SAS® Text Miner 14.1
High-Performance Procedures

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2015. *SAS® Text Miner 14.1: High-Performance Procedures*. Cary, NC: SAS Institute Inc.

SAS® Text Miner 14.1: High-Performance Procedures

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

July 2015

Contents

Credits and Acknowledgments

Credits

Documentation

Editing	Anne Baxter, Ed Huddleston
Documentation Support	Tim Arnold, Melanie Gratton

Software

The SAS high-Performance text mining procedures were implemented by the following members of the development staff. Program development includes design, programming, debugging, support, and documentation. In the following list, the names of the developers who currently provide primary support are listed first; other developers and previous developers are also listed.

HPTMINE	Zheng (Alan) Zhao, Russell Albright, David Bultman, Joshua Griffin, James Cox
HPTMSCORE	Zheng (Alan) Zhao, Russell Albright, David Bultman, Joshua Griffin, James Cox
High-performance computing foundation	Steve E. Krueger
High-performance analytics foundation	Robert Cohen, Georges H. Guirguis, Trevor Kearney, Richard Knight, Gang Meng, Oliver Schabenberger, Charles Shorb, Tom P. Weber
Numerical routines	Georges H. Guirguis

The following people contribute to the SAS high-Performance text mining procedures with their leadership and support: Saratendu Sethi, Bob Johnson.

Testing

Leslie Anderson, Tim Carter, Enzo D'Andreti, Alex Fang, Angela Jones, Paul Kovach, Jim McKenzie, Jim Metcalf, Kelly Miller, Laura Rock, Kim Stott, Carol Thompson.

Internationalization Testing

Alex Chai, Mi-Na Chu, Jacky Dong, Feng Gao, Masayuki Iizuka, David Li, Lan Luan, Haiyong Rong, Bin Sun, Frank Wang, Lina Xu, Catherine Yang.

Technical Support

Craig DeVault, Ann Kou.

Acknowledgments

Many people make significant and continuing contributions to the development of SAS software products.

The final responsibility for the SAS System lies with SAS alone. We hope that you will always let us know your opinions about the SAS System and its documentation. It is through your participation that SAS software is continuously improved.

Please visit us at `http://www.sas.com/software/high-performance-computing/`.

Chapter 1
What's New in SAS Text Miner 14.1 High-Performance Procedures

Contents

Overview

The HPBOOLRULE procedure has been added, and the HPTMINE procedure has been enhanced.

HPBOOLRULE Procedure

The HPBOOLRULE procedure is a high-performance procedure that enables you to extract Boolean rules from large-scale transactional data. PROC HPBOOLRULE adds essential capability to high-performance text mining for supervised rule based modeling. In the current release, you can use the HPBOOLRULE procedure to read data and extract rules only in single-machine mode.

The HPBOOLRULE procedure can automatically generate a set of Boolean rules by analyzing a text corpus that has been processed by the HPTMINE procedure and represented in a transactional format.

Enhancements to the HPTMINE Procedure

The HPTMINE procedure supports the following new languages, statement, and options:

- You can parse text data in the following newly supported languages: Chinese, Dutch, Finnish, French, Italian, Japanese, Korean, Portuguese, Russian, Spanish, and Turkish.
- You can use the HPTMINE procedure to generate a search index for a text corpus. The index can be input to the TMUTIL procedure for querying the text corpus.
- You can use the new SELECT statement to specify the parts of speech, entities, or attributes that you want to include in or exclude from your analysis.
- You can specify a terms table when running the HPTMINE procedure in SVD-only mode. The terms table is required by topic discovery.

Chapter 2
Introduction

Contents

Overview of the SAS Text Miner High-Performance procedures

The SAS Text Miner high-performance procedures provide text mining tools that have been specially developed to take advantage of parallel processing in both multithreaded single-machine mode and distributed multiple machine mode. The SAS Text Miner high-performance procedures provide full-spectrum support for text mining, including document parsing, term weighting and filtering, term-by-document matrix creation, dimensionality reduction via singular value de-composition (SVD), and scoring.

In addition to the procedures described in this book, SAS Text Miner includes high-performance utility procedures, which are described in *Base SAS Procedures Guide: High-Performance Procedures*. You can run all of these procedures in single-machine mode without licensing SAS High-Performance Text Miner. However, to run these procedures in distributed mode, you must license SAS High-Performance Text Miner.

Installation and Configuration

Before you can use the SAS Text Miner high-performance procedures, you need to install and configure the SAS High-Performance Analytics infrastrucure. For more information, see the *SAS High-Performance Analytics Infrastructure: Installation and Configuration Guide*.

Additional installation and configuration is required for SAS Text Miner high-performance procedures. For more information, see the section “System Configuration” on page 101 in Chapter 5, “The HPTMINE Procedure.”

About This Book

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

Chapter Organization

This book is organized as follows.

Chapter 2, this chapter, provides an overview of SAS Text Miner high-performance procedures.

Chapter 3, "Shared Concepts and Topics," describes the modes in which SAS Text Miner high-performance procedures can execute.

Subsequent chapters describe the SAS Text Miner high-performance procedures. These chapters appear in alphabetical order by procedure name. Each chapter is organized as follows:

- The "Overview" section provides a brief description of 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 miscellaneous details, such as ODS tables.
- The "Examples" section contains examples that use the procedure.
- The "References" section contains references for the methodology and for examples of the procedure.

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 the 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 for user-supplied values for options in the syntax definitions and in text.
VariableName	is used for the names of variables and data sets when they appear in the text.
bold	is used to refer to matrices and vectors.

italic	is used for terms that are defined in the 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:

```
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:

```
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.

Online Documentation

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

SAS Technical Support Services

As with all SAS products, the SAS Technical Support staff is available to respond to problems and answer technical questions regarding the use of the SAS Text Miner high-performance procedures. Go to `http://support.sas.com/techsup` for more information.

Chapter 3
Shared Concepts and Topics

Contents

Overview

This chapter describes the modes of execution in which SAS high-performance analytical procedures can execute. If you have SAS Text Miner installed, you can run any procedure in this book on a single machine.

However, to run procedures in this book in distributed mode, you must also have SAS High-Performance Text Miner software installed. For more information about these modes, see the next section.

This chapter provides details of how you can control the modes of execution and includes the syntax for the PERFORMANCE statement, which is common to all high-performance analytical procedures.

Processing Modes

Single-Machine Mode

Single-machine mode is a computing model in which multiple processors or multiple cores are controlled by a single operating system and can access shared resources, such as disks and memory. In this book, single-machine mode refers to an application running multiple concurrent threads on a multicore machine in order to take advantage of parallel execution on multiple processing units. More simply, single-machine mode for high-performance analytical procedures means multithreading on the client machine.

All high-performance analytical procedures are capable of running in single-machine mode, and this is the default mode when a procedure runs on the client machine. The procedure uses the number of CPUs (cores) on the machine to determine the number of concurrent threads. High-performance analytical procedures use different methods to map core count to the number of concurrent threads, depending on the analytic task. Using one thread per core is not uncommon for the procedures that implement data-parallel algorithms.

Distributed Mode

Distributed mode is a computing model in which several nodes in a distributed computing environment participate in the calculations. In this book, the distributed mode of a high-performance analytical procedure refers to the procedure performing the analytics on an appliance that consists of a cluster of nodes. This appliance can be one of the following:

- a database management system (DBMS) appliance on which the SAS High-Performance Analytics infrastructure is also installed
- a cluster of nodes that have the SAS High-Performance Analytics infrastructure installed but no DBMS software installed

Controlling the Execution Mode with Environment Variables and Performance Statement Options

You control the execution mode by using environment variables or by specifying options in the PERFORMANCE statement in high-performance analytical procedures, or by a combination of these methods.

The important environment variables follow:

- *grid host* identifies the domain name system (DNS) or IP address of the appliance node to which the SAS High-Performance Text Miner software connects to run in distributed mode.
- *installation location* identifies the directory where the SAS High-Performance Text Miner software is installed on the appliance.

You can set an environment variable directly from the SAS program by using the OPTION SET= command. For example, the following statements define the grid host and the location where the SAS High-Performance software is installed on the appliance:

```
option set=GRIDHOST      ="hpa.sas.com";
option set=GRIDINSTALLLOC="/opt/TKGrid";
```

Alternatively, you can set the parameters in the PERFORMANCE statement in high-performance analytical procedures. For example:

```
performance host       ="hpa.sas.com"
            install    ="/opt/TKGrid";
```

A specification in the PERFORMANCE statement overrides a specification of an environment variable without resetting its value. An environment variable that you set in the SAS session by using an OPTION SET= command remains in effect until it is modified or until the SAS session terminates.

The key variable that determines whether a high-performance analytical procedure executes in single-machine or distributed mode is the *grid host*. The installation location is needed to ensure that a connection to the grid host can be made, given that a host is specified. This book assumes that the installation location has been set by your system administrator.

The following sets of SAS statements are functionally equivalent:

```
proc hpreduce;
   reduce unsupervised x:;
   performance host="hpa.sas.com";
run;

option set=GRIDHOST="hpa.sas.com";
proc hpreduce;
   reduce unsupervised x:;
run;
```

Determining Single-Machine Mode or Distributed Mode

High-performance analytical procedures use the following rules to determine whether they run in single-machine mode or distributed mode:

- If a grid host is not specified, the analysis is carried out in single-machine mode on the client machine that runs the SAS session.

- If a grid host is specified, the behavior depends on whether the execution is alongside the database or alongside HDFS. If the data are local to the client (that is, not stored in the distributed database or HDFS on the appliance), you need to use the NODES= option in the PERFORMANCE statement to specify the number of nodes on the appliance or cluster that you want to engage in the analysis. If the procedure executes alongside the database or alongside HDFS, you do not need to specify the NODES= option.

The following example shows single-machine and client-data distributed configurations for a data set of 100,000 observations that are simulated from a logistic regression model. The following DATA step generates the data:

```
data simData;
   array _a{8} _temporary_ (0,0,0,1,0,1,1,1);
   array _b{8} _temporary_ (0,0,1,0,1,0,1,1);
   array _c{8} _temporary_ (0,1,0,0,1,1,0,1);
   do obsno=1 to 100000;
      x  = rantbl(1,0.28,0.18,0.14,0.14,0.03,0.09,0.08,0.06);
      a  = _a{x};
      b  = _b{x};
      c  = _c{x};
      x1 = int(ranuni(1)*400);
      x2 = 52 + ranuni(1)*38;
      x3 = ranuni(1)*12;
      lp = 6. -0.015*(1-a) + 0.7*(1-b) + 0.6*(1-c) + 0.02*x1 -0.05*x2 - 0.1*x3;
      y  = ranbin(1,1,(1/(1+exp(lp))));
      output;
   end;
   drop x lp;
run;
```

The following statements run PROC HPLOGISTIC to fit a logistic regression model:

```
proc hplogistic data=simData;
   class a b c;
   model y = a b c x1 x2 x3;
run;
```

Figure 3.1 shows the results from the analysis.

Figure 3.1 Results from Logistic Regression in Single-Machine Mode

The HPLOGISTIC Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Data Access Information

Data	Engine	Role	Path
WORK.SIMDATA	V9	Input	On Client

Figure 3.1 *continued*

Model Information	
Data Source	WORK.SIMDATA
Response Variable	y
Class Parameterization	GLM
Distribution	Binary
Link Function	Logit
Optimization Technique	Newton-Raphson with Ridging

Parameter Estimates

Parameter	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	5.7011	0.2539	Infty	22.45	<.0001
a 0	-0.01020	0.06627	Infty	-0.15	0.8777
a 1	0	.	.	.	.
b 0	0.7124	0.06558	Infty	10.86	<.0001
b 1	0	.	.	.	.
c 0	0.8036	0.06456	Infty	12.45	<.0001
c 1	0	.	.	.	.
x1	0.01975	0.000614	Infty	32.15	<.0001
x2	-0.04728	0.003098	Infty	-15.26	<.0001
x3	-0.1017	0.009470	Infty	-10.74	<.0001

The entries in the “Performance Information” table show that the HPLOGISTIC procedure runs in single-machine mode and uses four threads, which are chosen according to the number of CPUs on the client machine. You can force a certain number of threads on any machine that is involved in the computations by specifying the NTHREADS option in the PERFORMANCE statement. Another indication of execution on the client is the following message, which is issued in the SAS log by all high-performance analytical procedures:

```
NOTE: The HPLOGISTIC procedure is executing in single-machine mode.
```

The following statements use 10 nodes (in distributed mode) to analyze the data on the appliance; results appear in Figure 3.2:

```
proc hplogistic data=simData;
   class a b c;
   model y = a b c x1 x2 x3;
   performance host="hpa.sas.com" nodes=10;
run;
```

Figure 3.2 Results from Logistic Regression in Distributed Mode

The HPLOGISTIC Procedure

Performance Information	
Host Node	hpa.sas.com
Execution Mode	Distributed
Number of Compute Nodes	10
Number of Threads per Node	24

Data Access Information

Data	Engine	Role	Path
WORK.SIMDATA	V9	Input	From Client

Model Information	
Data Source	WORK.SIMDATA
Response Variable	y
Class Parameterization	GLM
Distribution	Binary
Link Function	Logit
Optimization Technique	Newton-Raphson with Ridging

Parameter Estimates

Parameter	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	5.7011	0.2539	Infty	22.45	<.0001
a 0	-0.01020	0.06627	Infty	-0.15	0.8777
a 1	0	.	.	.	.
b 0	0.7124	0.06558	Infty	10.86	<.0001
b 1	0	.	.	.	.
c 0	0.8036	0.06456	Infty	12.45	<.0001
c 1	0	.	.	.	.
x1	0.01975	0.000614	Infty	32.15	<.0001
x2	-0.04728	0.003098	Infty	-15.26	<.0001
x3	-0.1017	0.009470	Infty	-10.74	<.0001

The specification of a host causes the "Performance Information" table to display the name of the host node of the appliance. The "Performance Information" table also indicates that the calculations were performed in a distributed environment on the appliance. Twenty-four threads on each of 10 nodes were used to perform the calculations—for a total of 240 threads.

