  
      REPEATED option, 296, 385  
MTLEARN procedure, INPUT statement, 296  
MTLEARN procedure, OUTPUT statement, 297  
   COPYVARS= option, 297  
   OUT= option, 297  
MTLEARN procedure, PROC MTLEARN statement, 292  
   DATA= option, 293  
   GRAPHTYPEOUT= option, 293  
   GRAPHOUT= option, 293  
   GRAPHTABLE= option, 293  
   LEVEL= option, 296  
   MAXITERS= option, 294  
   MODELOUT= option, 294  
   NTTHREADS= option, 294  
   SEED= option, 294  
   TOLERANCE= option, 294  
MTLEARN procedure, SAVESTATE statement, 297  
   RSTORE= option, 297  
MTLEARN procedure, syntax, 292  
MTLEARN procedure, TARGET statement, 297  
   LEVEL= option, 298  
MU= option  
   RPCA statement, 316  
   PROC RPCA statement, 363  
MULTIPLY option  
   VIICODE statement (FOREST), 168  
   VIICODE statement (GRADBOOST), 233  
MULTITERM= option  
   PARSE statement, 473  
MWPCA procedure, DISPLAY statement
Syntax Index

CASESENSITIVE option, 313
EXCLUDE option, 313
EXCLUDEALL option, 313
TRACE option, 313

MWPCA procedure, DISPLAYOUT statement
  INCLUDETALL option, 313
  NOREPLACE option, 313
  REPEATED option, 313

MWPCA procedure, ID statement, 314
MWPCA procedure, INPUT statement, 314
MWPCA procedure, OUTPUT statement, 314

  NPC= option, 315
  OUT= option, 314
  STANDARDPC= option, 315

MWPCA procedure, PROC MWPCA statement, 311
  CENTER, 311
  CUMEIGPCTTOL= option, 311
  DATA= option, 311
  NTHREADS= option, 312
  ROBUST= option, 312
  STEPSIZE= option, 311
  WINDOWSIZE= option, 311

MWPCA procedure, RPCA statement, 315
  FIXEDMU option, 315
  LAMBDA= option, 315
  LAMBDAWEIGHT= option, 315
  MAXITER= option, 316
  METHOD= option, 316
  MU= option, 316
  TOLERANCE= option, 316

MWPCA procedure, SVD statement, 316
  MAXRANK= option, 316

MWPCA procedure, syntax, 310

NBINS option
  PROC SVDD statement, 399
NDIMENSIONS= option
  PROC TSNE statement, 521
NEWVARNAMES
  TEXTMINE statement, 471
NFACTORS= option
  PROC FACTMAC statement, 116
NFIS_RANGE= option
  PROC FISM statement, 142
NLHS_RANGE= option
  PROC MBANALYSIS statement, 277
NMATCH = option
  SOLVER statement, 403

NNET procedure, ARCHITECTURE statement, 329
  GLIM option, 329
  MLP DIRECT option, 329
  MLP option, 329

NNET procedure, AUTOTUNE statement, 330
  TUNINGPARAMETERS= option, 330

NNET procedure, CODE statement, 334
  COMMENT option, 19
  FILE= option, 19, 335
  FORMATWIDTH= option, 20
  INDENTSIZE= option, 20
  LABELID= option, 20
  LINESIZE= option, 20
  NOCOMPPGM option, 335
  NOTRIM option, 20
  OUT= option, 20

NNET procedure, CROSSVALIDATION statement, 335
  KFOLD= option, 335
  NOPARALLEL option, 335
  NSUBSESSIONWORKERS= option, 335

NNET procedure, HIDDEN statement, 335
  ACT= option, 335
  COMB= option, 336

NNET procedure, INPUT statement, 336
  LEVEL= option, 336

NNET procedure, OPTIMIZATION statement, 336
  ALGORITHM= option, 337
  ANNEALINGRATE= option, 337
  COMMFREQ= option, 337
  LEARNINGRATE= option, 337
  MAXITER= option, 338
  MAXTIME= option, 338
  MINIBATCHSIZE= option, 338
  MOMENTUM= option, 338
  REGL1=option, 338
  REGL2= option, 338
  SEED= option, 338
  USELOCKING option, 338

