SAS® Visual Data Mining and Machine Learning 8.1
Data Mining and Machine Learning Procedures
Chapter 1
Introduction

Contents

Overview of SAS Visual Data Mining and Machine Learning Procedures 1
Experimental Software 1
About This Book 2
Chapter Organization 2
Typographical Conventions 2
Options Used in Examples 3
Where to Turn for More Information 3
Online Documentation 3
SAS Technical Support Services 4

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 Data Mining and Machine Learning: Statistical Procedures.

Experimental Software

Experimental software is sometimes included as part of a production-release product. It is provided to (sometimes targeted) customers in order to obtain feedback. All experimental uses are marked Experimental in this document. Whenever an experimental procedure, statement, or option is used, a message is printed to the SAS log to indicate that it is experimental.
The design and syntax of experimental software might change before 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 1**, 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 2** 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;

ods pdf style=PearlJ;
   . . .
ods 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](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 2
Shared Concepts

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to Shared Concepts</td>
<td>5</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>5</td>
</tr>
<tr>
<td>Loading a SAS Data Set onto a CAS Server</td>
<td>6</td>
</tr>
<tr>
<td>Syntax Common to SAS Visual Data Mining and Machine Learning Procedures</td>
<td>7</td>
</tr>
<tr>
<td>CODE Statement</td>
<td>7</td>
</tr>
<tr>
<td>PARTITION Statement</td>
<td>9</td>
</tr>
<tr>
<td>Details for SAS Visual Data Mining and Machine Learning Procedures</td>
<td>10</td>
</tr>
<tr>
<td>Using Validation and Test Data</td>
<td>10</td>
</tr>
<tr>
<td>Multithreading</td>
<td>11</td>
</tr>
<tr>
<td>References</td>
<td>11</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 7 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 10 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:
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:

```
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:

```
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:**

  ```
  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
  ```
You can use the CASUTIL procedure as follows:

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

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

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

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

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

The Sample data table is accessible only in the mysess session. When you terminate the mysess session, the Sample data table is no longer accessible from the CAS server. If you want your Sample data table to be available to other CAS sessions, then you must promote your data table. For more information about data tables, see SAS Cloud Analytic Services: Accessing and Manipulating Data.

---

### Syntax Common to SAS Visual Data Mining and Machine Learning Procedures

#### CODE Statement

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

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

Table 2.1 summarizes the options you can specify in the CODE statement.
Table 2.1 CODE Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENT</td>
<td>Adds comments to the generated code</td>
</tr>
<tr>
<td>FILE=</td>
<td>Names the file where the generated code is saved</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>PCATALL</td>
<td>Generates probabilities for all levels of categorical response variables</td>
</tr>
</tbody>
</table>

If you do not specify the FILE= 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.

**FORMATWIDTH=width**

specifies the width to use in formatting derived numbers such as parameter estimates. You can specify a value in the range 4 to 32; the default is 20.

**INDENTSIZE=n**

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

**LABELID=value**

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

**LINESIZE=value**

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

PARTITION partition-option ;

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

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 10. 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.
Chapter 2: Shared Concepts

Details for SAS Visual Data Mining and Machine Learning Procedures

Using Validation and Test Data

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

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

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

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

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

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

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

```
proc logselect data=mycas.inData;
   partition Group=(0.25 validate=0.25);  /* Group 1 assigned for testing */
   ... 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: 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 that runs in CAS is determined by your installation.

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

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

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

### References

Chapter 3

The ASTORE Procedure

Overview: ASTORE Procedure

The ASTORE procedure is an interactive procedure in which each statement runs immediately. The ASTORE procedure describes, manages, and scores with an 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. Some procedures that support a SAVESTATE statement are the FACTMAC, FOREST, and SVMACHINE procedures. You can use the analytic store at a later time for scoring.

PROC ASTORE Features

The ASTORE procedure enables you to do the following:
Chapter 3: The ASTORE Procedure

- 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 set and produce an output data set by using a specified analytic store and optional DS2 scoring code that uses the analytic store
- 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
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 5 in Chapter 2, “Shared Concepts.”
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 9, “The SVMACHINE Procedure.”

This example uses 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 3.1 describes the variables in Hmeq.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Role</th>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad</td>
<td>Response</td>
<td>Binary</td>
<td>1 = customer defaulted on the loan or is seriously delinquent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = customer is current on loan payments</td>
</tr>
<tr>
<td>CLAge</td>
<td>Predictor</td>
<td>Interval</td>
<td>Age of oldest credit line in months</td>
</tr>
<tr>
<td>CLNo</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of credit lines</td>
</tr>
<tr>
<td>DebtInc</td>
<td>Predictor</td>
<td>Interval</td>
<td>Debt-to-income ratio</td>
</tr>
<tr>
<td>Delinq</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of delinquent credit lines</td>
</tr>
<tr>
<td>Derog</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of major derogatory reports</td>
</tr>
<tr>
<td>Job</td>
<td>Predictor</td>
<td>Nominal</td>
<td>Occupational category</td>
</tr>
<tr>
<td>Loan</td>
<td>Predictor</td>
<td>Interval</td>
<td>Requested loan amount</td>
</tr>
<tr>
<td>MortDue</td>
<td>Predictor</td>
<td>Interval</td>
<td>Amount due on existing mortgage</td>
</tr>
<tr>
<td>nInq</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of recent credit inquiries</td>
</tr>
<tr>
<td>Reason</td>
<td>Predictor</td>
<td>Binary</td>
<td>'DebtCon' = debt consolidation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'HomeImp' = home improvement</td>
</tr>
<tr>
<td>Value</td>
<td>Predictor</td>
<td>Interval</td>
<td>Value of current property</td>
</tr>
<tr>
<td>YoJ</td>
<td>Predictor</td>
<td>Interval</td>
<td>Years at present job</td>
</tr>
</tbody>
</table>

You can load the sampsio.hmeq 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 14), 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.

```plaintext
data mycas.hmeq;
  set sampsio.hmeq;
  id = _N_;
run;
```
Because distributed computing orders distributed data differently than traditional SAS procedures orders them, one or more variables that can act as the record identifier must be in the input data set. Here the record identifier is a single variable `id`.

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`.

```
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;
```

**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 3.1**.

```
proc print data=scoreout1(obs=5);
run;
```
The following statements are available in the ASTORE procedure:

```
PROC ASTORE ;
  SCORE score-options ;
  DESCRIBE describe-options ;
  DOWNLOAD download-options ;
  UPLOAD upload-options ;
```

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.

### DESCRIBE Statement

```
DESCRIBE STORE=local-file-name | 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 or in a data table stored in CAS. It 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:

---

**Figure 3.1** Scoring with PROC ASTORE

<table>
<thead>
<tr>
<th>Obs</th>
<th>id</th>
<th><em>P</em></th>
<th>P_BAD1</th>
<th>P_BAD0</th>
<th><em>BAD</em></th>
<th><em>WARN</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.0000000088</td>
<td>7.2815958E-8</td>
<td>0.9999999272</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.99999999691</td>
<td>9.2701136E-8</td>
<td>0.9999999073</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1.0000000096</td>
<td>7.2405865E-8</td>
<td>0.9999999276</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>-0.846156581</td>
<td>0.6410258508</td>
<td>0.3589741492</td>
<td>1</td>
<td>M</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1.0000000505</td>
<td>5.1990216E-8</td>
<td>0.999999948</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
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 14.

You can also specify the following describe-option:

EPCODE< =code-file-name>
sends DS2 language statement 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 allows 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 14.

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 is downloaded.

SCORE Statement

SCORE DATA=CAS-libref.data-table OUT=CAS-libref.data-table RSTORE=CAS-libref.data-table < 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 14.

`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. 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 14.

`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 14.

You can also specify the following option:

**EPISODE=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.

---

**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 14.
\textbf{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.

\section*{Details: ASTORE Procedure}

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

\begin{verbatim}
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;
\end{verbatim}

You can use the \texttt{preScoreRecord} method block to transform the input variables, and you can change or flag the scores in the \texttt{postScoreRecord} method block. If you do not intend to fill either of these two method blocks, then you do not need to specify the \texttt{EPCODE=code-file-name} option in the \texttt{SCORE} statement.

The \texttt{UPLOAD} statement produces output that includes the store key for future reference in the DS2 language code. The code that is produced by the \texttt{EPCODE=} option in the \texttt{DESCRIBE} statement includes the same key. If you upload the store to CAS and you specify the \texttt{EPCODE=} option in the \texttt{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.

\section*{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 3.2.
### Table 3.2  ODS Tables Produced by PROC ASTORE

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</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>
<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>SCORE</td>
</tr>
</tbody>
</table>

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

---

### Examples: ASTORE Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “Shared Concepts.”

### Example 3.1: Scoring a Single Record

This is a simple example that creates a table that contains one record whose ID is 100 and uses the analytic store that was produced by PROC SVMACHINE.

The following DATA step creates the mycas.hmeq1 table in CAS and extracts the record whose ID is 100.

```sas
   data mycas.hmeq1;
   set mycas.hmeq;
   if (id = 100);
   run;
```
Chapter 3: The ASTORE Procedure

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 3.1.1.

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

**Output 3.1.1** Scoring a Single Record

<table>
<thead>
<tr>
<th>Obs</th>
<th>id</th>
<th><em>P</em></th>
<th>P_BAD1</th>
<th>P_BAD0</th>
<th>I_BAD</th>
<th><em>WARN</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>1.0000000287</td>
<td>6.2877921E-8</td>
<td>0.9999999371</td>
<td>0</td>
<td>M</td>
</tr>
</tbody>
</table>

Example 3.2: 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 3.2.1 and sends the basic code to the file svmepcode.sas.

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

Output 3.2.1 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 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 3.2.1** Output Tables from the DESCRIBE statement

**The ASTORE Procedure**

<table>
<thead>
<tr>
<th>Key Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>14F2E0A7B9A73581F79904C6E900D60ABCC51EDB</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>
The following statements show the contents of the `svmeprcode.sas` file; this is DS2 language code:

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"n
    "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();
end;
enddata;

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. If you do not intend to edit either the preScoreRecord or postScoreRecord method block, you can still score. When you edit the file, you must follow the syntax of the DS2 programming language. The following statements show an example:

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 3.3: Downloading the Store to the Local File System

This example extracts the analytic store saved in the data table mycas.savehmeq from CAS to the local file svmlocalcopy in the local file system.

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

In addition to downloading the actual file, PROC ASTORE writes a note in the log that shows how many bytes were downloaded.

Example 3.4: Uploading the Local Store from the Local File System

This example sends the analytic store svmlocalcopy from the local file system to the data table mycas.savehmeq on CAS.

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

The UPLOAD statement produces the store key in the listing. You can use this key to construct the embedded processing code on your own, but it is simpler for you to use the EPCODE= option in the DESCRIBE statement to produce the resulting minimal code and then edit the contents.
## Chapter 4
The BOOLRULE Procedure

### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: BOOLRULE Procedure</td>
<td>28</td>
</tr>
<tr>
<td>PROC BOOLRULE Features</td>
<td>28</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>29</td>
</tr>
<tr>
<td>Getting Started: BOOLRULE Procedure</td>
<td>29</td>
</tr>
<tr>
<td>Syntax: BOOLRULE Procedure</td>
<td>34</td>
</tr>
<tr>
<td>PROC BOOLRULE Statement</td>
<td>34</td>
</tr>
<tr>
<td>DOCINFO Statement</td>
<td>37</td>
</tr>
<tr>
<td>OUTPUT Statement</td>
<td>38</td>
</tr>
<tr>
<td>SCORE Statement</td>
<td>38</td>
</tr>
<tr>
<td>TERMMINFO Statement</td>
<td>39</td>
</tr>
<tr>
<td>Details: BOOLRULE Procedure</td>
<td>39</td>
</tr>
<tr>
<td>BOOLLEAR for Boolean Rule Extraction</td>
<td>39</td>
</tr>
<tr>
<td>Term Ensemble Process</td>
<td>40</td>
</tr>
<tr>
<td>Rule Ensemble Process</td>
<td>41</td>
</tr>
<tr>
<td>Measurements Used in BOOLLEAR</td>
<td>43</td>
</tr>
<tr>
<td>Precision, Recall, and the F1 Score</td>
<td>43</td>
</tr>
<tr>
<td>g-Score</td>
<td>43</td>
</tr>
<tr>
<td>Estimated Precision</td>
<td>44</td>
</tr>
<tr>
<td>Improvability Test</td>
<td>44</td>
</tr>
<tr>
<td>Shrinking the Search Space</td>
<td>45</td>
</tr>
<tr>
<td>Feature Selection</td>
<td>45</td>
</tr>
<tr>
<td>Significance Testing</td>
<td>45</td>
</tr>
<tr>
<td>k-Best Search</td>
<td>45</td>
</tr>
<tr>
<td>Improvability Test</td>
<td>46</td>
</tr>
<tr>
<td>Early Stop Based on the F1 Score</td>
<td>46</td>
</tr>
<tr>
<td>Output Data Sets</td>
<td>46</td>
</tr>
<tr>
<td>CANDIDATETERMS= Data Table</td>
<td>46</td>
</tr>
<tr>
<td>RULES= Data Table</td>
<td>47</td>
</tr>
<tr>
<td>RULETERMS= Data Table</td>
<td>47</td>
</tr>
<tr>
<td>Scoring Data Set</td>
<td>48</td>
</tr>
<tr>
<td>OUTMATCH= Data Table</td>
<td>48</td>
</tr>
<tr>
<td>Examples: BOOLRULE Procedure</td>
<td>49</td>
</tr>
<tr>
<td>Example 4.1: Rule Extraction for Binary Targets</td>
<td>49</td>
</tr>
<tr>
<td>Example 4.2: Rule Extraction for a Multiclass Target</td>
<td>51</td>
</tr>
<tr>
<td>Example 4.3: Using Events in Rule Extraction</td>
<td>52</td>
</tr>
<tr>
<td>Example 4.4: Scoring</td>
<td>54</td>
</tr>
<tr>
<td>References</td>
<td>57</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:

```
(cut ^ rate ^ bank ^ percent ^ ~sell) or
(market ^ money ^ ~year ^ percent ^ ~sale) or
(repurchase ^ fee) or
(rate ^ prime rate) or
(federal ^ rate ^ maturity)
```

In this example, ^ indicates a logical “and,” and ~ 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:

```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 5 in Chapter 2, “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 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “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.
Chapter 4: The BOOLRULE Procedure

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 4.1** 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 4.2 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 47.

```
Obs TARGET RULEID RULE    F1  PRECISION  RECALL
1 apple_fruit 1 be & apple 0.85714 1 0.75
2 apple_fruit 2 tree       1.00000 1 1.00
```

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_
   termid = _termnum_
   score
      ruleterms = mycas.ruleterms
      outmatch = mycas.matches;
run;
proc print data=mycas.matches;
run;
```

Figure 4.3 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 48.
**Figure 4.3** The mycas.matches 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>d01</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>d06</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>d09</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>d11</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>d16</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>d17</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>d04</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>d04</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>d07</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>d14</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>d15</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>d19</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>d02</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>d03</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>d05</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>d08</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>d10</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>d12</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>d13</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>d18</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>d20</td>
<td>.</td>
<td>0</td>
</tr>
</tbody>
</table>
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 4.1 summarizes the `options` in the statement by function. The `options` are then described fully in alphabetical order.

<table>
<thead>
<tr>
<th><code>option</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic Options</strong></td>
<td></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the input data table (which must be in transactional format) for</td>
</tr>
<tr>
<td></td>
<td>rule extraction</td>
</tr>
<tr>
<td>DOCID=</td>
<td>Specifies the variable in the <code>DATA=</code> data table that contains the document</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>considered for rule extraction</td>
</tr>
<tr>
<td>GPOS=</td>
<td>Specifies the minimum ( g )-score needed for a positive term or a rule</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>ensemble process for creating a rule set</td>
</tr>
<tr>
<td>MINSUPPORTS=</td>
<td>Specifies the minimum number of documents in which a term needs to appear</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>terms</td>
</tr>
<tr>
<td>MPOS=</td>
<td>Specifies the ( m ) value for computing estimated precision for positive</td>
</tr>
<tr>
<td></td>
<td>terms</td>
</tr>
</tbody>
</table>
Table 4.1 continued

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

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 29.

- `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 29.

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 43.
Chapter 4: The BOOLRULE Procedure

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

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

**MAXTRIESIN=n**
specifies the \( k_i \) 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 45. 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 45. 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 \textit{variable} that contains the ID of each term. The \textit{variable} can be either a number or a string of characters. If the TERMINFO= option is not specified, \textit{variable} is also used as the label of terms.

**TERMINFO=CAS-libref.data-table**
names the input data table that contains information about terms. \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 29.

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

```
DOCINFO <options> ;
```

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

You can specify the following options:

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

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

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

- **TARGET=(variable, variable, . . .)**
  
specifies the target variables. A target variable can be either a numeric variable or a character variable.

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

- **TARGETTYPE=BINARY | MULTICLASS**
  
specifies the type of the target variables. You can specify the following values:

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

By default, TARGETTYPE=BINARY.
Chapter 4: The BOOLRULE Procedure

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 29.

  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 46.

- **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 29.

    For more information about this data table, see the section “Output Data Sets” on page 46.

- **RULETERMS=** `CAS-libref.data-table`
  - Specifies a data table to contain the terms in each rule that is generated by the BOOLRULE procedure. `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier, and `data-table` specifies the name of the output data table. For more information about this two-level name, see the `DATA=` option and the section “Using CAS Sessions and CAS Engine Librefs” on page 29.

    For more information about this data table, see the section “Output Data Sets” on page 46.

SCORE Statement

```
SCORE < options > ;
```

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

You can specify the following `options`:

- **OUTMATCH=** `CAS-libref.data-table`
  - Specifies a data table to contain the rule-matching results (that is, whether a document satisfies a rule). `CAS-libref.data-table` is a two-level name, where `CAS-libref` refers to the caslib and session identifier,
and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 29.

For more information about this data table, see the section “Scoring Data Set” on page 48.

RULETERMS=CAS-libref.data-table
specifies a data table that contains the terms in each rule that the BOOLRULE procedure generates. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 29.

For more information about this data table, see the section “RULETERMS= Data Table” on page 47.

TERMINFO Statement

TERMINFO < options > ;

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

You can specify the following options:

ID=variable
specifies the variable that contains the term ID. To fetch the text of terms, the values in variable are matched to the term ID variable that is specified in the 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 use 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 4.4 shows the flowchart of the term ensemble process.
Before adding terms to a rule, BOOLLEAR first sorts the candidate terms in descending order according to their \( g \)-score with respect to the target category. It then starts to add terms to the rule iteratively. In each iteration of the term ensemble process, BOOLLEAR takes a term \( t \) from the ordered candidate term list and determines whether adding the term to the current rule \( r \) can improve the rule’s estimated precision. To ensure that the term is good enough, BOOLLEAR tries \( k_{in} - 1 \) additional terms in the term list, where \( k_{in} \) is the maximum number of terms to examine for improvement. If none of these terms is better (results in a lower \( g \)-score of the current rule \( r \)) than term \( t \), the term is considered to be \( k \)-best, where \( k = k_{in} \), and BOOLLEAR updates the current rule \( r \) by adding term \( t \) to it. If one of the \( k_{in} - 1 \) additional terms is better than term \( t \), BOOLLEAR sets that term as \( t \) and tries \( k_{in} - 1 \) additional terms to determine whether this new \( t \) is better than all of those additional terms. BOOLLEAR repeats until the current term \( t \) is \( k \)-best or until it reaches the end of the term list. After a term is added to the rule, BOOLLEAR marks the term as used and continues to identify the next \( k \)-best term from the unused terms in the sorted candidate term list. When a \( k \)-best term is identified, BOOLLEAR adds it to the rule. BOOLLEAR keeps adding \( k \)-best terms until the rule cannot be further improved. By trying to identify a \( k \)-best term instead of the global best, BOOLLEAR shrinks its search space to improve its efficiency.

**Rule Ensemble Process**

The rule ensemble process iteratively creates and adds new rules to a rule set. When the process finishes, it returns the rule set, which can then be used for text categorization. Figure 4.5 shows the flowchart of the rule ensemble process.
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 $k_{out}$ 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 $k_{out} - 1$ 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 $k_{out} - 1$ rules that are generated after it. The obtained rule $r$ is called a $k$-best rule, where $k = k_{out}$. 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 43.
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.

\-Score

BOOLLEAR uses the \-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 \-test generates a \-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 \-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 \-score can be computed as

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

---

\(^2\)Accuracy is defined as \(\frac{TP+TN}{TP+FP+TN+FN}\).
\[
\begin{align*}
O(TP) &= TP \\
O(FP) &= FP \\
O(TN) &= TN \\
O(FN) &= FN \\
E(TP) &= \frac{(TP + FP) \times P}{P + N} \\
E(FP) &= \frac{(TP + FP) \times N}{P + N} \\
E(TN) &= \frac{(TN + FN) \times N}{P + N} \\
E(FN) &= \frac{(TN + FN) \times P}{P + N}
\end{align*}
\]

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

**Estimated Precision**

Estimated precision helps BOOLLEAR shorten its search path and avoid generating overly specific rules. The precision is estimated by a form of additive smoothing with additional correction (\( err_i \)) to favor shorter rules over longer rules:

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

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

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. \( TP_{i,t} \) and \( FP_{i,t} \) are the true-positive and false-positive of rule \( t \) when the length of the rule is \( i \).

**Improvability Test**

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

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

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

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

Feature Selection

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

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

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

Significance Testing

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

- **Minimum positive document coverage**: BOOLLEAR requires that a rule be satisfied by at least $s$ positive documents, where $s$ is the value of the \texttt{MINSUPPORTS=} option in the \texttt{PROC BOOLRULE} statement.

- **Early stop based on $g$-test**: BOOLLEAR stops searching when the $g$-score that is calculated for improving (or starting) a rule does not meet required statistical significance levels.

- **Early stop based on estimated precision**: BOOLLEAR stops building a rule when the estimated precision of the rule does not improve when the current best term is added to the rule. This strategy helps BOOLLEAR shorten its search path.

$k$-Best Search

In the worst case, BOOLLEAR could still examine an exponential number of rules, although the heuristics described here minimize that chance. But because the terms are ordered by predictiveness of the category beforehand, a $k$-best search is used to further improve the efficiency of BOOLLEAR: If BOOLLEAR tries unsuccessfully to expand (or start) a rule numerous times with the a priori “best” candidates, then the search can be prematurely ended. Two optional parameters, $k_{in}$ and $k_{out}$, determine the maximum number of terms and rules to examine for improvement. The $k_{in}$ parameter (which is specified in the \texttt{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 4.2 shows the fields in this data table.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>The category that the term is selected for (this field corresponds to the Target field in the RULES= data table)</td>
</tr>
<tr>
<td>Rank</td>
<td>The rank of the term in the ordered term list for the category (term rank starts from 1)</td>
</tr>
<tr>
<td>Term</td>
<td>A lowercase version of the term</td>
</tr>
<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 4.3 shows the fields in this data table.

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

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

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

RULETERMS= Data Table

The RULETERMS= option in the OUTPUT statement specifies a data table to contain the terms in the rules. The information in this data table is used in the scoring phase for scoring documents.
Table 4.4 Fields in the RULETERMS= 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><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 4.5 shows the fields in this data table.

Table 4.5 Fields in the OUTMATCH= Data Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Document</em></td>
<td>ID of the document that satisfies the rule</td>
</tr>
<tr>
<td><em>Target</em></td>
<td>ID of the target that the rule is generated for</td>
</tr>
<tr>
<td><em>Rule_ID</em></td>
<td>ID of the rule that the document satisfies</td>
</tr>
</tbody>
</table>
Example 4.1: Rule Extraction for Binary Targets

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

```
data mycas.reviews;
  infile datalines delimiter='|' missover;
  length text $300 category $20;
  input text$ positive category$ did;
  datalines;
  This is the greatest phone ever! love it!|1|electronics|1
  The phone's battery life is too short and screen resolution is low.|0|electronics|2
  The screen resolution is low, but I love this tv.|1|electronics|3
  The movie itself is great and I like it, although the resolution is low.|1|movies|4
  The movie's story is boring and the acting is poor.|0|movies|5
  I watched this movie on tv, it's not good on a small screen. |0|movies|6
  watched the movie first and loved it, the book is even better!|1|books |7
  I like the story in this book, they should put it on screen.|1|books|8
  I love the author, but this book is a waste of time, don't buy it.|0|books|9
run;
```

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

```
proc textmine data=mycas.reviews;
  doc_id did;
  var text;
  parse
    nonoungroups
    notagging
    entities = none
    outparent = mycas.reviews_bow
    outterms = mycas.reviews_terms
    reducef = 1;
run;
```
The following statements run PROC BOOLRULE to extract rules from the mycas.reviews_bow data table and run PROC PRINT to show the results. By default, TARGETTYPE=BINARY. One target variable, positive, is specified; this variable indicates whether the reviews are positive or negative.

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

Output 4.1.1 shows that the mycas.rules data table contains rules that are generated for the “positive” categories.

**Output 4.1.1** The mycas.rules Data Table

<table>
<thead>
<tr>
<th>Obs</th>
<th>TARGET</th>
<th>RULEID</th>
<th>RULE</th>
<th>F1</th>
<th>PRECISION</th>
<th>RECALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>positive</td>
<td>1</td>
<td>like</td>
<td>0.57143</td>
<td>1.00000</td>
<td>0.4</td>
</tr>
<tr>
<td>2</td>
<td>positive</td>
<td>2</td>
<td>better</td>
<td>0.75000</td>
<td>1.00000</td>
<td>0.6</td>
</tr>
<tr>
<td>3</td>
<td>positive</td>
<td>3</td>
<td>great</td>
<td>0.88889</td>
<td>1.00000</td>
<td>0.8</td>
</tr>
<tr>
<td>4</td>
<td>positive</td>
<td>4</td>
<td>love</td>
<td>0.90909</td>
<td>0.83333</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Example 4.2: Rule Extraction for a Multiclass Target

This example uses the same input table and the same TEXTMINE procedure call that are used in Example 4.1 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.

```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 movie itself is great and I like it, although the resolution is low.|1|movies|4
  The movie's story is boring and the acting is poor.|0|movies|5
  I watched this movie on tv, it's not good on a small screen.|0|movies|6
  watched the movie first and loved it, the book is even better!|1|books|7
  I like the story in this book, they should put it on screen.|1|books|8
  I love the author, but this book is a waste of time, don't buy it.|0|books|9
run;
```

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

```plaintext
proc textmine data=mycas.reviews;
  doc_id
    did;
  var
    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.
Chapter 4: The BOOLRULE Procedure

```sas
proc boolrule
  data = mycas.reviews_bow
docid = _document_
termid = _termnum_
docinfo = mycas.reviews
terminfo = mycas.reviews_terms
minsupports = 1
mpos = 1
gpos = 1;
docinfo
  id = did
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 4.2.1 shows that the mycas.rules data table contains rules that are generated for the “electronics,” “movies,” and “books” categories.

Output 4.2.1  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>0.80</td>
<td>1.00</td>
<td>0.66667</td>
</tr>
<tr>
<td>2</td>
<td>electronics</td>
<td>2</td>
<td>resolution</td>
<td>0.857</td>
<td>0.75</td>
<td>1.00000</td>
</tr>
<tr>
<td>3</td>
<td>movies</td>
<td>3</td>
<td>movie</td>
<td>0.857</td>
<td>0.75</td>
<td>1.00000</td>
</tr>
<tr>
<td>4</td>
<td>books</td>
<td>4</td>
<td>book</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00000</td>
</tr>
</tbody>
</table>

Example 4.3: Using Events in Rule Extraction

This example uses the same input table and the same TEXTMINE procedure call that are used in Example 4.1 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
  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:

```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.”