Another indication of distributed execution on the appliance is the following message, which is issued in the SAS log by all high-performance analytical procedures:

```
NOTE: The HPLOGISTIC procedure is executing in the distributed
      computing environment with 10 worker nodes.
```

You can override the presence of a grid host and force the computations into single-machine mode by specifying the NODES=0 option in the PERFORMANCE statement:

```
proc hplogistic data=simData;
   class a b c;
   model y = a b c x1 x2 x3;
   performance host="hpa.sas.com" nodes=0;
run;
```

Figure 3.3 shows the "Performance Information" table. The numeric results are not reproduced here, but they agree with the previous analyses, which are shown in Figure 3.1 and Figure 3.2.

Figure 3.3 Single-Machine Mode Despite Host Specification

The HPLOGISTIC Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Data Access Information

Data	Engine	Role	Path
WORK.SIMDATA	V9	Input	On Client

The "Performance Information" table indicates that the HPLOGISTIC procedure executes in single-machine mode on the client. This information is also reported in the following message, which is issued in the SAS log:

```
NOTE: The HPLOGISTIC procedure is executing in single-machine mode.
```

In the analysis shown previously in Figure 3.2, the data set Work.simData is local to the client, and the HPLOGISTIC procedure distributed the data to 10 nodes on the appliance. The High-Performance Analytics infrastructure does not keep these data on the appliance. When the procedure terminates, the in-memory representation of the input data on the appliance is freed.

When the input data set is large, the time that is spent sending client-side data to the appliance might dominate the execution time. In practice, transfer speeds are usually lower than the theoretical limits of the network connection or disk I/O rates. At a transfer rate of 40 megabytes per second, sending a 10-gigabyte data set to the appliance requires more than four minutes. If analytic execution time is in the range of seconds, the "performance" of the process is dominated by data movement.

The alongside-the-database execution model, unique to high-performance analytical procedures, enables you to read and write data in distributed form from the database that is installed on the appliance.

Data Access Modes

Single-Machine Data Access Mode

When high-performance analytical procedures run in single-machine mode, they access data in the same way as traditional SAS procedures. They use Base SAS to access input and output SAS data sets on the client machine, and they use the relevant SAS/ACCESS interface to bring data from other sources, such as third-party databases, Hadoop, and SAS LASR servers, to the client.

Distributed Data Access Mode

When high-performance analytical procedures run in distributed mode, input data must be brought to the computation that is performed on the nodes of the grid, and output data must be sent from the computational nodes. This can be accomplished in several ways:

- Client-data (local-data) mode: The input and output data for the analytic task are stored on the client machine where the high-performance procedure is invoked. When the procedure runs, the SAS High-Performance Analytics infrastructure sends input data from the client to the distributed computing environment and sends output data from the distributed computing environment to the client.
- Parallel symmetric mode: Input and output data are stored on the same nodes that are used for the distributed computation, and the data move in parallel from the data store to the computational nodes without crossing node boundaries. Parallel symmetric mode is available with the following distributed data sources:
  - Data in Greenplum databases that are collocated with the computational nodes. This access mode is also called alongside-the-database mode. For more information, see the section "Alongside-the-Database Execution" on page 16.
  - Data in SASHDAT format in the Hadoop Distributed File System (HDFS) that is collocated with the computational nodes. This access mode is also called alongside-HDFS mode. For more information, see the section "Alongside-HDFS Execution by Using the SASHDAT Engine" on page 25.
  - Data in a SAS LASR Analytic Server that is collocated with the computational nodes. This access mode is also called alongside-LASR mode. For more information, see the section "Running High-Performance Analytical Procedures Alongside a SAS LASR Analytic Server in Distributed Mode" on page 19.
- Parallel asymmetric mode: The primary reason for providing this mode is to enable you to manage and house data on appliances (the data appliances) and to run high-performance analytical procedures on a different appliance (the computing appliance). The high-performance analytical procedures run in a SAS process on the computing appliance. For each data source that is accessed in parallel asymmetric mode, a SAS Embedded Process must run on the associated data appliance. Data are requested by a SAS data feeder that runs on the computing appliance and communicates with the SAS Embedded Process on the data appliance. The SAS Embedded Process transfers the data in parallel to the SAS data feeder that runs on each of the nodes of the computing appliance. This mode is called asymmetric mode because the number of nodes on the data appliance does not need to match the number of nodes on the computing appliance. Parallel asymmetric mode is supported for data in Teradata, Greenplum, and Oracle databases and for data in HDFS and SAP HANA. In these cases, the parallel asymmetric access is somewhat loosely described as being asymmetric alongside access, even though the data storage and computation can occur on different appliances. For more information, see the section "Running High-Performance Analytical Procedures in Asymmetric Mode" on page 22.

- Through-the-client mode: When data can be accessed through a SAS/ACCESS interface but the data reside in a file system or in a distributed data source on which a SAS Embedded Process is not running, those data cannot be accessed in parallel in either symmetric or asymmetric mode. The SAS/ACCESS interface is used to transfer input data from the data source to the client machine on which the high-performance procedure is invoked, and the data are then sent to the distributed computing environment by the SAS High-Performance Analytics infrastructure. The data path is reversed for output data. This mode of data access is referred to as through-the-client access.

Determining the Data Access Mode

High-performance analytical procedures determine the data access mode individually for each data set that is used in the analysis. When high-performance analytical procedures run in distributed mode, parallel symmetric or parallel asymmetric mode is used whenever possible. There are two reasons why parallel access might not be possible. The first reason is that for a particular data set, the required SAS Embedded Process is not installed on the appliance that houses the data. In such cases, access to those data reverts to through-the-client access, and a note like the following is reported in the SAS log:

```
NOTE: The data MYLIB.MYDATA are being routed through the client because a
      SAS Embedded Process is not running on the associated data server.
```

The second reason why parallel data access might not be possible for a particular data set is that the required driver software might not be installed on the compute nodes. In this case, the required data feeder that moves the data from the compute nodes to the data source cannot be successfully loaded, and a note like the following is reported in the SAS log:

```
NOTE: The data MYLIB.MYDATA are being routed through the client because
      the ORACLE data feeder could not be loaded on the specified grid host.
```

For distributed data in SASHDAT format in HDFS or data in a SAS LASR Analytic Server, parallel symmetric access is used when the data nodes and compute nodes are collocated on the same appliance. For data in a LASR Analytic Server that cannot be accessed in parallel symmetric mode, through-the-client mode is used. Through-the-client access is not supported for data in SASHDAT format in HDFS.

For data in Greenplum databases, parallel symmetric access is used if the compute nodes and the data nodes are collocated on the same appliance and you do not specify the NODES=*n* option in a PERFORMANCE statement. In this case, the number of nodes that are used is determined by the number of nodes across which the data are distributed. If you specify NODES=*n*, then parallel asymmetric access is used.

High-performance analytical procedures produce a "Data Access Information" table that shows you how each data set that is used in the analysis is accessed. The following statements provide an example in which PROC HPDS2 is used to copy a distributed data set named Neuralgia (which is stored in SASHDAT format in HDFS) to a SAS data set on the client machine:

```
libname hdatlib sashdat
                host='hpa.sas.com';
                hdfs_path="/user/hps";

proc hpds2 data=hdatlib.neuralgia out=neuralgia;
```

```
      performance host='hpa.sas.com';
      data DS2GTF.out;
         method run();
        set DS2GTF.in;
      end;
     enddata;
   run;
```

Figure 3.4 shows the output that PROC HPDS2 produces. The “Performance Information” table shows that PROC HPDS2 ran in distributed mode on a 13-node grid. The “Data Access Information” table shows that the input data were accessed in parallel symmetric mode and the output data set was sent to the client, where the V9 (base) engine stored it as a SAS data set in the Work directory.

Figure 3.4 Performance Information and Data Access Information Tables

The HPDS2 Procedure

Performance Information	
Host Node	hpa.sas.com
Execution Mode	Distributed
Number of Compute Nodes	12
Number of Threads per Node	24

Data Access Information

Data	Engine	Role	Path
HDATLIB.NEURALGIA	SASHDAT	Input	Parallel, Symmetric
WORK.NEURALGIA	V9	Output	To Client

Alongside-the-Database Execution

High-performance analytical procedures interface with the distributed database management system (DBMS) on the appliance in a unique way. If the input data are stored in the DBMS and the grid host is the appliance that houses the data, high-performance analytical procedures create a distributed computing environment in which an analytic process is collocated with the nodes of the DBMS. Data then pass from the DBMS to the analytic process on each node. Instead of moving across the network and possibly back to the client machine, the data pass locally between the processes on each node of the appliance.

Because the analytic processes on the appliance are separate from the database processes, the technique is referred to as alongside-the-database execution in contrast to in-database execution, where the analytic code executes in the database process.

In general, when you have a large amount of input data, you can achieve the best performance from high-performance analytical procedures if execution is alongside the database.

Before you can run alongside the database, you must distribute the data to the appliance. The following statements use the HPDS2 procedure to distribute the data set Work.simData into the mydb database on the hpa.sas.com appliance. In this example, the appliance houses a Greenplum database.

```
option set=GRIDHOST="green.sas.com";
libname applianc greenplm
        server  ="green.sas.com"
        user    =XXXXXX
        password=YYYYY
        database=mydb;


option set=GRIDHOST="compute_appliance.sas.com";

proc datasets lib=applianc nolist; delete simData;
proc hpds2 data=simData
           out =applianc.simData(distributed_by='distributed randomly');
  performance commit=10000 nodes=all;
  data DS2GTF.out;
     method run();
        set DS2GTF.in;
     end;
  enddata;
run;
```

If the output table applianc.simData exists, the DATASETS procedure removes the table from the Greenplum database because a DBMS does not usually support replacement operations on tables.

Note that the libref for the output table points to the appliance. The data set option informs the HPDS2 procedure to distribute the records randomly among the data segments of the appliance. The statements that follow the PERFORMANCE statement are the DS2 program that copies the input data to the output data without further transformations.

Because you loaded the data into a database on the appliance, you can use the following HPLOGISTIC statements to perform the analysis on the appliance in the alongside-the-database mode. These statements are almost identical to the first PROC HPLOGISTIC example in a previous section, which executed in single-machine mode.

```
proc hplogistic data=applianc.simData;
   class a b c;
   model y = a b c x1 x2 x3;
run;
```

The subtle differences are as follows:

- The grid host environment variable that you specified in an OPTION SET= command is still in effect.
- The DATA= option in the high-performance analytical procedure uses a libref that identifies the data source as being housed on the appliance. This libref was specified in a prior LIBNAME statement.

Figure 3.5 shows the results from this analysis. The “Performance Information” table shows that the execution was in distributed mode, and the “Data Access Information” table shows that the data were accessed asymmetrically in parallel from the Greenplum database. The numeric results agree with the previous analyses, which are shown in Figure 3.1 and Figure 3.2.

Figure 3.5 Alongside-the-Database Execution on Greenplum

The HPLOGISTIC Procedure

Performance Information	
Host Node	compute_appliance.sas.com
Execution Mode	Distributed
Number of Compute Nodes	141
Number of Threads per Node	32

Data Access Information

Data	Engine	Role	Path
APPLIANC.SIMDATA	GREENPLM	Input	Parallel, Asymmetric

Model Information	
Data Source	APPLIANC.SIMDATA
Response Variable	y
Class Parameterization	GLM
Distribution	Binary
Link Function	Logit
Optimization Technique	Newton-Raphson with Ridging

Parameter Estimates

Parameter	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	5.7011	0.2539	Infty	22.45	<.0001
a 0	-0.01020	0.06627	Infty	-0.15	0.8777
a 1	0	.	.	.	.
b 0	0.7124	0.06558	Infty	10.86	<.0001
b 1	0	.	.	.	.
c 0	0.8036	0.06456	Infty	12.45	<.0001
c 1	0	.	.	.	.
x1	0.01975	0.000614	Infty	32.15	<.0001
x2	-0.04728	0.003098	Infty	-15.26	<.0001
x3	-0.1017	0.009470	Infty	-10.74	<.0001

Alongside-LASR Distributed Execution

You can execute high-performance analytical procedures in distributed mode alongside a SAS LASR Analytic Server. When high-performance analytical procedures run in this mode, the data are preloaded in distributed form in memory that is managed by a LASR Analytic Server. The data on the nodes of the appliance are accessed in parallel in the process that runs the LASR Analytic Server, and they are transferred to the process where the high-performance analytical procedure runs. In general, each high-performance analytical procedure copies the data to memory that persists only while that procedure executes. Hence, when a high-performance analytical procedure runs alongside a LASR Analytic Server, both the high-performance analytical procedure and the LASR Analytic Server have a copy of the subset of the data that is used by the high-performance analytical procedure. The advantage of running high-performance analytical procedures alongside a LASR Analytic Server (as opposed to running alongside a DBMS table or alongside HDFS) is that the initial transfer of data from the LASR Analytic Server to the high-performance analytical procedure is a memory-to-memory operation that is faster than the disk-to-memory operation when the procedure runs alongside a DBMS or HDFS. When the cost of preloading a table into a LASR Analytic Server is amortized by multiple uses of these data in separate runs of high-performance analytical procedures, using the LASR Analytic Server can result in improved performance.

Running High-Performance Analytical Procedures Alongside a SAS LASR Analytic Server in Distributed Mode

This section provides an example of steps that you can use to start and load data into a SAS LASR Analytic Server instance and then run high-performance analytical procedures alongside this LASR Analytic Server instance.

Starting a SAS LASR Analytic Server Instance

The following statements create a SAS LASR Analytic Server instance and load it with the simData data set that is used in the preceding examples. The data that are loaded into the LASR Analytic Server persist in memory across procedure boundaries until these data are explicitly deleted or until the server instance is terminated.

```
proc lasr port=54545
          data=simData
          path="/tmp/";
   performance host="hpa.sas.com" nodes=ALL;
run;
```

The PORT= option specifies a network port number to use. The PATH= option specifies the directory in which the server and table signature files are to be stored. The specified directory must exist on each machine in the cluster. The DATA= option specifies the name of a data set that is loaded into this LASR Analytic Server instance. (You do not need to specify the DATA= option at this time because you can add tables to the LASR Analytic Server instance at any stage of its life.) For more information about starting and using a LASR Analytic Server, see the *SAS LASR Analytic Server: Reference Guide*.