NNET procedure, OUTPUT statement, 338
  COPYVARS= option, 339
  OUT= option, 339
  ROLE= option, 339

NNET procedure, PARTITION statement, 339
  FRACTION option, 340
  ROLEVAR= option, 340

NNET procedure, PROC NNET statement, 327
  DATA= option, 327
  INMODEL= option, 328
  MISSING= option, 328
  NTHREADS= option, 328
  PRINTTARGET, 328
  STANDARDIZE= option, 328
  TARGETSTANDARDIZE= option, 329

NNET procedure, syntax, 327

NNET procedure, TARGET statement, 340
  ACT= option, 340
  COMB= option, 341
  ERROR= option, 341
  LEVEL= option, 341
Syntax Index

NNET procedure, TRAIN statement, 341
   DROPOUTHIDDEN= ratio, 342
   DROPOUTINPUT= option, 342
   NUMTRIES= option, 342
   OUTMODEL= option, 342
   RESUME option, 343
   STAGNATION= option, 343
   VALIDATION= option, 343
   VALIDGOAL= option, 343
   WSEED= option, 343
NNET procedure, WEIGHT statement, 343
NOCOMPPGM option
   CODE statement, 335
NOCUTOFFS option
   CODE statement, 335
SVD statement, 479
   CODE statement, 335
NONG option
   CODE statement, 473
NONNEGATIVE option
   CODE statement, 473
PROC FACTMAC statement, 116
NONOUNGROUPS option
   CODE statement, 473
NOPARALLEL option
   CODE statement, 473
   CROSSVALIDATION statement, 164, 229, 335
NPRINT
   CODE statement, 116
NPRINT option
   CODE statement, 335
PROC FACTMAC statement, 116
PROC FASTKNN statement, 133
PROC FOREST statement, 158
PROC GRADBOOST statement, 221
PROC SVDD statement, 399
PROC SVMACHINE statement, 448
PROC TSNE statement, 521
NOREPLACE 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, 315
NRHS_RANGE= option
   PROC MBANALYSIS statement, 277
NSAMP= 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, 143, 278
   OUTPUT statement (BNET), 65
NORM= option
   PROC MBANALYSIS statement, 277
NOSCALE option
   PROC SVMACHINE statement, 448
NOSTEMMING option
   CODE statement, 473
NOTAGGING option
   CODE statement, 473
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
   OUTPUT statement, 315
OUTPUT statement (FACTMAC), 122
OUTPUT statement (FASTKNN), 136
OUTPUT statement (FOREST), 166
OUTPUT statement (GMM), 199
OUTPUT statement (GRADBOOST), 230
OUTPUT statement (MTLEARN), 297
OUTPUT statement (MWPCA), 314
OUTPUT statement (SEMISUPLEARN), 385
OUTPUT statement (SVMACHINE), 452
OUTPUT statement (TSNE), 524
SCORE statement (ASTORE), 33
OUTCHILD= option
PARSE statement, 474
OUTCONFIG= option
PARSE statement, 474
OUTCP= option
PROC GV ARCLUS statement, 260
OUTDECOMP statement
RPCA procedure, 366
OUTDIST= option
PROC FASTKNN statement, 133
OUTDOCPRO= option
SVD statement, 479
OUTEDGE= option
PROC GV ARCLUS statement, 260
OUTERROR= option
PROC RPCA statement, 363
OUTFREQ= option
OUTPUT statement, 143, 278
OUTLIER_FRACTION= option
PROC SVDD statement, 399
OUTLOWRANK= option
PROC RPCA statement, 363
OUTMATCH= option
SCORE statement, 92
OUTMODEL
PROC FACTMAC statement, 116
OUTMODEL option
PROC GRADBOOST statement, 222
OUTMODEL= option
PROC FOREST statement, 158
TRAIN statement, 342
OUTPARENT= option
PARSE statement, 474
PROC TMSCORE statement, 515
OUTPOS= option
PARSE statement, 474
OUTPUT statement
BNET procedure, 64
BOOLRULE procedure, 91
FACTMAC procedure, 122
FASTKNN procedure, 135
FISM procedure, 143
FOREST procedure, 166
GRADBOOST procedure, 230
MBANALYSIS procedure, 278
MTLEARN procedure, 297
MWPCA procedure, 314
NNET procedure, 338
SEMISUPLEARN procedure, 385
SVMACHINE procedure, 452
TSNE procedure, 524
OUTRULE= option
OUTPUT statement, 279
OUTSPARSE= option
PROC RPCA statement, 363
OUTSV= option
PROC SVDD statement, 400
OUTTERMS= option
PARSE statement, 474
OUTTOPICS= option
SVD statement, 480
OUTTREE= option
PROC GV ARCLUS statement, 261
OUTVERT= option
PROC GV ARCLUS statement, 261
PARALLELIZATION= option
PROC FASTKNN statement, 133
PARSE statement
TEXTMINE procedure, 471
PARTITION statement
BNET procedure, 65
FOREST procedure, 20, 167
GRADBOOST procedure, 20, 231
NNET procedure, 339
SVMACHINE procedure, 453
PCLOADINGS= option
OUTDECOMP statement, 367
PCSCORES= option
OUTDECOMP statement, 367
PCTSUP= option
OUTDECOMP statement, 367
PCTSUP= option
PROC FISM statement, 142
PROC MBANALYSIS statement, 276
PERPLEXITY= option
PROC TSNE statement, 522
POLYNOMIAL option
PROC SVMACHINE statement, 452
POPSIZE= option
AUTOTUNE statement, 16, 17
POWER= option
SVD statement, 368
PREFIX= option
PROC RPCA statement, 363
PRINTTARGET
PROC NNET statement, 328
PRINTTARGET option
PROC BNET statement, 56
PROC FOREST statement, 159
PROC GRADBOOST statement, 222
PROC SVMACHINE statement, 448