```sas
proc boolrule
  data = mycas.reviews_bow
  docid = _document_
  termid = _termnum_
  docinfo = mycas.reviews
  terminfo = mycas.reviews_terms
  minsupports = 1
  mpos = 1
```
Chapter 4: The BOOLRULE Procedure

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

### Output 4.3.1 The mycas.rules Data Table

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

---

### Example 4.4: Scoring

This example uses the same input table and the same TEXTMINE procedure call that are used in Example 4.1 to illustrate how you can match extracted rules in documents. Then it adds the DATA step to generate testing data. The DATA step and procedure call are repeated here for convenience.

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

```
data mycas.reviews;
  infile datalines delimiter='|' missover;
  length text $300 category $20;
  input text$ positive category$ did;
datalines;
  This is the greatest phone ever! love it!|1|electronics|1
  The phone’s battery life is too short and screen resolution is low.|0|electronics|2
  The screen resolution is low, but I love this tv.|1|electronics|3
  The movie itself is great and I like it, although the resolution is low.|1|movies|4
  The movie’s story is boring and the acting is poor.|0|movies|5
  I watched this movie on tv, it’s not good on a small screen. |0|movies|6
```

---
Example 4.4: Scoring

watched the movie first and loved it, the book is even better!|1|books |7
I like the story in this book, they should put it on screen.|1|books|8
I love the author, but this book is a waste of time, don't buy it.|0|books|9

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

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

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

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

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

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

```plaintext
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:

```plaintext
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 4.4.1 shows which documents satisfy which rules.

**Output 4.4.1** 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 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:

    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 5 in Chapter 2, “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 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “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 three variables: car make, car model, 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 /level=nominal;
   target mpg_city /level=interval;
   output out=mycas.score_out copyvars=(make model 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 and model 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, and mpg_city variables be copied from the mycas.cars data table to the mycas.score_out data table.

Figure 5.1 shows the global bias, the bias values for each level, and the list of the factors for the first 48 observations.
62 F Chapter 5: The FACTMAC Procedure

Figure 5.1 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 ;
  ID variables ;
  INPUT variables <LEVEL=NOMINAL> ;
  OUTPUT OUT=CAS-libref.data-table <options> ;
  SAVESTATE RSTORE=CAS-libref.data-table ;
  TARGET variable <LEVEL=INTERVAL> ;
```

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

The PROC FACTMAC statement invokes the procedure. Table 5.1 summarizes the options available in the PROC FACTMAC statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA=</td>
<td>Specifies the input data table</td>
</tr>
<tr>
<td>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>
</tbody>
</table>

You can specify the following options:

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

names the input data table for PROC FACTMAC 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 5: The FACTMAC Procedure

statement. For more information about CAS-libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 60.

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

\textbf{LEARNSTEP=} \textit{number}

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

By default, \textbf{LEARNSTEP}=0.001.

\textbf{MAXITER=} \textit{number}

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

By default, \textbf{MAXITER}=1.

\textbf{NFACTORS=} \textit{number}

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

By default, \textbf{NFACTORS}=1.

\textbf{NOPRINT}

suppresses ODS output.

\textbf{NTHREADS=} \textit{number-of-threads}

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

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

specifies the output model data table to contain the computed factor parameters. \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 \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 60.

\textbf{SEED=} \textit{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 \textbf{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, \textbf{SEED}=0.

---

\textbf{CODE Statement}

\textbf{CODE FILE=} \textit{filename} ;

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

**FILE=filename**

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

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

---

**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=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=NOMINAL**

specifies the level of measurement of the variables.

PROC FACTMAC currently accepts only nominal input variables.

---

**OUTPUT Statement**

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

The OUTPUT statement creates an output data table to contain the results of the procedure run.

You must specify the following option:

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

names the output data table for PROC 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 60.

- **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.

---

**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 3, “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 60.

---

**TARGET Statement**

```plaintext
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

\[
\hat{y}(x) = w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j x_{j'} \sum_{f=1}^{k} v_{jf} v_{j'f} \tag{5.1}
\]

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

\[
\|w_j\|_\infty < B, \ j = 1, \ldots, p
\]

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 \texttt{SEED=} option and specify \texttt{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 \texttt{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 \texttt{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.
Chapter 5: The FACTMAC Procedure

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.

Output CAS Tables

When you specify the OUTPUT statement to create output tables on your CAS server, the FACTMAC procedure produces the output data table along with a table that lists the CAS library, the data table name, and the number of rows and columns in that data table.

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 5.2.

<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>OutCASTblFull</td>
<td>Summary table for output data; it contains the number of observations and the number of variables</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 5.3.
### Table 5.3  Output Data Table Produced by PROC FACTMAC

<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 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “Shared Concepts.”

### Example 5.1: Running PROC FACTMAC with the MovieLens Data Set

This example draws on data that are derived from companies that provide movies for online viewing. A company wants to offer its customers recommendations of movies that they might like. These recommendations are based on ratings that are provided by users. The MovieLens data set was developed by the GroupLens project at the University of Minnesota and is available at [http://grouplens.org/datasets/movielens](http://grouplens.org/datasets/movielens) (Harper and Konstan 2015). This example uses the MovieLens 100K version.

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

You can download the compressed archive file from the website at [http://files.grouplens.org/datasets/movielens/ml-100k.zip](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:

```sas
proc casutil;
  load file="/data/u.data" /casout="movlens"
  casutil;"Movlens"
  importoptions=(filetype="CSV" delimiter="TAB" getnames="FALSE"
                 vars=("userid" "itemid" "rating" "timestamp"));
run;
```

1 Disclaimer: SAS may reference other websites or content or resources for use at Customer’s sole discretion. SAS has no control over any websites or resources that are provided by companies or persons other than SAS. Customer acknowledges and agrees that SAS is not responsible for the availability or use of any such external sites or resources, and does not endorse any advertising, products, or other materials on or available from such websites or resources. Customer acknowledges and agrees that SAS is not liable for any loss or damage that may be incurred by Customer or its end users as a result of the availability or use of those external sites or resources, or as a result of any reliance placed by Customer or its end users on the completeness, accuracy, or existence of any advertising, products, or other materials on, or available from, such websites or resources.
The following statements show how to use PROC FACTMAC to predict movie ratings:

```sas
proc factmac data=mycas.movlens nfactors=10 learnstep=0.15
   maxiter=20  outmodel=mycas.factors;
   input userid itemid /level=nominal;
   target rating /level=interval;
   output out=mycas.out1 copyvars=(userid itemid rating);
run;
```

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 5.1.1.

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

**Output 5.1.1** 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></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 5.1.2.

```sas
proc print data=mycas.out1(obs=20);
run;
```
Output 5.1.2  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 6
## The FOREST Procedure

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: FOREST Procedure</td>
<td>74</td>
</tr>
<tr>
<td>PROC FOREST Features</td>
<td>74</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>75</td>
</tr>
<tr>
<td>Getting Started: FOREST Procedure</td>
<td>75</td>
</tr>
<tr>
<td>Syntax: FOREST Procedure</td>
<td>80</td>
</tr>
<tr>
<td>PROC FOREST Statement</td>
<td>80</td>
</tr>
<tr>
<td>AUTOTUNE Statement</td>
<td>84</td>
</tr>
<tr>
<td>CODE Statement</td>
<td>88</td>
</tr>
<tr>
<td>CROSSVALIDATION Statement</td>
<td>88</td>
</tr>
<tr>
<td>GROW Statement</td>
<td>89</td>
</tr>
<tr>
<td>ID Statement</td>
<td>90</td>
</tr>
<tr>
<td>INPUT Statement</td>
<td>90</td>
</tr>
<tr>
<td>OUTPUT Statement</td>
<td>90</td>
</tr>
<tr>
<td>PARTITION Statement</td>
<td>91</td>
</tr>
<tr>
<td>SAVESTATE Statement</td>
<td>91</td>
</tr>
<tr>
<td>TARGET Statement</td>
<td>92</td>
</tr>
<tr>
<td>Details: FOREST Procedure</td>
<td>92</td>
</tr>
<tr>
<td>Bagging the Data</td>
<td>92</td>
</tr>
<tr>
<td>Training a Decision Tree</td>
<td>93</td>
</tr>
<tr>
<td>Loh Method</td>
<td>93</td>
</tr>
<tr>
<td>Predicting an Observation</td>
<td>94</td>
</tr>
<tr>
<td>Measuring Prediction Error</td>
<td>94</td>
</tr>
<tr>
<td>Handling Missing Values</td>
<td>95</td>
</tr>
<tr>
<td>Strategies</td>
<td>95</td>
</tr>
<tr>
<td>Specifics</td>
<td>95</td>
</tr>
<tr>
<td>Handling Values That Are Absent from Training Data</td>
<td>96</td>
</tr>
<tr>
<td>Measuring Variable Importance</td>
<td>96</td>
</tr>
<tr>
<td>Residual Sum of Squares Importance Method</td>
<td>96</td>
</tr>
<tr>
<td>Random Branch Assignment Importance Method</td>
<td>98</td>
</tr>
<tr>
<td>Parameter Tuning</td>
<td>98</td>
</tr>
<tr>
<td>k-fold Cross Validation</td>
<td>99</td>
</tr>
<tr>
<td>Displayed Output</td>
<td>100</td>
</tr>
<tr>
<td>Model Information</td>
<td>100</td>
</tr>
<tr>
<td>Number of Observations</td>
<td>100</td>
</tr>
<tr>
<td>Variable Importance</td>
<td>100</td>
</tr>
<tr>
<td>RBA Variable Importance</td>
<td>100</td>
</tr>
</tbody>
</table>
Overview: FOREST Procedure

The FOREST procedure creates a predictive model called a forest (which consists of several decision trees) in SAS Viya. A predictive model defines a relationship between input variables and a target variable. The purpose of a predictive model is to predict a target value from inputs. The FOREST procedure trains the model; that is, it creates the model by using training data in which the target values are known. The model can then be applied to observations in which the target is unknown. If the predictions fit the new data well, the model is said to generalize well. Good generalization is the primary goal for predictive tasks. A predictive model might fit the training data well but generalize poorly.

A decision tree is a type of predictive model that has been developed independently in the statistics and artificial intelligence communities. The FOREST procedure creates a tree recursively: The procedure chooses an input variable and uses it to create a rule to split the data into two or more subsets. The process is then repeated in each subset, and then again in each new subset, and so on until some constraint is met. In the terminology of the tree metaphor, the subsets are nodes, the original data table is the root node, and the final unpartitioned subsets are leaves or terminal nodes. A node is an internal node if it is not a leaf. The data in a leaf determine the estimates of the value of the target variable. These estimates are subsequently applied to predict the target of a new observation that is assigned to the leaf.

The FOREST procedure creates multiple decision trees that differ from each other in two ways: First, the training data for each tree constitute a different sample; each sample is created by sampling with replacement observations from the original training data of the forest. Second, the input variables that are considered for splitting a node are randomly selected from all available inputs. Among these randomly selected variables, the FOREST procedure chooses a single variable, which is associated the most with the target, when it forms a splitting rule.

PROC FOREST Features

The FOREST procedure creates an ensemble of decision trees to predict a single target of either interval or nominal measurement level. An input variable can have an interval or nominal measurement level.

The FOREST procedure ignores any observation from the training data that has a missing target value.
Using CAS Sessions and CAS Engine Librefs

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

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

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

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

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

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

```sas
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 5 in Chapter 2, “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 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “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 6.1 describes the variables in `Hmeq`. 
Table 6.1 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 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>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 '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.

```latex
/* Convert variable names to mixed case */
data mycas.hmeq;
  length Bad Loan MortDue Value 8 Reason Job $7 YoJ Derog Delinq CLAge nlnq CLNo DebtInc 8;
  set sampsio.hmeq;
run;
```

Output 6.1 shows the first 10 observations of mycas.hmeq.

Figure 6.1 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>nlnq</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>1.0</td>
<td>HomeImp</td>
<td>10.5</td>
<td>0</td>
<td>0</td>
<td>94.367</td>
<td>1.0</td>
<td>9.0</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>5.0</td>
<td>HomeImp</td>
<td>5.0</td>
<td>3</td>
<td>2</td>
<td>77.100</td>
<td>1.0</td>
<td>17.0</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2000</td>
<td>.</td>
<td>62250</td>
<td>1.0</td>
<td>HomeImp Sales</td>
<td>16.0</td>
<td>0</td>
<td>0</td>
<td>115.800</td>
<td>13.0</td>
<td>0.0</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2000</td>
<td>45000</td>
<td>55000</td>
<td>3.0</td>
<td>HomeImp</td>
<td>3.0</td>
<td>0</td>
<td>0</td>
<td>86.067</td>
<td>2.0</td>
<td>25.0</td>
<td>.</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2200</td>
<td>24280</td>
<td>34687</td>
<td>0.0</td>
<td>HomeImp</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>300.867</td>
<td>0.0</td>
<td>8.0</td>
<td>.</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2300</td>
<td>28192</td>
<td>40150</td>
<td>4.5</td>
<td>HomeImp</td>
<td>4.5</td>
<td>0</td>
<td>0</td>
<td>54.600</td>
<td>1.0</td>
<td>16.0</td>
<td>.</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2400</td>
<td>50000</td>
<td>73395</td>
<td>5.0</td>
<td>HomeImp ProFEx</td>
<td>5.0</td>
<td>1</td>
<td>0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2400</td>
<td>.</td>
<td>17180</td>
<td>0.0</td>
<td>HomeImp</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>14.567</td>
<td>3.0</td>
<td>4.0</td>
<td>.</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>2500</td>
<td>15000</td>
<td>20200</td>
<td>18.0</td>
<td>HomeImp</td>
<td>18.0</td>
<td>0</td>
<td>0</td>
<td>136.067</td>
<td>1.0</td>
<td>19.0</td>
<td>.</td>
</tr>
</tbody>
</table>
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:

```sas
proc forest data=mycas.hmeq outmodel=mycas.savedModel;
   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 6.2 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 6.2 Model Information](image)

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 6.3 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 6.3  Number of Observations

<table>
<thead>
<tr>
<th></th>
<th>Training</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 6.4 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 6.4  Variable Importance

<table>
<thead>
<tr>
<th>Variable</th>
<th>Importance</th>
<th>Std Dev Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>DebtInc</td>
<td>274.92</td>
<td>83.5246</td>
</tr>
<tr>
<td>Delinq</td>
<td>90.6345</td>
<td>13.3056</td>
</tr>
<tr>
<td>Value</td>
<td>73.9882</td>
<td>24.1221</td>
</tr>
<tr>
<td>Derog</td>
<td>53.3015</td>
<td>7.7220</td>
</tr>
<tr>
<td>nlnq</td>
<td>42.7026</td>
<td>3.3223</td>
</tr>
<tr>
<td>CLNo</td>
<td>30.3675</td>
<td>3.8418</td>
</tr>
<tr>
<td>CLAge</td>
<td>26.3095</td>
<td>4.9389</td>
</tr>
<tr>
<td>Job</td>
<td>24.7731</td>
<td>2.4681</td>
</tr>
<tr>
<td>MortDue</td>
<td>23.1489</td>
<td>3.0192</td>
</tr>
<tr>
<td>YoJ</td>
<td>21.7237</td>
<td>2.2514</td>
</tr>
<tr>
<td>Loan</td>
<td>21.1239</td>
<td>6.0751</td>
</tr>
<tr>
<td>Reason</td>
<td>5.0869</td>
<td>1.5725</td>
</tr>
</tbody>
</table>

Figure 6.5 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 92. The listing shows that the out-of-bag error estimate is worse (larger) than the estimate that evaluates all observations on all trees. This is common.
### Figure 6.5  Fit Statistics

<table>
<thead>
<tr>
<th>Number of Trees</th>
<th>OOB Average 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>0.0939</td>
<td>0.0802</td>
<td>0.117</td>
<td>0.1023</td>
<td>0.725</td>
<td>0.497</td>
</tr>
<tr>
<td>2</td>
<td>0.1709</td>
<td>0.0711</td>
<td>0.122</td>
<td>0.0936</td>
<td>0.819</td>
<td>0.279</td>
</tr>
<tr>
<td>3</td>
<td>0.2149</td>
<td>0.0684</td>
<td>0.120</td>
<td>0.0901</td>
<td>0.840</td>
<td>0.243</td>
</tr>
<tr>
<td>4</td>
<td>0.2387</td>
<td>0.0656</td>
<td>0.118</td>
<td>0.0878</td>
<td>0.854</td>
<td>0.231</td>
</tr>
<tr>
<td>5</td>
<td>0.2566</td>
<td>0.0661</td>
<td>0.116</td>
<td>0.0846</td>
<td>0.849</td>
<td>0.233</td>
</tr>
<tr>
<td>6</td>
<td>0.2685</td>
<td>0.0646</td>
<td>0.114</td>
<td>0.0827</td>
<td>0.842</td>
<td>0.228</td>
</tr>
<tr>
<td>7</td>
<td>0.2760</td>
<td>0.0644</td>
<td>0.112</td>
<td>0.0820</td>
<td>0.826</td>
<td>0.227</td>
</tr>
<tr>
<td>8</td>
<td>0.2811</td>
<td>0.0652</td>
<td>0.112</td>
<td>0.0807</td>
<td>0.810</td>
<td>0.231</td>
</tr>
<tr>
<td>9</td>
<td>0.2849</td>
<td>0.0651</td>
<td>0.109</td>
<td>0.0785</td>
<td>0.817</td>
<td>0.231</td>
</tr>
<tr>
<td>10</td>
<td>0.2875</td>
<td>0.0648</td>
<td>0.107</td>
<td>0.0789</td>
<td>0.819</td>
<td>0.230</td>
</tr>
<tr>
<td>91</td>
<td>0.3052</td>
<td>0.0628</td>
<td>0.103</td>
<td>0.0827</td>
<td>0.852</td>
<td>0.222</td>
</tr>
<tr>
<td>92</td>
<td>0.3054</td>
<td>0.0627</td>
<td>0.103</td>
<td>0.0829</td>
<td>0.853</td>
<td>0.221</td>
</tr>
<tr>
<td>93</td>
<td>0.3053</td>
<td>0.0628</td>
<td>0.103</td>
<td>0.0829</td>
<td>0.852</td>
<td>0.222</td>
</tr>
<tr>
<td>94</td>
<td>0.3053</td>
<td>0.0627</td>
<td>0.102</td>
<td>0.0824</td>
<td>0.853</td>
<td>0.221</td>
</tr>
<tr>
<td>95</td>
<td>0.3054</td>
<td>0.0627</td>
<td>0.102</td>
<td>0.0826</td>
<td>0.853</td>
<td>0.222</td>
</tr>
<tr>
<td>96</td>
<td>0.3054</td>
<td>0.0627</td>
<td>0.103</td>
<td>0.0824</td>
<td>0.853</td>
<td>0.221</td>
</tr>
<tr>
<td>97</td>
<td>0.3056</td>
<td>0.0626</td>
<td>0.102</td>
<td>0.0824</td>
<td>0.853</td>
<td>0.221</td>
</tr>
<tr>
<td>98</td>
<td>0.3055</td>
<td>0.0626</td>
<td>0.102</td>
<td>0.0826</td>
<td>0.853</td>
<td>0.221</td>
</tr>
<tr>
<td>99</td>
<td>0.3055</td>
<td>0.0626</td>
<td>0.102</td>
<td>0.0827</td>
<td>0.853</td>
<td>0.221</td>
</tr>
<tr>
<td>100</td>
<td>0.3055</td>
<td>0.0626</td>
<td>0.103</td>
<td>0.0826</td>
<td>0.853</td>
<td>0.221</td>
</tr>
</tbody>
</table>
Syntax: FOREST Procedure

The following statements are available in the FOREST procedure:

**PROC FOREST** < options > ;
**AUTOTUNE** < options > ;
**CODE** < options > ;
**CROSSVALIDATION** < **KFO** LD=number > ;
**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 > ;

The **PROC FOREST**, **INPUT**, and **TARGET** statements are required. The **INPUT** statement can appear multiple times.

PROC FOREST Statement

**PROC FOREST** < options > ;

The **PROC FOREST** statement invokes the procedure. **Table 6.2** summarizes the options in the **PROC FOREST** statement.
Table 6.2 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>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>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>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>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**.

**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 75.

- **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 75.

When you use the INMODEL= option, the OUTPUT statement is required and any other options in the PROC FOREST statement, except for NOPRINT, 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.
**PROC FOREST Statement**

**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** = \( \text{number} \)
specifies the maximum depth of the tree to be grown. The number of levels in a tree is equal to the depth plus one.

By default, MAXDEPTH=20.

**MINLEAFSIZE** = \( \text{number} \)
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** = \( \text{number} \)
specifies a threshold for using missing values in the split search when ASSIGNMISSING=USEINSEARCH. If the number of observations in which the splitting variable has missing values is greater than or equal to \( \text{number} \), then PROC FOREST uses the USEINSEARCH policy to handle missing values for that variable.

By default, MINUSERINSEARCH=1.

**NOPRINT**
suppresses ODS output.

**NTREES** = \( \text{number} \)
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** = \( \text{number} \)
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** = \(< \text{CAS-libref.}> \text{data-table}\)
specifies the data table to which to save the forest model. \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 \text{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 75.

**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 96.
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.

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.

By default, VOTE=PROBABILITY.

AUTOTUNE Statement

AUTOTUNE <options> ;

The AUTOTUNE statement searches for the best combination of values of the INBAGFRACTION=, 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.

You can specify the following options:

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). 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 validation partition is usually sufficient for estimating error; a single partition is more efficient than cross validation in terms of the total execution time.

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

By default, FRACTION=0.3.
K F O L D = n u m b e r

specifies the number of partition folds 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 1 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).

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

M A X E V A L S = n u m b e r

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.

M A X I T E R = n u m b e r

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 might lead to termination before the value of the MAXEVALS= option or the MAXTIME= option is reached.

By default, MAXITER=5.

M A X T I M E = n u m b e r

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.

P O P S I Z E = n u m b e r

specifies the maximum number of evaluations in one iteration (population), where number must be greater than or equal to 1. In some cases, the tuner algorithm might generate a number of new configurations smaller than number.

By default, POPSIZE=10.

T U N I N G P A R A M E T E R S = ( s u b o p t i o n | . . . | < suboption > )

T U N E P A R M S = ( s u b o p t i o n | . . . | < 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.

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.

**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.

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.

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 to use for tuning the forest model. For more information, see the VARS_TO_TRY= option.

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.

USEPARAMETERS=tuning-parameter-option
specifies which set of parameters to tune.

You can specify the following tuning-parameter-options:

STANDARD tunes using the default bounds and initial values for all parameters.

CUSTOM tunes only the parameters that are specified in the TUNINGPARAMETERS= option.
Chapter 6: The FOREST Procedure

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

By default, USEPARAMETERS=COMBINED.

CODE Statement

CODE < options > ;

The CODE statement writes SAS DATA step code for computing predicted values of the fitted model either to a file or to a catalog entry. This code can then be included in a DATA step to score new data.

Table 6.3 summarizes the options available in the CODE statement.

Table 6.3  CODE Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENT</td>
<td>Adds comments to the generated code</td>
</tr>
<tr>
<td>FILE=</td>
<td>Names the file where the generated code is saved</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>PCATALL</td>
<td>Generates probabilities for all levels of categorical response variables</td>
</tr>
</tbody>
</table>

For more information about the syntax of the CODE statement, see the section “CODE Statement” on page 7 in Chapter 2, “Shared Concepts.”

CROSSVALIDATION Statement

CROSSVALIDATION < KFOLD=number > ;

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

You can specify the following option:

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.
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=NOMINAL | INTERVAL >;

The INPUT statement names input variables that share common options. The INPUT statement can be repeated.

You can specify the following option:

LEVEL=NOMINAL | INTERVAL

specifies the level of measurement of two variables. 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.

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 75.

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.

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 10 in Chapter 2, “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 3, “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 75.

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.

Details: FOREST Procedure

Bagging the Data

A decision tree in a forest trains on new training data that are derived from the original training data presented to the FOREST procedure. Using different data to train different trees reduces the correlation of the predictions of the trees, which in turn should improve the predictions of the forest.

The FOREST procedure samples the original data with replacement to create the training data for an individual tree. The convention of sampling with replacement originated with Leo Breiman’s bagging algorithm (Breiman 1996, 2001). The word bagging stems from “bootstrap aggregating,” where “bootstrap” refers to a process that uses sampling with replacement. Breiman refers to the observations that are excluded from the sample as out-of-bag (OOB) observations. Therefore, observations in the training sample are called the bagged observations, and the training data for a specific decision tree are called the bagged data.

The INBAGFRACTION= option in the PROC FOREST statement specifies the number of observations to sample with replacement into the bagged data.

Estimating the goodness of fit of the model by using the training data is usually too optimistic; the fit of the model to new data is usually worse than the fit to the training data. Estimating the goodness of fit by using the out-of-bag data is usually too pessimistic at first. With enough trees, the out-of-bag estimates are an unbiased estimate of the generalization fit.
Training a Decision Tree

The FOREST procedure trains a decision tree by splitting the bagged data, then splitting each of the resulting segments, and so on recursively until some constraint is met.

Splitting involves the following subtasks:

1. selecting candidate inputs
2. computing the association of each input with the target
3. searching for the best split that uses the most highly associated inputs

PROC FOREST randomly selects \( m \) candidate input variables independently in every node, where \( m \) is the value of the VARS_TO_TRY= option in the PROC FOREST statement. If you specify \( L \) as the value of the LOH= option and \( L < m \), then PROC FOREST chooses the best \( L \) input variables from the \( m \) variables according to the criterion described in section “Loh Method” on page 93. A split search is performed on all \( L \) or \( m \) variables, and the best rule is kept to split the node.

The reason for searching fewer input variables for a splitting rule instead of searching all inputs and choosing the best split is to improve prediction on new data. An input that offers more splitting possibilities provides the search routine more chances to find a spurious split. Loh and Shih (1997) demonstrate the bias towards spurious splits that result. They also demonstrate that preselecting the input variable and then searching only on that one input reduces the bias. You can choose to preselect a number of input variables by using the LOH= option.

The split search seeks to maximize the reduction in the gain for a nominal target and the reduction in variance of an interval target.

Loh Method

Specify the LOH= option to use ideas developed by Loh in a series of papers (Loh and Shih 1997; Loh 2002, 2009).

This method selects the number of variables that have the smallest \( p \)-value of a chi-square test of association in a contingency table. These variables are selected from the VARS_TO_TRY=\( m \) randomly selected variables that are chosen for a single decision tree in the forest.

Let \( Y \) and \( X \) denote the target variable and input variable, respectively. Let \( Y_i \) and \( X_i \) denote their values in observation \( i \). If \( Y \) is categorical, let \( J \) denote the number of values. Similarly, if \( X \) is categorical, let \( K \) denote the number of values.

If both \( Y \) and \( X \) are categorical, then form the \( J \times K \) contingency table of the frequencies of the observations and compute the \( p \)-value. If \( Y \) has an interval measurement level, then note whether \( Y_i \) is greater than or less than the average of \( Y \), \( \bar{Y} \), in the node, and then form the \( 2 \times K \) table of frequencies and compute the \( p \)-value.

If \( X \) has an interval measurement level, then let

\[
K = \begin{cases} 
3 & \text{if } N < 20J \\
4 & \text{otherwise}
\end{cases}
\]

where \( N \) is the number of observations in the calculations and \( J = 2 \) if \( Y \) has an interval measurement level.
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

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

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

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

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

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

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

The definitions are valid whether \(\hat{y}_i\) is the usual model prediction or the out-of-bag prediction. The \(\text{ASE}_{\text{int}}\) that is based on the usual model predictions of the original training data is usually optimistic and smaller than what its value will be 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.

Residual Sum of Squares Importance Method

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

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

where

- $i$ is an observation on leaf $\lambda$
- $y_i$ is the predicted value of the response variable of observation $i$
• $\bar{y}_\lambda$ is the actual value of the response variable on leaf $\lambda$

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

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

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 is

$$\Delta_d = RSS_d - \sum_i RSS^d_i$$

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
• $RSS^d_i$ 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 tree is then defined as

$$\sqrt{\sum_{d=1}^D \Delta_d}$$

where $D$ is the total number of nodes.

The RSS variable importance for the forest is the average of the RSS variable importance across all trees in the forest.
Random Branch Assignment Importance Method

The random branch assignment (RBA) method computes the importance of an input variable $v$ by comparing how the data fit the predictions with how the data fit modified predictions. 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, $P(\text{branch})$, is proportional to the number of training observations that were assigned to the branch during construction of the model.

The RBA importance can be expressed mathematically as

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

where $\tilde{y}_i$ 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 square error loss and absolute error loss. For a categorical target, PROC FOREST computes square error loss and negative margin loss.

Neville and Tan (2014) motivate and introduce the RBA method of variable importance.

Parameter Tuning

The quality of the predictive model that PROC FOREST 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. The AUTOTUNE statement engages the optimization algorithm (tuner), which searches for the best possible combination of values of these select hyperparameters while trying to minimize the objective function. The objective function is a validation error estimate (misclassification error for nominal targets or average square error for interval targets). The tuning process includes multiple iterations; each iteration usually involves multiple objective function evaluations. Each objective function evaluation can consist of one or several training and scoring executions as follows:

- If you specify the PARTITION statement, the tuner uses a single-partition validation set as defined in that statement. 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.

- If you specify the FRACTION= option, the tuner uses a single-partition validation set. 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.

- If KFOLD=k is specified, the tuner uses $k$-fold cross validation. In this process, the tuner partitions all the data into $k$-fold subsets (folds). For each fold, a new model is trained on each of the ($k$-1) folds and then validated using the selected (holdout) fold. The objective function value is averaged over each set of training and scoring executions to obtain a single error estimate value.
The optimization tuner algorithm is based on a genetic algorithm (GA), which applies the principles of natural selection and evolution to find an improved configuration. The tuner performs the following sequence of actions:

1. A default model configuration (default values of select model tuning parameters) 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. The initial set of configurations, also called a “population,” is generated using a technique called random Latin hypercube sampling (LHS). In a Latin hypercube sample, each configuration of hyperparameters is evaluated, and their objective function values are again recorded for comparison. This becomes Iteration 1.

3. The best model configurations from the initial population are used to generate the next population of model configurations, Iteration 2, which are then evaluated. This process is repeated for the remaining iterations, as long as the maximum number of evaluations or the maximum time is not reached.

4. The best model configuration is reevaluated by executing a single training and model 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 ODS table, as described in the section “ODS Table Names” on page 101.

You can tune the following hyperparameter values when you specify the AUTOTUNE statement:

- the NTREES= option for the number of trees to grow in the forest.
- the INBAGFRACTION= option for the bootstrap sample size for building each tree in the forest.
- the 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 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 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 96. 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 98. 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 94. 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 Results

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

Output CAS Tables

When you specify the OUTPUT statement or the OUTMODEL= option in the PROC FOREST statement to create output tables on your CAS server, the FOREST procedure produces the output data table along with a table that lists the CAS library, the data table name, and the number of rows and columns in that data table.

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 6.4.

<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>CrossValidationResults</td>
<td>Average error rate (misclassification error or average square error) of ( k )-fold cross validation</td>
<td>CROSSVALIDATION</td>
<td>Default</td>
</tr>
<tr>
<td>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>
</tbody>
</table>
### Table 6.4  continued

<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</td>
<td>PROC FOREST</td>
<td>Default</td>
</tr>
<tr>
<td>OutCASTbl</td>
<td>Summary table for output data; it contains the number of observations and the number of variables</td>
<td>PROC FOREST / OUTPUT</td>
<td>OUTMODEL= / Default</td>
</tr>
<tr>
<td>RBAVariableImportance</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>
Example: 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 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “Shared Concepts.”

Example 6.1: 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;
  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 6.1.1. The table shows the training and out-of-bag statistics.
Chapter 6: The FOREST Procedure

Output 6.1.1  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>0.114</td>
<td>0.1052</td>
<td>0.148</td>
<td>0.1448</td>
<td>0.538</td>
<td>0.430</td>
</tr>
<tr>
<td>2</td>
<td>0.192</td>
<td>0.0926</td>
<td>0.140</td>
<td>0.1274</td>
<td>0.769</td>
<td>0.337</td>
</tr>
<tr>
<td>3</td>
<td>0.226</td>
<td>0.0857</td>
<td>0.133</td>
<td>0.1150</td>
<td>0.787</td>
<td>0.282</td>
</tr>
<tr>
<td>4</td>
<td>0.254</td>
<td>0.0849</td>
<td>0.132</td>
<td>0.1121</td>
<td>0.823</td>
<td>0.280</td>
</tr>
<tr>
<td>5</td>
<td>0.275</td>
<td>0.0831</td>
<td>0.132</td>
<td>0.1089</td>
<td>0.849</td>
<td>0.277</td>
</tr>
<tr>
<td>6</td>
<td>0.284</td>
<td>0.0803</td>
<td>0.122</td>
<td>0.1054</td>
<td>0.844</td>
<td>0.270</td>
</tr>
<tr>
<td>7</td>
<td>0.293</td>
<td>0.0807</td>
<td>0.118</td>
<td>0.1000</td>
<td>0.834</td>
<td>0.272</td>
</tr>
<tr>
<td>8</td>
<td>0.298</td>
<td>0.0797</td>
<td>0.115</td>
<td>0.0963</td>
<td>0.841</td>
<td>0.271</td>
</tr>
<tr>
<td>9</td>
<td>0.302</td>
<td>0.0794</td>
<td>0.114</td>
<td>0.0982</td>
<td>0.843</td>
<td>0.271</td>
</tr>
<tr>
<td>10</td>
<td>0.306</td>
<td>0.0795</td>
<td>0.114</td>
<td>0.0976</td>
<td>0.850</td>
<td>0.272</td>
</tr>
<tr>
<td>91</td>
<td>0.332</td>
<td>0.0788</td>
<td>0.103</td>
<td>0.0878</td>
<td>0.892</td>
<td>0.274</td>
</tr>
<tr>
<td>92</td>
<td>0.332</td>
<td>0.0787</td>
<td>0.103</td>
<td>0.0876</td>
<td>0.892</td>
<td>0.274</td>
</tr>
<tr>
<td>93</td>
<td>0.331</td>
<td>0.0787</td>
<td>0.103</td>
<td>0.0880</td>
<td>0.892</td>
<td>0.274</td>
</tr>
<tr>
<td>94</td>
<td>0.331</td>
<td>0.0787</td>
<td>0.102</td>
<td>0.0880</td>
<td>0.892</td>
<td>0.274</td>
</tr>
<tr>
<td>95</td>
<td>0.331</td>
<td>0.0787</td>
<td>0.102</td>
<td>0.0878</td>
<td>0.891</td>
<td>0.274</td>
</tr>
<tr>
<td>96</td>
<td>0.331</td>
<td>0.0787</td>
<td>0.102</td>
<td>0.0885</td>
<td>0.891</td>
<td>0.274</td>
</tr>
<tr>
<td>97</td>
<td>0.331</td>
<td>0.0787</td>
<td>0.103</td>
<td>0.0887</td>
<td>0.892</td>
<td>0.274</td>
</tr>
<tr>
<td>98</td>
<td>0.331</td>
<td>0.0787</td>
<td>0.103</td>
<td>0.0887</td>
<td>0.892</td>
<td>0.273</td>
</tr>
<tr>
<td>99</td>
<td>0.331</td>
<td>0.0788</td>
<td>0.103</td>
<td>0.0887</td>
<td>0.891</td>
<td>0.274</td>
</tr>
<tr>
<td>100</td>
<td>0.331</td>
<td>0.0787</td>
<td>0.104</td>
<td>0.0887</td>
<td>0.892</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 6.1.2 match those previously seen in Output 6.1.1.
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.

References


Chapter 7
The GRADBOOST Procedure

Contents

Overview: GRADBOOST Procedure ............................................. 108
Using CAS Sessions and CAS Engine Librefs .......................... 108
Getting Started: GRADBOOST Procedure ................................. 109
Syntax: GRADBOOST Procedure .................................................. 113
  PROC GRADBOOST Statement .............................................. 113
  AUTOTUNE Statement ...................................................... 117
  CODE Statement .......................................................... 122
  CROSSVALIDATION Statement .......................................... 123
  ID Statement .............................................................. 123
  INPUT Statement .......................................................... 123
  OUTPUT Statement ........................................................ 124
  PARTITION Statement ..................................................... 124
  SAVESTATE Statement ..................................................... 125
  TARGET Statement ........................................................ 125
Details: GRADBOOST Procedure ................................................ 126
  Subsampling the Data ...................................................... 126
  Training a Decision Tree .................................................. 126
  Boosting ............................................................................. 126
  Measuring Prediction Error ................................................ 127
  Handling Missing Values .................................................... 127
    Strategies ..................................................................... 127
    Specifics ......................................................................... 128
  Measuring Variable Importance ........................................... 128
    Residual Sum of Squares Importance Method ..................... 129
  Parameter Tuning ............................................................ 130
  k-fold Cross Validation ..................................................... 131
  Displayed Output ............................................................. 132
    Model Information ........................................................ 132
    Number of Observations ............................................... 132
    Variable Importance ..................................................... 132
    Fit Statistics ............................................................... 132
    Tuner Information ......................................................... 132
    Tuner Summary ............................................................ 132
    Tuner Timing ............................................................... 132
    Best Configuration ....................................................... 132
    Tuner Results ............................................................. 133
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 126.

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 5 in Chapter 2, “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 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “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 7.1 describes the variables in *Hmeq*.

#### Table 7.1 Variables in the Home Equity (Hmeq) Data Table

<table>
<thead>
<tr>
<th>Variable</th>
<th>Role</th>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
</table>
| Bad      | Response | Binary  | 1 = applicant defaulted on the loan or is seriously delinquent  
            |         |         | 0 = applicant paid off the loan                        |
| CLAge    | Predictor | Interval | Age of oldest credit line in months                      |
| CLNo     | Predictor | Interval | Number of credit lines                                  |
| DebtInc  | Predictor | Interval | Debt-to-income ratio                                    |
| Delinq   | Predictor | Interval | Number of delinquent credit lines                       |
| Derog    | Predictor | Interval | Number of major derogatory reports                      |
| Job      | Predictor | Nominal  | Occupational category                                   |
| Loan     | Predictor | Interval | Requested loan amount                                   |
| MortDue  | Predictor | Interval | Amount due on mortgage                                  |
| nlnq     | Predictor | Interval | Number of recent credit inquiries                        |
| Reason   | Predictor | Binary   | 'DebtCon' = debt consolidation                           |
            |         |         | 'HomeImp' = home improvement                            |
| Value    | Predictor | Interval | Value of property                                        |
| YoJ      | Predictor | Interval | Years at present job                                    |
The following statements load the `mycas.hmeq` data into your CAS session. For this example, the statements assume that your CAS engine libref is named `mycas`, but you can substitute any appropriately defined CAS engine libref.

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

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

Output 7.1 shows the first 10 observations of `mycas.hmeq`.

![Figure 7.1 Partial Listing of the mycas.hmeq Data](image)

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`:

```sas
proc gradboost data=mycas.hmeq outmodel=mycas.savedModel;
    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 7.2 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 7.2 Model Information**

The GRADBOOST Procedure

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Trees</td>
</tr>
<tr>
<td>Learning Rate</td>
</tr>
<tr>
<td>Subsampling Rate</td>
</tr>
<tr>
<td>Number of Variables Per Split</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>Seed</td>
</tr>
</tbody>
</table>

Figure 7.3 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 7.3 Number of Observations**

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

Figure 7.4 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.
Chapter 7: The GRADBOOST Procedure

Figure 7.4 Variable Importance

<table>
<thead>
<tr>
<th>Variable</th>
<th>Importance</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>DebtInc</td>
<td>26.9257</td>
<td>53.6874</td>
</tr>
<tr>
<td>Delinq</td>
<td>7.3450</td>
<td>5.1979</td>
</tr>
<tr>
<td>nlnq</td>
<td>6.1775</td>
<td>1.8946</td>
</tr>
<tr>
<td>Job</td>
<td>5.4479</td>
<td>1.7540</td>
</tr>
<tr>
<td>Derog</td>
<td>5.3054</td>
<td>3.0248</td>
</tr>
<tr>
<td>CLAge</td>
<td>5.0686</td>
<td>2.6648</td>
</tr>
<tr>
<td>Value</td>
<td>4.9606</td>
<td>3.2630</td>
</tr>
<tr>
<td>CLNo</td>
<td>4.9280</td>
<td>1.4812</td>
</tr>
<tr>
<td>YoJ</td>
<td>4.8388</td>
<td>1.5184</td>
</tr>
<tr>
<td>MortDue</td>
<td>4.7727</td>
<td>1.6993</td>
</tr>
<tr>
<td>Loan</td>
<td>3.2080</td>
<td>1.5420</td>
</tr>
<tr>
<td>Reason</td>
<td>1.1742</td>
<td>1.3284</td>
</tr>
</tbody>
</table>

Figure 7.5 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.

Figure 7.5 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>0.1443</td>
<td>0.1995</td>
<td>0.455</td>
</tr>
<tr>
<td>2</td>
<td>0.1324</td>
<td>0.1995</td>
<td>0.424</td>
</tr>
<tr>
<td>3</td>
<td>0.1222</td>
<td>0.1995</td>
<td>0.399</td>
</tr>
<tr>
<td>4</td>
<td>0.1147</td>
<td>0.1961</td>
<td>0.380</td>
</tr>
<tr>
<td>5</td>
<td>0.1084</td>
<td>0.1763</td>
<td>0.364</td>
</tr>
<tr>
<td>6</td>
<td>0.1033</td>
<td>0.1473</td>
<td>0.351</td>
</tr>
<tr>
<td>7</td>
<td>0.0981</td>
<td>0.1352</td>
<td>0.337</td>
</tr>
<tr>
<td>8</td>
<td>0.0941</td>
<td>0.1201</td>
<td>0.326</td>
</tr>
<tr>
<td>9</td>
<td>0.0908</td>
<td>0.1124</td>
<td>0.316</td>
</tr>
<tr>
<td>10</td>
<td>0.0877</td>
<td>0.1059</td>
<td>0.307</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>91</td>
<td>0.0338</td>
<td>0.0436</td>
<td>0.126</td>
</tr>
<tr>
<td>92</td>
<td>0.0336</td>
<td>0.0436</td>
<td>0.125</td>
</tr>
<tr>
<td>93</td>
<td>0.0333</td>
<td>0.0433</td>
<td>0.125</td>
</tr>
<tr>
<td>94</td>
<td>0.0330</td>
<td>0.0421</td>
<td>0.123</td>
</tr>
<tr>
<td>95</td>
<td>0.0326</td>
<td>0.0416</td>
<td>0.122</td>
</tr>
<tr>
<td>96</td>
<td>0.0324</td>
<td>0.0421</td>
<td>0.122</td>
</tr>
<tr>
<td>97</td>
<td>0.0322</td>
<td>0.0421</td>
<td>0.121</td>
</tr>
<tr>
<td>98</td>
<td>0.0320</td>
<td>0.0418</td>
<td>0.120</td>
</tr>
<tr>
<td>99</td>
<td>0.0319</td>
<td>0.0423</td>
<td>0.119</td>
</tr>
<tr>
<td>100</td>
<td>0.0317</td>
<td>0.0409</td>
<td>0.119</td>
</tr>
</tbody>
</table>
The following statements are available in the GRADBOOST procedure:

- **PROC GRADBOOST** <options> ;
- **AUTOTUNE** <options> ;
- **CODE** <options> ;
- **CROSSVALIDATION** <KFOLD=number> ;
- **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 > ;

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 7.2 summarizes the options in the PROC GRADBOOST statement.
## Table 7.2 PROC GRADBOOST 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>INMODEL=</td>
<td>Specifies a saved gradient boosting model to use to score new data.</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>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><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 also specify the following options:

**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 126. 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.

**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 108.

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

**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 108.

When you use the INMODEL= option, both the DATA= option and the OUTPUT statement are required, and any other options, except for NOPRINT, are ignored.

**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.

**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 108.

** 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.

AUTOTUNE Statement

AUTOTUNE < 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.

You can specify the following options:

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). 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 validation partition is usually sufficient for estimating error; a single partition is more efficient than cross validation in terms of the total execution time.

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

By default, FRACTION=0.3.

KFOLD=number
specifies the number of partition folds 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 1 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).
You cannot specify this option in combination with the FRACTION= option. If a PARTITION statement is specified, the validation partition defined in that statement is used, and this option is ignored.

**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 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.

**POPSIZE**=`number`

specifies the maximum number of evaluations in one iteration (population), where `number` must be greater than or equal to 1. In some cases, the tuner algorithm might generate a number of new configurations smaller than `number`.

By default, POPSIZE=10.

**TUNINGPARAMETERS**=`(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.

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.

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.

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.

**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.

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.

**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.

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.

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.
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.

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.
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.
Chapter 7: The GRADBOOST Procedure

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.

USEPARAMETERS=tuning-parameter-option
specifies which set of parameters to tune.

You can specify the following tuning-parameter-options:

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

By default, USEPARAMETERS=COMBINED.

CODE Statement

CODE < options > ;

The CODE statement writes SAS DATA step code for computing predicted values of the fitted model either to a file or to a catalog entry. This code can then be included in a DATA step to score new data.

Table 7.3 summarizes the options available in the CODE statement.
Table 7.3  CODE Statement Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENT</td>
<td>Adds comments to the generated code</td>
</tr>
<tr>
<td>FILE=</td>
<td>Names the file where the generated code is saved</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>PCATALL</td>
<td>Generates probabilities for all levels of categorical response variables</td>
</tr>
</tbody>
</table>

For more information about the syntax of the CODE statement, see the section “CODE Statement” on page 7 in Chapter 2, “Shared Concepts.”

CROSSVALIDATION Statement

CROSSVALIDATION <KFDOL=number> ;

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

You can specify the following option:

KFDOL=number
  specifies the number of partition folds in the cross validation process, where number must be between 2 and 20, inclusive.
  By default, KFDOL=5.

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=NOMINAL | INTERVAL > ;

The INPUT statement names input variables that share common options. The INPUT statement can be repeated.

You can specify the following option:
LEVEL=NOMINAL | INTERVAL
specifies the level of measurement of two variables. 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.

OUTPUT Statement

OUTPUT OUT=CAS-libref.data-table < option >;

The OUTPUT statement creates an output data table that contains the results of running PROC GRADBOOST.

You must specify the following option:

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

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about CAS-libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 108.

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.

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 10 in Chapter 2, “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:
**SAVESTATE Statement**

```plaintext
SAVESTATE RSTORE=CAS-libref.data-table ;
```

The SAVESTATE statement creates an analytic store for the model and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to score new data. For more information, see Chapter 3, “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 108.

**TARGET Statement**

```plaintext
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.
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).

PROC GRADBOOST 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 126. The sequence of trees and how each tree affects a subsequent tree are discussed in Hastie, Tibshirani, and Friedman (2001) and Friedman (2001).
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 calculates the variable importance by using the change in the residual sum of square errors.

**Residual Sum of Squares Importance Method**

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

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

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

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

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 is

$$\Delta_d = \text{RSS}_d - \sum_i \text{RSS}_i^d$$

where

- $d$ denotes the node
- $i$ denotes the index of a child that this node includes
- $\text{RSS}_d$ is the RSS if the node is treated as a leaf
- $\text{RSS}_i^d$ is the RSS of the node after it has been split
If the change in RSS is negative (which is possible when you use the validation set), then the change is set to 0.

The RSS-based importance for a single tree is then defined as

\[
\sqrt{\sum_{d=1}^{D} \Delta_d}
\]

where \( D \) is the total number of nodes.

The RSS variable importance for the boosting model is the average of the RSS variable importance across all trees in the boosting model.

---

**Parameter Tuning**

The quality of the predictive model that PROC GRADBOOST 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. The AUTOTUNE statement engages the optimization algorithm (tuner), which searches for the best possible combination of values of these select hyperparameters while trying to minimize the objective function. The objective function is a validation error estimate (misclassification error for nominal targets or average square error for interval targets). The tuning process includes multiple iterations; each iteration usually involves multiple objective function evaluations. Each objective function evaluation can consist of one or several training and scoring executions as follows:

- If you specify the PARTITION statement, the tuner uses a single-partition validation set as defined in that statement. 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.

- If you specify the FRACTION= option, the tuner uses a single-partition validation set. 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.

- If KFOLD=k is specified, the tuner uses \( k \)-fold cross validation. In this process, the tuner partitions all the data into \( k \)-fold subsets (folds). For each fold, a new model is trained on each of the \((k-1)\) folds and then validated using the selected (holdout) fold. The objective function value is averaged over each set of training and scoring executions to obtain a single error estimate value.

The optimization tuner algorithm is based on a genetic algorithm (GA), which applies the principles of natural selection and evolution to find an improved configuration. The tuner performs the following sequence of actions:
1. A default model configuration (default values of select model tuning parameters) 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. The initial set of configurations, also called a “population,” is generated using a technique called random Latin hypercube sampling (LHS). In a Latin hypercube sample, each configuration of hyperparameters is evaluated, and their objective function values are again recorded for comparison. This becomes Iteration 1.

3. The best model configurations from the initial population are used to generate the next population of model configurations, Iteration 2, which are then evaluated. This process is repeated for the remaining iterations, as long as the maximum number of evaluations or the maximum time is not reached.

4. The best model configuration is reevaluated by executing a single training and model 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 ODS table, as described in the section “ODS Table Names” on page 133.

You can tune the following hyperparameter values when you specify the AUTOTUNE statement:

- the LASSO= option for the L1 regularization parameter
- the RIDGE= option for the L2 regularization parameter
- the LEARNINGRATE= option for the learning rate parameter
- the NTREES= option for the number of trees
- the SAMPLINGRATE= option for the proportion of the training data to sample
- the 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 128. This table is produced by default.

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

Output Table
The “Output Table” table describes tables that are created either when you specify the OUTMODEL option in the PROC GRADBOOST statement or when you specify the OUTPUT 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 7.4.

<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>CrossValidationResults</td>
<td>Average error rate (misclassification error or average square error) of $k$-fold cross validation</td>
<td>CROSSVALIDATION</td>
<td>Default</td>
</tr>
<tr>
<td>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>OutCASTbl</td>
<td>Output table</td>
<td>PROC GRADBOOST or OUTPUT</td>
<td>OUTMODEL=</td>
</tr>
<tr>
<td>TunerInfo</td>
<td>Setup values used by the tuner</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
</tbody>
</table>
### Table 7.4  continued

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>TunerResults</td>
<td>Values of the hyperparameters, the objective function for the default configuration (Iteration 0), and up to 10 best found configurations</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerSummary</td>
<td>Statistics about the tuning process</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerTiming</td>
<td>Total time spent on different tasks while tuning</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>VariableImportance</td>
<td>Residual sum of squares variable importance</td>
<td>PROC GRADBOOST</td>
<td>Default</td>
</tr>
</tbody>
</table>

### Example: GRADBOOST Procedure

**NOTE:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “Shared Concepts.”

### Example 7.1: Scoring New Data by Using a Previous Boosting Model

This example illustrates how you can use the OUTMODEL= option to save a model table, and later use the model table to score a data table. It uses the JunkMail data set in the Sashelp library.

The JunkMail data set comes from a study that classifies whether an email is junk email (coded as 1) or not (coded as 0). The data set contains 4,601 observations with 59 variables. The response variable is a binary indicator of whether an email is considered spam or not. There are 57 predictor variables that record the frequencies of some common words and characters and the lengths of uninterrupted sequences of capital letters in emails.

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

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

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

```sas
proc gradboost data=mycas.junkmail outmodel=mycas.gradboost_model;
    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 7.1.1. The table shows the training statistics.

**Output 7.1.1**  Fit Statistics, Fit at Run Time

<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>0.2144</td>
<td>0.3940</td>
<td>0.620</td>
</tr>
<tr>
<td>2</td>
<td>0.1953</td>
<td>0.2467</td>
<td>0.580</td>
</tr>
<tr>
<td>3</td>
<td>0.1800</td>
<td>0.2039</td>
<td>0.548</td>
</tr>
<tr>
<td>4</td>
<td>0.1669</td>
<td>0.1506</td>
<td>0.520</td>
</tr>
<tr>
<td>5</td>
<td>0.1563</td>
<td>0.1361</td>
<td>0.496</td>
</tr>
<tr>
<td>6</td>
<td>0.1472</td>
<td>0.1384</td>
<td>0.475</td>
</tr>
<tr>
<td>7</td>
<td>0.1391</td>
<td>0.1326</td>
<td>0.456</td>
</tr>
<tr>
<td>8</td>
<td>0.1326</td>
<td>0.1308</td>
<td>0.441</td>
</tr>
<tr>
<td>9</td>
<td>0.1270</td>
<td>0.1302</td>
<td>0.427</td>
</tr>
<tr>
<td>10</td>
<td>0.1217</td>
<td>0.1221</td>
<td>0.413</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>91</td>
<td>0.0570</td>
<td>0.0706</td>
<td>0.199</td>
</tr>
<tr>
<td>92</td>
<td>0.0569</td>
<td>0.0706</td>
<td>0.199</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>0.0553</td>
<td>0.0702</td>
<td>0.194</td>
</tr>
</tbody>
</table>
The following statements use a previously saved model to score new data:

```
proc gradboost data=mycas.junkmail inmodel=mycas.gradboost_model;
  output out=mycas.score_later;
  ods output FitStatistics=fit_later;
run;
```

When you specify the INMODEL= option to use a previously created boosting model, you see the statistics for the scored data if the target exists in the newly scored data table. In this example, the scored data are the same as the training data, so you can see that the statistics in Output 7.1.2 match those previously seen in Output 7.1.1.