The NODES=ALL option in the PERFORMANCE statement specifies that the LASR Analytic Server run on all the nodes on the appliance. You can start a LASR Analytic Server on a subset of the nodes on an appliance, but this might affect whether high-performance analytical procedures can run alongside the LASR Analytic Server. For more information, see the section "Alongside-LASR Distributed Execution on a Subset of the Appliance Nodes" on page 21.

Figure 3.6 shows the "Performance Information" and "Data Access Information" tables, which show that the LASR procedure ran in distributed mode on 13 nodes and that the data were sent from the client to the appliance.

Figure 3.6 Performance and Data Access Information

The LASR Procedure

Performance Information	
Host Node	hpa.sas.com
Execution Mode	Distributed
Number of Compute Nodes	12

Data Access Information

Data	Engine	Role	Path
WORK.SIMDATA	V9	Input	From Client

Associating a SAS Libref with the SAS LASR Analytic Server Instance

The following statements use a LIBNAME statement that associates a SAS libref (named MyLasr) with tables on the server instance as follows:

```
libname MyLasr sasiola port=54545 host="hpa.sas.com";
```

The SASIOLA option requests that the MyLasr libref use the SASIOLA engine, and the PORT= value associates this libref with the appropriate server instance. For more information about creating a libref that uses the SASIOLA engine, see the *SAS LASR Analytic Server: Reference Guide*.

Running a High-Performance Analytical Procedure Alongside the SAS LASR Analytic Server Instance

You can use the MyLasr libref to specify the input data for high-performance analytical procedures. You can also create output data sets in the SAS LASR Analytic Server instance by using this libref to request that the output data set be held in memory by the server instance as follows:

```
proc hplogistic data=MyLasr.simData;
   class a b c;
   model y = a b c x1 x2 x3;
   output out=MyLasr.simulateScores pred=PredictedProbabliity;
run;
```

Because you previously specified the GRIDHOST= environment variable and the input data are held in distributed form in the associated server instance, this PROC HPLOGISTIC step runs in distributed mode alongside the LASR Analytic Server, as indicated in the “Performance Information” table shown in Figure 3.7.

Figure 3.7 Performance and Data Access Information

The HPLOGISTIC Procedure

Performance Information	
Host Node	hpa.sas.com
Execution Mode	Distributed
Number of Compute Nodes	12
Number of Threads per Node	24

Data Access Information

Data	Engine	Role	Path
MYLASR.SIMDATA	SASIOLA	Input	Parallel, Symmetric
MYLASR.SIMULATESCORES	SASIOLA	Output	Parallel, Symmetric

The "Data Access Information" table shows that both the input and output data were read and written, respectively, in parallel symmetric mode.

The preceding OUTPUT statement creates an output table that is added to the LASR Analytic Server instance. Output data sets do not have to be created in the same server instance that holds the input data. You can use a different LASR Analytic Server instance to hold the output data set. However, in order for the output data to be created in parallel symmetric mode, all the nodes that are used by the server instance that holds the input data must also be used by the server instance that holds the output data.

Terminating a SAS LASR Analytic Server Instance

You can continue to run high-performance analytical procedures and add and delete tables from the SAS LASR Analytic Server instance until you terminate the server instance as follows:

```
proc lasr term port=54545;
run;
```

Alongside-LASR Distributed Execution on a Subset of the Appliance Nodes

When you run PROC LASR to start a SAS LASR Analytic Server, you can specify the NODES= option in a PERFORMANCE statement to control how many nodes the LASR Analytic Server executes on. Similarly, a high-performance analytical procedure can execute on a subset of the nodes either because you specify the NODES= option in a PERFORMANCE statement or because you run alongside a DBMS or HDFS with an input data set that is distributed on a subset of the nodes on an appliance. In such situations, if a high-performance analytical procedure uses nodes on which the LASR Analytic Server is not running, then running alongside LASR is not supported. You can avoid this issue by specifying the NODES=ALL in the PERFORMANCE statement when you use PROC LASR to start the LASR Analytic Server.

Running High-Performance Analytical Procedures in Asymmetric Mode

This section provides examples of how you can run high-performance analytical procedures in asymmetric mode.

Asymmetric mode is commonly used when the data appliance and the computing appliance are distinct appliances. In order to be able to use an appliance as a data provider for high-performance analytical procedures that run in asymmetric mode on another appliance, it is not necessary that SAS High-Performance Text Miner be installed on the data appliance. However, it is essential that a SAS Embedded Process be installed on the data appliance and that SAS High-Performance Text Miner be installed on the computing appliance.

The following examples use a 24-node data appliance named "data_appliance.sas.com," which houses a Teradata DBMS and has a SAS Embedded Process installed.

The following statements load the simData data set of the preceding sections onto the data appliance:

```
libname dataLib teradata
        server  ="tera2650"
        user    =XXXXXX
        password=YYYYY
        database=mydb;

data dataLib.simData;
   set simData;
run;
```

NOTE: You can provision the appliance with data even if SAS High-Performance Text Miner software is not installed on the appliance.

The following subsections show how you can run the HPLOGISTIC procedure asymmetrically on distinct data and computing appliances.

Running in Asymmetric Mode on Distinct Appliances

Usually, there is no advantage to executing high-performance analytical procedures in asymmetric mode on one appliance, because data might have to be unnecessarily moved between nodes. The following example demonstrates the more typical use of asymmetric mode. In this example, the specified grid host "compute_appliance.sas.com" is a 142-node computing appliance that is different from the 24-node data appliance "data_appliance.sas.com," which houses the Teradata DBMS where the data reside.

The advantage of using different computing and data appliances is that the data appliance is not affected by the execution of high-performance analytical procedures except during the initial parallel data transfer. A potential disadvantage of this asymmetric mode of execution is that the performance can be limited by the bandwidth with which data can be moved between the appliances. However, because this data movement takes place in parallel from the nodes of the data appliance to the nodes of the computing appliance, this potential performance bottleneck can be overcome with appropriately provisioned hardware. The following statements show how this is done:

```
proc hplogistic data=dataLib.simData;
   class a b c;
   model y = a b c x1 x2 x3;
   performance host = "compute_appliance.sas.com" nodes=30;
run;
```

Figure 3.8 shows the “Performance Information” and “Data Access Information” tables.

Figure 3.8 Asymmetric Mode with Distinct Data and Computing Appliances

The HPLOGISTIC Procedure

Performance Information	
Host Node	compute_appliance.sas.com
Execution Mode	Distributed
Number of Compute Nodes	30
Number of Threads per Node	32

Data Access Information

Data	Engine	Role	Path
DATALIB.simData	TERADATA	Input	Parallel, Asymmetric

PROC HPLOGISTIC ran on 30 nodes of the computing appliance, even though the data were partitioned across the 24 nodes of the data appliance. The numeric results are not reproduced here, but they agree with the previous analyses shown in Figure 3.1 and Figure 3.2.

Every time you run a high-performance analytical procedure in asymmetric mode that uses different computing and data appliances, data are transferred between these appliances. If you plan to make repeated use of the same data, then it might be advantageous to temporarily persist the data that you need on the computing appliance. One way to persist the data is to store them as a table in a SAS LASR Analytic Server that runs on the computing appliance. By running PROC LASR in asymmetric mode, you can load the data in parallel from the data appliance nodes to the nodes on which the LASR Analytic Server runs on the computing appliance. You can then use a LIBNAME statement that associates a SAS libref with tables on the LASR Analytic Server. The following statements show how you do this:

```
proc lasr port=54345
          data=dataLib.simData
          path="/tmp/";
   performance host ="compute_appliance.sas.com" nodes=30;
run;

libname MyLasr sasiola tag="dataLib" port=54345 host="compute_appliance.sas.com" ;
```

Figure 3.9 show the “Performance Information” and “Data Access Information” tables.

Figure 3.9 PROC LASR Running in Asymmetric Mode

The LASR Procedure

Performance Information	
Host Node	compute_appliance.sas.com
Execution Mode	Distributed
Number of Compute Nodes	30

Data Access Information			
Data	**Engine**	**Role**	**Path**
DATALIB.simData	TERADATA	Input	Parallel, Asymmetric

By default, all the nodes on the computing appliance would be used. However, because NODES=30 was specified in the PERFORMANCE statement, PROC LASR ran on only 30 nodes of the computing appliance. The data were loaded asymmetrically in parallel from the 24 data appliance nodes to the 30 compute nodes on which PROC LASR ran.

After the data are loaded into a LASR Analytic Server that runs on the computing appliance, you can run high-performance analytical procedures alongside this LASR Analytic Server as shown by the following statements:

```
proc hplogistic data=MyLasr.simData;
   class a b c;
   model y = a b c x1 x2 x3;
   output out=MyLasr.myOutputData pred=myPred;
   performance host = "compute_appliance.sas.com";
run;
```

The following note, which appears in the SAS log, confirms that the output data set is created successfully:

```
NOTE: The table DATALIB.MYOUTPUTDATA has been added to the LASR Analytic Server
      with port 54345. The Libname is MYLASR.
```

You can use the dataLib libref that you used to load the data onto the data appliance to create an output data set on the data appliance.

```
proc hplogistic data=MyLasr.simData;
   class a b c;
   model y = a b c x1 x2 x3;
   output out=dataLib.myOutputData pred=myPred;
   performance host  = "compute_appliance.sas.com";
run;
```

The following note, which appears in the SAS log, confirms that the output data set is created successfully on the data appliance:

```
NOTE: The data set DATALIB.myOutputData has 100000 observations and 1 variables.
```

When you run a high-performance analytical procedure on a computing appliance and either read data from or write data to a different data appliance on which a SAS Embedded Process is running, the Read and Write operations take place in parallel without any movement of data to and from the SAS client.

When you no longer need the data in the SAS LASR Analytic Server, you should terminate the server instance as follows:

```
proc lasr term port=54345;
    performance host="compute_appliance.sas.com";
run;
```

If you configured Hadoop on the computing appliance, then you can create output data tables that are stored in the HDFS on the computing appliance. You can do this by using the SASHDAT engine as described in the section "Alongside-HDFS Execution" on page 25.

Alongside-HDFS Execution

Running high-performance analytical procedures alongside HDFS shares many features with running alongside the database. You can execute high-performance analytical procedures alongside HDFS by using either the SASHDAT engine or the Hadoop engine.

You use the SASHDAT engine to read and write data that are stored in HDFS in a proprietary SASHDAT format. In SASHDAT format, metadata that describe the data in the Hadoop files are included with the data. This enables you to access files in SASHDAT format without supplying any additional metadata. Additionally, you can also use the SASHDAT engine to read data in CSV (comma-separated value) format, but you need supply metadata that describe the contents of the CSV data. The SASHDAT engine provides highly optimized access to data in HDFS that are stored in SASHDAT format.

The Hadoop engine reads data that are stored in various formats from HDFS and writes data to HDFS in CSV format. This engine can use metadata that are stored in Hive, which is a data warehouse that supplies metadata about data that are stored in Hadoop files. In addition, this engine can use metadata that you create by using the HDMD procedure.

The following subsections provide details about using the SASHDAT and Hadoop engines to execute high-performance analytical procedures alongside HDFS.

Alongside-HDFS Execution by Using the SASHDAT Engine

If the grid host is a cluster that houses data that have been distributed by using the SASHDAT engine, then high-performance analytical procedures can analyze those data in the alongside-HDFS mode. The procedures use the distributed computing environment in which an analytic process is collocated with the nodes of the cluster. Data then pass from HDFS to the analytic process on each node of the cluster.

Before you can run a procedure alongside HDFS, you must distribute the data to the cluster. The following statements use the SASHDAT engine to distribute to HDFS the simData data set that was used in the previous two sections:

```
option set=GRIDHOST="hpa.sas.com";

libname hdatLib sashdat
        path="/hps";

 data hdatLib.simData (replace = yes) ;
     set simData;
 run;
```

In this example, the GRIDHOST is a cluster where the SAS Data in HDFS Engine is installed. If a data set that is named simData already exists in the hps directory in HDFS, it is overwritten because the REPLACE=YES data set option is specified. For more information about using this LIBNAME statement, see the section “LIBNAME Statement for the SAS Data in HDFS Engine” in the *SAS LASR Analytic Server: Reference Guide*.

The following HPLOGISTIC procedure statements perform the analysis in alongside-HDFS mode. These statements are almost identical to the PROC HPLOGISTIC example in the previous two sections, which executed in single-machine mode and alongside-the-database distributed mode, respectively.

Figure 3.10 shows the “Performance Information” and “Data Access Information” tables. You see that the procedure ran in distributed mode and that the input data were read in parallel symmetric mode. The numeric results shown in Figure 3.11 agree with the previous analyses shown in Figure 3.1, Figure 3.2, and Figure 3.5.

Figure 3.10 Alongside-HDFS Execution Performance Information

The HPLOGISTIC Procedure

Performance Information	
Host Node	hpa.sas.com
Execution Mode	Distributed
Number of Compute Nodes	12
Number of Threads per Node	24

Data Access Information

Data	Engine	Role	Path
HDATLIB.SIMDATA	SASHDAT	Input	Parallel, Symmetric

Figure 3.11 Alongside-HDFS Execution Model Information

Model Information	
Data Source	HDATLIB.SIMDATA
Response Variable	y
Class Parameterization	GLM
Distribution	Binary
Link Function	Logit
Optimization Technique	Newton-Raphson with Ridging

Figure 3.11 *continued*

Parameter Estimates

Parameter	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	5.7011	0.2539	Infty	22.45	<.0001
a 0	-0.01020	0.06627	Infty	-0.15	0.8777
a 1	0	.	.	.	.
b 0	0.7124	0.06558	Infty	10.86	<.0001
b 1	0	.	.	.	.
c 0	0.8036	0.06456	Infty	12.45	<.0001
c 1	0	.	.	.	.
x1	0.01975	0.000614	Infty	32.15	<.0001
x2	-0.04728	0.003098	Infty	-15.26	<.0001
x3	-0.1017	0.009470	Infty	-10.74	<.0001

Alongside-HDFS Execution by Using the Hadoop Engine

The following LIBNAME statement sets up a libref that you can use to access data that are stored in HDFS and have metadata in Hive:

```
libname hdoopLib hadoop
        server       = "hpa.sas.com"
        user         = XXXXX
        password     = YYYYY
        database     = myDB
        config       = "demo.xml" ;
```

For more information about LIBNAME options available for the Hadoop engine, see the LIBNAME topic in the Hadoop section of *SAS/ACCESS for Relational Databases: Reference*. The configuration file that you specify in the CONFIG= option contains information that is needed to access the Hive server. It also contains information that enables this configuration file to be used to access data in HDFS without using the Hive server. This information can also be used to specify replication factors and block sizes that are used when the engine writes data to HDFS.