PROC ASTORE statement
ASTORE procedure, 31

PROC BNET statement
  ALPHA= option, 53
  BESTMODEL option, 53
  BNET procedure, 52
  INDEPTTEST= option, 54
  MIALPHA= option, 54
  MISSINGINT= option, 54
  MISSINGNOM= option, 55
  NTHREADS= option, 55
  NUMBIN= option, 55
  OUTNETWORK= option, 55
  PARENTING= option, 55
  PRESCREENING= option, 55
  STRUCTURE= option, 56
  VARESELECT= option, 56

PROC BOOLRULE statement
BOOLRULE procedure, 87

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 MTLREARN statement, see MTLREARN procedure
PROC MWPCA statement, see MWPCA procedure
PROC NNET statement, see NNET procedure
PROC RPCA statement, see RPCA procedure
PROC SEMISUPLEARN statement, see SEMISUPLEARN procedure
PROC SVDD statement, see SVDD procedure
PROC SVMACHINE statement, see SVMACHINE procedure
PROC TEXTMINE statement
TEXTMINE procedure, 470

PROC TMSCORE statement
TMSCORE procedure, 514
PROC TSNE statement, see TSNE procedure
PROJECTONTYPE= option
  CODE statement, 364

QUERY= option
  PROC FASTKNN statement, 133

RBAIMP option
  PROC FOREST statement, 159
  PROC GRADBOOST statement, 222

RBF option
  KERNEL statement, 401

REDUCEF= option
  PARSE statement, 475

REGL1= option
  PROC MTLREARN statement, 294

REGL2= option
  OPTIMIZATION statement, 338

REPEATED option
  DISPLAYOUT statement (FACTMAC), 121
  DISPLAYOUT statement (FASTKNN), 135
  DISPLAYOUT statement (GMM), 198
  DISPLAYOUT statement (GVARCLUS), 263
  DISPLAYOUT statement (MTLEARN), 296, 385
  DISPLAYOUT statement (MWPCA), 313
  DISPLAYOUT statement (RPCA), 366
  DISPLAYOUT statement (TSNE), 523

RES= option
  SVD statement, 480

RESOLUTION= option
  SVD statement, 480

RESUME option
  TRAIN statement, 343

RHO= option
  PROC GVARCLUS statement, 261

RIDGE= option
  PROC GRADBOOST statement, 222

ROBUST= option
  PROC MWPCA statement, 312

ROLE option
  OUTPUT statement, 166, 230

ROLE= option
  OUTPUT statement, 65, 339, 453

ROLEVAR= option
  PARTITION statement, 340
  PARTITION statement (BNET), 66
  PARTITION statement (FOREST), 21, 167
  PARTITION statement (GRADBOOST), 21, 231
  PARTITION statement (SVMACHINE), 453

ROTATION= option
  SVD statement, 480

ROW= option
  SVD statement, 480

RPCA procedure, CODE statement, 364
  FILE= option, 364
  PROJECTONTYPE= option, 364

RPCA procedure, DISPLAY statement
CASESENSITIVE option, 365
EXCLUDE option, 365
EXCLUDEALL option, 365
TRACE option, 365
RPCA procedure, DISPLAYOUT statement
  INCLUDEALL option, 366