**Output 7.1.2**  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>0.2144</td>
<td>0.3940</td>
<td>0.620</td>
</tr>
<tr>
<td>2</td>
<td>0.1953</td>
<td>0.2467</td>
<td>0.580</td>
</tr>
<tr>
<td>3</td>
<td>0.1800</td>
<td>0.2039</td>
<td>0.548</td>
</tr>
<tr>
<td>4</td>
<td>0.1669</td>
<td>0.1506</td>
<td>0.520</td>
</tr>
<tr>
<td>5</td>
<td>0.1563</td>
<td>0.1361</td>
<td>0.496</td>
</tr>
<tr>
<td>6</td>
<td>0.1472</td>
<td>0.1384</td>
<td>0.475</td>
</tr>
<tr>
<td>7</td>
<td>0.1391</td>
<td>0.1326</td>
<td>0.456</td>
</tr>
<tr>
<td>8</td>
<td>0.1326</td>
<td>0.1308</td>
<td>0.441</td>
</tr>
<tr>
<td>9</td>
<td>0.1270</td>
<td>0.1302</td>
<td>0.427</td>
</tr>
<tr>
<td>10</td>
<td>0.1217</td>
<td>0.1221</td>
<td>0.413</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>91</td>
<td>0.0570</td>
<td>0.0706</td>
<td>0.199</td>
</tr>
<tr>
<td>92</td>
<td>0.0569</td>
<td>0.0706</td>
<td>0.199</td>
</tr>
<tr>
<td>93</td>
<td>0.0568</td>
<td>0.0702</td>
<td>0.199</td>
</tr>
<tr>
<td>94</td>
<td>0.0565</td>
<td>0.0698</td>
<td>0.198</td>
</tr>
<tr>
<td>95</td>
<td>0.0564</td>
<td>0.0698</td>
<td>0.197</td>
</tr>
<tr>
<td>96</td>
<td>0.0561</td>
<td>0.0700</td>
<td>0.196</td>
</tr>
<tr>
<td>97</td>
<td>0.0559</td>
<td>0.0702</td>
<td>0.196</td>
</tr>
<tr>
<td>98</td>
<td>0.0557</td>
<td>0.0702</td>
<td>0.195</td>
</tr>
<tr>
<td>99</td>
<td>0.0555</td>
<td>0.0704</td>
<td>0.194</td>
</tr>
<tr>
<td>100</td>
<td>0.0553</td>
<td>0.0702</td>
<td>0.194</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.
References


# Chapter 8
## The NNET Procedure

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: NNET Procedure</td>
<td>140</td>
</tr>
<tr>
<td>PROC NNET Features</td>
<td>140</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>141</td>
</tr>
<tr>
<td>Getting Started: NNET Procedure</td>
<td>141</td>
</tr>
<tr>
<td>Syntax: NNET Procedure</td>
<td>143</td>
</tr>
<tr>
<td>PROC NNET Statement</td>
<td>143</td>
</tr>
<tr>
<td>ARCHITECTURE Statement</td>
<td>144</td>
</tr>
<tr>
<td>AUTOTUNE Statement</td>
<td>145</td>
</tr>
<tr>
<td>CODE Statement</td>
<td>151</td>
</tr>
<tr>
<td>CROSSVALIDATION Statement</td>
<td>151</td>
</tr>
<tr>
<td>HIDDEN Statement</td>
<td>151</td>
</tr>
<tr>
<td>INPUT Statement</td>
<td>152</td>
</tr>
<tr>
<td>OPTIMIZATION Statement</td>
<td>152</td>
</tr>
<tr>
<td>PARTITION Statement</td>
<td>154</td>
</tr>
<tr>
<td>SCORE Statement</td>
<td>155</td>
</tr>
<tr>
<td>TARGET Statement</td>
<td>156</td>
</tr>
<tr>
<td>TRAIN Statement</td>
<td>157</td>
</tr>
<tr>
<td>WEIGHT Statement</td>
<td>159</td>
</tr>
<tr>
<td>Details: NNET Procedure</td>
<td>159</td>
</tr>
<tr>
<td>Computational Method</td>
<td>159</td>
</tr>
<tr>
<td>Parameter Tuning</td>
<td>160</td>
</tr>
<tr>
<td><em>k</em>-fold Cross Validation</td>
<td>161</td>
</tr>
<tr>
<td>Displayed Output</td>
<td>161</td>
</tr>
<tr>
<td>Iteration History</td>
<td>162</td>
</tr>
<tr>
<td>Convergence Status</td>
<td>162</td>
</tr>
<tr>
<td>Model Information</td>
<td>162</td>
</tr>
<tr>
<td>Score Information</td>
<td>162</td>
</tr>
<tr>
<td>Tuner Information</td>
<td>162</td>
</tr>
<tr>
<td>Tuner Results</td>
<td>162</td>
</tr>
<tr>
<td>Best Configuration</td>
<td>162</td>
</tr>
<tr>
<td>Tuner Summary</td>
<td>162</td>
</tr>
<tr>
<td>TunerTiming</td>
<td>162</td>
</tr>
<tr>
<td>Cross Validation Results</td>
<td>162</td>
</tr>
<tr>
<td>ODS Table Names</td>
<td>163</td>
</tr>
<tr>
<td>Examples: NNET Procedure</td>
<td>164</td>
</tr>
<tr>
<td>Example 8.1: Binary Target Classification with Partition</td>
<td>164</td>
</tr>
</tbody>
</table>
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:

```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 5 in Chapter 2, “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 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “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:
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:

proc nnet data=mycas.iris;
   input SepalLength SepalWidth PetalLength PetalWidth;
   target Species / level=nom;
   hidden 2;
   train outmodel=mycas.nnetModel_gs seed=635117188;
   partition fraction(validate=0.3 seed=103873735);
run;

Figure 8.1 shows the model information for the neural network.

**Figure 8.1 Model Information**

<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>93</td>
</tr>
<tr>
<td>Number of Observations Read</td>
<td>93</td>
</tr>
<tr>
<td>Target/Response Variable</td>
<td>Species</td>
</tr>
<tr>
<td>Number of Nodes</td>
<td>9</td>
</tr>
<tr>
<td>Number of Input Nodes</td>
<td>4</td>
</tr>
<tr>
<td>Number of Output Nodes</td>
<td>3</td>
</tr>
<tr>
<td>Number of Hidden Nodes</td>
<td>2</td>
</tr>
<tr>
<td>Number of Hidden Layers</td>
<td>1</td>
</tr>
<tr>
<td>Number of Weight Parameters</td>
<td>14</td>
</tr>
<tr>
<td>Number of Bias Parameters</td>
<td>5</td>
</tr>
<tr>
<td>Architecture</td>
<td>MLP</td>
</tr>
<tr>
<td>Number of Neural Nets</td>
<td>1</td>
</tr>
<tr>
<td>Objective Value</td>
<td>0.169573494</td>
</tr>
<tr>
<td>Misclassification Error for Validation (%)</td>
<td>3.5087719298</td>
</tr>
</tbody>
</table>

Figure 8.2 shows the misclassification error for the training sample.

**Figure 8.2 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>93</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>93</td>
</tr>
<tr>
<td>Misclassification Error (%)</td>
<td>2.1505376344</td>
</tr>
</tbody>
</table>

Figure 8.3 shows the misclassification error for the validation sample.
Syntax: NNET Procedure

The following statements are available in the NNET procedure:

```
PROC NNET < options > ;
  INPUT variables < / LEVEL=INT | NOM > ;
  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 < KFOLD=number > ;
  SCORE 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, SCORE, and CODE statements are allowed.

PROC NNET Statement

```
PROC NNET < options > ;
```

The PROC NNET statement invokes the procedure. You can specify the following options in the PROC NNET statement:

You can specify the following options:

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

names the input data table for PROC NNET to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

*CAS-libref* refers to a collection of information that is defined in the LIBNAME statement and includes the *caslib*, which includes a path to the data, and a session identifier, which
defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about CAS-libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 141.

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

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

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

specifies a model for stand-alone scoring or coding. 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 141.

**MISSING=**MIN | MAX | MEAN

specifies the statistic with which to impute missing values for interval input variables or the target variable. If you specify this option, missing values for nominal input variables or the target variable are treated as a valid level. If you specify this option and do not want PROC NNET to impute the target variable, you need to filter out observations that have missing values for the target variable before calling PROC NNET.

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

**NTHREADS=**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 number-of-threads can be from 1 to 64, inclusive.

**STANDARDIZE=**NONE | STD | (MIDRANGE | MID)

specifies the method to use for standardizing interval inputs.

You can specify the following methods:

**NONE** specifies the method in which the variables are not altered.

**STD** specifies the method in which the variables are scaled such that their mean is 0 and the standard deviation is 1.

**MIDRANGE | MID** specifies the method in which the variables are scaled 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.

By default, STANDARDIZE=MIDRANGE.

---

**ARCHITECTURE Statement**

**ARCHITECTURE** architecture-option ;

The ARCHITECTURE statement specifies the architecture of the neural network to be trained.

You can specify one of the following architecture-options:
GLIM
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.

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, the ARCHITECTURE statement is optional. This statement is not allowed when you use PROC NNET to perform stand-alone scoring.

NOTE: If you specify the AUTOTUNE statement, PROC NNET uses the architecture suggested by the tuning optimization. As a result, it might use a different architecture from the one specified in the ARCHITECTURE statement.

AUTOTUNE Statement

AUTOTUNE < options> ;

The AUTOTUNE statement activates the tuning optimization algorithm, which searches for the best hidden layers and regularization parameters based on the problem and specified options. If ALGORITHM=SGD, the algorithm also searches for the best values of the learning rate and annealing rate. When you specify AUTOTUNE statement, PROC NNET might ignore any specified HIDDEN statements. You cannot specify both the AUTOTUNE statement and the CROSSVALIDATION statement in the same procedure run.

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). 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 validation partition is usually sufficient for estimating error; a single partition is more efficient than cross validation in terms of the total execution time.

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

By default, FRACTION=0.3.

KFOLD=number
specifies the number of partition folds 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 1 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).
You cannot specify this option in combination with the FRACTION= option. If a PARTITION statement is specified, the validation partition defined in that statement is used, and this option is ignored.

**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 on a validation partition, or a number of training and scoring executions equal to the value of KFOLD= option for cross validation. The MAXEVALS= option might lead to termination before the value of the MAXITER= or 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 usually involves a number of objective evaluations up to the value of the POPSIZE option. The MAXITER= option might lead to termination before the value of the MAXEVALS= or 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 can significantly exceed number. The MAXTIME= option might lead to termination before the value of the MAXEVALS= or MAXITER= option is reached.

By default, MAXTIME=36000.

**POPSIZE=**number

specifies the maximum number of evaluations in one iteration (population), where number must be greater than or equal to 1. In some cases, the tuner algorithm might generate a number of new configurations smaller than number.

By default, POPSIZE=10.

**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. If options LB, UB, VALUES, and INIT are specified, they will be ignored. 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 ith 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.
\textbf{UB=number}

specifies the maximum number neurons in the $i$th hidden layer, where \textit{number} must be a nonnegative integer. If you specify this suboption, you cannot also specify the VALUES= suboption.

By default, UB=$\min \left(3n, 100\right)$, where $n$ is the number of model inputs.

\textbf{VALUES=value-list}

specifies a list of candidate numbers of neurons to be searched in the tuning process, where \textit{value-list} is a space-separated list of nonnegative integers. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

\textbf{INIT=number}

specifies the initial number of neurons in hidden layer $i$.

By default, INIT=1.

\textbf{EXCLUDE}

excludes the number of neurons in hidden layer $i$ from the tuning process. If you specify this suboption, any specified LB=, UB=, VALUES=, and INIT= suboptions are ignored. If you specify this suboption, you cannot specify the NHIDDEN suboption or any other NUNITS$i$ option.

\textbf{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:

\textbf{LB=number}

specifies the minimum L1 regularization value in the tuning process, where \textit{number} is a nonnegative double. If you specify this suboption, you cannot also specify the VALUES= suboption.

By default, LB=0.

\textbf{UB=number}

specifies the maximum L1 regularization value in the tuning process, where \textit{number} is a nonnegative double. If you specify this suboption, you cannot also specify the VALUES= suboption.

By default, UB=10.

\textbf{VALUES=value-list}

specifies a list of L1 regularization values to be searched in the tuning process, where \textit{value-list} is a space-separated list of nonnegative doubles. If you specify this suboption, you cannot specify either the LB= or UB= suboption.

\textbf{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.

**USEPARAMETERS=tuning-parameter-option**
specifies which set of parameters should be tuned.

You can specify the following *tuning-parameter-options*:

**STANDARD**
tunes using the default bounds and initial values for all parameters.

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

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

By default, USEPARAMETERS=COMBINED.
**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 option **FRACTION** option in the **PARTITION** statement is included in the score code.

**CROSSVALIDATION Statement**

```sas
CROSSVALIDATION <K FOLD=number> ;
```

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

You can specify the following option:

- **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, K FOLD=5.

**HIDDEN Statement**

```sas
HIDDEN number </options> ;
```

The **HIDDEN** statement specifies the number of neurons or units in a hidden layer. 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.
RECTIFIER (Experimental) 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=INT | NOM> ;

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=INT | NOM
specifies the variables in the input data table. You can specify the following methods:

INT specifies that the variables are interval variables, which must be numeric.
NOM specifies that the variables are nominal variables, also known as classification variables, which can be numeric or character.

By default, LEVEL=INT.

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 < 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

$$w_{k+1} = w_k - \frac{\eta}{\|I_k\|} \sum_{(x_i, y_i) \in I_k} \nabla L(w_k; x_i, y_i)$$

where $w_k$ is the current weight vector, $w_{k+1}$ is the new weight vector, $I_k$ is the minibatch used during iteration $k$, and $L(w_k; x_i, y_i)$ is the loss associated with the $i$th observation.

If you see a huge objective value 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.
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.

REGL\(=\)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.

REGL\(=\)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.

PARTITION Statement

PARTITION <partition-options> ;
The **PARTITION** statement specifies how observations in the input data table are logically partitioned into disjoint subsets for model training, validation, and testing. Either you can designate a variable in the input data table and a set of formatted values of that variable to determine the role of each observation, or you can specify proportions to use for random assignment of observations for each role. Alternatively, you can use a separate validation data table in the **TRAIN** statement to do validation.

You can specify the following mutually exclusive **partition-options**:

**ROLEVAR=** `variable` *(TRAIN= **value** VALIDATE= **value** `< TEST= **value** )*)

names the variable in the input data table whose values are used to assign roles to each observation. The formatted values of this variable, which are used to assign observations roles, are specified in the **TEST=**, **TRAIN=**, and **VALIDATION=** suboptions. The **VALIDATE=** suboptions is required; the **TRAIN=** and **TEST=** suboptions are optional. If you do not specify the **TRAIN=** suboption, the training subset that **PROC NNET** uses is the complement set of the **VALIDATE=** suboption, or the complement set of the **VALIDATE=** and **TEST=** suboptions if they are both specified.

**FRACTION(VALIDATE= **fraction** TEST= **fraction** `< SEED= **random-seed** )**

randomly assigns the specified proportions of the observations in the input data table to training and validation roles. You specify the proportions for testing and validation by using the **TEST=** and **VALIDATE=** suboptions. The **VALIDATE=** suboption is required, and the **TEST=** suboption is optional. If you specify both the **TEST=** and **VALIDATE=** suboptions, then the sum of the specified fractions must be less than 1 and the remaining fraction of the observations are assigned to the training role. Otherwise, the **PARTITION** statement is ignored. The range of the **VALIDATE=** and **TEST=** suboptions is from 1E–5 to 1 – (1E–5), inclusive.

**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. **_Fraction_PartInd_** is a reserved partition variable name for the **FRACTION** option. **PROC NNET** terminates with an error if you specify the **FRACTION** option and the input data table includes a variable named **_Fraction_PartInd_**. If you specify the **FRACTION** option, the **_Fraction_PartInd_** variable is automatically included in the scored output data table.

---

**SCORE Statement**

```
SCORE OUT=CAS-libref.data-table <option> ;
```

The **SCORE** statement creates a new data table that is the result of prediction 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.data-table** is a two-level name, where
Chapter 8: The NNET Procedure

**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 141.

**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.

---

**TARGET Statement**

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

The **TARGET** statement specifies the target variable for the neural network. This statement is required.

You can specify the following options:

- **LEVEL=INT | NOM**
  
  specifies the variable type. You can specify the following values:
  
  - **INT** specifies that the variable is interval, which must be numeric.
  - **NOM** specifies that the variable is nominal, also known as a classification variable, which can be numeric or character.

  By default, LEVEL=INT.

- **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.
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.

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.

TRAIN Statement

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

specifies the final model from training. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the caslib and session identifier, and \texttt{data-table} specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 141.

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 \texttt{options}:

\texttt{DROPOUTHIDDEN=ratio (Experimental)}

specifies the dropout \texttt{ratio} of hidden layers. This option is valid only when ALGORITHM=SGD and all the connections use the linear combination function. The \texttt{ratio} must be between 0 and 1, inclusive.

By default, DROPOUTHIDDEN=0.

\texttt{DROPOUTINPUT=ratio (Experimental)}

specifies the dropout \texttt{ratio} of input layers. This option is valid only when ALGORITHM=SGD and all the connections use the linear combination function. The \texttt{ratio} must be between 0 and 1, inclusive.

By default, DROPOUTINPUT=0.

\texttt{NUMTRIES=number}

specifies the \texttt{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 \texttt{number} must be an integer between 1 and 20,000, inclusive. By default, NUMTRIES=1.

\texttt{NOTE}: When NUMTRIES > 1, the ODS tables “OptIterHistory” and “ConvergenceStatus” are suppressed.

\texttt{RESUME (Experimental)}

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.

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

specifies a separate data table for validation during training. \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 DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 141.

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.

\texttt{WSEED=random-seed}
\texttt{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**

```plaintext
WEIGHT variable ;
```

If you specify a WEIGHT statement, `variable` identifies a numeric variable in the input data table that contains the weight to be placed on the prediction error (the difference between the output of the network and the target value specified in the input data table) for each observation during training.

If the `variable` is less than or equal to 0 or is missing, the observation is not used for training. When you perform scoring, PROC NNET scores the observation even if the weight is less than or equal to 0 or missing. The WEIGHT statement is optional. If a WEIGHT statement is not included, all observations are assigned a weight of 1.

**Details: NNET Procedure**

**Computational Method**

PROC NNET trains a multilayer perceptron neural network that contains one or more hidden layers. For more information about multilayer perceptron neural networks, see Bishop (1995).

The NNET procedure does not have many parameters that you must specify for training. You must specify where the training data are (in the DATA= option in the PROC NNET statement), the names and types of the input variables (in the INPUT statement), the names and types of the target variables (in the TARGET statement), the number of hidden layers (in the HIDDEN statement), and the number of neurons in each hidden layer (in the HIDDEN statement).

Optionally, you can also specify where to write the score file that contains targets from the input file and predicted target variables from the trained network and where to write the model file that contains the parameters of the trained network (in the SCORE 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.

Parameter Tuning

The quality of the predictive model that PROC NNET creates depends on the values specified in 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. The AUTOTUNE statement engages the optimization algorithm (tuner), which searches for the best possible combination of values of these select hyperparameters while trying to minimize the objective function. The objective function is a validation error estimate (misclassification error for nominal targets or average square error for interval targets). The tuning process includes multiple iterations; each iteration usually involves multiple objective function evaluations. Each objective function evaluation can consist of one or several training and scoring executions as follows:

- If you specify the PARTITION statement, the tuner uses a single-partition validation set as defined in that statement. 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.

- If you specify the FRACTION= option, the tuner uses a single-partition validation set. 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.

- If KFOLD=\(k\) is specified, the tuner uses \(k\)-fold cross validation. In this process, the tuner partitions all the data into \(k\)-fold subsets (folds). For each fold, a new model is trained on each of the \((k-1)\) folds and then validated using the selected (holdout) fold. The objective function value is averaged over each set of training and scoring executions to obtain a single error estimate value.

The optimization tuner algorithm is based on a genetic algorithm (GA), which applies the principles of natural selection and evolution to find an improved configuration. The tuner performs the following sequence of actions:

1. A default model configuration (default values of select model tuning parameters) 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. The initial set of configurations, also called a “population,” is generated using a technique called random Latin hypercube sampling (LHS). In a Latin hypercube sample, each configuration of hyperparameters is evaluated, and their objective function values are again recorded for comparison. This becomes Iteration 1.
3. The best model configurations from the initial population are used to generate the next population of model configurations, Iteration 2, which are then evaluated. This process is repeated for the remaining iterations, as long as the maximum number of evaluations or the maximum time is not reached.

4. The best model configuration is reevaluated by executing a single training and model 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 ODS table, as described in the section “ODS Table Names” on page 163.

You can tune the following hyperparameter values when you specify the AUTOTUNE statement:

- the number of hidden layers
- the number of hidden units in each hidden layer
- the REGL1= option for the L1 regularization parameter
- the REGL2= option for the L2 regularization parameter
- the ANNEALINGRATE= option for the annealing rate parameter used by the training algorithm of the SGD optimizer when you specify ALGORITHM=SGD in the OPTIMIZATION statement
- the LEARNINGRATE= option for the learning rate parameter used by the training algorithm of the SGD optimizer when you specify ALGORITHM=SGD in the OPTIMIZATION statement

### k-fold Cross Validation

The CROSSVALIDATION statement uses 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 partitions (folds), where k is the value of the KFOLD= option. For each of the folds, a new model is trained on the (KFOLD−1) folds and then validated using the selected (holdout) 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**

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 error 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.
Iterate History

The “Iteration History” table contains the iteration history from 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 error or mean square error for training, validation, or testing partition sets.

Tuner Information

The “Tuner Information” table contains the values of options used by tuner.

Tuner Results

The “Tuner Results” table contains the values of hyperparameters, objective function for the default configuration (Iteration 0), and up to 10 best configurations found from tuner.

Best Configuration

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

Tuner Summary

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

Tuner Timing

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

Cross Validation Results

The “Cross Validation Results” table contains the average error rate (misclassification error or average square error) of $k$-fold cross validation.
**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 8.1.

### Table 8.1 ODS Tables Produced by PROC NNET

<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>CrossValidationResults</td>
<td>Average error rate (misclassification error or average square error) of $k$-fold cross validation</td>
<td>CROSSVALIDATION</td>
<td>Default</td>
</tr>
<tr>
<td>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 error or mean square error for partition sets</td>
<td>SCORE, PARTITION</td>
<td>VALIDATION=</td>
</tr>
<tr>
<td>TunerInfo</td>
<td>Setup values used by the tuner</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerResults</td>
<td>Values of the hyperparameters, the objective function for the default configuration (Iteration 0), and up to 10 best configurations found</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerSummary</td>
<td>Statistics about the tuning process</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
<tr>
<td>TunerTiming</td>
<td>Total time spent on different tasks during tuning</td>
<td>AUTOTUNE</td>
<td>Default</td>
</tr>
</tbody>
</table>
Examples: NNET Procedure

**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 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “Shared Concepts.”

Example 8.1: Binary Target Classification with Partition

This example demonstrates how to use PROC NNET to predict whether a mortgage applicant will default on a loan. The data table **Hmeq**, which is in the **Sampsio** library that SAS provides, contains observations for 5,960 mortgage applicants. A variable named **Bad** indicates whether the applicant, after being approved for a loan, paid off or defaulted on the loan.

The following DATA steps load the **Hmeq** data set into a CAS session by naming a CAS engine libref in the first statement of each step:

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

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

The following statements run PROC NNET and output the results to ODS tables. Based on the specified percentages, the PARTITION statement randomly splits the **hmeq** data set into three partitions: training, validation, and testing. The error rates of the validation data set are computed during the training process as one of the stopping criteria in order to prevent overfitting.

```sas
  proc nnet data=mycas.hmeq standardize=midrange missing=mean;
    architecture mlp;
    input job reason / level=nominal;
    input debtinc delinq loan mortdue value yoj derog clage clno;
    hidden 7;
    target bad / level=nominal;
    optimization algorithm=lbfgs maxiter=500;
    train outmodel=mycas.nnetmodel1 seed=12345;
    partition fraction(validate=0.2 test=0.1 seed=54321);
  run;
```

The **PROC NNET** call creates the model, **nnetModel1**, from the training data; this model contains all the weight of the neural network. The example is run on one controller node.

**Output 8.1.1** shows the model information for the neural network.
Example 8.1: Binary Target Classification with Partition

Output 8.1.1 Model Information

The NNET Procedure

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
</tr>
<tr>
<td>Number of Observations Used</td>
</tr>
<tr>
<td>Number of Observations Read</td>
</tr>
<tr>
<td>Target/Response Variable</td>
</tr>
<tr>
<td>Number of Nodes</td>
</tr>
<tr>
<td>Number of Input Nodes</td>
</tr>
<tr>
<td>Number of Output Nodes</td>
</tr>
<tr>
<td>Number of Hidden Nodes</td>
</tr>
<tr>
<td>Number of Hidden Layers</td>
</tr>
<tr>
<td>Number of Weight Parameters</td>
</tr>
<tr>
<td>Number of Bias Parameters</td>
</tr>
<tr>
<td>Architecture</td>
</tr>
<tr>
<td>Number of Neural Nets</td>
</tr>
<tr>
<td>Objective Value</td>
</tr>
<tr>
<td>Misclassification Error (%)</td>
</tr>
</tbody>
</table>

Output 8.1.2 shows the misclassification error of the training data.

Output 8.1.2 Score Information for Training Data

<table>
<thead>
<tr>
<th>Score Information for Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
</tr>
<tr>
<td>Number of Observations Used</td>
</tr>
<tr>
<td>Misclassification Error (%)</td>
</tr>
</tbody>
</table>

Output 8.1.3 shows the misclassification error of the validation data.

Output 8.1.3 Score Information for Validation Data

<table>
<thead>
<tr>
<th>Score Information for Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
</tr>
<tr>
<td>Number of Observations Used</td>
</tr>
<tr>
<td>Misclassification Error (%)</td>
</tr>
</tbody>
</table>

Output 8.1.4 shows the misclassification error of the testing data.

Output 8.1.4 Score Information for Testing Data

<table>
<thead>
<tr>
<th>Score Information for Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
</tr>
<tr>
<td>Number of Observations Used</td>
</tr>
<tr>
<td>Misclassification Error (%)</td>
</tr>
</tbody>
</table>
Example 8.2: 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 141. 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.

```sas
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.

```sas
proc nnet data=mycas.iris;
  input SepalLength SepalWidth PetalLength PetalWidth;
  target Species / level=nom;
  train outmodel=mycas.nnetModel2 seed=1517878693;
  autotune useparameters=custom
    tuningparameters=(nhidden(LB=0 UB=2 INIT=0)
      nunits1(LB=1 UB=10 INIT=1)
      nunits2(LB=2 UB=15 INIT=2)
      regl1(LB=1e-03 UB=1e-02 INIT=1e-03)
      regl2(LB=1e-03 UB=1e-02 INIT=1e-03)
    );
  optimization algorithm=LBFGS maxiter=100;
run;
```

Output 8.2.1 shows the setup values used by the tuner.

<table>
<thead>
<tr>
<th>Tuner Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Type                      Neural Net</td>
</tr>
<tr>
<td>Tuner Objective Function          Misclassification Error Percentage</td>
</tr>
<tr>
<td>Maximum Evaluations              50</td>
</tr>
<tr>
<td>Population Size                  10</td>
</tr>
<tr>
<td>Maximum Iterations               5</td>
</tr>
<tr>
<td>Maximum Tuning Time in Seconds    36000</td>
</tr>
<tr>
<td>Validation Type                  Single Partition</td>
</tr>
<tr>
<td>Validation Partition Fraction     0.3</td>
</tr>
<tr>
<td>Log Level                        2</td>
</tr>
<tr>
<td>Seed                             1517878693</td>
</tr>
</tbody>
</table>
Output 8.2.2 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 8.2.2 Tuner Results

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Hidden Layers</th>
<th>Neurons in Hidden Layer 1</th>
<th>Neurons in Hidden Layer 2</th>
<th>L1 Regularization</th>
<th>L2 Regularization</th>
<th>Misclassification Error Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.001000</td>
<td>0.001000</td>
<td>4.76</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.006863</td>
<td>0.006020</td>
<td>0.00</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.007992</td>
<td>0.008279</td>
<td>0.00</td>
</tr>
<tr>
<td>31</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.007936</td>
<td>0.007580</td>
<td>0.00</td>
</tr>
<tr>
<td>36</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0.010000</td>
<td>0.010000</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>0</td>
<td>0.007483</td>
<td>0.002430</td>
<td>1.59</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>0.002243</td>
<td>0.003835</td>
<td>1.59</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0.004824</td>
<td>0.006129</td>
<td>1.59</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.006185</td>
<td>0.006866</td>
<td>1.59</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>10</td>
<td>15</td>
<td>0.010000</td>
<td>0.005195</td>
<td>1.59</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>0.002035</td>
<td>0.003050</td>
<td>1.59</td>
</tr>
</tbody>
</table>

Output 8.2.3 shows the best values of the tuning parameters from the tuning process.

### Output 8.2.3 Best Configuration

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Hidden Layers</th>
<th>Neurons in Hidden Layer 1</th>
<th>Neurons in Hidden Layer 2</th>
<th>L1 Regularization</th>
<th>L2 Regularization</th>
<th>Misclassification Error Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0.00686269</td>
<td>0.006020</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Output 8.2.4 shows the tuner summary.

### Output 8.2.4 Tuner Summary

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Configuration Objective Value</td>
<td>4.7619</td>
</tr>
<tr>
<td>Best Configuration Objective Value</td>
<td>0</td>
</tr>
<tr>
<td>Worst Configuration Objective Value</td>
<td>33.3333</td>
</tr>
<tr>
<td>Initial Configuration Evaluation Time in Seconds</td>
<td>1.2884</td>
</tr>
<tr>
<td>Best Configuration Evaluation Time in Seconds</td>
<td>0.2535</td>
</tr>
<tr>
<td>Number of Improved Configurations</td>
<td>2</td>
</tr>
<tr>
<td>Number of Evaluated Configurations</td>
<td>46</td>
</tr>
<tr>
<td>Total Tuning Time in Seconds</td>
<td>32.1485</td>
</tr>
</tbody>
</table>
Output 8.2.5 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.

### Output 8.2.5 Tuner Timing

<table>
<thead>
<tr>
<th>Task</th>
<th>Tuner Task Timing</th>
<th>Seconds</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Training</td>
<td>Tuner Task Timing</td>
<td>30.97</td>
<td>96.34</td>
</tr>
<tr>
<td>Model Scoring</td>
<td>Tuner Task Timing</td>
<td>1.17</td>
<td>3.63</td>
</tr>
<tr>
<td>Total Objective Evaluations</td>
<td>Tuner Task Timing</td>
<td>32.14</td>
<td>99.98</td>
</tr>
<tr>
<td>Tuner</td>
<td>Tuner Task Timing</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>Total Time</td>
<td>Tuner Task Timing</td>
<td>32.15</td>
<td>100.00</td>
</tr>
</tbody>
</table>

### References


Chapter 9
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 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:

```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 5 in Chapter 2, “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 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “Shared Concepts.”