The following DATA step uses the Hadoop engine to distribute to HDFS the simData data set that was used in the previous sections. The engine creates metadata for the data set in Hive.

```
data hdoopLib.simData;
    set simData;
run;
```

After you have loaded data or if you are accessing preexisting data in HDFS that have metadata in Hive, you can access this data alongside HDFS by using high-performance analytical procedures. The following HPLOGISTIC procedure statements perform the analysis in alongside-HDFS mode. These statements are similar to the PROC HPLOGISTIC example in the previous sections.

```
proc hplogistic data=hdoopLib.simData;
   class a b c;
   model y = a b c x1 x2 x3;
   performance host = "compute_appliance.sas.com";
run;
```

Figure 3.12 shows the "Performance Information" and "Data Access Information" tables. You see that the procedure ran in distributed mode and that the input data were read in parallel asymmetric mode. The numeric results shown in Figure 3.13 agree with the previous analyses.

Figure 3.12 Alongside-HDFS Execution by Using the Hadoop Engine

The HPLOGISTIC Procedure

Performance Information	
Host Node	compute_appliance.sas.com
Execution Mode	Distributed
Number of Compute Nodes	141
Number of Threads per Node	32

Data Access Information

Data	Engine	Role	Path
GRIDLIB.SIMDATA	HADOOP	Input	Parallel, Asymmetric

Figure 3.13 Alongside-HDFS Execution by Using the Hadoop Engine

Model Information	
Data Source	GRIDLIB.SIMDATA
Response Variable	y
Class Parameterization	GLM
Distribution	Binary
Link Function	Logit
Optimization Technique	Newton-Raphson with Ridging

Parameter Estimates

Parameter	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	5.7011	0.2539	Infty	22.45	<.0001
a 0	-0.01020	0.06627	Infty	-0.15	0.8777
a 1	0	.	.	.	.
b 0	0.7124	0.06558	Infty	10.86	<.0001
b 1	0	.	.	.	.
c 0	0.8036	0.06456	Infty	12.45	<.0001
c 1	0	.	.	.	.
x1	0.01975	0.000614	Infty	32.15	<.0001
x2	-0.04728	0.003098	Infty	-15.26	<.0001
x3	-0.1017	0.009470	Infty	-10.74	<.0001

The Hadoop engine also enables you to access tables in HDFS that are stored in various formats and that are not registered in Hive. You can use the HDMD procedure to generate metadata for tables that are stored in the following file formats:

- delimited text
- fixed-record length binary
- sequence files
- XML text

To read any other kind of file in Hadoop, you can write a custom file reader plug-in in Java for use with PROC HDMD. For more information about LIBNAME options available for the Hadoop engine, see the LIBNAME topic in the Hadoop section of *SAS/ACCESS for Relational Databases: Reference*.

The following example shows how you can use PROC HDMD to register metadata for CSV data independently from Hive and then analyze these data by using high-performance analytical procedures. The CSV data in the table csvExample.csv is stored in HDFS in the directory /user/demo/data. Each record in this table consists of the following fields, in the order shown and separated by commas.

1. a string of at most six characters
2. a numeric field with values of 0 or 1
3. a numeric field with real numbers

Suppose you want to fit a logistic regression model to these data, where the second field represents a target variable named Success, the third field represents a regressor named Dose, and the first field represents a classification variable named Group.

The first step is to use PROC HDMD to create metadata that are needed to interpret the table, as in the following statements:

```
libname hdoopLib hadoop
                 server       = "hpa.sas.com"
                 user         = XXXXX
                 password     = YYYYY
                 HDFS_PERMDIR = "/user/demo/data"
                 HDFS_METADIR = "/user/demo/meta"
                 config       = "demo.xml"
                 DBCREATE_TABLE_EXTERNAL=YES;

proc hdmd name=hdoopLib.csvExample data_file='csvExample.csv'
          format=delimited encoding=utf8 sep = ',';

     column Group    char(6);
     column Success  double;
     column Dose     double;
run;
```

The metadata that are created by PROC HDMD for this table are stored in the directory /user/demo/meta that you specified in the HDFS_METADIR = option in the preceding LIBNAME statement. After you create the metadata, you can execute high-performance analytical procedures with these data by using the hdoopLib libref. For example, the following statements fit a logistic regression model to the CSV data that are stored in csvExample.csv table.

```
proc hplogistic data=hdoopLib.csvExample;
    class Group;
    model Success = Dose;
    performance host     = "compute_appliance.sas.com"
                gridmode = asym;
run;
```

Figure 3.14 shows the results of this analysis. You see that the procedure ran in distributed mode and that the input data were read in parallel asymmetric mode. The metadata that you created by using the HDMD procedure have been used successfully in executing this analysis.

Figure 3.14 Alongside-HDFS Execution with CSV Data

The HPLOGISTIC Procedure

Performance Information	
Host Node	compute_appliance.sas.com
Execution Mode	Distributed
Number of Compute Nodes	141
Number of Threads per Node	32

Data Access Information

Data	Engine	Role	Path
GRIDLIB.CSVEXAMPLE	HADOOP	Input	Parallel, Asymmetric

Model Information	
Data Source	GRIDLIB.CSVEXAMPLE
Response Variable	Success
Class Parameterization	GLM
Distribution	Binary
Link Function	Logit
Optimization Technique	Newton-Raphson with Ridging

Class Level Information

Class	Levels	Values
Group	3	group1 group2 group3

Number of Observations Read	1000
Number of Observations Used	1000

Parameter Estimates

Parameter	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	0.1243	0.1295	Infty	0.96	0.3371
Dose	-0.2674	0.2216	Infty	-1.21	0.2277

Output Data Sets

In the alongside-the-database mode, the data are read in distributed form, minimizing data movement for best performance. Similarly, when you write output data sets and a high-performance analytical procedure executes in distributed mode, the data can be written in parallel into the database.

For example, in the following statements, the HPLOGISTIC procedure executes in distributed mode by using eight nodes on the appliance to perform the logistic regression on work.simData:

```
proc hplogistic data=simData;
   class a b c;
   model y = a b c x1 x2 x3;
   id a;
   output out=applianc.simData_out pred=p;
   performance host="hpa.sas.com" nodes=8;
run;
```

The output data set applianc.simData_out is written in parallel into the database. Although the data are fed on eight nodes, the database might distribute the data on more nodes.

When a high-performance analytical procedure executes in single-machine mode, all output objects are created on the client. If the libref of the output data sets points to the appliance, the data are transferred to the database on the appliance. This can lead to considerable performance degradation compared to execution in distributed mode.

Many procedures in SAS software add the variables from the input data set when an observationwise output data set is created. The assumption of high-performance analytical procedures is that the input data sets can be large and contain many variables. For performance reasons, the output data set contains the following:

- variables that are explicitly created by the statement
- variables that are listed in the ID statement, as described in Chapter 4, “Shared Statistical Concepts” (*SAS/STAT User’s Guide: High-Performance Procedures*)
- distribution keys or hash keys that are transferred from the input data set

Including this information enables you to add to the output data set information necessary for subsequent SQL joins without copying the entire input data set to the output data set.

Working with Formats

You can use SAS formats and user-defined formats with high-performance analytical procedures as you can with other procedures in the SAS System. However, because the analytic work is carried out in a distributed environment and might depend on the formatted values of variables, some special handling can improve the efficiency of work with formats.

High-performance analytical procedures examine the variables that are used in an analysis for association with user-defined formats. Any user-defined formats that are found by a procedure are transmitted automatically to the appliance. If you are running multiple high-performance analytical procedures in a SAS session and the analysis variables depend on user-defined formats, you can preprocess the formats. This step involves generating an XML stream (a file) of the formats and passing the stream to the high-performance analytical procedures.

Suppose that the following formats are defined in your SAS program:

```
proc format;
     value YesNo        1='Yes'        0='No';
     value checkThis    1='ThisisOne'  2='ThisisTwo';
     value $cityChar    1='Portage'    2='Kinston';
run;
```

The next group of SAS statements create the XML stream for the formats in the file *Myfmt.xml*, associate that file with the file reference myxml, and pass the file reference with the FMTLIBXML= option in the PROC HPLOGISTIC statement:

```
filename myxml 'Myfmt.xml';
libname  myxml XML92 xmltype=sasfmt tagset=tagsets.XMLsuv;
proc format cntlout=myxml.allfmts;
run;


proc hplogistic data=six fmtlibxml=myxml;
   class wheeze cit age;
   format wheeze best4. cit $cityChar.;
   model wheeze = cit age;
run;
```

Generation and destruction of the stream can be wrapped in convenience macros:

```
%macro Make_XMLStream(name=tempxml);
    filename &name 'fmt.xml';
    libname  &name XML92 xmltype=sasfmt tagset=tagsets.XMLsuv;
    proc format cntlout=&name..allfmts;
    run;
%mend;

%macro Delete_XMLStream(fref);
    %let rc=%sysfunc(fdelete(&fref));
%mend;
```

If you do not pass an XML stream to a high-performance analytical procedure that supports the FMTLIBXML= option, the procedure generates an XML stream as needed when it is invoked.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables that describe the distributed computing environment, and requests detailed results about the performance characteristics of a high-performance analytical procedure.

You can also use the PERFORMANCE statement to control whether a high-performance analytical procedure executes in single-machine or distributed mode.

You can specify the following *performance-options* in the PERFORMANCE statement:

COMMIT=*n*

requests that the high-performance analytical procedure write periodic updates to the SAS log when observations are sent from the client to the appliance for distributed processing.

High-performance analytical procedures do not have to use input data that are stored on the appliance. You can perform distributed computations regardless of the origin or format of the input data, provided that the data are in a format that can be read by the SAS System (for example, because a SAS/ACCESS engine is available).

In the following example, the HPREG procedure performs LASSO variable selection where the input data set is stored on the client:

```
proc hpreg data=work.one;
   model y = x1-x500;
   selection method=lasso;
   performance nodes=10 host='mydca' commit=10000;
run;
```

In order to perform the work as requested using 10 nodes on the appliance, the data set Work.One needs to be distributed to the appliance.

High-performance analytical procedures send the data in blocks to the appliance. Whenever the number of observations sent exceeds an integer multiple of the COMMIT= size, a SAS log message is produced. The message indicates the actual number of observations distributed, and not an integer multiple of the COMMIT= size.

DETAILS

requests a table that shows a timing breakdown of the procedure steps.

GRIDHOST=*“name”*

HOST=*“name”*

specifies the name of the appliance host in single or double quotation marks. If this option is specified, it overrides the value of the GRIDHOST environment variable.

GRIDMODE=SYM | ASYM

MODE=SYM | ASYM

is a deprecated option that specifies whether to run the high-performance analytical procedure in symmetric (SYM) mode or asymmetric (ASYM) mode. This option overrides the GRIDMODE environment variable.

GRIDTIMEOUT=*s*

TIMEOUT=*s*

specifies the time-out in seconds for a high-performance analytical procedure to wait for a connection to the appliance and establish a connection back to the client. The default is 120 seconds. If jobs are submitted to the appliance through workload management tools that might suspend access to the appliance for a longer period, you might want to increase the time-out value.

INSTALL=*"name"*

INSTALLLOC=*"name"*

specifies the directory in which the shared libraries for the high-performance analytical procedure are installed on the appliance. Specifying the INSTALL= option overrides the GRIDINSTALLLOC environment variable.

LASRSERVER=*"path"*

LASR=*"path"*

specifies the fully qualified path to the description file of a SAS LASR Analytic Server instance. If the input data set is held in memory by this LASR Analytic Server instance, then the procedure runs alongside LASR. This option is not needed to run alongside LASR if the DATA= specification of the input data uses a libref that is associated with a LASR Analytic Server instance. For more information, see the section "Alongside-LASR Distributed Execution" on page 18 and the *SAS LASR Analytic Server: Reference Guide*.

NODES=ALL | *n*

NNODES=ALL | *n*

specifies the number of nodes in the distributed computing environment, provided that the data are not processed alongside the database.

Specifying NODES=0 indicates that you want to process the data in single-machine mode on the client machine. If the input data are not alongside the database, this is the default. The high-performance analytical procedures then perform the analysis on the client. For example, the following sets of statements are equivalent:

```
proc hplogistic data=one;
   model y = x;
run;
```

```
proc hplogistic data=one;
   model y = x;
   performance nodes=0;
run;
```

If the data are not read alongside the database, the NODES= option specifies the number of nodes on the appliance that are involved in the analysis. For example, the following statements perform the analysis in distributed mode by using 10 units of work on the appliance that is identified in the HOST= option:

```
proc hplogistic data=one;
   model y = x;
   performance nodes=10 host="hpa.sas.com";
run;
```

If the number of nodes can be modified by the application, you can specify a NODES=*n* option, where *n* exceeds the number of physical nodes on the appliance. The SAS High-Performance Text Miner software then *oversubscribes* the nodes and associates nodes with multiple units of work. For example, on a system that has 16 appliance nodes, the following statements oversubscribe the system by a factor of 3:

```
proc hplogistic data=one;
   model y = x;
   performance nodes=48 host="hpa.sas.com";
run;
```

Usually, it is not advisable to oversubscribe the system because the analytic code is optimized for a certain level of multithreading on the nodes that depends on the CPU count. You can specify NODES=ALL if you want to use all available nodes on the appliance without oversubscribing the system.