  NOREPLACE option, 366
  REPEATED option, 366
RPCA procedure, ID statement, 366
RPCA procedure, INPUT statement, 366
RPCA procedure, OUTDECOMP statement, 366
  PCLOADINGS= option, 367
  PCSCORES= option, 367
  SVDDIAG= option, 366
  SVDLEFT= option, 367
  SVDRIGHT= option, 367
RPCA procedure, PROC RPCA statement, 360
  CENTER option, 361
  COLSTATISTICS= option, 361
  CUMEIGPCTTOL= option, 361
  DATA= option, 362
  DECOMP= option, 362
  FIXEDMU option, 362
  LAMBDA= option, 362
  LAMBDAWEIGHT= option, 362
  MAXITER= option, 362
  METHOD= option, 363
  MU= option, 363
  NTHREADS= option, 363
  OUTERROR= option, 363
  OUTLOWRANK= option, 363
  OUTSPARSE= option, 363
  PREFIX= option, 363
RPCA procedure, SAVESTATE statement, 367
  RSTORE= option, 367
RPCA procedure, SVD statement, 368
  METHOD= option, 368
RPCA procedure, syntax, 360
RPCA statement
  MWPCA procedure, 315
RSS option
  GROW statement (FOREST), 165
RSTORE= option
  DOWNLOAD statement, 32
  PROC SVDD statement, 402
  SAVESTATE statement, 66, 123, 167, 199, 231, 279, 297, 367, 454, 476
  UPLOAD statement, 37
RSTORE= option
  DESCRIBE statement, 32
RTOL= option
  SOLVER statement, 403
RULES= option
  OUTPUT statement, 91
RULETERMS= option
  OUTPUT statement, 92
  SCORE statement, 92
SAMPLESIZE= option
  AUTOTUNE statement, 17
SAMPLINGRATE option
  PROC GRADBOOST statement, 222
SAVESTATE statement
  BNET procedure, 66
  FACTMAC procedure, 123
  FOREST procedure, 167
  GMM procedure, 199
  GRADBOOST procedure, 231
  MBANALYSIS procedure, 279
  MTLearn procedure, 297
  RPCA procedure, 367
  SVDD procedure, 402
  SVMACHINE procedure, 454, 476
SCALE option
  PROC RPCA statement, 363
  PROC SVMACHINE statement, 448
SCORE option
  PROC SVDD statement, 399
SCORE statement
  ASTORE procedure, 33
  BOOLRULE procedure, 92
  GMM procedure, 199
SEARCHMETHOD= option
  AUTOTUNE statement, 18
SEED= option
  PROC FACTMAC statement, 117
  PROC FOREST statement, 159
  PROC GMM statement, 197
  PROC GRADBOOST statement, 222
  PROC MTLearn statement, 294
  SOLVER statement, 403
  SVD statement, 368
SEED= option
  OPTIMIZATION statement, 338
SELECT statement
  TEXTMINE procedure, 476
SELECT= option
  PROC GVARCLUS statement, 261
SEMISUPLEARN procedure, INPUT statement, 385
SEMISUPLEARN procedure, OUTPUT statement, 385
  COPYVARS= option, 386
  OUT= option, 385
SEMISUPLEARN procedure, PROC
  SEMISUPLEARN statement, 382
  DATA= option, 383
  GAMMA= option, 383
Syntax Index