This example trains the model by using German credit benchmark data, which are available in the samp-sio.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-sio.dmagecr data set into your CAS session by specifying your CAS engine libref in the second statement in the following DATA step:

```sas
data mycas.dmagecr;
  set sampsio.dmagecr;
run;
```

These statements assume that your CAS engine libref is named mycas, as in the section “Using CAS Sessions and CAS Engine Librefs” on page 170, 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 9.1 through Figure 9.3.

```sas
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, Telephon, 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 9.1 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.
The “Misclassification Matrix” table in Figure 9.2 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 9.3.

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 > ;
  CODE FILE=filename ;
  ID variables ;
  INPUT variables / < LEVEL=INTERVAL | NOMINAL > ;
  KERNEL kernel-type / < kernel-parameter > ;
  OUTPUT OUT=CAS-libref.data-table < 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 170.

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

- **MAXITER=number**
  - specifies the maximum number of iterations before the process stops, where number is a positive integer.
  - By default, MAXITER=25.
**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=\textit{number-of-threads}**
specifies the number of threads that are used in the computation. The default value is the number of CPUs available in the machine.

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

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

**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.

**TOLERANCE=\textit{number}**
specifies the minimal absolute tolerance at which the iteration stops. The tolerance \textit{number} must be equal to or greater than $1.0 \times 10^{-12}$.

By default, TOLERANCE=$1.0 \times 10^{-6}$.

---

**CODE Statement**

**CODE FILE=\textit{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=\textit{filename}**

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

The CODE statement is optional.

---

**ID Statement**

**ID \textit{variables} ;**

The ID statement lists one or more variables that are to be copied from the input data table to the output data tables that are specified in the OUT= option in the OUTPUT statement and the RSTORE= option in the SAVESTATE statement.
**INPUT Statement**

**INPUT** variables / < LEVEL=INTERVAL | NOMINAL > ;

The INPUT statement specifies the names of variables to be used in training. Only interval, binary, and nominal variables are accepted. If you want to use different options for different variables, you can specify multiple INPUT statements.

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

**LEVEL=INTERVAL | NOMINAL**

specifies whether the specified input variables are continuous or categorical. You can specify the following values:

**INTERVAL**  
specifies that the input variables are continuous.

**NOMINAL**  
specifies that the input variables are categorical.

By default, LEVEL=INTERVAL for numeric variables and LEVEL=NOMINAL for categorical variables. Binary variables are considered to be categorical variables.

**KERNEL Statement**

**KERNEL**  
kernel-type / < kernel-parameter > ;

The KERNEL statement specifies the type of kernel and any associated parameters to be used during training.

You can specify the following kernel-types:

**LINEAR**

uses a linear kernel during training. No kernel-parameter is needed. The kernel is defined as

\[ k(x_1, x_2) = \langle x_1, x_2 \rangle \]

where \( x_1 \) and \( x_2 \) are two vectors and \( \langle, \rangle \) is the inner product.

**POLYNOMIAL / 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.
Chapter 9: The SVMACHINE Procedure

OUTPUT Statement

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

The OUTPUT statement creates an output data table that contains the predicted values of the training 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 170.

**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.

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 3, “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 170.

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

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

where $X$ denotes the $m \times n$ matrix whose rows correspond to the observations, $D$ denotes a diagonal matrix whose diagonals are 1 or –1, $z$ denotes the slack variables that relax the classification constraints, and $C$ denotes the penalty term. The corresponding dual optimization problem is

$$ \begin{align*}
\text{minimize} & \quad \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\
\text{subject to} & \quad d^T \alpha = 0 \\
& \quad 0 \leq \alpha \leq C
\end{align*} $$

where $Q = DXX^T D$ and $\alpha$ is the Lagrange multiplier. For a more general discussion about duality in the context of quadratic programming, see the chapter "The OPTQP Procedure" in *SAS/OR User’s Guide: Mathematical Programming*. At the solution, $w$ is related to $\alpha$ by the equation $w = X^T D \alpha$. Observations that correspond to nonzero entries in $\alpha$ are called support vectors. Observations that correspond to entries in $\alpha$ that are active at their upper bound are called support vectors on the margin.

In the nonlinear case, the dual optimization problem satisfies $Q_{ij} = d_i d_j k(x_i, x_j)$, where $k(x, y)$ denotes the selected kernel function, which is defined in the section "KERNEL Statement" on page 175. 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.
**Interior-Point Method Optimization Technique**

A popular approach for accurately obtaining the global solution for an SVM optimization problem is the interior-point method. Interior-point methods are attractive in that the required number of iterations is relatively small and does not grow dramatically with problem size. The cost per iteration for interior-point methods can be quite high for large-scale problems unless there exists an underlying structure that can be exploited. For example, interior-point methods can be extremely efficient for problems where the number of variables are small in comparison to the number of constraints. When both the number of variables and number of constraints is large, interior-point methods become intractable for dense problems.

PROC SVMACHINE applies a primal-dual interior-point method to linear and polynomial kernels of degree 2 and 3. In the polynomial kernel case, \( X \) is obtained by explicitly projecting each observation in the design matrix. The resulting number of columns is given by the binomial coefficient

\[
\binom{n + p}{p} = \frac{(n + p)!}{p!n!}
\]

where \( n \) denotes the number of columns in the levelized design matrix and \( p \) denotes the polynomial degree. Thus, KERNEL POLYNOMIAL/DEGREE=2 is not recommended when the number of columns in the design matrix is much greater than 100. Similarly, KERNEL POLYNOMIAL/DEGREE=3 is not recommended when the number of columns in the design matrix is much greater than 32.

Primal-dual interior-point methods perturb the optimality conditions in order to create a system of nonlinear equations that satisfy the requirements of Newton’s method. This perturbed system of nonlinear equations has the property that an interior solution (with respect to the inequality constraints) always exists; safeguards are then wrapped around Newton’s method to ensure that the interior (and hence feasible) approximate solution is obtained. Interior-point methods have the additional property that as the perturbation term goes to 0, the approximate solution converges to the true solution.

The dominant cost for each iteration comes from the need to solve a system of linear equations of size \((m + n) \times (m + n)\). Because PROC SVMACHINE assumes that \( m \) might be very large, it uses block reduction strategies similar to those described in Gertz and Griffin (2005, 2010) to reduce the size of this system to a matrix of size \( n \times n \), where \( n \) denotes the number of columns in the design (or projected kernel) matrix. Then it performs dense matrix factorization on the resulting system. For problems where \( m \gg n \), the dominant computational cost occurs during the block-row reduction step. To reduce the solution time, this operation is both distributed and threaded.

**Scoring Process**

The scoring process for the interior-point method is straightforward. As long as the training weight parameters and bias are known, the scoring process is just a linear combination. The event or nonevent is decided by the decision function. If the decision function is less than or equal to 0, then the prediction is an event; otherwise, it is a nonevent.

For the interior-point method, the score code is provided for a linear kernel and for a polynomial kernel of degree 2 and 3.
Displayed Output

The following sections describe the output that PROC SVMACHINE produces by default. The output is organized into various tables, which are discussed in the order of their appearance.

Model Information

The “Model Information” table contains the initial training settings, such as task type, optimization technique, and kernel function type. If the kernel function type is polynomial, then the kernel degree is also displayed.

Number of Observations

The “Number of Observations” table contains the number of observations and the number of observations used.

Training Results

The “Training Results” table 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.

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 9.1.
### Table 9.1: ODS Tables Produced by PROC SVMACHINE

<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</td>
<td>PROC SVMACHINE</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td>training</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IterHistory</td>
<td>Iteration history</td>
<td>PROC SVMACHINE</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Misclassification</td>
<td>Misclassification matrix table</td>
<td>PROC SVMACHINE</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ModellInfo</td>
<td>Basic model information for the</td>
<td>PROC SVMACHINE</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td>training</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NObs</td>
<td>Observation information about</td>
<td>PROC SVMACHINE</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td>the input data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TrainingResult</td>
<td>Displays the training results</td>
<td>PROC SVMACHINE</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></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 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “Shared Concepts.”

### Example 9.1: 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 9.2 describes the variables in Hmeq.
Example 9.1: Home Equity Loan Case

Table 9.2  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>nInq</td>
<td>Predictor</td>
<td>Interval</td>
<td>Number of recent credit inquiries</td>
</tr>
<tr>
<td>Reason</td>
<td>Predictor</td>
<td>Binary</td>
<td>'DebtCon' = debt consolidation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'HomeImp' = home improvement</td>
</tr>
<tr>
<td>Value</td>
<td>Predictor</td>
<td>Interval</td>
<td>Value of current property</td>
</tr>
<tr>
<td>YoJ</td>
<td>Predictor</td>
<td>Interval</td>
<td>Years at present job</td>
</tr>
</tbody>
</table>

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

```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 9.1.1 through Output 9.1.5.

The “Model Information” table in Output 9.1.1 shows that the kernel function is linear, and the penalty parameter value is 1 (both of which are default values).
Output 9.1.1  Model Information

The SVMACHINE Procedure

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Type</td>
</tr>
<tr>
<td>Optimization Technique</td>
</tr>
<tr>
<td>Scale</td>
</tr>
<tr>
<td>Kernel Function</td>
</tr>
<tr>
<td>Penalty Method</td>
</tr>
<tr>
<td>Penalty Parameter</td>
</tr>
<tr>
<td>Maximum Iterations</td>
</tr>
<tr>
<td>Tolerance</td>
</tr>
</tbody>
</table>

The observations table in Output 9.1.2 shows that the total number of observations is 5,960 and the number of observations used in the training is 3,364.

Output 9.1.2  Number of Observations

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

The “Training Results” table in Output 9.1.3 shows the inner product of weights, bias, total slack, and so on.

Output 9.1.3  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 9.1.4 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 9.1.4  Misclassification Matrix

<table>
<thead>
<tr>
<th>Misclassification Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed</td>
</tr>
<tr>
<td>Observed</td>
</tr>
<tr>
<td>Observed</td>
</tr>
<tr>
<td>Total</td>
</tr>
</tbody>
</table>
The “Fit Statistics” table in Output 9.1.5 shows information about the accuracy, error, sensitivity, and specificity.

Output 9.1.5  Fit Statistics

<table>
<thead>
<tr>
<th>Statistic</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 9.2: 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 9.2.1 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 9.2.1 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 9.2.2 shows the accuracy (92.85%) and the error (7.15%) of the model.

Output 9.2.2 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 10
## The TEXTMINE Procedure

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: TEXTMINE Procedure</td>
<td>186</td>
</tr>
<tr>
<td>PROC TEXTMINE Features</td>
<td>186</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>187</td>
</tr>
<tr>
<td>Getting Started: TEXTMINE Procedure</td>
<td>187</td>
</tr>
<tr>
<td>Syntax: TEXTMINE Procedure</td>
<td>192</td>
</tr>
<tr>
<td>PROC TEXTMINE Statement</td>
<td>192</td>
</tr>
<tr>
<td>DOC_ID Statement</td>
<td>193</td>
</tr>
<tr>
<td>PARSE Statement</td>
<td>193</td>
</tr>
<tr>
<td>SAVESTATE Statement</td>
<td>198</td>
</tr>
<tr>
<td>SELECT Statement</td>
<td>198</td>
</tr>
<tr>
<td>SVD Statement</td>
<td>199</td>
</tr>
<tr>
<td>TARGET Statement</td>
<td>203</td>
</tr>
<tr>
<td>VARIABLES Statement</td>
<td>203</td>
</tr>
<tr>
<td>Details: TEXTMINE Procedure</td>
<td>204</td>
</tr>
<tr>
<td>Natural Language Processing</td>
<td>204</td>
</tr>
<tr>
<td>Stemming</td>
<td>204</td>
</tr>
<tr>
<td>Part-of-Speech Tagging</td>
<td>204</td>
</tr>
<tr>
<td>Noun Group Extraction</td>
<td>205</td>
</tr>
<tr>
<td>Entity Identification</td>
<td>205</td>
</tr>
<tr>
<td>Multiword Terms Handling</td>
<td>207</td>
</tr>
<tr>
<td>Language Support</td>
<td>207</td>
</tr>
<tr>
<td>Term and Cell Weighting</td>
<td>207</td>
</tr>
<tr>
<td>Sparse Format</td>
<td>208</td>
</tr>
<tr>
<td>Coordinate List (COO) Format</td>
<td>208</td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td>208</td>
</tr>
<tr>
<td>Applications in Text Mining</td>
<td>208</td>
</tr>
<tr>
<td>Computation</td>
<td>209</td>
</tr>
<tr>
<td>SVD-Only Mode</td>
<td>209</td>
</tr>
<tr>
<td>Topic Discovery</td>
<td>209</td>
</tr>
<tr>
<td>Output Data Tables</td>
<td>210</td>
</tr>
<tr>
<td>The OUTCHILD= Data Table</td>
<td>210</td>
</tr>
<tr>
<td>The OUTCONFIG= Data Table</td>
<td>210</td>
</tr>
<tr>
<td>The OUTDOCPRO= Data Table</td>
<td>211</td>
</tr>
<tr>
<td>The OUTPARENT= Data Table</td>
<td>211</td>
</tr>
<tr>
<td>The OUTPOS= Data Table</td>
<td>212</td>
</tr>
<tr>
<td>The OUTTERMS= Data Table</td>
<td>212</td>
</tr>
</tbody>
</table>
Overview: TEXTMINE Procedure

The TEXTMINE procedure integrates natural language processing and statistical analysis to analyze large-scale textual data in SAS Viya. PROC TEXTMINE supports a wide range of fundamental text analysis features, which include tokenizing, stemming, part-of-speech tagging, noun group extraction, default or customized stop lists and start lists, entity parsing, multiword tokens, synonym lists, term weighting, term-by-document matrix creation, dimension reduction with singular value decomposition (SVD), and topic discovery.

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, 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 5 in Chapter 2, “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 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “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.
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:

    data mycas.engstop;
      set sashelp.engstop;
    run;

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 only the terms that appear at least twice in the document collection 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.

    proc textmine data=mycas.getstart;
      doc_id did;
      variables text;
      parse
        outterms  = mycas.terms
        reducef  = 2
        stop     = mycas.engstop;
      svd
        k        = 3
        outdocpro = mycas.docpro
        outtopics = mycas.topics
        numLabels = 4;
    run;

The output from this analysis is presented in Figure 10.2, Figure 10.3 and Figure 10.4.
**Figure 10.1** 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. The log shows that the mycas/terms data table contains 24 observations. This means that the TEXTMINE procedure identified 24 individual terms in the input document collection. Because $K=3$ in the SVD statement, 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 small collection and the use of the default cutoff threshold causes the values in the table to all be either 0 or 1.

**Figure 10.1** 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 TERMWG'T option is specified. TERMWG'T=ENTROPY will be run by default.
NOTE: No CELLWG'T option is specified. CELLWG'T=LOG will be run by default.
NOTE: No ENTITIES option is specified. ENTITIES=NONE will be run by default.
NOTE: The dense SVD solver was used for this calculation.
NOTE: The Cloud Analytic Services server processed the request in 2.37275 seconds.
NOTE: The data set MYCAS_TERMS has 24 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.

---

The following statements use PROC PRINT in Base SAS to show the contents of the first 10 rows of the mycas/docpro data table that is generated by the TEXTMINE procedure:

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

**Figure 10.2** shows the output of PROC PRINT. For information about the output of the OUTDOCPRO= option, see the section “The OUTDOCPRO= Data Table” on page 211.

**Figure 10.2** 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>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The following statements use a DATA step and PROC PRINT to show the contents of the mycas.topics data table that is generated by the TEXTMINE procedure:

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

Figure 10.3 shows the output of PROC PRINT. The three discovered topics are listed with four descriptive terms to characterize each topic.

**Figure 10.3 The mycas.topics Data Table**

<table>
<thead>
<tr>
<th>Obs</th>
<th>_topicid</th>
<th>_termCutOff</th>
<th>_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.354</td>
<td>easy deployment, deployment, +ensure, easy</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.357</td>
<td>maintenance, +reduce, +cost, revenue</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.359</td>
<td>sas, data, viya, business</td>
</tr>
</tbody>
</table>

The following statements use a DATA step and the SORT and PRINT procedures to show the contents of the mycas.terms data table that is generated by the TEXTMINE procedure:

```
data terms; set mycas.terms; run;
proc sort data = terms; by key; run;
proc print data = terms;
var term role freq numdocs key parent;
run;
```

Figure 10.4 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 example, the output shows that the key of “ensures” is 19 and its parent’s key is 4, which is the term “ensure.” The TEXTMINE procedure also identified that “sas” is a proper noun and that “easy deployment” is a noun group. For information about the output of the OUTTERMS= option, see the section “The OUTTERMS= Data Table” on page 212.
### Figure 10.4  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>cost</td>
<td>Noun</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>cost</td>
<td>Noun</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>fast</td>
<td>Adj</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>data</td>
<td>Noun</td>
<td>7</td>
<td>6</td>
<td>3</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>ensure</td>
<td>Verb</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>.</td>
</tr>
<tr>
<td>6</td>
<td>cloud</td>
<td>Noun</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>.</td>
</tr>
<tr>
<td>7</td>
<td>vlya</td>
<td>Prop</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>.</td>
</tr>
<tr>
<td>8</td>
<td>revenue</td>
<td>Noun</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>business</td>
<td>Noun</td>
<td>3</td>
<td>3</td>
<td>8</td>
<td>.</td>
</tr>
<tr>
<td>10</td>
<td>easy</td>
<td>Adj</td>
<td>2</td>
<td>2</td>
<td>9</td>
<td>.</td>
</tr>
<tr>
<td>11</td>
<td>maintenance</td>
<td>Noun</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>.</td>
</tr>
<tr>
<td>12</td>
<td>easy deployment</td>
<td>NOUN_GROUP</td>
<td>2</td>
<td>2</td>
<td>11</td>
<td>.</td>
</tr>
<tr>
<td>13</td>
<td>deployment</td>
<td>Noun</td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>.</td>
</tr>
<tr>
<td>14</td>
<td>help</td>
<td>Verb</td>
<td>2</td>
<td>2</td>
<td>13</td>
<td>.</td>
</tr>
<tr>
<td>15</td>
<td>analytics</td>
<td>Noun</td>
<td>2</td>
<td>2</td>
<td>14</td>
<td>.</td>
</tr>
<tr>
<td>16</td>
<td>sas</td>
<td>Prop</td>
<td>4</td>
<td>4</td>
<td>15</td>
<td>.</td>
</tr>
<tr>
<td>17</td>
<td>reduce</td>
<td>Verb</td>
<td>2</td>
<td>2</td>
<td>16</td>
<td>.</td>
</tr>
<tr>
<td>18</td>
<td>reduce</td>
<td>Verb</td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>19</td>
<td>reduces</td>
<td>Verb</td>
<td>1</td>
<td>1</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>20</td>
<td>faster</td>
<td>Adj</td>
<td>2</td>
<td>2</td>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>ensures</td>
<td>Verb</td>
<td>2</td>
<td>2</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>22</td>
<td>helps</td>
<td>Verb</td>
<td>1</td>
<td>1</td>
<td>20</td>
<td>13</td>
</tr>
<tr>
<td>23</td>
<td>costs</td>
<td>Noun</td>
<td>1</td>
<td>1</td>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>helping</td>
<td>Verb</td>
<td>1</td>
<td>1</td>
<td>22</td>
<td>13</td>
</tr>
</tbody>
</table>
Syntax: TEXTMINE Procedure

The following statements are available in the TEXTMINE procedure:

```plaintext
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