If the data are read alongside the distributed database on the appliance, specifying a nonzero value for the NODES= option has no effect. The number of units of work in the distributed computing environment is then determined by the distribution of the data and cannot be altered. For example, if you are running alongside an appliance with 24 nodes, the NODES= option in the following statements is ignored:

```
libname GPLib greenplm server=gpdca user=XXX password=YYY
              database=ZZZ;
proc hplogistic data=gplib.one;
   model y = x;
   performance nodes=10 host="hpa.sas.com";
run;
```

NTHREADS=*n*

THREADS=*n*

specifies the number of threads for analytic computations and overrides the SAS system option THREADS | NOTHREADS. If you do not specify the NTHREADS= option, the number of threads is determined based on the number of CPUs on the host on which the analytic computations execute. The algorithm by which a CPU count is converted to a thread count is specific to the high-performance analytical procedure. Most procedures create one thread per CPU for the analytic computations.

By default, high-performance analytical procedures run in multiple concurrent threads unless multithreading has been turned off by the NOTHREADS system option or you force single-threaded execution by specifying NTHREADS=1. The largest number that can be specified for *n* is 256. Individual high-performance analytical procedures can impose more stringent limits if called for by algorithmic considerations.

NOTE: The SAS system options THREADS | NOTHREADS apply to the client machine on which the SAS high-performance analytical procedures execute. They do not apply to the compute nodes in a distributed environment.

Chapter 4
The HPBOOLRULE Procedure

Contents

Overview: HPBOOLRULE Procedure

The HPBOOLRULE procedure is a high-performance procedure that enables you to extract Boolean rules from large-scale transactional data. In the current release, you can use the HPBOOLRULE procedure to read data and extract rules only in single-machine mode.

The HPBOOLRULE procedure can automatically generate a set of Boolean rules by analyzing a text corpus that has been processed by the HPTMINE 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 HPBOOLRULE procedure has three advantages when you use it to analyze your large-scale transactional data with a supervised rule-based model. 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 HPBOOLRULE Features

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

- Boolean rules are automatically extracted from large-scale transactional data.
- The extracted rules are human understandable and tunable.
- Important features are identified for each category.

[1] A data set is imbalanced if it contains many more negative samples than positive samples, or vice versa.

- Imbalanced data are handled robustly.
- Binary-class, multiclass, and multi-label categorization are supported.
- Events for defining labels for documents are supported.
- All processing phases use a high degree of multi threading.

PROC HPBOOLRULE Contrasted with Other SAS Procedures

This section compares the HPBOOLRULE procedure with the ARBOR, HPTREE, and TAXONOMY procedures in SAS Enterprise Miner.

The ARBOR and HPTREE procedures enable you to create a decision tree. The branches of the tree that lead to a positive category also define a rule set. These procedures and the HPBOOLRULE procedure differ as follows:

- The ARBOR and HPSPLIT procedures take as input a relational format data set, which is in the standard dense form of observations by variables. PROC HPBOOLRULE is designed to handle a very large number of variables, which are represented in sparse form as a transaction data set that indicates row, column, and value.
- PROC HPBOOLRULE is designed to work with binary attributes only: they are either present or absent. The other procedures can take categorical or interval inputs. It is possible to create binary attributes from categorical data by dummying, or from interval inputs by binning, but such creation is not supported directly by the procedure and is not tailored to the data.
- PROC HPBOOLRULE is designed to work with a very large number of variables, so it works much faster than a decision tree approach in that situation.
- Decision trees model positive and negative samples simultaneously. Their performance degenerates when the input data are highly imbalanced. In contrast, the HPBOOLRULE procedure focuses on modeling the positive samples. Therefore, it performs robustly even when the data are highly imbalanced.
- Decision trees normally work by building a tree greedily and then pruning back after the tree is built. PROC HPBOOLRULE examines only the most promising candidate rules at the first level, and becomes fully greedy at deeper levels. Pruning is done in the process of building the rules.

The TAXONOMY procedure mines transaction data for association rules. This is essentially an unsupervised process for discovering items that appear together frequently. In contrast, the HPBOOLRULE procedure is a supervised modeling procedure. Because PROC HPBOOLRULE identifies terms that are highly special to the target category, it can be used to discriminate positive samples from negative samples. Also, in the modeling process, the HPBOOLRULE procedure uses both the presence and absence (by using the negation operator) of a term to create rules, whereas the TAXONOMY procedure only uses the presence of an item to create rules.

Getting Started: HPBOOLRULE Procedure

The following DATA step 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 set represents a document for analysis.

```
data getstart;
    infile cards delimiter='|' missover;
    length text $150;
    input text$ apple_fruit did$;
    cards;
        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 tasts 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;
```

The following statements use the HPTMINE procedure to parse the input text data. The generated term-by-document matrix is stored in a SAS data set named BOW. The summary information about the terms in the document collection is stored in a SAS data set named Terms.

```
proc hptmine data=getstart language="english";
    doc_id
        did;
    var
        text;
    parse
        nonoungroups
        entities    = none
        outparent   = bow
        outterms    = keys
        reducef     = 0
        stop        = sashelp.engstop;
    performance
        details;
run;
```

The following statements use the HPBOOLRULE procedure to extract rules:

```
proc hpboolrule
        data        =   bow
        docid       =   _document_
        termid      =   _termnum_
        docinfo     =   getstart
        terminfo    =   keys
        minsupports =   1
        mpos        =   1
        gpos        =   1;
    docinfo
        id          =   did
        targets     =   (apple_fruit);
    terminfo
        id          =   key
        label       =   term;
    output
        rules       =   rules;
    performance
        details;
run;
```

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

The DOCINFO statement specifies the following information about the GetStart data set:

- 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 HPBOOLRULE statement 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 keys data set:

- 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 HPBOOLRULE statement 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 set Rules.

The output from this analysis is presented in Figure 4.1 through Figure 4.6.

Figure 4.1 shows the “Performance Information” table, which indicates that PROC HPBOOLRULE executes in single-machine mode. That is, PROC HPBOOLRULE runs on the machine where the SAS System is running. The table also shows that four threads are used for computing.

Figure 4.1 Performance Information

The HPBOOLRULE Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Figure 4.2 shows the "Data Access Information" table, which provides the access information about the data sets that the HPBOOLRULE procedure has accessed and generated.

Figure 4.2 Data Access Information

Data Access Information

Data	Engine	Role	Path
WORK.BOW	V9	Input	On Client
WORK.GETSTART	V9	Input	On Client
WORK.KEYS	V9	Input	On Client
WORK.RULES	V9	Output	On Client

Figure 4.3 shows the "Data Information" table, which provides the information about the input data sets that the HPBOOLRULE procedure used for modeling.

Figure 4.3 Data Information

Data Information

Description	Value
Number of documents read from BOW	20
Number of terms read from BOW	56
Number of document indices specified in both BOW and GETSTART	20
Number of terms indices specified in both BOW and KEYS	56
Number of documents in category apple_fruit	8

Figure 4.4 shows the "Procedure Task Timing" table, which provides details about how much time is used by each processing step.

Figure 4.4 Procedure Task Timing

Procedure Task Timing

Task	Seconds	Percent
Read Data	0.01	100.0%
Feature Selection	0.00	0.00%
Output Selected Features	0.00	0.00%
Rule Generation	0.00	0.00%
Output Rules	0.00	0.00%
Output Rule Terms	0.00	0.00%

Figure 4.5 shows the SAS log that is generated by PROC HPBOOLRULE; 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 Rules data set contains four observations, indicating that the HPBOOLRULE procedure identified three rules for the apple_fruit category.

Figure 4.5 SAS Log

```
NOTE: Input data set WORK.BOW.DATA is used for training.
NOTE: No SPARSEFORMAT option is specified. SPARSEFORMAT=COO will be run by
      default.
NOTE: Neither SEQCOVER nor NOSEQCOVER is specified. SEQCOVER is used by default.
NOTE: The HPBOOLRULE procedure is executing in single-machine mode.
NOTE: There were 87 observations read from the data set WORK.BOW.
NOTE: There were 20 observations read from the data set WORK.GETSTART.
NOTE: There were 75 observations read from the data set WORK.KEYS.
NOTE: The data set WORK.RULES has 4 observations and 15 variables.
```

The following statements use PROC PRINT to show the contents of the Rules data set that is generated by the HPBOOLRULE procedure:

```
proc print data = rules;
var target ruleid rule F1 precision recall;
run;
```

Figure 4.6 shows the output of PROC PRINT, which contains four single-term rules. For information about the output of the RULES= option, see the section "RULES= Data Set" on page 56.

Figure 4.6 The rules Data Set

Obs	TARGET	RULEID	RULE	F1	PRECISION	RECALL
1	apple_fruit	1	tree	0.54545	1	0.375
2	apple_fruit	2	fruit	0.85714	1	0.750
3	apple_fruit	3	fuji	0.93333	1	0.875
4	apple_fruit	4	sweet	1.00000	1	1.000

Syntax: HPBOOLRULE Procedure

The following statements are available in the HPBOOLRULE procedure:

PROC HPBOOLRULE < *options* > ;
  DOCINFO < *options* > ;
  TERMINFO < *options* > ;
  OUTPUT < *options* > ;
  PERFORMANCE < *performance-options* > ;

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

PROC HPBOOLRULE Statement

PROC HPBOOLRULE < *options* > ;

The PROC HPBOOLRULE 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 4.1 PROC HPBOOLRULE Statement Options

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

Table 4.1 *continued*

Option	**Description**
TERMID=	Specifies the variable in the DATA= data set that contains the term ID
TERMINFO=	Specifies the input data set that contains information about terms
Output Options	
NOPRINT	Suppresses ODS output

You can specify the following *options*:

DATA=*SAS-data-set*

DOC=*SAS-data-set*

names the input SAS data set to be used by PROC HPBOOLRULE for rule extraction. The default is the most recently created data set. Each row of the input data set 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 text variable. The HPBOOLRULE procedure does not assume that the data set is sorted by either document ID or term ID.

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=*SAS-data-set*

names the input SAS data set that contains information about documents. Each row of the input data set must contain one variable for the document ID. The HPBOOLRULE procedure uses the document ID in the DATA= data set to search for the document ID variable in this data set 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 53.

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 Table 4.8 is larger than g-*value*. By default, GPOS=8.

MAXCANDIDATES=*n*

MAXCANDS=*n*

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

MAXTRIESIN=*n*

specifies the k_{in} value for the k-best search in the term ensemble process for creating rules. For more information about the k-best search, see the section "k-Best Search" on page 55. 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 about the k-best search, see the section "k-Best Search" on page 55. 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.

NOPRINT

suppresses the generation of ODS output.

TERMID=*variable*

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

TERMINFO=*SAS-data-set*

names the input SAS data set that contains information about terms. Each row of the input data set must contain one variable for the term ID. If this option is specified, the TERMINFO statement must be used to specify which variables in the data set contain the term ID and the term label, respectively. The HPBOOLRULE procedure uses the term ID in the DATA= data set to search for the term ID variable in this data set 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 set that is specified in the DOCINFO= option in the PROC HPBOOLRULE 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=MULTICLASS, the *values* are considered as categories of interest for rule extraction.
- When TARGETTYPE=BINARY and the target variable is a numerical variable, "1" is considered as a positive event by default.
- When TARGETTYPE=BINARY and the target variable is a text variable, "Y" is considered as a positive event by default.
- When TARGETTYPE=MULTICLASS, each level of the target variable corresponds to a category of interest.

ID=*variable*

specifies the *variable* that contains the document ID. The values in the *variable* are matched to the document ID variable that is specified in the DOCID= option in the PROC HPBOOLRULE statement in order to fetch the target information about documents. The *variable* can be either a numerical variable or a text variable. Its type must match the type of the variable that is specified in the DOCID= option in the PROC HPBOOLRULE statement.

TARGET=(*variable***,** *variable***, . . .)**

specifies the target *variables*. A target variable can be either a numerical variable or a text variable.

- When TARGETTYPE=BINARY, multiple target variables can be specified, and each target variable corresponds to a category.
- When TARGETTYPE=MULTICLASS, only one target variable can be specified, and each level of the target variable corresponds 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 variables can be specified and each level of the target variable corresponds a category.

By default, TARGETTYPE=BINARY.

OUTPUT Statement

OUTPUT < *options* > **;**

The OUTPUT statement specifies the SAS data sets that contain the results generated by the HPBOOLRULE procedure.

You can specify the following *options*:

CANDIDATETERMS=*SAS-data-set*
specifies a SAS data set to contain the terms that have been selected by the HPBOOLRULE procedure for rule creation. If MAXCANDIDATES=p in the HPBOOLRULE statement, the procedure selects at most p terms for each category to be considered for rule extraction. For more information about this data set, see the section "Output Data Sets" on page 55.

RULES=*SAS-data-set*
specifies a SAS data set to contain the rules that have been generated by the HPBOOLRULE procedure for each category. For more information about this data set, see the section "Output Data Sets" on page 55.

RULETERMS=*SAS-data-set*
specifies a SAS data set to contain the terms in each rule that is generated by the HPBOOLRULE procedure. For more information about this data set, see the section "Output Data Sets" on page 55.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables that describe the distributed computing environment, and requests detailed results about the performance characteristics of the HPBOOLRULE procedure. The PERFORMANCE statement is documented further in the section "PERFORMANCE Statement" on page 33 of Chapter 3, "Shared Concepts and Topics." For the current release, the HPBOOLRULE procedure supports only single-machine mode.

TERMINFO Statement

TERMINFO < *options* > **;**

The TERMINFO statement specifies the information about the TERMINFO= data set that is specified in the PROC HPBOOLRULE statement. If you specify the TERMINFO= data set in the PROC HPBOOLRULE statement, you must also include this statement to specify which variables in the data set 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. The values in the variable are matched to the term ID variable that is specified in the TERMID= option in the PROC HPBOOLRULE statement to fetch the text of terms. The ID variable can be either a numerical variable or a text variable. Its type must match the type of the variable that is specified in the TERMID= option in the PROC HPBOOLRULE statement.