LABEL= option, 383
MAXITER= option, 383
NTHREADS= option, 383

SEMISUPLEARN procedure, syntax, 382
SEMISUPLEARN procedure, TARGET statement, 386

SEPARATOR= option
PROC MBANALYSIS statement, 277

SETOPTION statement
ASTORE procedure, 34
SHOWDROPPEDTERMS= option
PARSE statement, 475

SHRINKAGE= option
TRANSFERLEARN statement, 232

SOLVER statement
SVDD procedure, 402

STAGNATION= option
TRAIN statement, 343

STANDARDIZE= option
PROC NNET statement, 328

STANDARDPC= option
OUTPUT statement, 315

START= option
PARSE statement, 475

STD option
PROC SVDD statement, 399

STEPSIZE= option
PROC MWPCA statement, 311

STOCHS option
PROC SVDD statement, 402

STOL= option
SOLVER statement, 402

STOP= option
PARSE statement, 475
PROC GVARCLUS statement, 261

STORE= option
DESCRIBE statement, 32
DOWNLOAD statement, 33
UPLOAD statement, 37

SUBTRACT option
VIICODE statement (FOREST), 168
VIICODE statement (GRADBOOST), 233

SUP_LIFT= option
PROC MBANALYSIS statement, 277

SUPPORT= option
PROC FISM statement, 142
PROC MBANALYSIS statement, 276

SVD statement
MWPCA procedure, 316
RPCA procedure, 368
TEXTMINE procedure, 477

SVDD procedure, CODE statement, 400
FILE= option, 400

SVDD procedure, ID statement, 400

SVDD procedure, INPUT statement, 400
LEVEL= option, 401

SVDD procedure, KERNEL statement, 401
RBF option, 401

SVDD procedure, PROC SVDD statement, 398
DATA= option, 399
NBINS, 399
NOPRINT, 399
NTHREADS= option, 399
OUTLIER_FRACTION= option, 399
OUTSV=, 400
RSTORE=, 402
SCORE, 399
STD option, 399
UCC option, 399

SVDD procedure, PROC SVDD statement, SOLVER statement
CTOL= option, 403
MAXITER= option, 402
MAXSV= option, 403
NMATCH= option, 403
NSAMP= option, 403
RTOL= option, 403
SEED option, 403
STOL= option, 402