```plaintext
PROC TEXTMINE DATA=CAS-libref.data-table <options>;
```

The PROC TEXTMINE statement invokes the procedure. Table 10.1 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>Basic Options</td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>DOC=</td>
</tr>
<tr>
<td>LANGUAGE=</td>
<td>Specifies the language that the input data table of documents uses</td>
</tr>
<tr>
<td>NEWVARNAMES</td>
<td>Specifies that the new-style variable names should be used on tables</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multithreading Options</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NTHREADS=</td>
<td>Specifies number of threads</td>
</tr>
</tbody>
</table>

You must specify the following option:

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

names the input data table for PROC TEXTMINE to use. The default is the most recently created data table. `CAS-libref.data-table` is a two-level name, where
CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about CAS-libref, see the section “Using CAS Sessions and CAS Engine Librefs” on page 187.

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

Each row of the input data table must contain one text variable and one ID variable that correspond to the text and the unique ID of a document, respectively.

When you specify the SVD statement but not the PARSE statement, PROC TEXTMINE runs in SVD-only mode. In this mode, the DATA= option names the input SAS data table that contains the term-by-document matrix that is generated by the OUTPARENT= option in the PARSE statement.

You can also specify the following options:

LANGUAGE=language
names the language that is used by the documents in the input SAS data table. Languages supported in the current release are Chinese, Dutch, English, Finnish, French, German, Italian, Japanese, Korean, Portuguese, Russian, Spanish, and Turkish. By default, LANGUAGE=ENGLISH.

NEWVARNAMES
adds leading and trailing blanks to variable names in the input and output tables.

NTHREADS=nthreads
specifies the number of threads to be used. By default, the number of threads is the same as the number of CPUs on the CAS server.

DOC_ID Statement

DOC_ID variable ;

The DOC_ID statement specifies the variable that contains the ID of each document. In the input data table, each row corresponds to one document. The ID of each document must be unique; it can be either a number or a string of characters.

PARSE Statement

PARSE <parse-options> ;

The PARSE statement specifies the options for parsing the input documents and creating the term-by-document matrix. Table 10.2 summarizes the parse-options in the statement by function. The parse-options are then described fully in alphabetical order.
### Table 10.2 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 207.
- **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 187. 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 multiword’s case variations to the list before using the TEXTMINE procedure to create consistent multiword terms. The multi-term 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 multi-term lists; it represents compound words.

NONOUNGROUPS
NONG
suppresses standard noun group extraction. By default, the TEXTMINE procedure extracts noun groups, returns noun phrases without determiners or prepositions, and (unless the NOSTEMMING option is specified) stems noun group elements.

NOSTEMMING
suppresses stemming of words. By default, words are stemmed; that is, terms such as “advises” and “advising” are mapped to the parent term “advise.” The TEXTMINE procedure uses dictionary-based stemming (also known as lemmatization).

NOTAGGING
suppresses tagging of terms. By default, terms are tagged and the TEXTMINE procedure identifies a term’s part of speech based on context clues. The identified part of speech is provided in the Role variable of the OUTTERMS= data table.
OUTCHILD=CAS-libref.data-table
specifies the output data table to contain a compressed representation of the sparse term-by-document matrix. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 187. 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 210.

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 187. 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 210.

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 187. 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 211.

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 187. For more information about this data table, see the section “The OUTPOS= Data Table” on page 212.

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 187. For more information about this data table, see the section “Output Data Tables” on page 210.
**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 187. 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 187. 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 187. 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 207.

SAVESTATE Statement

SAVESTATE RSTORE=CAS-libref.data-model;

The SAVESTATE statement creates a text mining model and saves it as a binary object in a data table. You can apply the model by using the ASTORE procedure.

You must specify the following option:

RSTORE=CAS-libref.data-model specifies a data table in which to save the text mining model. CAS-libref.data-table is a two-level name, where CAS-libref refers to the caslib and session identifier, and data-table specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 187.

SELECT Statement

SELECT label-list / GROUP=group-option> KEEP | IGNORE ;

The SELECT statement enables you to specify the parts of speech or entities or attributes that you want to include in or exclude from your analysis. Exclusion by the SELECT statement is different from exclusion that is indicated by the _keep variable in the OUTTERMS= data table. Terms that are excluded by the SELECT statement cannot be included in the OUTTERMS= data table, whereas terms that have _keep=N can be included in the OUTTERMS= data table if the SHOWDROPPEDTERMS option is specified. Terms excluded by the SELECT statement are excluded from the OUTPOS= data table, but terms that have _keep=N are included in OUTPOS= data table. Table 10.3 summarizes the options you can specify in the SELECT statement. The options are then described fully in syntactic order.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label-list</td>
<td>Specifies one or more labels of terms that are to be ignored or kept in your analysis</td>
</tr>
<tr>
<td>GROUP=</td>
<td>Specifies whether the labels are parts of speech, entities, or attributes</td>
</tr>
<tr>
<td>IGNORE</td>
<td>Ignores terms whose labels are specified in the label-list</td>
</tr>
<tr>
<td>KEEP</td>
<td>Keeps terms whose labels are specified in the label-list</td>
</tr>
</tbody>
</table>

You must specify a label-list and either the IGNORE or KEEP option:
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 10.5 shows all possible part-of-speech tags. Table 10.6 shows all valid English entities. The attribute variable in Table 10.12 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 10.4 summarizes the svd-options in the statement by function. The svd-options are then described fully in alphabetical order.

<table>
<thead>
<tr>
<th>Table 10.4</th>
<th>SVD Statement Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>svd-option</strong></td>
<td><strong>Description</strong></td>
</tr>
<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 10.4  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></td>
<td></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></td>
<td></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 187. The data table should have the variables that are described in Table 10.12. The terms are required to generate topic names in the OUTTOPICS= data table. This option is only for topic discovery in SVD-only mode. This option conflicts with the PARSE statement, and only one of the two can be specified. If you want to run SVD-only mode without topic discovery, then you do not need to specify this option.

\[ K=k \]

specifies the number of columns in the matrices \( U \), \( V \), and \( S \). This value is the number of dimensions of the data table after SVD is performed. If the value of \( k \) is too large, then the TEXTMINE procedure runs for an unnecessarily long time. This option takes precedence over the MAX_K= option. This option also controls the number of topics that are extracted from the text corpus when the ROTATION= option is specified.

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

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.

\[ \text{NO}CUTOFFS \]

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.

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

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

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 187. 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 187.

RESOLUTION=LOW | MED | HIGH
RES=LOW | MED | HIGH
specifies how to calculate the recommended number of dimensions (resolution) for the singular value decomposition. If you specify this option, you must also specify the MAX_K= option. A low-resolution singular value decomposition returns fewer dimensions than a high-resolution singular value decomposition. This option recommends the value of $k$ (the number of columns in the matrices $U$, $V$, and $S$) heuristically based on the value specified in the MAX_K= option. Assume that the MAX_K= option is set to $n$ and a singular value decomposition that has $n$ dimensions accounts for $t\%$ of the total variance. You can specify the following values:

- **HIGH** always recommends the maximum number of dimensions; that is, $k = n$.
- **MED** recommends a $k$ that explains $(5/6) \times t\%$ of the total variance.
- **LOW** recommends a $k$ that explains $(2/3) \times t\%$ of the total variance.

By default, RESOLUTION=HIGH.

ROTATION=VARIMAX | PROMAX
specifies the type of rotation to be used in order to maximize the explanatory power of each topic. You can specify the following values:

- **PROMAX** does an oblique rotation on the original left singular vectors and generates topics that might be correlated.
- **VARIMAX** does an orthogonal rotation on the original left singular vectors and generates uncorrelated topics.

By default, ROTATION=VARIMAX.

ROW=variable
specifies the variable that contains the row indices of the term-by-document matrix. You must specify this option when you run PROC TEXTMINE in SVD-only mode (that is, when you specify the SVD statement but not the PARSE statement).
SVDS=\texttt{CAS-libref.data-table}

specifies the output data table to contain the calculated singular values. \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 187.

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

specifies the data table to contain the calculated left singular vectors. \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 187.

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

specifies the data table to contain the calculated right singular vectors. \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 187.

TOL=\epsilon

specifies the maximum allowable tolerance for the singular value. Let $A$ be a matrix. Suppose $\lambda_i$ is the $i$th singular value of $A$ and $\xi_i$ is the corresponding right singular vector. The SVD computation terminates when for all $i \in \{1, \ldots, k\}$, $\lambda_i$ and $\xi_i$ satisfy $\| A^\top A \xi - \lambda \xi \|_2^2 \leq \epsilon$. The default value of $\epsilon$ is $10^{-6}$, which is more than adequate for most text mining problems.

\textbf{TARGET Statement}

\texttt{TARGET \ variable ;}

This statement specifies the \texttt{variable} that contains the information about the category that a document belongs to. The target \texttt{variable} can be any nominal or ordinal variable; it is used in calculating mutual information term weighting.

\textbf{VARIABLES Statement}

\texttt{VARIABLES \ variable ;}

\texttt{VAR \ variable ;}

This statement specifies the \texttt{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 10.5 shows all possible part-of-speech tags.

<table>
<thead>
<tr>
<th>Part-of-Speech Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABBR</td>
<td>Abbreviation</td>
</tr>
<tr>
<td>ADJ</td>
<td>Adjective</td>
</tr>
<tr>
<td>ADV</td>
<td>Adverb</td>
</tr>
<tr>
<td>AUX</td>
<td>Auxiliary or modal term</td>
</tr>
<tr>
<td>CONJ</td>
<td>Conjunction</td>
</tr>
<tr>
<td>DET</td>
<td>Determiner</td>
</tr>
<tr>
<td>INTERJ</td>
<td>Interjection</td>
</tr>
<tr>
<td>NOUN</td>
<td>Noun</td>
</tr>
<tr>
<td>NUM</td>
<td>Number or numeric expression</td>
</tr>
<tr>
<td>PART</td>
<td>Infinitive marker, negative participle, or possessive marker</td>
</tr>
<tr>
<td>PREF (Korean only)</td>
<td>Prefix</td>
</tr>
<tr>
<td>PREP</td>
<td>Preposition</td>
</tr>
<tr>
<td>PRON</td>
<td>Pronoun</td>
</tr>
<tr>
<td>PROP</td>
<td>Proper noun</td>
</tr>
<tr>
<td>PUNCT</td>
<td>Punctuation</td>
</tr>
<tr>
<td>VERB</td>
<td>Verb</td>
</tr>
<tr>
<td>VERBADJ</td>
<td>Verbal adjective</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, “Person,” “Location,” “Company,” and “Measurement” are identified as classes for “George W. Bush,” “Boston,” “SAS Institute,” “2.5 inches,” respectively. Table 10.6 shows all valid entities for English. Not all languages support all entities. Table 10.7, Table 10.8, and Table 10.9 indicate the languages that are available for each entity.

Table 10.6 All Valid English Entities

<table>
<thead>
<tr>
<th>Entities</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS</td>
<td>Postal address or number and street name</td>
</tr>
<tr>
<td>COMPANY</td>
<td>Company name</td>
</tr>
<tr>
<td>CURRENCY</td>
<td>Currency or currency expression</td>
</tr>
<tr>
<td>INTERNET</td>
<td>Email address or URL</td>
</tr>
<tr>
<td>LOCATION</td>
<td>City, county, state, political, or geographical place or region</td>
</tr>
<tr>
<td>MEASURE</td>
<td>Measurement or measurement expression</td>
</tr>
<tr>
<td>NOUN_GROUP</td>
<td>Phrases that contain multiple words</td>
</tr>
<tr>
<td>ORGANIZATION</td>
<td>Government, legal, or service agency</td>
</tr>
<tr>
<td>PERCENT</td>
<td>Percentage or percentage expression</td>
</tr>
<tr>
<td>PERSON</td>
<td>Person’s name</td>
</tr>
<tr>
<td>PHONE</td>
<td>Telephone number</td>
</tr>
<tr>
<td>PROP_MISC</td>
<td>Proper noun with an ambiguous classification</td>
</tr>
<tr>
<td>SSN</td>
<td>Social Security number</td>
</tr>
<tr>
<td>TIME</td>
<td>Time or time expression</td>
</tr>
<tr>
<td>TIME_PERIOD</td>
<td>Measure of time expressions</td>
</tr>
<tr>
<td>TITLE</td>
<td>Person’s title or position</td>
</tr>
<tr>
<td>VEHICLE</td>
<td>Motor vehicle, including color, year, make, and model</td>
</tr>
</tbody>
</table>

Table 10.7 Supported Language-Entity Pairs, Part 1

<table>
<thead>
<tr>
<th>Language</th>
<th>Address</th>
<th>Company</th>
<th>Currency</th>
<th>Date</th>
<th>Internet</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chinese</td>
<td></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>
<td>✔</td>
</tr>
<tr>
<td>English</td>
<td>✔</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>
<td></td>
</tr>
<tr>
<td>French</td>
<td>✔</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>
<td>✔</td>
</tr>
<tr>
<td>Italian</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Table 10.7  continued

<table>
<thead>
<tr>
<th>Language</th>
<th>Address</th>
<th>Company</th>
<th>Currency</th>
<th>Date</th>
<th>Internet</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Japanese</td>
<td>✓</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>
<td>✓</td>
</tr>
<tr>
<td>Portuguese</td>
<td>✓</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>
<td></td>
</tr>
<tr>
<td>Spanish</td>
<td>✓</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>
<td></td>
</tr>
</tbody>
</table>

Table 10.8  Supported Language-Entity Pairs, Part 2

<table>
<thead>
<tr>
<th>Language</th>
<th>Measure</th>
<th>Noun_Group</th>
<th>Organization</th>
<th>Percent</th>
<th>Person</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chinese</td>
<td></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>
<td>✓</td>
</tr>
<tr>
<td>English</td>
<td>✓</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>
<td></td>
</tr>
<tr>
<td>French</td>
<td>✓</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>
<td>✓</td>
</tr>
<tr>
<td>Italian</td>
<td>✓</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>
<td>✓</td>
</tr>
<tr>
<td>Korean</td>
<td></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>
<td>✓</td>
</tr>
<tr>
<td>Russian</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spanish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Turkish</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10.9  Supported Language-Entity Pairs, Part 3

<table>
<thead>
<tr>
<th>Language</th>
<th>Prop_Misc</th>
<th>SSN</th>
<th>Time</th>
<th>Time_Period</th>
<th>Title</th>
<th>Vehicle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chinese</td>
<td>✓</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>
<td></td>
</tr>
<tr>
<td>English</td>
<td>✓</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>
<td></td>
</tr>
<tr>
<td>French</td>
<td>✓</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>
<td></td>
</tr>
<tr>
<td>Italian</td>
<td>✓</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>
<td></td>
</tr>
<tr>
<td>Korean</td>
<td>✓</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>
<td></td>
</tr>
<tr>
<td>Russian</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spanish</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Turkish</td>
<td></td>
<td>✓</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 Chinese, Dutch, English, Finnish, French, German, Italian, Japanese, Korean, Portuguese, Russian, Spanish, and Turkish. 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 \(A = U\Sigma V^\top\). 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. In the current release, computing singular value decomposition requires the input data to contain at least 25 documents and at least as many documents as there are nodes in the grid. If $p$ nodes are used for computing singular value decomposition in a distributed computing environment, then the input data must contain at least $\max(p, 25)$ documents. 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
Chapter 10: The TEXTMINE Procedure

of the \( V \) matrix characterize the strength of the association of each document with each topic. Finally, the TEXTMINE procedure can output a topic table that contains the best set of descriptor terms for each topic.

Because topic discovery is derived from the \( U \) matrix of SVD (each column of the \( U \) matrix is rotated and corresponds to a topic), topic discovery options are specified in the SVD statement.

Output Data Tables

This section describes the output data tables that PROC TEXTMINE produces when you specify the corresponding option.

The OUTCHILD= Data Table

The OUTCHILD= option in the PARSE statement specifies the data table to contain a compressed representation of the sparse term-by-document matrix, which is usually very sparse. To save space, this matrix is stored in COO format.

If you do not specify the SHOWDROPPEDTERMS option in the PARSE statement, this data table saves only the kept terms.\(^1\)

The child frequencies are not attributed to their corresponding parent (as they are in the data table specified in the OUTPARENT= option). Using the example in the previous section, the data table that is generated by the OUTCHILD= option will have two entries:

\[
\begin{align*}
& t1 & d1 & 8 \\
& t2 & d1 & 1
\end{align*}
\]

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 10.10 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>
</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 10.10  continued

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagging</td>
<td>Whether tagging is used: “Y” indicates that tagging is used, and “N” indicates that it is not used</td>
</tr>
<tr>
<td>NG</td>
<td>Whether noun grouping is used: “Y” indicates that noun grouping is used, and “N” indicates that it is not used</td>
</tr>
<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  | d1  | 9 |

---

2Many elements of the matrix are 0.
The term count of “said” in d1 is attributed to its parent, “say.”

**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 10.11 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>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 10.12 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>
</tbody>
</table>
### Table 10.12  continued

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attribute</strong></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>
<tr>
<td><strong>Freq</strong></td>
<td>The frequency of a term in the entire document collection</td>
</tr>
<tr>
<td><strong>Numdocs</strong></td>
<td>The number of documents that contain the term</td>
</tr>
<tr>
<td><strong>_keep</strong></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><strong>Key</strong></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><strong>Parent</strong></td>
<td>The Key value of the term’s parent or a “.” (period):</td>
</tr>
<tr>
<td>• If a term has a parent, this variable contains the term number of that parent.</td>
<td></td>
</tr>
<tr>
<td>• If a term does not have a parent, this value is a “.” (period).</td>
<td></td>
</tr>
<tr>
<td>• If the values of Key, Parent, and Parent_id are identical, the parent occurs as itself.</td>
<td></td>
</tr>
<tr>
<td>• If the values of Parent and Parent_id are identical but differ from Key, the observation is a child.</td>
<td></td>
</tr>
<tr>
<td><strong>Parent_id</strong></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><strong>_ispar</strong></td>
<td>An indication of term’s status as a parent, child, or neither:</td>
</tr>
<tr>
<td>• A “+” (plus sign) indicates that the term is a parent.</td>
<td></td>
</tr>
<tr>
<td>• A “.” (period) indicates that the term is a child.</td>
<td></td>
</tr>
<tr>
<td>• A missing value indicates that the term is neither a parent nor a child.</td>
<td></td>
</tr>
<tr>
<td><strong>Weight</strong></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

If you specify the ROTATION= option in the SVD statement, 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.

System Configuration

Prerequisites for Running PROC TEXTMINE

To use the TEXTMINE procedure, you must have a valid SAS Text Miner license, and the language binary files that are provided under that license must be available on the grid for parsing text.

Configuring for Language Binary Files

PROC TEXTMINE needs to find the grid location of the language binary files that are used in parsing text. These binary files must be deployed to the grid. The GRID_TEXTANALYTICS_BIN_LOC macro variable can be specified to indicate the location of the binary files on the grid. If the macro variable is not specified, then PROC TEXTMINE uses the default installation location.

Deploying Language Binary Files on the Grid

The language binary files must be deployed to the grid. The SAS grid installation script can automatically install the language binary files.

If you choose to manually deploy the binary files, you must copy the binary files from the $SASROOT$ folder to the grid either by placing the files in a location that is accessible by all nodes of the grid or by placing a copy of the files on each node of the grid. The binary files are very large, and a shared location requires less space to store them. However, using a shared location means that the files must be distributed to the grid when PROC TEXTMINE runs, and this can be time-consuming. Consult your grid administrator for a recommended binary location on the grid.

Language Binary Files

The language binary files are originally installed in the following location: $SASROOT/misc/tktg.$

When you manually deploy the language binary files, you need to copy to the grid only the binary files that correspond to the language you plan to use. The following binary files are used for English processing:
● en-compounds.txt
● en-ne.li
● en-sastags.txt
● en-utf8.stkzo
● en-utf8.tkzo
● en-utf8-AnalInfSinFas.mdic
● en-utf8-std.htagger
● case_mapping.bin
● chlangid.bin
● chmap.bin
● tix.config
● language-data.yml

If you have licensed SAS Text Miner languages other than English, you will see other files in your installation directory. The filenames begin with the corresponding two-letter language codes. For any language that you want to use on the grid, you need to copy its corresponding language binary files to the grid.

The GRID_TEXTANALYTICS_BIN_LOC Macro

You can use the GRID_TEXTANALYTICS_BIN_LOC macro to tell the TEXTMINE procedure where to find the language binary files. If the macro variable is not specified, then the default installation location will be used.

Assume that the grid administrator has installed the language binary files to a directory named /global_dir/tktg/misc, which is accessible to all nodes of the grid. To tell PROC TEXTMINE the location of the language binary files, insert the following statement before calling the procedure:

```sas
%let GRID_TEXTANALYTICS_BIN_LOC=/global_dir/tktg/misc;
```

When storage space permits, you can ensure optimal performance by placing a copy of the language binary files on each node and using a relative pathname for the GRID_TEXTANALYTICS_BIN_LOC macro variable. For example, the grid administrator can create a directory whose pathname is /local_dir/tktg/misc on each node and can store the language binary files in that directory. When the following statement is specified, the TEXTMINE procedure goes to the directory /local_dir/tktg/misc on each node to load the binary files:

```sas
%let GRID_TEXTANALYTICS_BIN_LOC=/local_dir/tktg/misc;
```
Chapter 10: The TEXTMINE Procedure

Examples: TEXTMINE Procedure

Example 10.1: Parsing with No Options Turned On

This example parses five documents, which are in a generated data table. The following DATA step generates the five documents:

```plaintext
/* 1) create data table */

data mycas.CarNominations;
  infile datalines delimiter='|' missover;
  length text $70 ;
  input text$ i;
  datalines;
  The Ford Taurus is the World Car of the Year. |1
  Hyundai won the award last year.  |2
  Toyota sold the Toyota Tacoma in bright green. |3
  The Ford Taurus is sold in all colors except for lime green. |4
  The Honda Insight was World Car of the Year in 2008. |5
  ;
run;
```

The following statements run PROC TEXTMINE to parse the documents.

```plaintext
/* 2) starting code */
proc textmine data=mycas.CarNominations;
  doc_id i;
  var text;
  parse
    nostemming notagging nonoungroups
    termwgt = none
    cellwgt = none
    reducef = 1
    entities = none
    outparent = mycas.outparent
    outterms = mycas.outterms
    outchild = mycas.outchild
    outconfig = mycas.outconfig
  ;
run;

/* 3) print outterms data table */
data outterms; set mycas.outterms; run;
proc print data=outterms; run;
```
Output 10.1.1 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 10.1.1 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>.</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>toyota</td>
<td>Alpha</td>
<td>2</td>
<td>1</td>
<td>Y</td>
<td>2</td>
<td>.</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ford</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>3</td>
<td>.</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>tacoma</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>4</td>
<td>.</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>year</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>5</td>
<td>.</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>taurus</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>6</td>
<td>.</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>won</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>honda</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>8</td>
<td>.</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>bright</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>9</td>
<td>.</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>sold</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>10</td>
<td>.</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>colors</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>11</td>
<td>.</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>lime</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>12</td>
<td>.</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>except</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>hyundai</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>in</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>is</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>for</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>17</td>
<td>.</td>
<td>17</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>world</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>green</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>19</td>
<td>.</td>
<td>19</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>the</td>
<td>Alpha</td>
<td>8</td>
<td>5</td>
<td>Y</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>of</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>21</td>
<td>.</td>
<td>21</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>award</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>was</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>car</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>insight</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>25</td>
<td>.</td>
<td>25</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>last</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>26</td>
<td>.</td>
<td>26</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Example 10.2: Parsing with Stemming

This example uses the data table that is generated in Example 10.1. 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 10.2.1 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 10.2.1  The mycas.outterms Data Table with Stemming

<table>
<thead>
<tr>
<th>Obs</th>
<th>Term</th>
<th>Role</th>
<th>Attribute</th>
<th>Freq_numdocs</th>
<th>_keep</th>
<th>Key</th>
<th>Parent</th>
<th>Parent_id</th>
<th>_ispars</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>all</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>win</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>2</td>
<td>.</td>
<td>2</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>toyota</td>
<td>Alpha</td>
<td>2</td>
<td>1</td>
<td>Y</td>
<td>3</td>
<td>.</td>
<td>3</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>ford</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>4</td>
<td>.</td>
<td>4</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>tacoma</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>6</td>
<td>year</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>6</td>
<td>.</td>
<td>6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>taurus</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>won</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>26</td>
<td>2</td>
<td>.</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>honda</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>8</td>
<td>.</td>
<td>8</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>bright</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>9</td>
<td>.</td>
<td>9</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>be</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>10</td>
<td>.</td>
<td>10</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>sold</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>27</td>
<td>11</td>
<td>.</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>sell</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>11</td>
<td>.</td>
<td>11</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>colors</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>28</td>
<td>23</td>
<td>.</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>lime</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>12</td>
<td>.</td>
<td>12</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>except</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>hyundai</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>in</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>is</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>29</td>
<td>10</td>
<td>.</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>for</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>world</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>17</td>
<td>.</td>
<td>17</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>green</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>the</td>
<td>Alpha</td>
<td>8</td>
<td>5</td>
<td>Y</td>
<td>19</td>
<td>.</td>
<td>19</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>of</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>25</td>
<td>award</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>26</td>
<td>was</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>30</td>
<td>10</td>
<td>.</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>car</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>color</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>insight</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>last</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>25</td>
<td>.</td>
<td>25</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Example 10.3: Adding Entities and Noun Groups

This example uses the data table that is generated in Example 10.1. The following statements run PROC TEXTMINE to parse the documents. Because the NONOUNGROUPS option is not specified in the PARSE statement, noun groups are extracted, and because the ENTITIES=STD option is specified, entities are identified.

```sas
/* create data table */
data mycas.CarNominations;
infile datalines delimiter='|' missover;
length text $70 ;
input text$ i;
datalines;
The Ford Taurus is the World Car of the Year. |1
Hyundai won the award last year. |2
Toyota sold the Toyota Tacoma in bright green. |3
The Ford Taurus is sold in all colors except for lime green. |4
The Honda Insight was World Car of the Year in 2008. |5
;
run;

proc textmine data=mycas.CarNominations;
doc_id i;
var text;
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 10.3.1 shows the content of the mycas.outterms data table. Compared to Output 10.2.1, the mycas.outterms data table is longer, because it contains entities and noun groups. For example, “honda insight” is included in the mycas.outterms data table as an entity with Role=Vehicle, and “bright green” is also included in the mycas.outterms data table as a noun group.
Output 10.3.1  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</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>lime green</td>
<td>NOUN_GROUP</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>all</td>
<td>Alpha</td>
<td>Alpha</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>win</td>
<td>Alpha</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>4</td>
<td>toyota</td>
<td>Alpha</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>5</td>
<td>ford</td>
<td>Alpha</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>5</td>
<td>.</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>toyota tacoma</td>
<td>VEHICLE</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>7</td>
<td>tacoma</td>
<td>Alpha</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>8</td>
<td>year</td>
<td>Alpha</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>8</td>
<td>.</td>
<td>8</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>taurus</td>
<td>Alpha</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>9</td>
<td>.</td>
<td>9</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>won</td>
<td>Alpha</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>33</td>
<td>3</td>
<td>3 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>honda</td>
<td>Alpha</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>10</td>
<td>.</td>
<td>10</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>bright</td>
<td>Alpha</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>13</td>
<td>be</td>
<td>Alpha</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>12</td>
<td>.</td>
<td>12 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>sold</td>
<td>Alpha</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>34</td>
<td>15</td>
<td>15 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>bright green</td>
<td>NOUN_GROUP</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>16</td>
<td>ford taurus</td>
<td>VEHICLE</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>sell</td>
<td>Alpha</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>18</td>
<td>hyundai</td>
<td>COMPANY</td>
<td>Entity</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>19</td>
<td>colors</td>
<td>Alpha</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>35</td>
<td>30</td>
<td>30 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>lime</td>
<td>Alpha</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>17</td>
<td>.</td>
<td>17</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>except</td>
<td>Alpha</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>22</td>
<td>honda insight</td>
<td>VEHICLE</td>
<td>Entity</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>23</td>
<td>in</td>
<td>Alpha</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>is</td>
<td>Alpha</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>36</td>
<td>12</td>
<td>12 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>for</td>
<td>Alpha</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>26</td>
<td>toyota</td>
<td>COMPANY</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>world</td>
<td>Alpha</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>green</td>
<td>Alpha</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>29</td>
<td>the</td>
<td>Alpha</td>
<td>Alpha</td>
<td>8</td>
<td>5</td>
<td>Y</td>
<td>25</td>
<td>.</td>
<td>25</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>of</td>
<td>Alpha</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>26</td>
<td>.</td>
<td>26</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>world car</td>
<td>PROP_MISC</td>
<td>Entity</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>32</td>
<td>award</td>
<td>Alpha</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>28</td>
<td>.</td>
<td>28</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>was</td>
<td>Alpha</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>37</td>
<td>12</td>
<td>12 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>car</td>
<td>Alpha</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>29</td>
<td>.</td>
<td>29</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>color</td>
<td>Alpha</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>30</td>
<td>.</td>
<td>30 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>insight</td>
<td>Alpha</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>31</td>
<td>.</td>
<td>31</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>last</td>
<td>Alpha</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>32</td>
<td>.</td>
<td>32</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Example 10.4: Adding Part-of-Speech Tagging

This example uses the data table that is generated in Example 10.1. The following statements run PROC TEXTMINE to parse the documents. Because the NOTAGGING option is not specified in the PARSE statement, PROC TEXTMINE uses context clues to determine a term’s part of speech.

```sas
/* create data table */
data mycas.CarNominations;
 infile datalines delimiter='|' missover;
 length text $70 ;
 input text$ i;
datalines;
   The Ford Taurus is the World Car of the Year. |1
   Hyundai won the award last year. |2
   Toyota sold the Toyota Tacoma in bright green. |3
   The Ford Taurus is sold in all colors except for lime green. |4
   The Honda Insight was World Car of the Year in 2008. |5
; run;

proc textmine data=mycas.CarNominations;
doc_id i;
var text;
pars
  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 10.4.1 shows the content of the mycas.outterms data table. Compared to Output 10.3.1, the mycas.outterms data table also contains the part-of-speech tag for the terms.
## Output 10.4.1  The mycas.outterms Data Table with Part-of-Speech Tagging

<table>
<thead>
<tr>
<th>Obs</th>
<th>Term</th>
<th>Role</th>
<th>Attribute</th>
<th>Freq</th>
<th>numdocs</th>
<th>_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>car</td>
<td>Prop</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>lime green</td>
<td>NOUN_GROUP</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>2</td>
<td>.</td>
<td>2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>insight</td>
<td>Prop</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>4</td>
<td>last</td>
<td>Adj</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>5</td>
<td>lime</td>
<td>Adj</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>6</td>
<td>sell</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>6</td>
<td>.</td>
<td>6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>toyota tacoma</td>
<td>VEHICLE</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>bright</td>
<td>Adj</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>9</td>
<td>honda</td>
<td>Prop</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>10</td>
<td>sold</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>34</td>
<td>6</td>
<td>6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>colors</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>35</td>
<td>11</td>
<td>11</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>for</td>
<td>Prep</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>13</td>
<td>color</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>11</td>
<td>.</td>
<td>11</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>bright green</td>
<td>NOUN_GROUP</td>
<td>Alpha</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>15</td>
<td>the</td>
<td>Det</td>
<td>Alpha</td>
<td>8</td>
<td>5</td>
<td>Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>is</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>36</td>
<td>23</td>
<td>23</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>ford taurus</td>
<td>VEHICLE</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>hyundai</td>
<td>COMPANY</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>toyota</td>
<td>Prop</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>20</td>
<td>except</td>
<td>Verb</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>21</td>
<td>in</td>
<td>Prep</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>was</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>37</td>
<td>23</td>
<td>23</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>honda insight</td>
<td>VEHICLE</td>
<td>Entity</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>24</td>
<td>won</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>38</td>
<td>27</td>
<td>27</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>is</td>
<td>Aux</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>39</td>
<td>26</td>
<td>26</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>ford</td>
<td>Prop</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>27</td>
<td>world</td>
<td>Prop</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>21</td>
<td>.</td>
<td>21</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>year</td>
<td>Noun</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>be</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>all</td>
<td>Adj</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>toyota</td>
<td>COMPANY</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>25</td>
<td>.</td>
<td>25</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>be</td>
<td>Aux</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>26</td>
<td>.</td>
<td>26</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>win</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>27</td>
<td>.</td>
<td>27</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>taurus</td>
<td>Noun</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>28</td>
<td>.</td>
<td>28</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>35</td>
<td>world car</td>
<td>PROP_MISC</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>29</td>
<td>.</td>
<td>29</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>36</td>
<td>of</td>
<td>Prep</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>30</td>
<td>.</td>
<td>30</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>37</td>
<td>award</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>31</td>
<td>.</td>
<td>31</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>38</td>
<td>tacoma</td>
<td>Prop</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>32</td>
<td>.</td>
<td>32</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>39</td>
<td>green</td>
<td>Noun</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>33</td>
<td>.</td>
<td>33</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Example 10.5: Adding Synonyms

This example uses the data table that is generated in Example 10.1. So far, by looking at the mycas.outterms data tables that are generated by Example 10.1 to Example 10.4, you can see that the data are very “vehicle focused.” But suppose what you really care about are the companies. You can use a synonym list in parsing in order to map each vehicle to the company that produces it. The following DATA step generates the synonym list, and the following statements show this mapping:

```plaintext
/* create data table */
data mycas.CarNominations;
infile datalines delimiter='|' missover;
length text $70 ;
input text$ i;
datalines;
The Ford Taurus is the World Car of the Year. |1
Hyundai won the award last year. |2
Toyota sold the Toyota Tacoma in bright green. |3
The Ford Taurus is sold in all colors except for lime green. |4
The Honda Insight was World Car of the Year in 2008. |5
;
run;
/* create synonym list */
data mycas.synds;
infile datalines delimiter=',';
length Term $13;
input Term $ TermRole $ Parent $ ParentRole$;
datalines;
honda insight, VEHICLE, honda, COMPANY,
ford taurus, VEHICLE, ford, COMPANY,
toyota tacoma, VEHICLE, toyota, COMPANY,
;
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 10.5.1 shows the content of the mycas.outterms data table. You can see that the term “honda insight” (key=39) is assigned the parent term “honda” (key=6). Only the term “honda” appears in the mycas.outparent 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>car</td>
<td>Prop</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>lime green</td>
<td>NOUN_GROUP</td>
<td>Alpha</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>insight</td>
<td>Prop</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>4</td>
<td>last</td>
<td>Adj</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>5</td>
<td>lime</td>
<td>Adj</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>6</td>
<td>honda</td>
<td>COMPANY</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>7</td>
<td>sell</td>
<td>Verb</td>
<td>Alpha</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>toyota tacoma</td>
<td>VEHICLE</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>33</td>
<td>23</td>
<td>23 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>bright</td>
<td>Adj</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>8</td>
<td>.</td>
<td>8</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>honda</td>
<td>Prop</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>11</td>
<td>sold</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>34</td>
<td>7</td>
<td>7 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>colors</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>35</td>
<td>11</td>
<td>11 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>for</td>
<td>Prep</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>10</td>
<td>.</td>
<td>10</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>color</td>
<td>Noun</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>15</td>
<td>bright green</td>
<td>NOUN_GROUP</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>16</td>
<td>the</td>
<td>Det</td>
<td>Alpha</td>
<td>8</td>
<td>5</td>
<td>Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>is</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>36</td>
<td>21</td>
<td>21 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>ford taurus</td>
<td>VEHICLE</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>37</td>
<td>26</td>
<td>26 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>hyundai</td>
<td>COMPANY</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>toyota</td>
<td>Prop</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>21</td>
<td>except</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>in</td>
<td>Prep</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>17</td>
<td>.</td>
<td>17</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>was</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>38</td>
<td>21</td>
<td>21 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>honda insight</td>
<td>VEHICLE</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>39</td>
<td>6</td>
<td>6 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>won</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>40</td>
<td>25</td>
<td>25 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>is</td>
<td>Aux</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>41</td>
<td>24</td>
<td>24 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>ford</td>
<td>Prop</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>world</td>
<td>Prop</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>19</td>
<td>.</td>
<td>19</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>year</td>
<td>Noun</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>be</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>21</td>
<td>.</td>
<td>21 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>all</td>
<td>Adj</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>toyota</td>
<td>COMPANY</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>23</td>
<td>.</td>
<td>23 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>toyota</td>
<td>COMPANY</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>23</td>
<td>23</td>
<td>23 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>be</td>
<td>Aux</td>
<td>Alpha</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>35</td>
<td>win</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>25</td>
<td>.</td>
<td>25 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>ford</td>
<td>COMPANY</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>26</td>
<td>.</td>
<td>26 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>taurus</td>
<td>Noun</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>38</td>
<td>world car</td>
<td>PROP_MISC</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>28</td>
<td>.</td>
<td>28</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>of</td>
<td>Prep</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>29</td>
<td>.</td>
<td>29</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>award</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>30</td>
<td>.</td>
<td>30</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>tacoma</td>
<td>Prop</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>31</td>
<td>.</td>
<td>31</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>green</td>
<td>Noun</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>32</td>
<td>.</td>
<td>32</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Example 10.6: Adding a Stop List

This example uses the data table that is generated in Example 10.1. If you want to eliminate the entity TOYOTA from the analysis, you can enter the parent term “Toyota” with role=COMPANY in a stop list. When this stop list is an input, PROC TEXTMINE drops the term “Toyota” (role=COMPANY) and all its children terms in the mycas.outterms data table by marking _keep=N for these terms. The following DATA steps generate the synonym list and the stop list:

```plaintext
/* create data table */
data mycas.CarNominations;
infile datalines delimiter='|' missover;
length text $70 ;
input text$ i;
datalines;
The Ford Taurus is the World Car of the Year. |1
Hyundai won the award last year. |2
Toyota sold the Toyota Tacoma in bright green. |3
The Ford Taurus is sold in all colors except for lime green. |4
The Honda Insight was World Car of the Year in 2008. |5
;
r;

/* create synonym list */
data mycas.synds;
infile datalines delimiter=',';
length Term $13;
input Term $ TermRole $ Parent $ ParentRole$;
datalines;
honda insight, VEHICLE, honda, COMPANY,
ford taurus, VEHICLE, ford, COMPANY,
toyota tacoma, VEHICLE, toyota, COMPANY,
;
r;

/* create stop list*/
data mycas.stopList;
infile datalines delimiter='|' missover;
length term $25 role $40;
input term$ role$ ;
datalines;
toyota| COMPANY
;
r;

proc textmine data=mycas.CarNominations;
doc_id i;
var text;
parse
termwgt = none
cellwgt = none
reducef = 1
entities = std
```
Example 10.6: Adding a Stop List

    synonym    = mycas.synds
    stop       = mycas.stopList
    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 10.6.1 shows the content of the mycas.outterms data table. You can see that the term “Toyota, COMPANY” and the term “Toyota Tacoma, VEHICLE” are removed from the mycas.outterms data table. However, the term “Toyota, Prop” remains in the data table because its role is not COMPANY. The mycas.outparent data table is shorter than the one generated in Example 10.5.