LABEL=*variable*

specifies the variable that contains the text of the terms. This variable must be a text variable.

Details: HPBOOLRULE Procedure

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

BOOLLEAR for Boolean Rule Extraction

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

1. Use an information gain criterion to form an ordered term candidate list. The term that best predicts the category is first on the list, and so on. Terms that do not have a significant relationship to the category are removed from this list. Set the current term to the first term.

2. Determine the "estimated precision" of the current term. The estimated precision is the projected percentage of the term's occurrence with the category in out-of-sample data, using additive smoothing. Create a rule that consists of that term.

3. If the "estimated precision" of the current rule could not possibly be improved by adding more terms as qualifiers, then go to step 6.

4. Starting with the next term on the list, determine whether the conjunction of the current rule with that term (via either term presence or term absence) significantly improves the information gain and also improves estimated precision.

5. If there is at least one combination that meets the criterion in step 4, choose the combination that yields the best estimated precision, and go to step 3 with that combination. Otherwise, continue to step 6.

6. If the best rule obtained in step 3 has a higher estimated precision than the current "highest precision" rule, replace the current rule with the new rule.

7. Increment the current term to the next term on 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.

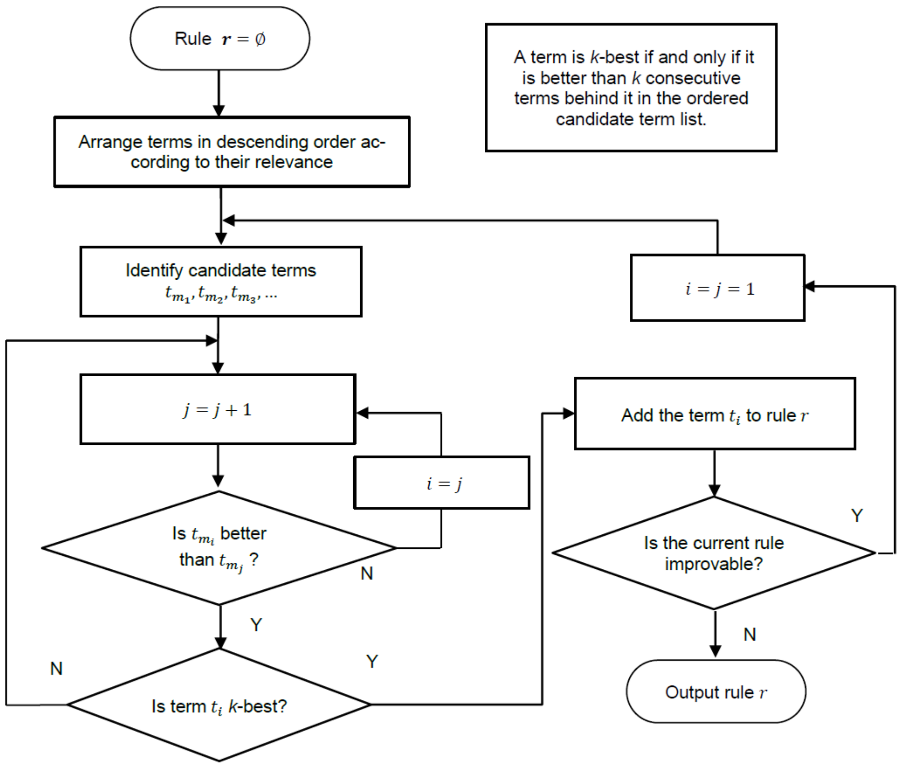
9. If so, remove all documents that match the new rule from the document set, add this rule to the rule set, and go to step 1 to start creating the next rule in the rule set.

BOOLLEAR contains two essential processes for rule extraction: a term ensemble process (step 4 and step 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. The following figure shows the flowchart of the process.

Figure 4.7 Term Ensemble Process for Creating a Rule



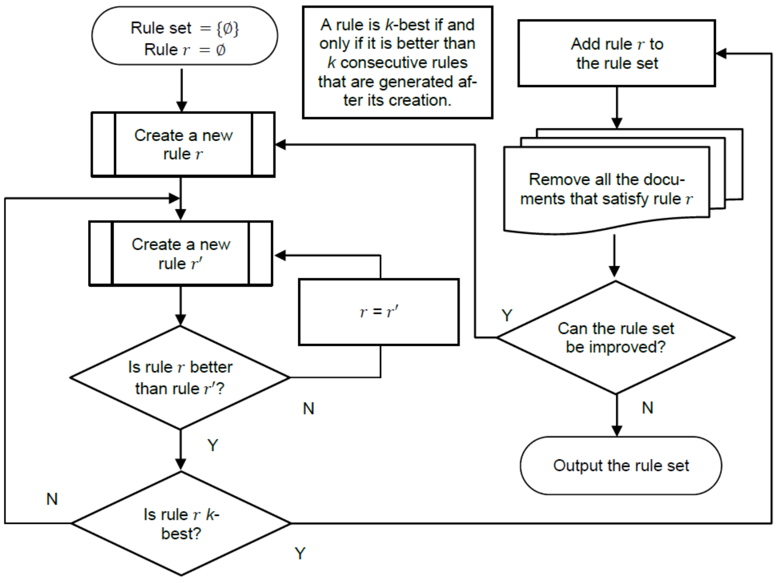
Rule $r = \emptyset$
A term is *k*-best if and only if it is better than *k* consecutive terms behind it in the ordered candidate term list.
Arrange terms in descending order according to their relevance
Identify candidate terms $t_{m_1}, t_{m_2}, t_{m_3}, \dots$
$i = j = 1$
$j = j + 1$
Add the term t_i to rule r
$i = j$
Is t_{m_i} better than t_{m_j} ?
N
Y
Is the current rule improvable?
Y
N
Y
N
Is term t_i *k*-best?
Output rule r

Before adding terms to a rule, BOOLLEAR first sorts the candidate terms in descending order according to their g-score 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 $\mathbf{r}$ 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 $\mathbf{r}$) than term t, the term is considered as k-best, where $k = k_{in}$, and BOOLLEAR updates the current rule $\mathbf{r}$ 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 growing the rule by 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 add new rules to a rule set. When the process finishes, it returns the rule set, which can be then used for text categorization. The following figure shows the flowchart of the rule ensemble process.

Figure 4.8 Rule Ensemble for Creating a Rule Set


Rule set $= \{\emptyset\}$
Rule $r = \emptyset$
A rule is k-best if and only if it is better than k consecutive rules that are generated after its creation.
Add rule r to the rule set
Create a new rule r
Remove all the documents that satisfy rule r
Create a new rule r'
$r = r'$
Y
Can the rule set be improved?
Is rule r better than rule r'?
N
N
Y
Output the rule set
N
Is rule r k-best?
Y

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 HPBOOLRULE 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 to generate another rule using the updated corpus. 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 52.

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

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

where TP is true-positive (the number of documents that are predicted to be positive and are actually positive), FP is false-positive (the number of documents that are predicted to be positive but are actually negative), TN is true-negative (the number of documents that are predicted to be negative and are actually negative), and FN is 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[2] when the data are imbalanced, because a classifier can obtain very high accuracy by predicting that all samples belong to the majority category.

[2]Accuracy is defined as $\frac{TP+TN}{TP+FP+TN+FN}$

g-Score

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

$$g = 2 \times \sum\nolimits_{i=\{TP,TN,FP,FN\}} O\,(i) \log\left(\frac{O(i)}{E(i)}\right)$$

$$\begin{aligned}
O(TP) &= TP\\
O(FP) &= FP\\
O(TN) &= TN\\
O(FN) &= FN\\
E(TP) &= \frac{(TP + FP) \times P}{P + N}\\
E(FP) &= \frac{(TP + FP) \times N}{P + N}\\
E(TN) &= \frac{(TN + FN) \times N}{P + N}\\
E(FN) &= \frac{(TN + FN) \times P}{P + N}
\end{aligned}$$

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:

$$precision_i^m(t) = \frac{TP_{i,\mathbf{t}} + \frac{P}{N+P} \times m}{TP_{i,\mathbf{t}} + FP_{i,\mathbf{t}} + m} - err_{i-1}$$

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

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

TP	0
0	FP

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 is added to the rule. If the g-score is smaller than a number that you specify to indicate a maximum p-value for significance, BOOLLEAR considers the rule to be unimprovable.

Shrinking the Search Space

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

Feature Selection

BOOLLEAR uses the g-test to evaluate terms. Assume that MAXCANDIDATES=p and MINSUPPORTS=c in the PROC HPBOOLRULE 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 does pruning "in-process." The following three checks are a form of in-process pruning, in that rules are not expanded when their expansion does not meet these basic requirements. These requirements help BOOLLEAR truncate its search path and avoid generating overly specific rules:

- **Minimum positive document coverage**: BOOLLEAR requires that a rule be satisfied by at least s positive documents, where s is the value of the MINSUPPORTS= option in the PROC HPBOOLRULE 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 growing a rule when the estimated precision of the rule does not improve when the current best term is added to the rule. This strategy helps BOOLLEAR shorten its search path.

k-Best Search

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

Improvability Test

This 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 growing 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 HPBOOLRURE produces.

CANDIDATETERMS= Data Set

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

Table 4.2 shows the fields in this data set.

Table 4.2 Fields in the CANDIDATETERMS= Data Set

Fields	Description
Target	The category that the term is selected for (this field corresponds to the Target field in the RULES= data set)

Table 4.2 *continued*

Fields	Description
Rank	The rank of the term in the ordered term list for the category (term rank starts from 1)
Term	A lowercase version of the term
Key	The term identifier of the term
GScore	The *g*-score of the term that is obtained for the target category
Support	The number of documents in which the term appears
TP	The number of positive documents in which the term appears
FP	The number of negative documents in which the term appears

RULES= Data Set

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

Table 4.3 shows the fields in this data set.

Table 4.3 Fields in the RULES= Data Set

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

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 of the data. The second rule is trained on the data that did not satisfy the first rule. And subsequent rules are learned only after removing any 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 TARGETTYPE=MULTICLASS is specified in the DOCINFO statement, each target level of the target variable defines a category and the target field contains the same content of the Target_val field. When TARGETTYPE=BINARY is specified in the DOCINFO statement, each target variable defines a category and the target field contains the same content of the Target_var field.

RULETERMS= Data Set

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

Table 4.4 Fields in the RULETERMS= Data Set

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

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 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.

Displayed Output

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

Performance Information

The "Performance Information" table is produced by default. It displays information about the grid host for distributed execution and information about whether the procedure executes in single-machine mode, distributed mode, or alongside-the-database mode. The numbers of computing nodes and threads are also displayed, depending on the environment. In the current release, the HPBOOLRULE procedure supports only single-machine mode.

Data Access Information

The "Data Access Information" table is produced by default; it displays the library engine, role, and path of each data set that is used by the HPBOOLRULE procedure.

Data Information

The "Data Information" table is produced by default. It displays the following information:

- number of documents that are read from the DATA= data set
- number of terms that are read from the DATA= data set
- number of documents that appear in both the DATA= data set and the DOCINFO= data set
- number of terms that appear in both the DATA= data set and the TERMINFO= data set
- number of documents in each category

Procedure Task Timing

When the DETAILS option is specified in the PERFORMANCE statement, PROC HPBOOLRULE produces a "Procedure Task Timing" table, which displays the elapsed time (absolute and relative) for the main tasks.

ODS Table Names

Each table that is created by the HPBOOLRULE procedure has a name associated with it, and you must use this name to refer to the table when you use ODS statements. These names are listed in Table 4.5.

Table 4.5 ODS Tables Produced by PROC HPBOOLRULE

Table Name	Description	Required Statement / Option
DataAccessInfo	Information about the data sets that are used by the procedure	Default output
DataInfo	Information about the number of documents and the number of terms that are used as inputs	Default output
PerformanceInfo	Information about the high-performance computing environment	Default output
Timing	Absolute and relative times for tasks performed by the procedure	DETAILS option in the PERFORMANCE statement

Examples: HPBOOLRULE Procedure

Example 4.1: Rule Extraction for Binary Targets

This example generates rules for the (news) data set named sampsio. If the sampsio libref is unavailable to you, you can define it by running the SAS command: **`libname sampsio "!sasroot\tmine\sample";`**. The following DATA step generates the news_key data set by adding the _document_ variable to the (news) data set. The _document_ variable contains the ID of documents. The HPTMINE procedure parses the news_key data set. The term-by-document matrix is stored in the news_key_bow_coo data set in transactional format. Terms that appeared in the news_key data set are stored in the news_key_terms data set.

```
data news_key;
    set sampsio.news;
    _document_ = _n_;
run;


proc hptmine data=news_key;
    doc_id
        _document_;
    var
        text;
    parse
        nonoungroups
        notagging
        stop         = sashelp.engstop
        entities     = none
        outparent    = news_key_bow_coo
        outterms     = news_key_terms;
    performance
        details;
run;
```

The following statements run PROC HPBOOLRULE to extract rules from the news_key_bow_coo data set. By default, TARGETTYPE=BINARY. Two target variables, graphics and hockey, are specified, each of which defines a category:

```
proc hpboolrule
        data        =   news_key_bow_coo
        docid       =   _document_
        termid      =   _termnum_
        docinfo     =   news_key
        terminfo    =   news_key_terms;
    docinfo
        id          =   key
        targets     =   (graphics hockey);
    terminfo
        id          =   key
        label       =   term;
    output
        ruleterms   =   ruleterms
        rules       =   rules;
    performance
        details;
run;

proc print data=rules;
    var target ruleid rule F1 precision recall;
run;
```

Output 4.1.1 shows that the rules data set contains rules that are generated for the “graphics” and “hockey” categories.