SVDD procedure, SAVESTATE statement, 402

SVDD procedure, SOLVER statement, 402
ACTSET option, 402
STOCHS option, 402

SVDD procedure, syntax, 398

SVDD procedure, WEIGHT statement, 403

SVDDIAG= option
OUTDECOMP statement, 366

SVDDOCPRO= option
PROC TMSCORE statement, 515

SVDELEFT= option
OUTDECOMP statement, 367

SVDRIGHT= option
OUTDECOMP statement, 367

SVDS= option
SVD statement, 480

SVDU= option
PROC TMSCORE statement, 515
SVD statement, 481

SVDV= option
SVD statement, 481

SVMACHINE procedure, AUTOTUNE statement, 448
TUNINGPARAMETERS= option, 449

SVMACHINE procedure, CODE statement, 451
COMMENT option, 19
FILE= option, 19, 451
FORMATWIDTH= option, 20
INDENTSIZER= option, 20
LABELID= option, 20
LINESIZE= option, 20
NOTRIM option, 20
OUT= option, 20
SVMACHINE procedure, ID statement, 451
SVMACHINE procedure, INPUT statement, 451
LEVEL= option, 451
SVMACHINE procedure, KERNEL statement, 452
SVMACHINE procedure, OUTPUT statement, 452
COPYVARS= option, 453
OUT= option, 452
ROLE= option, 453
SVMACHINE procedure, PARTITION statement
FRACTION option, 453
ROLEVAR= option, 453
SVMACHINE procedure, PROC SVMACHINE
statement, 447
C= option, 447
DATA= option, 447
ITERATIONREPORT, 447
LINEAR option, 452
MAXITER= option, 448
NOPRINT, 448
NOSCALE, 448
NTHREADS= option, 448
POLYNOMIAL option, 452
PRINTTARGET option, 448
SCALE option, 448
TOLERANCE option, 448
USEMISS option, 448
SVMACHINE procedure, SAVESTATE statement,
454, 476
RSTORE= option, 454, 476
SVMACHINE procedure, syntax, 447
SVMACHINE procedure, TARGET statement, 454
ASCENDING option, 454
DESCENDING option, 454
SYNONYM= option
PARSE statement, 475
syntax
BOOLRULE procedure, 87
TEXTMINE procedure, 470
TMSCORE procedure, 514
TARGET statement
BNET procedure, 66
FACTMAC procedure, 123
FISM procedure, 143
FOREST procedure, 168
GRADBOOST procedure, 232
MBANALYSIS procedure, 279
MTLEARN procedure, 297
NNET procedure, 340
SEMISUPLEARN procedure, 386
SV машине procedure, 454
TEXTMINE procedure, 481
TARGET= option
DOCINFO statement, 91
TARGETEVENT= option
AUTOTUNE statement, 18
TARGETSTANDARDIZE= option
PROC NNET statement, 329
TARGETTYPE= option
DOCINFO statement, 91
TERMID= option
PROC BOOLRULE statement, 90
TERMINF0 statement
BOOLRULE procedure, 92
TERMINF0= option
PROC BOOLRULE statement, 90
TERMS= option
PROC TMSCORE statement, 515
TERMWGt= option
PARSE statement, 475
TEXTMINE procedure, 470
PARSE statement, 471
PROC TEXTMINE statement, 470
SELECT statement, 476
SVD statement, 477
syntax, 470
TEXTMINE procedure, CODE statement
COMMENT option, 19
FILE= option, 19
FORMATWIDTH= option, 20
INDENTSIZE= option, 20
LABELID= option, 20
LINESIZE= option, 20
NOTRIM option, 20
OUT= option, 20
TEXTMINE procedure, DOC_ID statement, 471
TEXTMINE procedure, PARSE statement, 471
CELLWGt= option, 472
ENTITIES= option, 473
MULTITERM= option, 473
NONG option, 473
NONOUNGROUPS option, 473
NOSTEMMING option, 473
NOTAGGING option, 473
OUTCHILD= option, 474
OUTCONFIG= option, 474
OUTPARENT= option, 474
OUTPOS= option, 474
OUTTERMS= option, 474
REDUCEF= option, 475
SHOWDROPPEDTERMS= option, 475
START= option, 475
STOP= option, 475
SYNONYM= option, 475
TERMWGt= option, 475
<table>
<thead>
<tr>
<th>Procedure/Statement</th>
<th>Option/Attribute</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXTMINE</td>
<td>DATA=</td>
<td>470</td>
</tr>
<tr>
<td></td>
<td>LANGUAGE=</td>
<td>471</td>
</tr>
<tr>
<td></td>
<td>NEWVARNAMES=</td>
<td>471</td>
</tr>
<tr>
<td></td>
<td>NTHREADS=</td>
<td>471</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>GROUP=</td>
<td>476</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>IGNORE=</td>