Output 10.6.1 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</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>car</td>
<td>Prop</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>lime green</td>
<td>NOUN_GROUP</td>
<td>Alpha</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>insight</td>
<td>Prop</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>4</td>
<td>last</td>
<td>Adj</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>5</td>
<td>lime</td>
<td>Adj</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>6</td>
<td>honda</td>
<td>COMPANY</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>7</td>
<td>sell</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>bright</td>
<td>Adj</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>9</td>
<td>honda</td>
<td>Prop</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>10</td>
<td>sold</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>32</td>
<td>7</td>
<td>7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>colors</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>33</td>
<td>11</td>
<td>11</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>for</td>
<td>Prep</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>13</td>
<td>color</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>11</td>
<td>.</td>
<td>11 +</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>bright green</td>
<td>NOUN_GROUP</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>15</td>
<td>the</td>
<td>Det</td>
<td>Alpha</td>
<td>8</td>
<td>5</td>
<td>Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>is</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>34</td>
<td>21</td>
<td>21</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>ford taurus</td>
<td>VEHICLE</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>35</td>
<td>25</td>
<td>25 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>hyundai</td>
<td>COMPANY</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>toyota</td>
<td>Prop</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>except</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>in</td>
<td>Prep</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>17</td>
<td>.</td>
<td>17</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>was</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>36</td>
<td>21</td>
<td>21</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>honda insight</td>
<td>VEHICLE</td>
<td>Entity</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>37</td>
<td>6</td>
<td>6 .</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>won</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>38</td>
<td>24</td>
<td>24</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>is</td>
<td>Aux</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>39</td>
<td>23</td>
<td>23</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>ford</td>
<td>Prop</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>27</td>
<td>world</td>
<td>Prop</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>19</td>
<td>.</td>
<td>19</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>year</td>
<td>Noun</td>
<td>Alpha</td>
<td>3</td>
<td>3</td>
<td>Y</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>be</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>21</td>
<td>.</td>
<td>21 +</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>all</td>
<td>Adj</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>be</td>
<td>Aux</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>32</td>
<td>win</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>24</td>
<td>.</td>
<td>24 +</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>ford</td>
<td>COMPANY</td>
<td>Entity</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>25</td>
<td>.</td>
<td>25 +</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>taurus</td>
<td>Noun</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>world car</td>
<td>PROP_MISC</td>
<td>Entity</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>36</td>
<td>of</td>
<td>Prep</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>28</td>
<td>.</td>
<td>28</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>37</td>
<td>award</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>29</td>
<td>.</td>
<td>29</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>38</td>
<td>tacoma</td>
<td>Prop</td>
<td>Alpha</td>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>30</td>
<td>.</td>
<td>30</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>39</td>
<td>green</td>
<td>Noun</td>
<td>Alpha</td>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>31</td>
<td>.</td>
<td>31</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Example 10.7: 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 10.1 to show how to use the MULTITERM= option. The following DATA steps generate a synonym list, a stop list, and 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 synonym list */
data mycas.synds;
infile datalines delimiter=',';
  length Term $13;
  input Term $ TermRole $ Parent $ ParentRole$;
datalines;
  honda insight, VEHICLE, honda, COMPANY,
  ford taurus, VEHICLE, ford, COMPANY,
  toyota tacoma, VEHICLE, toyota, COMPANY,
;run;
/* create stop list */
data mycas.stopList;
infile datalines delimiter='|' missover;
  length term $25 role $40;
  input term$ role$;
datalines;
  toyota| COMPANY
;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;
```
Chapter 10: The TEXTMINE Procedure

```
parse
termwgt  = none
cellwgt  = none
reducef  = 1
entities  = std
synonym  = mycas.synds
stop     = mycas.stopList
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 10.7.1 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 10.7.1 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>car</td>
<td>Prop</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>1 Y</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>lime green</td>
<td>NOUN_GROUP</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>2 Y</td>
<td>.</td>
<td>2</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>insight</td>
<td>Prop</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>3</td>
<td>.</td>
<td>3</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>last</td>
<td>Adj</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>4</td>
<td>.</td>
<td>4</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>lime</td>
<td>Adj</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>5</td>
<td>.</td>
<td>5</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>honda</td>
<td>COMPANY</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>6</td>
<td>.</td>
<td>6</td>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>sell</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>7</td>
<td>.</td>
<td>7</td>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>bright</td>
<td>Adj</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>8</td>
<td>.</td>
<td>8</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>honda</td>
<td>Prop</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>9</td>
<td>.</td>
<td>9</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>sold</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>31</td>
<td>7</td>
<td>7</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>colors</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>32</td>
<td>11</td>
<td>11</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>except for</td>
<td>Prep</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>10</td>
<td>.</td>
<td>10</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>color</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>11</td>
<td>.</td>
<td>11</td>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>bright green</td>
<td>NOUN_GROUP</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>12</td>
<td>.</td>
<td>12</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>the</td>
<td>Det</td>
<td>Alpha</td>
<td>8</td>
<td>5 Y</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>is</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>33</td>
<td>20</td>
<td>20</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>ford taurus</td>
<td>VEHICLE</td>
<td>Entity</td>
<td>2</td>
<td>2 Y</td>
<td>34</td>
<td>24</td>
<td>24</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>hyundai</td>
<td>COMPANY</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>14</td>
<td>.</td>
<td>14</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>toyota</td>
<td>Prop</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>15</td>
<td>.</td>
<td>15</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>in</td>
<td>Prep</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>was</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>35</td>
<td>20</td>
<td>20</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>honda insight</td>
<td>VEHICLE</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>36</td>
<td>6</td>
<td>6</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>won</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>37</td>
<td>23</td>
<td>23</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>is</td>
<td>Aux</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>38</td>
<td>22</td>
<td>22</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>ford</td>
<td>Prop</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>17</td>
<td>.</td>
<td>17</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>world</td>
<td>Prop</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>year</td>
<td>Noun</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>19</td>
<td>.</td>
<td>19</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>be</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>20</td>
<td>.</td>
<td>20</td>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>all</td>
<td>Adj</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>21</td>
<td>.</td>
<td>21</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>be</td>
<td>Aux</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>22</td>
<td>.</td>
<td>22</td>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>win</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>23</td>
<td>.</td>
<td>23</td>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>ford</td>
<td>COMPANY</td>
<td>Entity</td>
<td>2</td>
<td>2 Y</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>taurus</td>
<td>Noun</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>25</td>
<td>.</td>
<td>25</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>world car</td>
<td>PROP_MISC</td>
<td>Entity</td>
<td>2</td>
<td>2 Y</td>
<td>26</td>
<td>.</td>
<td>26</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>35</td>
<td>of</td>
<td>Prep</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>27</td>
<td>.</td>
<td>27</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>36</td>
<td>award</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>28</td>
<td>.</td>
<td>28</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>37</td>
<td>tacoma</td>
<td>Prop</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>29</td>
<td>.</td>
<td>29</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>38</td>
<td>green</td>
<td>Noun</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>30</td>
<td>.</td>
<td>30</td>
<td>.</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example 10.8: Selecting Parts of Speech and Entities to Ignore

This example uses the data table that is generated in Example 10.1. 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 “PROP_MISC,” you can add another SELECT statement that includes “PROP_MISC” 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;

/* create synonym list */
data mycas.synds;
infile datalines delimiter=',';
length Term $13;
input Term $ TermRole $ Parent $ ParentRole$;
datalines;
   honda insight, VEHICLE, honda, COMPANY,
   ford taurus, VEHICLE, ford, COMPANY,
   toyota tacoma, VEHICLE, toyota, COMPANY,
;
run;

/* create stop list */
data mycas.stopList;
infile datalines delimiter='|' missover;
length term $25 role $40;
input term$ role$ ;
datalines;
   toyota| COMPANY
;
run;

proc textmine data=mycas.CarNominations;
doc_id i;
var text;
parse
   termwgt = none
   cellwgt = none
   reducef = 1
   entities = std
   synonym = mycas.synds
```

Example 10.8: Selecting Parts of Speech and Entities to Ignore

```plaintext
stop = mycas.stopList
outparent = mycas.outparent
outterms = mycas.outterms
outchild = mycas.outchild
outconfig = mycas.outconfig
;
select "prep" "det" "prop"/ignore;
select "prop_misc"/group="entities" ignore;
run;

data outterms; set mycas.outterms; run;
proc print data= outterms; run;
```

Output 10.8.1 shows the content of the mycas.outterms data table. You can see that prepositions, determiners, and proper nouns are excluded. Terms that are labeled “PROP_MISC” are also excluded.

Output 10.8.1 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>lime green</td>
<td>NOUN_GROUP</td>
<td>Alpha</td>
<td>1</td>
<td>1 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>last</td>
<td>Adj</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>2 .</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>lime</td>
<td>Adj</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>3 .</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>honda</td>
<td>COMPANY</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>4 .</td>
<td>4 +</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>sell</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>5 .</td>
<td>5 +</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>bright</td>
<td>Adj</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>6 .</td>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>sold</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>20</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>colors</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>21</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>color</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>7 .</td>
<td>7 +</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>bright green</td>
<td>NOUN_GROUP</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>8 .</td>
<td>8</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>is</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>22</td>
<td>12</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>ford taurus</td>
<td>VEHICLE</td>
<td>Entity</td>
<td>2</td>
<td>2 Y</td>
<td>23</td>
<td>16</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>hyundai</td>
<td>COMPANY</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>9 .</td>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>except</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>was</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>24</td>
<td>12</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>honda insight</td>
<td>VEHICLE</td>
<td>Entity</td>
<td>1</td>
<td>1 Y</td>
<td>25</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>won</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>26</td>
<td>15</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>is</td>
<td>Aux</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>27</td>
<td>14</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>year</td>
<td>Noun</td>
<td>Alpha</td>
<td>3</td>
<td>3 Y</td>
<td>11</td>
<td>11</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>be</td>
<td>Verb</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>12</td>
<td>12 +</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>all</td>
<td>Adj</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>13</td>
<td>13</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>be</td>
<td>Aux</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>14</td>
<td>14 +</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>win</td>
<td>Verb</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>15</td>
<td>15 +</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>ford</td>
<td>COMPANY</td>
<td>Entity</td>
<td>2</td>
<td>2 Y</td>
<td>16</td>
<td>16 +</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>taurus</td>
<td>Noun</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>17</td>
<td>17</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>award</td>
<td>Noun</td>
<td>Alpha</td>
<td>1</td>
<td>1 Y</td>
<td>18</td>
<td>18</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>green</td>
<td>Noun</td>
<td>Alpha</td>
<td>2</td>
<td>2 Y</td>
<td>19</td>
<td>19</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Chapter 11
The TMSCORE Procedure

Contents
Overview: TMSCORE Procedure .................................................. 235
PROC TMSCORE Features ......................................................... 235
Using CAS Sessions and CAS Engine Librefs .......................... 236
Getting Started: TMSCORE Procedure ...................................... 236
Syntax: TMSCORE Procedure .................................................... 239
PROC TMSCORE Statement ..................................................... 240
DOC_ID Statement ................................................................. 242
VARIABLES Statement ......................................................... 242
Details: TMSCORE Procedure .................................................... 242
System Configuration .............................................................. 242
Prerequisites for Running PROC TMSCORE ...................... 242
Configuration for Language Binary Files ............................ 242

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 Svd, 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 240.

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 5 in Chapter 2, “Shared Concepts.”

Getting Started: TMSCORE Procedure

**Note:** Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table
name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 5 and “Loading a SAS Data Set onto a CAS Server” on page 6 in Chapter 2, “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.

```sas
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;
```
Chapter 11: The TMSCORE Procedure

data mycas.getstart_score;
    infile datalines delimeter='|' missover;
    length text $150;
    input text$ did;
    datalines;
    Big data according to SAS
    At SAS, we consider two other dimensions when thinking about big data:
    Variability. In addition to the increasing velocities and varieties of data, data flows can be highly inconsistent with periodic peaks.
    Is something big trending in the social media?
    Perhaps there is a high-profile IPO looming.
    Maybe swimming with pigs in the Bahamas is suddenly the must-do vacation activity. Daily, seasonal and event-triggered peak data loads can be challenging to manage - especially with social media involved.
    Complexity. When you deal with huge volumes of data, it comes from multiple sources. It is quite an undertaking to link, match, cleanse and transform data across systems. However, it is necessary to connect and correlate relationships, hierarchies and multiple data linkages or your data can quickly spiral out of control. Data governance can help you determine how disparate data relates to common definitions and how to systematically integrate structured and unstructured data assets to produce high-quality information that is useful, appropriate and up-to-date.
    Ultimately, regardless of the factors involved, we believe that the term big data is relative whenever an organization's ability to handle, store and analyze data exceeds its current capacity.

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`.

```plaintext
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 11.1.

The following statements use PROC PRINT to show the content of the first 10 rows of the `mycas.docpro` data table, which is generated by the TMSCORE procedure:

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

Figure 11.1 shows the output of PROC PRINT.

---

**Figure 11.1** 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>3</td>
<td>0.8539143279</td>
<td>-0.324165052</td>
<td>-0.392891285</td>
<td>-0.104971802</td>
<td>0.0190970858</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0.7919376615</td>
<td>0.1438623939</td>
<td>0.2905376543</td>
<td>0.49632457</td>
<td>-0.146246862</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>0.8764284867</td>
<td>-0.323410455</td>
<td>0.0020293279</td>
<td>0.2184107934</td>
<td>0.2820840172</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>0.4401286327</td>
<td>-0.325098542</td>
<td>0.3095168259</td>
<td>-0.261390564</td>
<td>0.7324425111</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>0.8893052136</td>
<td>-0.195906296</td>
<td>-0.357729233</td>
<td>0.0169109904</td>
<td>0.2061571602</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>0.8917319278</td>
<td>-0.238277292</td>
<td>-0.293958253</td>
<td>-0.228301642</td>
<td>-0.097493626</td>
</tr>
<tr>
<td>7</td>
<td>18</td>
<td>0.9171698276</td>
<td>-0.317438567</td>
<td>0.0746891277</td>
<td>-0.171043677</td>
<td>-0.152308432</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>0.8893052136</td>
<td>-0.195906296</td>
<td>-0.357729233</td>
<td>0.0169109904</td>
<td>0.2061571602</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>0.8566937355</td>
<td>-0.104229717</td>
<td>0.2763641007</td>
<td>-0.370132146</td>
<td>-0.204541165</td>
</tr>
<tr>
<td>10</td>
<td>26</td>
<td>0.6359676823</td>
<td>-0.035894584</td>
<td>-0.121093532</td>
<td>0.5380064809</td>
<td>0.5386483722</td>
</tr>
</tbody>
</table>

---

**Syntax: TMSCORE Procedure**

The following statements are available in the TMSCORE procedure:

```plaintext
PROC TMSCORE DATA=CAS-libref.data-table <options> ;
   VARIABLES variable ;
   DOC_ID variable ;
```
PROC TMSCORE Statement

PROC TMSCORE DATA=\texttt{CAS-libref.data-table} <options> ;

The PROC TMSCORE statement invokes the procedure. Table 11.1 summarizes the \textit{options} in the statement by function. The \textit{options} are then described fully in alphabetical order.

<table>
<thead>
<tr>
<th>option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{Basic Options}</td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>Specifies the input document data table</td>
</tr>
<tr>
<td>DOC=</td>
<td>Specifies the data table that contains the terms to be used for scoring</td>
</tr>
<tr>
<td>TERMS=</td>
<td>Specifies the data table that contains the configuration information</td>
</tr>
<tr>
<td>CONFIG=</td>
<td>Specifies the data table that contains the U matrix whose columns are the left singular vectors</td>
</tr>
<tr>
<td>SVDU=</td>
<td>Specifies the data table that contains the term-by-document frequency matrix that is used to model the document collection. In this matrix, the child terms are not represented and child terms’ frequencies are attributed to their corresponding parents.</td>
</tr>
<tr>
<td>\textbf{Output Options}</td>
<td></td>
</tr>
<tr>
<td>OUTPARENT=</td>
<td>Specifies the data table that contains the projections of the documents</td>
</tr>
</tbody>
</table>

You must specify the following option:

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

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

names the input data table for PROC TMSCORE 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 236.

\texttt{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 \textit{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 236. 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 210 of Chapter 10, “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 236. 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 211 of Chapter 10, “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 236. 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 236. 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 10, “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 236. Specify the table that was generated by the OUTTERMS= option in the PARSE statement of the TEXTMINE procedure during training. This data table conveys to PROC TMSCORE which terms should be used in the analysis and whether they should be mapped to a parent. The data table also assigns to each term a key that corresponds to the key that is used in the input data table that is specified by the SVDU= option. For more information about this data table, see the section “The OUTTERMS= Data Table” on page 212 of Chapter 10, “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 204 of Chapter 10, “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.

Configuration for Language Binary Files

PROC TMSCORE needs to find the grid location of the language binary files that are used in parsing text. These binary files must be deployed to the grid. The GRID_TEXTANALYTICS_BIN_LOC macro can be specified to indicate the location of the binary files on your grid. If the macro variable is not set, then the default installation location `$SASROOT/misc/tktg` will be used. For more information, see the section “Configuring for Language Binary Files” on page 214 of Chapter 10, “The TEXTMINE Procedure.”
Subject Index

*k-fold cross validation*
- FOREST procedure, 99
- GRADBOOST procedure, 131
- NNET procedure, 161

**ASTORE procedure**
- input data tables, 19
- output data tables, 19

**bagging the data**
- FOREST procedure, 92

**best configuration**
- FOREST procedure, 101
- GRADBOOST procedure, 132
- NNET procedure, 162

**boosting**
- GRADBOOST procedure, 126

**CODE statement**
- syntax (Shared Concepts), 7

**computational method**
- FACTMAC procedure, 11
- FOREST procedure, 11
- GRADBOOST procedure, 11
- NNET procedure, 11, 159
- SVMACHINE procedure, 11
- TEXTMINE procedure, 11
- TMSCORE procedure, 11

**convergence status**
- NNET procedure, 162

**cross validation results**
- FOREST procedure, 101
- GRADBOOST procedure, 133
- NNET procedure, 162

**displayed output**
- FACTMAC procedure, 67
- FOREST procedure, 100
- GRADBOOST procedure, 132
- NNET procedure, 161
- SVMACHINE procedure, 179

**FACTMAC procedure, 59**
- computational method, 11
- displayed output, 67
- final exact loss, 68
- input data tables, 63
- interval variables, 68
- iteration history, 67

**FOREST procedure, 74**
- *k-fold cross validation*, 99
- bagging the data, 92
- best configuration, 101
- computational method, 11
- cross validation results, 101
- displayed output, 100
- fit statistics, 100
- handling missing values, 95
- handling values that are absent from training data, 96
- input data tables, 82
- measuring variable importance, 96
- model information, 100
- multithreading, 11
- number of observations, 100
- ODS table names, 101
- output data tables, 90
- output table, 101
- parameter tuning, 98
- preselection, 93
- splitting criteria, 89
- test data, 10
- training a decision tree, 93
- tuner information, 100
- tuner results, 101
- tuner summary, 100
- tuner timing, 101
- validation, 10
- variable importance, 100

**GRADBOOST procedure, 108**
- *k-fold cross validation*, 131
best configuration, 132
boosting, 126
computational method, 11
cross validation results, 133
displayed output, 132
fit statistics, 132
handling missing values, 127
input data tables, 115
measuring prediction error, 127
measuring variable importance, 128
model information, 132
multithreading, 11
number of observations, 132
ODS table names, 133
output data tables, 124
output table, 133
parameter tuning, 130
subsampling the data, 126
test data, 10
training a decision tree, 126
tuner information, 132
tuner results, 133
tuner summary, 132
tuner timing, 132
validation, 10
variable importance, 132

handling missing values
FOREST procedure, 95
GRADBOOST procedure, 127

handling values that are absent from training data
FOREST procedure, 96

input variables
NNET procedure, 152

interval variables
FACTMAC procedure, 68

iteration history
NNET procedure, 162

iteration history table
SVMACHINE procedure, 179

linear kernels
SVMACHINE procedure, 177

measuring prediction error
GRADBOOST procedure, 127
measuring variable importance
FOREST procedure, 96
GRADBOOST procedure, 128

misclassification matrix table
SVMACHINE procedure, 179

model information
FOREST procedure, 100
GRADBOOST procedure, 132

NNET procedure, 162
model information table
SVMACHINE procedure, 179

multithreading
FACTMAC procedure, 11
FOREST procedure, 11
GRADBOOST procedure, 11
NNET procedure, 11
SVMACHINE procedure, 11
TEXTMINE procedure, 11
TMSCORE procedure, 11

NNET procedure, 140
k-fold cross validation, 161
best configuration, 162
computational method, 11, 159
convergence status, 162
cross validation results, 162
displayed output, 161
input data tables, 143
iteration history, 162
model information, 162
multithreading, 11
ODS table names, 163
output data tables, 155
parameter tuning, 160
score information, 162
test data, 10
tuner information, 162
tuner results, 162
tuner summary, 162
validation, 10

nonlinear kernels
SVMACHINE procedure, 177
number of hidden neurons
NNET procedure, 151
number of observations
FOREST procedure, 100
GRADBOOST procedure, 132
number of observations table
SVMACHINE procedure, 179

optimization
SVMACHINE procedure, 178

options summary
PARSE statement, 193
PROC FACTMAC statement, 63
PROC FOREST statement, 80
PROC GRADBOOST statement, 113
PROC TEXTMINE statement, 192
PROC TMSCORE statement, 240
SELECT statement, 198
SVD statement, 199

output CAS tables
FACTMAC procedure, 68
output data tables
FACTMAC procedure, 68
output table
FOREST procedure, 101
GRADBOOST procedure, 133
parameter tuning
FOREST procedure, 98
GRADBOOST procedure, 130
NNET procedure, 160
PARTITION statement
syntax (Shared Concepts), 9
preselection
FOREST procedure, 93
PROC ASTORE, 13
PROC ASTORE features, 13
PROC SVMACHINE, 169
PROC SVMACHINE features, 170
score information
NNET procedure, 162
scoring process
SVMACHINE procedure, 178
Shared Concepts
CODE statement, 7
PARTITION statement, 9
sparse matrix
TEXTMINE procedure, 211
splitting criteria
FOREST procedure, 89
subsampling the data
GRADBOOST procedure, 126
SVMACHINE procedure
computational method, 11
displayed output, 179
fit statistics table, 179
input data tables, 173
iteration history table, 179
linear kernels, 177
misclassification matrix table, 179
model information table, 179
multithreading, 11
nonlinear kernels, 177
number of observations table, 179
ODS table names, 179
optimization, 178
output data tables, 176
scoring process, 178
test data, 10
training results table, 179
validation, 10
target variables
NNET procedure, 156
test data
FACTMAC procedure, 10
FOREST procedure, 10
GRADBOOST procedure, 10
NNET procedure, 10
SVMACHINE procedure, 10
TEXTMINE procedure, 10
TMSCORE procedure, 10
TEXTMINE procedure, 186
cell weight, 194
computational method, 11
coordinate list (COO) format, 208
entity, 195
filtering term by frequency, 197
input data tables, 192
language used by input data tables, 193
multiterm words list, 195
multithreading, 11
noun groups, 195
number of threads, 193
show dropped terms, 197
sparse format, 208
sparse matrix, 211
start list, 197
stemming, 195
stop list, 197
SVD, singular value decomposition, 208
synonym list, 197
system configuration, 214
tagging, 195
term weight, 197
test data, 10
transactional style, 211
validation, 10
variable name style, 193
TEXTMINE procedure, system configuration
configuration for language binary files, 214
deploying language binary files, 214
GRID_TEXTANALYTICS_BIN_LOC macro, 215
language binary files, 214
prerequisite, 214
TMSCORE procedure, 235
computational method, 11
input data tables, 240
multithreading, 11
system configuration, 242
test data, 10
validation, 10
TMSCORE procedure, system configuration
configuration for language binary files, 242
prerequisite, 242
training a decision tree
FOREST procedure, 93
GRADBOOST procedure, 126

training results table

SVMACHINE procedure, 179

transactional style

TEXTMINE procedure, 211

tuner information

FOREST procedure, 100
GRADBOOST procedure, 132
NNET procedure, 162

tuner results

FOREST procedure, 101
GRADBOOST procedure, 133
NNET procedure, 162

tuner summary

FOREST procedure, 100
GRADBOOST procedure, 132
NNET procedure, 162

tuner timing

FOREST procedure, 101
GRADBOOST procedure, 132

validation

FACTMAC procedure, 10
FOREST procedure, 10
GRADBOOST procedure, 10
NNET procedure, 10
SVMACHINE procedure, 10
TEXTMINE procedure, 10
TMSCORE procedure, 10

variable importance

FOREST procedure, 100
GRADBOOST procedure, 132

weight variable

NNET procedure, 159
Syntax Index

ACT= option
   HIDDEN statement, 151
   TARGET statement, 156
ALGORITHM= option
   OPTIMIZATION statement, 153
ANNEALINGRATE= option
   OPTIMIZATION statement, 153
ARCHITECTURE statement
   NNET procedure, 144
ASCENDING option
   TARGET statement, 177
ASSIGNMISSING= option
   PROC FOREST statement, 81
PROC GRADBOOST statement, 114
ASTORE procedure
   PROC ASTORE statement, 17
ASTORE procedure, DESCRIBE statement, 17
EPCODE= option, 18
RSTORE= option, 18
STORE= option, 18
ASTORE procedure, DOWNLOAD statement, 18
RSTORE= option, 18
STORE= option, 18
ASTORE procedure, OUTPUT statement
   OUT= option, 19
ASTORE procedure, PROC ASTORE statement, 17
DATA= option, 19
ASTORE procedure, SCORE statement, 18
ASTORE procedure, syntax, 17
ASTORE procedure, UPLOAD statement, 19
RSTORE= option, 19
STORE= option, 20
AUTOTUNE statement
   FOREST procedure, 84
   GRADBOOST procedure, 117
   NNET procedure, 145
BOOLRULE procedure, 34
   DOCINFO statement, 37
PROC BOOLRULE statement, 34
SCORE statement, 38
syntax, 34
TERMINFO statement, 39
BOOLRULE procedure, DOCINFO statement, 37
   EVENTS= option, 37
   ID= option, 37
   TARGET= option, 37
   TARGETTYPE= option, 37
BOOLRULE procedure, OUTPUT statement, 38
   CANDIDATETERMS= option, 38
   RULES= option, 38
   RULETERMS= option, 38
BOOLRULE procedure, PROC BOOLRULE statement, 34
   DATA= option, 35
   DOC= option, 35
   DOCID= option, 35
   DOCINFO= option, 35
   GNEG= option, 35
   GPOS= option, 36
   MAXCANDIDATES= option, 36
   MAXCANDS= option, 36
   MAXTRIESIN= option, 36
   MAXTRIESOUT= option, 36
   MINSUPPORTS= option, 36
   MNEG= option, 36
   MPOS= option, 36
   TERMINFO= option, 36
BOOLRULE procedure, SCORE statement, 38
   OUTMATCH= option, 38
   RULETERMS= option, 39
BOOLRULE procedure, TERMINFO statement, 39
   ID= option, 39
   LABEL= option, 39
C= option
   PROC SVMACHINE statement, 173
CANDIDATETERMS= option
   OUTPUT statement, 38
CELLWGT= option
   PARSE statement, 194
CHAID option
   GROW statement (FOREST), 89
CHISQUARE option
   GROW statement (FOREST), 89
CODE statement
   ASTORE procedure, 17
   FACTMAC procedure, 7, 64
   FOREST procedure, 7, 88
   GRADBOOST procedure, 7, 122
   NNET procedure, 7, 151
   SVMACHINE procedure, 7, 174
   TEXTMINE procedure, 7
   TMSCORE procedure, 7
COL= option
    SVD statement, 200