Output 4.1.1 The rules Data Set

Obs	TARGET	RULEID	RULE	F1	PRECISION	RECALL
1	graphics	1	graphics	0.38525	0.97917	0.23980
2	graphics	2	image	0.55797	0.96250	0.39286
3	graphics	3	algorithm	0.60839	0.96667	0.44388
4	graphics	4	file	0.68590	0.92241	0.54592
5	graphics	5	software	0.72561	0.90152	0.60714
6	graphics	6	ibm	0.74850	0.90580	0.63776
7	graphics	7	mode	0.77193	0.90411	0.67347
8	graphics	8	define	0.78963	0.90728	0.69898
9	graphics	9	chip	0.80226	0.89873	0.72449
10	graphics	10	code	0.81440	0.89091	0.75000
11	graphics	11	screen	0.82514	0.88824	0.77041
12	graphics	12	yeh	0.83469	0.89017	0.78571
13	graphics	13	sphere	0.84409	0.89205	0.80102
14	graphics	14	pov	0.85333	0.89385	0.81633
15	graphics	15	ftp	0.86243	0.89560	0.83163
16	graphics	16	polygon	0.87139	0.89730	0.84694
17	graphics	17	fractal	0.88021	0.89894	0.86224
18	graphics	18	cs	0.88601	0.90000	0.87245
19	hockey	19	team	0.56429	0.98750	0.39500
20	hockey	20	hockey	0.65552	0.98990	0.49000
21	hockey	21	cup	0.71565	0.99115	0.56000
22	hockey	22	rangers	0.75926	0.99194	0.61500
23	hockey	23	fan	0.79762	0.98529	0.67000
24	hockey	24	lemieux	0.82558	0.98611	0.71000
25	hockey	25	playoff	0.84900	0.98675	0.74500
26	hockey	26	montreal	0.86517	0.98718	0.77000
27	hockey	27	rauser.734062608@sfu.ca	0.88089	0.98758	0.79500
28	hockey	28	trade	0.89315	0.98788	0.81500
29	hockey	29	wing	0.90270	0.98235	0.83500
30	hockey	30	sport	0.91153	0.98266	0.85000
31	hockey	31	puck	0.92021	0.98295	0.86500
32	hockey	32	darling	0.92593	0.98315	0.87500
33	hockey	33	game	0.92268	0.95213	0.89500
34	hockey	34	potvin	0.92821	0.95263	0.90500
35	hockey	35	score	0.93367	0.95313	0.91500

Example 4.2: Rule Extraction for a Multiclass Target

The following statements run PROC HPBOOLRULE to extract rules from the news_key_bow_coo data set, which is built in "Example 4.1: Rule Extraction for Binary Targets" on page 59. TARGET-TYPE=MULTICLASS is specified and newsgroup is specified as the target variable, which contains three levels: "graphics," "hockey," and "medical." Each level defines a category for the HPBOOLRULE procedure to extract rules for.

```
proc hpboolrule
        data        =   news_key_bow_coo
        docid       =   _document_
        termid      =   _termnum_
        docinfo     =   news_key
        terminfo    =   news_key_terms;
    docinfo
        id          =   key
        targettype  =   multiclass
        targets     =   (newsgroup);
    terminfo
        id          =   key
        label       =   term;
    output
        ruleterms   =   ruleterms
        rules       =   rules;
    performance
        details;
run;

proc print data=rules;
    var target ruleid rule F1 precision recall;
run;
```

Output 4.2.1 shows that the rules data set contains rules that are generated for the "graphics," "hockey," and "medical" categories.

Output 4.2.1 The rules Data Set

Obs	TARGET	RULEID	RULE	F1	PRECISION	RECALL
1	graphics	1	graphics	0.38525	0.97917	0.23980
2	graphics	2	image	0.55797	0.96250	0.39286
3	graphics	3	algorithm	0.60839	0.96667	0.44388
4	graphics	4	file	0.68590	0.92241	0.54592
5	graphics	5	software	0.72561	0.90152	0.60714
6	graphics	6	ibm	0.74850	0.90580	0.63776
7	graphics	7	mode	0.77193	0.90411	0.67347
8	graphics	8	define	0.78963	0.90728	0.69898
9	graphics	9	chip	0.80226	0.89873	0.72449
10	graphics	10	code	0.81440	0.89091	0.75000
11	graphics	11	screen	0.82514	0.88824	0.77041
12	graphics	12	yeh	0.83469	0.89017	0.78571
13	graphics	13	sphere	0.84409	0.89205	0.80102
14	graphics	14	pov	0.85333	0.89385	0.81633
15	graphics	15	ftp	0.86243	0.89560	0.83163
16	graphics	16	polygon	0.87139	0.89730	0.84694
17	graphics	17	fractal	0.88021	0.89894	0.86224
18	graphics	18	cs	0.88601	0.90000	0.87245
19	hockey	19	team	0.56429	0.98750	0.39500
20	hockey	20	hockey	0.65552	0.98990	0.49000
21	hockey	21	cup	0.71565	0.99115	0.56000
22	hockey	22	rangers	0.75926	0.99194	0.61500
23	hockey	23	fan	0.79762	0.98529	0.67000
24	hockey	24	lemieux	0.82558	0.98611	0.71000
25	hockey	25	playoff	0.84900	0.98675	0.74500
26	hockey	26	montreal	0.86517	0.98718	0.77000
27	hockey	27	rauser.734062608@sfu.ca	0.88089	0.98758	0.79500
28	hockey	28	trade	0.89315	0.98788	0.81500
29	hockey	29	wing	0.90270	0.98235	0.83500
30	hockey	30	sport	0.91153	0.98266	0.85000
31	hockey	31	puck	0.92021	0.98295	0.86500
32	hockey	32	darling	0.92593	0.98315	0.87500
33	hockey	33	game	0.92268	0.95213	0.89500
34	hockey	34	potvin	0.92821	0.95263	0.90500
35	hockey	35	score	0.93367	0.95313	0.91500
36	medical	36	gordon	0.44961	1.00000	0.29000
37	medical	37	doctor	0.56940	0.98765	0.40000
38	medical	38	msg	0.63729	0.98947	0.47000
39	medical	39	health	0.69055	0.99065	0.53000
40	medical	40	medicine	0.73186	0.99145	0.58000
41	medical	41	merrill	0.76687	0.99206	0.62500
42	medical	42	pain	0.80000	0.99259	0.67000
43	medical	43	drug	0.83095	0.97315	0.72500
44	medical	44	infection	0.85070	0.97419	0.75500
45	medical	45	russell	0.86740	0.96914	0.78500
46	medical	46	disease	0.87738	0.96407	0.80500
47	medical	47	treat	0.88649	0.96471	0.82000
48	medical	48	migraine	0.89544	0.96532	0.83500

Output 4.2.1 *continued*

Obs	TARGET	RULEID	RULE	F1	PRECISION	RECALL
49	medical	49	lung	0.90133	0.96571	0.84500
50	medical	50	diet	0.90716	0.96610	0.85500
51	medical	51	ccreegan@ecsvax.uncecs.edu	0.91293	0.96648	0.86500
52	medical	52	med	0.91864	0.96685	0.87500

Example 4.3: Using Events in Rule Extraction

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 used the EVENTS= option to specify the categories for which you want to extract rules. The following statements run PROC HPBOOLRULE to extract rules from the news_key_bow_coo data set. TARGETTYPE=MULTICLASS is specified, and the target variable is specified as newsgroup, which contains three levels: “graphics,” “hockey,” and “medical.” Because the “graphics” and “hockey” levels are specified in the EVENTS= option, PROC HPBOOLRULE procedure extracts rules for “graphics” and “hockey,” but not “medical.”

```
proc hpboolrule
        data        =   news_key_bow_coo
        docid       =   _document_
        termid      =   _termnum_
        docinfo     =   news_key
        terminfo    =   news_key_terms;
    docinfo
        id          =   key
        targettype  =   multiclass
        targets     =   (newsgroup)
        events      =   ("graphics" "hockey");
    terminfo
        id          =   key
        label       =   term;
    output
        ruleterms   =   ruleterms
        rules       =   rules;
    performance
        details;
run;

proc print data=rules;
    var target ruleid rule F1 precision recall;
run;
```

Output 4.3.1 shows that the rules data set contains rules that are generated for the “graphics” and “hockey” categories.

Output 4.3.1 The rules Data Set

Obs	TARGET	RULEID	RULE	F1	PRECISION	RECALL
1	graphics	1	graphics	0.38525	0.97917	0.23980
2	graphics	2	image	0.55797	0.96250	0.39286
3	graphics	3	algorithm	0.60839	0.96667	0.44388
4	graphics	4	file	0.68590	0.92241	0.54592
5	graphics	5	software	0.72561	0.90152	0.60714
6	graphics	6	ibm	0.74850	0.90580	0.63776
7	graphics	7	mode	0.77193	0.90411	0.67347
8	graphics	8	define	0.78963	0.90728	0.69898
9	graphics	9	chip	0.80226	0.89873	0.72449
10	graphics	10	code	0.81440	0.89091	0.75000
11	graphics	11	screen	0.82514	0.88824	0.77041
12	graphics	12	yeh	0.83469	0.89017	0.78571
13	graphics	13	sphere	0.84409	0.89205	0.80102
14	graphics	14	pov	0.85333	0.89385	0.81633
15	graphics	15	ftp	0.86243	0.89560	0.83163
16	graphics	16	polygon	0.87139	0.89730	0.84694
17	graphics	17	fractal	0.88021	0.89894	0.86224
18	graphics	18	cs	0.88601	0.90000	0.87245
19	hockey	19	team	0.56429	0.98750	0.39500
20	hockey	20	hockey	0.65552	0.98990	0.49000
21	hockey	21	cup	0.71565	0.99115	0.56000
22	hockey	22	rangers	0.75926	0.99194	0.61500
23	hockey	23	fan	0.79762	0.98529	0.67000
24	hockey	24	lemieux	0.82558	0.98611	0.71000
25	hockey	25	playoff	0.84900	0.98675	0.74500
26	hockey	26	montreal	0.86517	0.98718	0.77000
27	hockey	27	rauser.734062608@sfu.ca	0.88089	0.98758	0.79500
28	hockey	28	trade	0.89315	0.98788	0.81500
29	hockey	29	wing	0.90270	0.98235	0.83500
30	hockey	30	sport	0.91153	0.98266	0.85000
31	hockey	31	puck	0.92021	0.98295	0.86500
32	hockey	32	darling	0.92593	0.98315	0.87500
33	hockey	33	game	0.92268	0.95213	0.89500
34	hockey	34	potvin	0.92821	0.95263	0.90500
35	hockey	35	score	0.93367	0.95313	0.91500

References

Cox, J., and Zhao, Z. (2014). "System for Efficiently Generating *k*-Maximally Predictive Association Rules with a Given Consequent." US Patent Number 20140337271.

Chapter 5
The HPTMINE Procedure

Contents

Overview: HPTMINE Procedure

The HPTMINE procedure is a high-performance procedure that analyzes large-scale textual data. PROC HPTMINE provides an essential capability for high-performance text mining and 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, and dimension reduction by term filtering and singular value decomposition (SVD).

PROC HPTMINE integrates the functionalities that are provided by the TGPARSE, TMUTIL, and SPSVD procedures from SAS Text Miner, and achieves high efficiency and scalability through parallel processing. The HPTMINE procedure can also generate results that can be used to facilitate scoring by the HPTMSCORE procedure. These results include a configuration data set, a term data set, and a data set that contains the SVD projection matrix.

You can use the HPTMINE procedure to read data in distributed form and perform text analysis in parallel in single-machine mode or distributed mode. For more information about how to configure the execution mode of SAS high-performance analytical procedures, see the section "Processing Modes" on page 8 in Chapter 3, "Shared Concepts and Topics."

NOTE: Distributed mode requires SAS High-Performance Text Mining.

PROC HPTMINE Features

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

- 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 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.
- Analysis can be performed on a massively parallel SAS high-performance appliance.
- All phases of processing make use of a high degree of multithreading.
- Input data can be read in parallel when the data source is the appliance database.

PROC HPTMINE Contrasted with Other SAS Procedures

The following remarks compare the HPTMINE procedure with the TGPARSE, TMUTIL, and SPSVD procedures in SAS Text Miner software.

PROC HPTMINE maximizes performance by combining the key functionalities that are found in the TGPARSE, TMUTIL, and SPSVD procedures. By functioning as an "all-in-one" procedure, PROC HPTMINE avoids expensive I/O operations. The PARSE option in the HPTMINE procedure effectively replaces the necessary options that are provided by specifying PROC TGPARSE followed by PROC TMUTIL in the conventional SAS environment. You can use the PARSE option to control parsing, accumulation, and ultimately the construction of the underlying weighted term-document frequency matrix. This matrix serves as input for singular value decomposition, which is controlled by the SVD statement in the HPTMINE procedure.

PROC HPTMINE's functionality also replaces many of the core functions that were available in the SPSVD procedure. Both PROC HPTMINE and PROC SPSVD produce an output that holds the reduced dimensional representation for the input documents. They also generate an output **U** matrix that can be used to project new documents in this same reduced space during scoring.