<td>477</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>KEEP=</td>
<td>477</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>LABELS=</td>
<td>477</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>COL=</td>
<td>478</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>ENTRY=</td>
<td>478</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>EXACTWEIGHT=</td>
<td>478</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>IN_TERMS=</td>
<td>478</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>K=</td>
<td>479</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>KEEPVARS, KEEPVARIABLES=</td>
<td>479</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>MAX_K=</td>
<td>479</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>NOCUTOFFS=</td>
<td>479</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>NUMLABELS=</td>
<td>479</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>OUTDOCPRO=</td>
<td>479</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>OUTTOPICS=</td>
<td>480</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>RES=</td>
<td>480</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>RESOLUTION=</td>
<td>480</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>ROTATION=</td>
<td>480</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>ROW=</td>
<td>480</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>SVDS=</td>
<td>480</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>SVDU=</td>
<td>481</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>SVDV=</td>
<td>481</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>TOL=</td>
<td>481</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>DOC=</td>
<td>514</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>OUTPARENT=</td>
<td>515</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>SVDDOCPRO=</td>
<td>515</td>
</tr>
<tr>
<td>TEXTMINE</td>
<td>TERMS=</td>
<td>515</td>
</tr>
<tr>
<td>TMSCORE</td>
<td>CONFIG=</td>
<td>515</td>
</tr>
<tr>
<td>TMSCORE</td>
<td>FILE=</td>
<td>514</td>
</tr>
<tr>
<td>TMSCORE</td>
<td>DATA=</td>
<td>514</td>
</tr>
<tr>
<td>TMSCORE</td>
<td>DOC=</td>
<td>514</td>
</tr>
<tr>
<td>TMSCORE</td>
<td>INCLUDEALL=</td>
<td>523</td>
</tr>
<tr>
<td>TMSCORE</td>
<td>LEARNINGRATE=</td>
<td>521</td>
</tr>
<tr>
<td>TMSCORE</td>
<td>MAXITERS=</td>
<td>521</td>
</tr>
</tbody>
</table>
NDIMENSIONS= option, 521
NOPRINT option, 521
NTHREADS= option, 522
PERPLEXITY= option, 522
TSNE procedure, syntax, 520
TUNINGPARAMETERS= option
   AUTOTUNE statement, 18, 58, 118, 160, 224, 330, 449

UCC option
   PROC SVDD statement, 399
UPLOAD statement
   ASTORE procedure, 37
USELOCKING option
   OPTIMIZATION statement, 338
USEMISS option
   PROC SVMACHINE statement, 448
USEPARAMETERS= option
   AUTOTUNE statement, 18

VALIDATION= option
   TRAIN statement, 343
VALIDGOAL= option
   TRAIN statement, 343
VAR statement
   TEXTMINE procedure, 481
   TMSCORE procedure, 516
VARIABLES statement
   TEXTMINE procedure, 481
   TMSCORE procedure, 516
VARS_TO_TRY= option
   PROC FOREST statement, 159
   PROC GRADBOOST statement, 223
VII= option
   PROC FOREST statement, 159
   PROC GRADBOOST statement, 223
VIICODE statement
   FOREST procedure, 168
   GRADBOOST procedure, 232
VOTE= option
   PROC FOREST statement, 159

WEIGHT statement
   FOREST procedure, 169
   GRADBOOST procedure, 233
   NNET procedure, 343
   SVDD procedure, 403
WINDOWSIZE= option
   PROC MWPCA statement, 311
WNSUMMARY= option
   OUTPUT statement, 315
WSEED= option
   TRAIN statement, 343
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.