COMB= option
    HIDDEN statement, 152
    TARGET statement, 157

COMMENT option
    CODE statement (FACTMAC), 8
    CODE statement (FOREST), 8
    CODE statement (GRADBOOST), 8
    CODE statement (NNET), 8
    CODE statement (SVMACHINE), 8
    CODE statement (TEXTMINE), 8
    CODE statement (TMSCORE), 8

COMM_FREQ= option
    OPTIMIZATION statement, 153

CONFIG= option
    TMSCORE statement, 241

COPYVARS= option
    OUTPUT statement, 66, 91, 124, 156, 176

CROSSVALIDATION statement
    FOREST procedure, 88
    GRADBOOST procedure, 123
    NNET procedure, 151

DATA= option
    PROC ASTORE statement, 19
    PROC BOOLRULE statement, 35
    PROC FACTMAC statement, 63
    PROC FOREST statement, 82
    PROC GRADBOOST statement, 115
    PROC NNET statement, 143
    PROC SVMACHINE statement, 173
    PROC TEXTMINE statement, 192
    PROC TMSCORE statement, 240

DESCENDING option
    TARGET statement, 177

DOC= option
    PROC BOOLRULE statement, 35
    PROC TMSCORE statement, 240

DOC_ID statement
    TEXTMINE procedure, 193
    TMSCORE procedure, 242

DOC_ID= option
    PROC BOOLRULE statement, 35

DOCINFO statement
    BOOLRULE procedure, 37

DOCINFO= option
    PROC BOOLRULE statement, 35

DOWNLOAD statement
    ASTORE procedure, 18

DROPOUT_HIDDEN= option
    TRAIN statement, 158

DROPOUT_INPUT= option
    TRAIN statement, 158

ENTITIES= option
    PARSE statement, 195

ENTROPY option
    GROW statement (FOREST), 89

ENTRY= option
    SVD statement, 200

EP_CODE= option
    DESCRIBE statement, 18

ERROR= option
    TARGET statement, 157

EVENTS= option
    DOCINFO statement, 157

EXACTWEIGHT option
    SVD statement, 200

FACTMAC procedure, CODE statement, 64
    COMMENT option, 8
    FILE= option, 8
    FORMATWIDTH= option, 8
    INDENTSIZE= option, 8
    LABELID= option, 8
    LINESIZE= option, 8
    NOTRIM option, 9
    PCATALL option, 9

FACTMAC procedure, ID statement, 65

FACTMAC procedure, INPUT statement, 65

FACTMAC procedure, OUTPUT statement, 65
    COPYVARS= option, 66
    LEVEL= option, 66
    OUT= option, 65

FACTMAC procedure, PROC FACTMAC statement, 63
    DATA= option, 63
    FILE= option, 65
    LEARNSTEP= option, 64
    LEVEL= option, 65
    MAXITER= option, 64
    NFACTORS= option, 64
    NOPRINT, 64
    NTHREADS= option, 64
    NTHREADS= option, 64
    OUTMODEL, 64
    SEED= option, 64

FACTMAC procedure, SAVESTATE statement, 66
    RSTORE= option, 66

FACTMAC procedure, syntax, 63

FACTMAC procedure, TARGET statement, 66

FILE= option
    CODE statement, 151, 174
    CODE statement (FACTMAC), 8
    CODE statement (FOREST), 8
    CODE statement (GRADBOOST), 8
    CODE statement (NNET), 8
    CODE statement (SVMACHINE), 8
    CODE statement (TEXTMINE), 8
CODE statement (TMSCORE), 8
PROC FACTMAC statement, 65
FOREST procedure, AUTOTUNE statement, 84
   FRACTION= option, 84, 117, 145
   KFOLD= option, 85, 117, 145
   MAXEVALS= option, 85
   MAXITER= option, 85
   MAXTIME= option, 85
   POPSIZE= option, 85
   TUNINGPARAMETERS= option, 85
   USEPARAMETERS= option, 87
FOREST procedure, CODE statement, 88
   COMMENT option, 8
   FILE= option, 8
   FORMATWIDTH= option, 8
   INDENTSIZE= option, 8
   LABELID= option, 8
   LINESIZE= option, 8
   NOTRIM option, 9
   PCATALL option, 9
FOREST procedure, CROSSVALIDATION statement, 88
   KFOLD= option, 88
FOREST procedure, GROW statement, 89
   CHAID option, 89
   CHISQUARE option, 89
   ENTROPY option, 89
   FTEST option, 89
   GINI option, 89
   IGR option, 89
   RSS option, 89
FOREST procedure, ID statement, 90
FOREST procedure, INPUT statement, 90
   LEVEL= option, 90
FOREST procedure, OUTPUT statement, 90
   COPYV ARS= option, 91
   OUT= option, 90
FOREST procedure, PARTITION statement
   FRACTION= option, 9, 91
   ROLEVAR= option, 9, 91
FOREST procedure, PROC FOREST statement, 80
   ASSIGNMISSING= option, 81
   DATA= option, 82
   INBAGFRACTION= option, 82
   INMODEL= option, 82
   LOH= option, 82
   MAXBRANCH= option, 83
   MAXDEPTH= option, 83
   MINLEAFSIZE= option, 83
   MINUSEINSEARCH= option, 83
   NOPRINT option, 83
   NTREES= option, 83
   NUMBIN= option, 83
   OUTMODEL= option, 83
   RBAIMP option, 83
   SEED= option, 84
   VARS_TO_TRY= option, 84
   VOTE= option, 84
FOREST procedure, SAVESTATE statement, 91
   RSTORE= option, 91
FOREST procedure, syntax, 80
FOREST procedure, TARGET statement, 92
   LEVEL= option, 92
FOREST procedure, CODE statement (FACTMAC), 8
FOREST procedure, CODE statement (FOREST), 8
FOREST procedure, CODE statement (GRADBOOST), 8
FOREST procedure, CODE statement (NNET), 8
FOREST procedure, CODE statement (SVMACHINE), 8
FOREST procedure, CODE statement (TEXTMINE), 8
FOREST procedure, CODE statement (TMSCORE), 8
FOREST procedure, CODE statement, 88
   COMMENT option, 8
   FILE= option, 8
   FORMATWIDTH= option, 8
   INDENTSIZE= option, 8
   LABELID= option, 8
   LINESIZE= option, 8
   NOTRIM option, 9
   PCATALL option, 9
FOREST procedure, CROSSVALIDATION statement, 88
   KFOLD= option, 88
FOREST procedure, GROW statement, 89
   CHAID option, 89
   CHISQUARE option, 89
   ENTROPY option, 89
   FTEST option, 89
   GINI option, 89
   IGR option, 89
   RSS option, 89
FOREST procedure, ID statement, 90
FOREST procedure, INPUT statement, 90
   LEVEL= option, 90
FOREST procedure, OUTPUT statement, 90
   COPYV ARS= option, 91
   OUT= option, 90
FOREST procedure, PARTITION statement
   FRACTION= option, 9, 91
   ROLEVAR= option, 9, 91
FOREST procedure, PROC FOREST statement, 80
   ASSIGNMISSING= option, 81
   DATA= option, 82
   INBAGFRACTION= option, 82
   INMODEL= option, 82
   LOH= option, 82
   MAXBRANCH= option, 83
   MAXDEPTH= option, 83
   MINLEAFSIZE= option, 83
   MINUSEINSEARCH= option, 83
   NOPRINT option, 83
   NTREES= option, 83
   NUMBIN= option, 83
   OUTMODEL= option, 83
   RBAIMP option, 83
   SEED= option, 84
   VARS_TO_TRY= option, 84
   VOTE= option, 84
FOREST procedure, SAVESTATE statement, 91
   RSTORE= option, 91
FOREST procedure, syntax, 80
FOREST procedure, TARGET statement, 92
   LEVEL= option, 92
FOREST procedure, CROSSVALIDATION statement, 123
Syntax Index

K FOLD= option, 123
GRADBOOST procedure, ID statement, 123
GRADBOOST procedure, INPUT statement, 123
LEVEL= option, 124
GRADBOOST procedure, OUTPUT statement, 124
COPYV ARS= option, 124
OUT= option, 124
GRADBOOST procedure, PARTITION statement
FRACTION option, 9, 125
ROLEVAR= option, 9, 125
GRADBOOST procedure, PROC GRADBOOST statement, 113
ASSIGNMISSING= option, 114
DATA= option, 115
INMODEL= option, 115
LASSO= option, 115
LEARNINGRATE= option, 115
MAXBRANCH= option, 116
MAXDEPTH= option, 116
MINLEAFSIZE= option, 116
MINUSEINSEARCH= option, 116
NOPRINT option, 116
NTREES= option, 116
NUMBIN= option, 116
OUTMODEL option, 116
RIDGE= option, 116
SAMPLINGRATE option, 117
SEED= option, 117
V ARS_TO_TRY= option, 117
GRADBOOST procedure, SAVESTATE statement, 125
RSTORE= option, 125
GRADBOOST procedure, syntax, 113
GRADBOOST procedure, TARGET statement, 125
LEVEL= option, 125
GROUP= option
SELECT statement, 199
GROW statement
FOREST procedure, 89
HIDDEN statement
NNET procedure, 151
ID statement
FACTMAC procedure, 65
FOREST procedure, 90
GRADBOOST procedure, 123
SVMACHINE procedure, 174
ID= option
DOCINFO statement, 37
TERMINFO statement, 39
IGNORE option
SELECT statement, 199
IGR option
GROW statement (FOREST), 89
IN_TERMS= option
SVD statement, 200
INBAGFRACTION= option
PROC FOREST statement, 82
INDENTSIZE= option
PROC FOREST statement, 82
PROC GRADBOOST statement, 115
PROC NNET statement, 144
INPUT statement
FACTMAC procedure, 65
FOREST procedure, 90
GRADBOOST procedure, 123
NNET procedure, 152
SVMACHINE procedure, 175
K= option
SVD statement, 201
KEEP option
SELECT statement, 199
KEEPV ARS, KEEPV ARIABLES
SVD statement, 201
KERNEL statement
SVMACHINE procedure, 175
K FOLD= option
AUTOTUNE statement, 85, 117, 145
CROSSVALIDATION statement, 88, 123, 151
LABEL= option
TERMINFO statement, 39
LABELID= option
PROC TEXTMINE statement, 193
LANGUAGE= option
PROC TEXTMINE statement, 193
LASSO= option
PROC GRADBOOST statement, 115
LEARNINGRATE= option
PROC GRADBOOST statement, 115
LEARNINGRATE=option
  OPTIMIZATION statement, 153
LEARNSTEP= option
  PROC FACTMAC statement, 64
LEVEL= option
  INPUT statement, 90, 124, 152, 175
  OUTPUT statement, 66
  PROC FACTMAC statement, 65
  TARGET statement, 92, 125, 156
LINEAR option
  PROC SVMACHINE statement, 175
LINESIZE= option
  CODE statement (FACTMAC), 8
  CODE statement (FOREST), 8
  CODE statement (GRADBOOST), 8
  CODE statement (NNET), 8
  CODE statement (SVMACHINE), 8
  CODE statement (TEXTMINE), 8
  CODE statement (TMSCORE), 8
LOH= option
  PROC FOREST statement, 82
MAX_K= option
  SVD statement, 201
MAXBRANCH= option
  PROC FOREST statement, 83
  PROC GRADBOOST statement, 116
MAXCANDIDATES= option
  PROC BOOLRULE statement, 36
MAXCANDS= option
  PROC BOOLRULE statement, 36
MAXDEPTH= option
  PROC FOREST statement, 83
  PROC GRADBOOST statement, 116
MAXEVALS= option
  AUTOTUNE statement, 85, 118, 146
MAXITER= option
  AUTOTUNE statement, 85, 118, 146
  OPTIMIZATION statement, 154
  PROC FACTMAC statement, 64
  PROC SVMACHINE statement, 173
MAXTIME= option
  AUTOTUNE statement, 85, 118, 146
  OPTIMIZATION statement, 154
MAXTRIESIN= option
  PROC BOOLRULE statement, 36
MAXTRIESOUT= option
  PROC BOOLRULE statement, 36
MINIBATCHSIZE= option
  OPTIMIZATION statement, 154
MINLEAFSIZE= option
  PROC FOREST statement, 83
  PROC GRADBOOST statement, 116
MINSUPPORTS= option
  PROC BOOLRULE statement, 36
MINUSEINSEARCH= option
  PROC FOREST statement, 83
  PROC GRADBOOST statement, 116
MISSING= option
  PROC NNET statement, 144
MLP DIRECT option
  ARCHITECTURE statement, 145
MLP option
  ARCHITECTURE statement, 145
MNEG= option
  PROC BOOLRULE statement, 36
MOMENTUM= option
  OPTIMIZATION statement, 154
MPOS= option
  PROC BOOLRULE statement, 36
MULTITITERM= option
  PARSE statement, 195
NEWVARNAMES
  TEXTMINE statement, 193
NFACTORS= option
  PROC FACTMAC statement, 64
NNET procedure, ARCHITECTURE statement, 144
  GLIM option, 145
  MLP DIRECT option, 145
  MLP option, 145
NNET procedure, AUTOTUNE statement, 145
  MAXEVALS= option, 146
  MAXITER= option, 146
  MAXTIME= option, 146
  POPSIZE= option, 146
  TUNINGPARAMETERS= option, 146
  USEPARAMETERS= option, 150
NNET procedure, CODE statement, 151
  COMMENT option, 8
  FILE= option, 8, 151
  FORMATWIDTH= option, 8
  INDENTSIZE= option, 8
  LABELID= option, 8
  LINESIZE= option, 8
  NOCOMPPGM option, 151
  NOTRIM option, 9
  PCATALL option, 9
NNET procedure, CROSSVALIDATION statement, 151
  KFOLD= option, 151
NNET procedure, HIDDEN statement, 151
  ACT= option, 151
  COMB= option, 152
NNET procedure, INPUT statement, 152
  LEVEL= option, 152
NNET procedure, OPTIMIZATION statement, 152
  ALGORITHM= option, 153
ANNEALINGRATE= option, 153
COMMFFREQ= option, 153
LEARNINGRATE= option, 153
MAXITER= option, 154
MAXTIME= option, 154
MINIBATCHSIZE= option, 154
MOMENTUM= option, 154
REGL1= option, 154
REGL2= option, 154
SEED= option, 154
USELOCKING option, 154
NNET procedure, OUTPUT statement
  COPYVARS= option, 156
  OUT= option, 155
NNET procedure, PARTITION statement, 154
  FRACTION option, 9, 155
  ROLEVAR= option, 9, 155
NNET procedure, PROC NNET statement, 143
  DATA= option, 143
  INMODEL= option, 144
  MISSING= option, 144
  NTHREADS= option, 144
  STANDARDIZE= option, 144
NNET procedure, SCORE statement, 155
NNET procedure, syntax, 143
NNET procedure, TARGET statement, 156
  ACT= option, 156
  COMB= option, 157
  ERROR= option, 157
  LEVEL= option, 156
NNET procedure, TRAIN statement, 157
  DROPOUTHIDDEN= ratio, 158
  DROPOUTINPUT= option, 158
  NUMTRIES= option, 158
  OUTMODEL= option, 158
  RESUME option, 158
  VALIDATION= option, 158
  WSEED= option, 158
NNET procedure, WEIGHT statement, 159
NOCOMPPGM option
  CODE statement, 151
NOCUTOFFS option
  SVD statement, 201
NONG option
  PARSE statement, 195
NONOUNGROUPS option
  PARSE statement, 195
NOPRINT
  PROC FACTMAC statement, 64
NOPRINT option
  PROC FOREST statement, 83
  PROC GRADBOOST statement, 116
  PROC SVMACHINE statement, 174
NOSCALE option
  PROC SVMACHINE statement, 174
NOSTEMMING option
  PARSE statement, 195
NOTAGGING option
  PARSE statement, 195
NOTRIM option
  CODE statement (FACTMAC), 9
  CODE statement (FOREST), 9
  CODE statement (GRADBOOST), 9
  CODE statement (NNET), 9
  CODE statement (SVMACHINE), 9
  CODE statement (TEXTMINE), 9
  CODE statement (TMSCORE), 9
NTHREADS= option
  PROC FACTMAC statement, 64
  PROC NNET statement, 144
  PROC SVMACHINE statement, 174
  PROC TEXTMINE statement, 193
NTREES= option
  PROC FOREST statement, 83
  PROC GRADBOOST statement, 116
NUMBIN= option
  PROC FOREST statement, 83
  PROC GRADBOOST statement, 116
NUMLABELS= option
  SVD statement, 201
NUMTRIES= option
  TRAIN statement, 158
OPTIMIZATION statement
  NNET procedure, 152
OUT= option
  OUTPUT statement (ASTORE), 19
  OUTPUT statement (FACTMAC), 65
  OUTPUT statement (FOREST), 90
  OUTPUT statement (GRADBOOST), 124
  OUTPUT statement (NNET), 155
  OUTPUT statement (SVMACHINE), 176
OUTCHILD= option
  PARSE statement, 196
OUTCONFIG= option
  PARSE statement, 196
OUTDOCPRO= option
  SVD statement, 201
OUTMATCH= option
  SCORE statement, 38
OUTMODEL
  PROC FACTMAC statement, 64
  OUTMODEL option
  PROC GRADBOOST statement, 116
  OUTMODEL= option
  PROC FOREST statement, 83
  TRAIN statement, 158
OUTPARENT= option
PARSE statement, 196
PROC TMSCORE statement, 241
OUTPOS= option
    PARSE statement, 196
OUTPUT statement
    BOOLRULE procedure, 38
    FACTMAC procedure, 65
    FOREST procedure, 90
    GRADBOOST procedure, 124
    SVMACHINE procedure, 176
OUTTERMS= option
    PARSE statement, 196
OUTTOPICS= option
    SVD statement, 202
PARSE statement
    TEXTMINE procedure, 193
PARTITION statement
    FACTMAC procedure, 9
    FOREST procedure, 9, 91
    GRADBOOST procedure, 9, 124
    NNET procedure, 9, 154
    SVMACHINE procedure, 9
    TEXTMINE procedure, 9
    TMSCORE procedure, 9
PCATALL option
    CODE statement (FACTMAC), 9
    CODE statement (FOREST), 9
    CODE statement (GRADBOOST), 9
    CODE statement (NNET), 9
    CODE statement (SVMACHINE), 9
    CODE statement (TEXTMINE), 9
    CODE statement (TMSCORE), 9
POLYNOMIAL option
    PROC SVMACHINE statement, 175
POPSIZE= option
    AUTOTUNE statement, 85, 118, 146
PROC ASTORE statement
    ASTORE procedure, 17
PROC BOOLRULE statement
    BOOLRULE procedure, 34
PROC FACTMAC statement, see FACTMAC procedure
PROC FOREST statement, see FOREST procedure
PROC GRADBOOST statement, see GRADBOOST procedure
PROC NNET statement, see NNET procedure
PROC SVMACHINE statement, see SVMACHINE procedure
PROC TEXTMINE statement
    TEXTMINE procedure, 192
PROC TMSCORE statement
    TMSCORE procedure, 240
RBAIMP option
    PROC FOREST statement, 84
PROC FOREST statement, 83
REDUCEF= option
    PARSE statement, 197
REGL1= option
    OPTIMIZATION statement, 154
REGL2= option
    OPTIMIZATION statement, 154
RES= option
    SVD statement, 202
RESOLUTION= option
    SVD statement, 202
RESUME option
    TRAIN statement, 158
RIDGE= option
    PROC GRADBOOST statement, 116
ROLEVAR= option
    PARTITION statement, 155
    PARTITION statement (FOREST), 9, 91
    PARTITION statement (GRADBOOST), 9, 125
    PARTITION statement (NNET), 9
ROTATION= option
    SVD statement, 202
ROW= option
    SVD statement, 202
RSS option
    GROW statement (FOREST), 89
RSTORE= option
    DOWNLOAD statement, 18
    SAVESTATE statement, 66, 91, 125, 176, 198
    UPLOAD statement, 19
RSTORE= option
    DESCRIIBE statement, 18
RULES= option
    OUTPUT statement, 38
RULETERMS= option
    OUTPUT statement, 38
    SCORE statement, 39
SAMPLINGRATE option
    PROC GRADBOOST statement, 117
SAVESTATE statement
    FACTMAC procedure, 66
    FOREST procedure, 91
    GRADBOOST procedure, 125
    SVMACHINE procedure, 176, 198
SCALE option
    PROC SVMACHINE statement, 174
SCORE statement
    ASTORE procedure, 18
    BOOLRULE procedure, 38
    NNET procedure, 155
SEED= option
    PROC FACTMAC statement, 64
    PROC FOREST statement, 84
PROC GRADBOOST statement, 117
SEED=option
   OPTIMIZATION statement, 154
SELECT statement
   TEXTMINE procedure, 198
SHOWDROPPEDTERMS= option
   PARSE statement, 197
STANDARDIZE= option
   PROC NNET statement, 144
START= option
   PARSE statement, 197
STOP= option
   PARSE statement, 197
STORE= option
   DESCRIBE statement, 18
   DOWNLOAD statement, 18
   UPLOAD statement, 20
SVD statement
   TEXTMINE procedure, 199
SVDDOCPRO= option
   PROC TMSCORE statement, 241
SVDS= option
   SVD statement, 203
SVDU= option
   PROC TMSCORE statement, 241
   SVD statement, 203
SVDV= option
   SVD statement, 203
SVMACHINE procedure, CODE statement, 174
   COMMENT option, 8
   FILE= option, 8, 174
   FORMATWIDTH= option, 8
   INDENTSIZE= option, 8
   LABELID= option, 8
   LINESIZE= option, 8
   NOTRIM option, 9
   PCATALL option, 9
SVMACHINE procedure, ID statement, 174
SVMACHINE procedure, INPUT statement, 175
   LEVEL= option, 175
SVMACHINE procedure, KERNEL statement, 175
SVMACHINE procedure, OUTPUT statement, 176
   COPYVARS= option, 176
   OUT= option, 176
SVMACHINE procedure, PROC SVMACHINE statement, 173
   C= option, 173
   DATA= option, 173
   LINEAR option, 175
   MAXITER= option, 173
   NOPRINT, 174
   NOSCALE, 174
   NTHREADS= option, 174
   POLYNOMIAL option, 175
   SCALE option, 174
   TOLERANCE option, 174
   USEMISS option, 174
SVMACHINE procedure, SAVESTATE statement, 176, 198
   RSTORE= option, 176, 198
SVMACHINE procedure, syntax, 173
SVMACHINE procedure, TARGET statement, 176
   ASCENDING option, 177
   DESCENDING option, 177
SYNONYM= option
   PARSE statement, 197
syntax
   BOOLRULE procedure, 34
   TEXTMINE procedure, 192
   TMSCORE procedure, 239
TARGET statement
   FACTMAC procedure, 66
   FOREST procedure, 92
   GRADBOOST procedure, 125
   NNET procedure, 156
   SVMACHINE procedure, 176
   TEXTMINE procedure, 203
   DOCINFO statement, 37
TARGETTYPE= option
   DOCINFO statement, 37
TERMID= option
   PROC BOOLRULE statement, 36
TERMINFO statement
   BOOLRULE procedure, 39
TERMINFO= option
   PROC BOOLRULE statement, 36
TERMS= option
   PROC TMSCORE statement, 241
TERMWGT= option
   PARSE statement, 197
TEXTMINE procedure, 192
   PARSE statement, 193
   PROC TEXTMINE statement, 192
   SELECT statement, 198
   SVD statement, 199
   syntax, 192
TEXTMINE procedure, CODE statement
   COMMENT option, 8
   FILE= option, 8
   FORMATWIDTH= option, 8
   INDENTSIZE= option, 8
   LABELID= option, 8
   LINESIZE= option, 8
   NOTRIM option, 9
   PCATALL option, 9
TEXTMINE procedure, DOC_ID statement, 193
TEXTMINE procedure, PARSE statement, 193
  CELLWGT= option, 194
  ENTITIES= option, 195
  MULTITERM= option, 195
  NONG option, 195
  NONOUNGROUPS option, 195
  NOSTEMMING option, 195
  NOTAGGING option, 195
  OUTCHILD= option, 196
  OUTCONFIG= option, 196
  OUTPARENT= option, 196
  OUTPOS= option, 196
  OUTTERMS= option, 196
  REDUCEF= option, 197
  SHOWDROPPEDTERMS= option, 197
  START= option, 197
  STOP= option, 197
  SYNONYM= option, 197
  TERMWGT= option, 197

TEXTMINE procedure, PROC TEXTMINE statement, 192
  DATA= option, 192
  LANGUAGE= option, 193
  NEWVARNAMES, 193
  NTHREADS= option, 193

TEXTMINE procedure, SELECT statement, 198
  GROUP= option, 199
  IGNORE option, 199
  KEEP option, 199
  LABELS option, 199

TEXTMINE procedure, SVD statement, 199
  COL= option, 200
  ENTRY= option, 200
  EXACTWEIGHT option, 200
  IN_TERMS= option, 200
  K= option, 201
  KEEPVARS, KEEPVARIABLES, 201
  MAX_K= option, 201
  NOCUTOFFS option, 201
  NUMLABELS= option, 201
  OUTDOCPRO= option, 201
  OUTTOPICS= option, 202
  RES= option, 202
  RESOLUTION= option, 202
  ROTATION= option, 202
  ROW= option, 202
  SVDS= option, 203
  SVDDOCPRO= option, 201
  SVDU= option, 241
  TOL= option

TEXTMINE procedure, TARGET statement, 203

TMSCORE procedure, 239
  PROC TMSCORE statement, 240
  syntax, 239

TMSCORE procedure, CODE statement
  COMMENT option, 8
  FILE= option, 8
  FORMATWIDTH= option, 8
  INDENTSIZE= option, 8
  LABELID= option, 8
  LINESIZE= option, 8
  NOTRIM option, 9
  PCATALL option, 9

TMSCORE procedure, DOC_ID statement, 242

TMSCORE procedure, PROC TMSCORE statement, 240
  DATA= option, 240
  DOC= option, 240
  OUTPARENT= option, 241
  SVDDOCPRO= option, 241
  SVDU= option, 241
  TERMS= option, 241

TMSCORE procedure, TMSCORE statement
  CONFIG= option, 241

TMSCORE procedure, VAR statement, 242

TMSCORE procedure, VARIABLES statement, 242
  TOL= option

SVD statement, 203
  SVD statement, 203
  TOLERANCE option
  PROC SVMACHINE statement, 174
  TRAIN statement
  NNET procedure, 157
  TUNINGPARAMETERS= option
  AUTOTUNE statement, 85, 118, 146

UPLOAD statement
  ASTORE procedure, 19
  USELOCKING option
  OPTIMIZATION statement, 154
  USEMISS option
  PROC SVMACHINE statement, 174
  USEPARAMETERS= option
  AUTOTUNE statement, 87, 122, 150

VALIDATION= option
  TRAIN statement, 158

VAR statement
  TEXTMINE procedure, 203
  TMSCORE procedure, 242

VARIABLES statement
  TEXTMINE procedure, 203
  TMSCORE procedure, 242

VARS_TO_TRY= option
  PROC FOREST statement, 84
  PROC GRADBOOST statement, 117

VOTE= option
PROC FOREST statement, 84

WEIGHT statement
   NNET procedure, 159
WSEED= option
   TRAIN statement, 158
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.