Getting Started: HPTMINE Procedure

The following DATA step contains 36 observations that have two variables. The text variable contains the input documents, and the did variable contains the ID of the documents. Each row in the data set represents a document for analysis.

```
data getstart;
    infile cards delimiter='|' missover;
    length text $150;
    input text$ did$;
    cards;
        High-performance analytics hold the key to |d01
        unlocking the unprecedented business value of big data.|d02
        Organizations looking for optimal ways to gain insights|d03
        from big data in shorter reporting windows are turning to SAS.|d04
        As the gold-standard leader in business analytics |d05
        for more than 36 years,|d06
        SAS frees enterprises from the limitations of |d07
        traditional computing and enables them |d08
        to draw instant benefits from big data.|d09
        Faster Time to Insight.|d10
        From banking to retail to health care to insurance, |d11
        SAS is helping industries glean insights from data |d12
        that once took days or weeks in just hours, minutes or seconds.|d13
        It's all about getting to and analyzing relevant data faster.|d14
        Revealing previously unseen patterns, sentiments and relationships.|d15
        Identifying unknown risks.|d16
        And speeding the time to insights.|d17
        High-Performance Analytics from SAS Combining industry-leading |d18
        analytics software with high-performance computing technologies|d19
        produces fast and precise answers to unsolvable problems|d20
        and enables our customers to gain greater competitive advantage.|d21
        SAS In-Memory Analytics eliminate the need for disk-based processing|d22
        allowing for much faster analysis.|d23
        SAS In-Database executes analytic logic into the database itself |d24
        for improved agility and governance.|d25
        SAS Grid Computing creates a centrally managed,|d26
        shared environment for processing large jobs|d27
        and supporting a growing number of users efficiently.|d28
        Together, the components of this integrated, |d29
        supercharged platform are changing the decision-making landscape|d30
        and redefining how the world solves big data business problems.|d31
        Big data is a popular term used to describe the exponential growth,|d32
        availability and use of information,|d33
        both structured and unstructured.|d34
        Much has been written on the big data trend and how it can |d35
        serve as the basis for innovation, differentiation and growth.|d36
run;
```

The following statements use singular value decomposition to parse the input text data and generate a lower-dimensional representation of the documents. The statements specify that only the terms that appear at least twice in the document collection be kept for generating the term-by-document matrix. The summary information about the terms in the document collection is stored in a SAS data set named terms. The SVD statement requests that the top 10 singular values and singular vectors be computed. The projection of the documents is stored in a SAS data set named docpro.

```
proc hptmine data=getstart;
doc_id      did;
variables   text;
parse
            outterms  = terms
            reducef   = 2;
svd
            k         = 10
            outdocpro = docpro;
performance details;
run;
```

The output from this analysis is presented in Figure 5.1 through Figure 5.6.

Figure 5.1 shows the “Performance Information” table, which indicates that PROC HPTMINE executes in single-machine mode. That is, PROC HPTMINE runs on the machine where the SAS system is running. The table also shows that four threads are used for computing.

Figure 5.1 Performance Information

The HPTMINE Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Figure 5.2 shows the “Procedure Task Timing” table, which provides details about how much time is used by each processing step.

Figure 5.2 Procedure Task Timing

Procedure Task Timing

Task	Seconds	Percent
Parse documents	9.55	99.76%
Analyze terms	0.00	0.01%
Obtain term frequency	0.01	0.10%
Filter terms	0.00	0.02%
Generate term-document matrix	0.00	0.03%
Output OUTTERMS table	0.00	0.01%
Compute SVD	0.01	0.05%
Output OUTDOCPRO table	0.00	0.01%

Figure 5.3 shows the “Data Access Information” table, which provides the information about the data sets that the HPTMINE procedure has accessed and generated.

Figure 5.3 Data Access Information

Data Access Information			
Data	**Engine**	**Role**	**Path**
WORK.GETSTART	V9	Input	On Client
WORK.TERMS	V9	Output	On Client
WORK.DOCPRO	V9	Output	On Client

Figure 5.4 shows the SAS log that is generated by PROC HPTMINE; 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 terms data set contains 43 observations. This means that the HPTMINE procedure identified 43 individual terms in the input document collection. Because K=10 in the SVD statement, the docpro data set contains 11 variables: the first variable is the document ID, and the remaining 10 variables are obtained by projecting the original document to the 10 left singular vectors that are computed by singular value decomposition.

Figure 5.4 SAS Log

```
NOTE: No SPARSEFORMAT option is specified. SPARSEFORMAT=COO will be run by
      default.
NOTE: Stemming will be used in parsing.
NOTE: Tagging will be used in parsing.
NOTE: Noun groups will be used in parsing.
NOTE: No TERMWGT option is specified. TERMWGT=ENTROPY will be run by default.
NOTE: No CELLWGT option is specified. CELLWGT=LOG will be run by default.
NOTE: No ENTITIES option is specified. ENTITIES=NONE will be run by default.
NOTE: The HPTMINE procedure is executing in single-machine mode.
NOTE: There were 36 observations read from the data set WORK.GETSTART.
NOTE: The data set WORK.TERMS has 43 observations and 11 variables.
NOTE: The data set WORK.DOCPRO has 36 observations and 11 variables.
```

The following statements use PROC PRINT to show the contents of the first 10 rows of the docpro data set that is generated by the HPTMINE procedure:

```
proc print data = docpro (obs=10) round;
run;
```

Figure 5.5 shows the output of PROC PRINT. For information about the output of the OUTDOCPRO= option, see the section "The OUTDOCPRO= Data Set" on page 97.

Figure 5.5 The docpro Data Set

Obs	did	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8	COL9	COL10
1	d01	0.26	-0.74	0.06	-0.33	0.34	-0.26	0.08	0.11	-0.03	-0.26
2	d02	0.74	0.22	0.41	-0.13	-0.11	0.07	-0.07	-0.14	-0.41	0.03
3	d03	0.23	-0.21	-0.68	0.02	-0.15	0.43	-0.34	-0.13	0.04	-0.33
4	d04	0.75	-0.07	0.18	0.40	0.10	0.09	-0.31	0.12	0.17	0.29
5	d05	0.30	-0.24	0.37	-0.61	-0.17	0.19	-0.35	0.33	0.11	0.16
6	d06	0.10	-0.23	-0.35	-0.10	-0.75	0.29	0.20	-0.07	0.34	-0.03
7	d07	0.45	-0.37	0.17	0.40	-0.20	0.07	-0.11	-0.11	-0.61	0.20
8	d08	0.09	-0.15	-0.24	-0.33	0.27	0.10	-0.14	-0.76	0.14	0.32
9	d09	0.86	0.09	0.05	0.35	0.19	-0.12	-0.21	0.00	-0.03	0.18
10	d10	0.23	-0.03	-0.69	-0.12	0.19	0.05	0.07	0.57	-0.27	0.11

The following statements use PROC SORT and PROC PRINT to show the contents of the terms data set that is generated by the HPTMINE procedure. In the output, only the terms that have a key value from 1 to 35 are shown. And for these terms only the values of the variables term, role, freq, numdocs, key, and parent are displayed.

```
proc sort data = terms; by key; run;
proc print data = terms (obs=35);
var term role freq numdocs key parent;
run;
```

Figure 5.6 shows the output of PROC PRINT, which provides details about the terms that are identified by the HPTMINE procedure. For example, it shows that the key of "problems" is 32 and its parent's key is 9, which is the term "problem." The HPTMINE procedure also identified that "sas" is a proper noun and that "big data" and "high-performance analytics" are noun groups. For information about the output of the OUTTERMS= option, see the section "The OUTTERMS= Data Set" on page 98.

Figure 5.6 The terms Data Set

Obs	Term	Role	Freq	numdocs	Key	Parent
1	high-performance	Adj	3	3	1	.
2	big data	NOUN_GROUP	5	5	2	.
3	to	Prep	13	11	3	.
4	fast	Adj	4	4	4	4
5	fast	Adj	1	1	4	.
6	data	Noun	8	8	5	.
7	gain	Verb	2	2	6	.
8	big	Adj	6	6	7	.
9	for	Prep	7	7	8	.
10	problem	Noun	2	2	9	9
11	as	Prep	2	2	10	.
12	business	Noun	3	3	11	.
13	analytics	Prop	2	2	12	.
14	the	Det	13	13	13	.
15	time	Noun	2	2	14	.
16	in	Prep	3	3	15	.
17	process	Verb	2	2	16	16
18	and	Conj	13	13	17	.
19	insight	Noun	3	3	18	18
20	a	Det	3	3	19	.
21	compute	Verb	2	2	20	20
22	be	Verb	2	2	21	21
23	high-performance analytics	NOUN_GROUP	2	2	22	.
24	be	Aux	4	4	23	23
25	from	Prep	6	6	24	.
26	how	Conj	2	2	25	.
27	growth	Noun	2	2	26	.
28	it	Pron	2	2	27	.
29	enable	Verb	2	2	28	28
30	of	Prep	5	5	29	.
31	analytics	Noun	3	3	30	.
32	sas	Prop	7	7	31	.
33	problems	Noun	2	2	32	9
34	insights	Noun	3	3	33	18
35	enables	Verb	2	2	34	28

Syntax: HPTMINE Procedure

The following statements are available in the HPTMINE procedure:

> **PROC HPTMINE** < *options* > **;**
>   **VARIABLES** *variable* **;**
>   **TARGET** *variable* **;**
>   **DOC_ID** *variable* < *docid-option* > **;**
>   **PARSE** < *parse-options* > **;**
>   **SELECT** *label-list* /< **GROUP=***group-option* > **KEEP | IGNORE ;**
>   **SVD** < *svd-options* > **;**
>   **PERFORMANCE** < *performance-options* > **;**

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

PROC HPTMINE Statement

PROC HPTMINE < *options* > **;**

The PROC HPTMINE statement invokes the procedure. Table 5.1 summarizes the *options* in the statement by function. The *options* are then described fully in alphabetical order.

Table 5.1 PROC HPTMINE Statement Options

option	**Description**
Basic Options	
DATA | DOC=	Specifies the input document data set
LANGUAGE=	Specifies the language that the input data set of documents uses
Output Options	
NOPRINT	Suppresses ODS output
SPARSEFORMAT | SPFMT=	Specifies the format used for storing sparse matrices

You can specify the following *options*:

DATA=*SAS-data-set*

DOC=*SAS-data-set*

names the input SAS data set of documents to be used by PROC HPTMINE. The default is the most recently created data set. If PROC HPTMINE executes in distributed mode, the input data are distributed to memory on the appliance nodes and analyzed in parallel, unless the data are already distributed in the appliance database. When data are already distributed, PROC HPTMINE reads the data alongside the distributed database. For more information, see the sections “Processing Modes” on page 8 and “Alongside-the-Database Execution” on page 16 in Chapter 3, “Shared Concepts and

Topics." Each row of the input data set must contain one text field and one ID field 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 HPTMINE runs in SVD-only mode. In this mode, the DATA= option names the input SAS data set that contains the term-by-document matrix that is generated by the OUTPARENT= option in the PARSE statement. This functionality is supported when PROC HPTMINE runs in both single-machine and distributed mode.

LANGUAGE=*language*

names the language that is used by the documents in the input SAS data set. 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.

NOPRINT

suppresses the generation of ODS output.

SPARSEFORMAT=COO | BESR < (MAXTEXTLEN=*n* **) >**

SPFMT=COO | BESR < (MAXTEXTLEN=*n* **) >**

names the sparse format for storing the sparse matrix. You can specify the following values:

COO uses the coordinate list format.

BESR uses the Base64-encoded sparse row format.

When you specify SPARSEFORMAT=BESR, each row of the sparse matrix is stored as a string. You can use the MAXTEXTLEN= suboption to specify the maximum length of the text variable that will be used to store the string. If the length of the string is longer than *n*, multiple text variables are used so that the total length of the text variables is longer than the length of the string. By default, MAXTEXTLEN=32758.

By default, SPARSEFORMAT=COO.

DOC_ID Statement

DOC_ID *variable* < *docid-option* > **;**

The DOC_ID statement specifies the *variable* that contains the ID of each document. In the input data set, 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.

You can specify the following *docid-option*:

DUPLICATIONCHECK < (WARNING | STOP) >

DUPCHK < (WARNING | STOP) >

checks whether the ID of each document is unique. When you do not specify this option, no duplication check is performed. When you specify DUPLICATIONCHECK(WARNING), the HPTMINE procedure outputs a warning message that shows the number of duplicate document IDs that have been detected, unless the procedure finds no duplicate IDs, in which case no message is generated. When you specify DUPLICATIONCHECK(STOP), the HPTMINE procedure outputs an error message that shows the number of duplicate document IDs that have been detected, and then the procedure terminates. If you specify the DUPLICATIONCHECK option without the WARNING or STOP keyword, the HPTMINE procedure takes the WARNING keyword by default.

PARSE Statement

PARSE < *parse-options* > **;**

The PARSE statement specifies the options for parsing the input documents and creating the term-by-document matrix. Table 5.2 summarizes the *parse-options* in the statement by function. The *parse-options* are then described fully in alphabetical order.

Table 5.2 PARSE Statement Options

parse-option	**Description**
Parsing Options	
ENTITIES=	Specifies whether to extract entities in parsing
LITILIST=	Specifies the .li files to be used for custom entity extraction
MULTITERM=	Specifies the multiword term list
NAMEDFILE	Indicates that the variable specified in the VAR statement is a reference to an external file
NONOUNGROUPS | NONG	Specifies no noun group extraction in parsing
NOSTEMMING	Specifies no stemming in parsing
NOTAGGING	Specifies no part-of-speech tagging in parsing
SHOWDROPPEDTERMS=	Includes dropped terms in the OUTTERMS= data set
START=	Specifies the start list
STOP=	Specifies the stop list
SYNONYM | SYN=	Specifies the synonym list
Term-by-Document Matrix Creation Options	
CELLWGT=	Specifies how cells are weighted
REDUCEF=	Specifies the frequency for term filtering
TERMWGT=	Specifies how terms are weighted
Search Index	
BUILDINDEX	Builds an index for documents during parsing
INDEXPATH=	Specifies the file system location where index files are to be output
INDEXNAME=	Specifies the prefix name of the index files
Output Options	
OUTCHILD=	Specifies the data set to contain the raw term-by-document matrix. All kept terms, whether or not they are child terms, are represented in the data set along with their corresponding frequency.
OUTCONFIG=	Specifies the data set to contain the option settings that PROC HPTMINE uses in the current run

Table 5.2 *continued*

parse-option	**Description**
OUTPARENT=	Specifies the data set to contain the term-by-document matrix. Child terms are not represented in the data set. The frequencies of child terms are attributed to their corresponding parents.
OUTTERMS=	Specifies the data set to contain the summary information about the terms in the document collection
OUTPOS=	Specifies the data set to contain the position information about the child terms' occurrences in the document collection

You can specify the following *parse-options*.

BUILDINDEX

builds an index for documents during parsing. This option requests that the HPTMINE procedure build a search index that can be used in the TMUTIL procedure, which can then process queries against this index. The index consists of a set of files that are written to the same directory on the file system and whose names all start with the same prefix. This option is supported only in single-machine mode.

CELLWGT=LOG | NONE

specifies how the elements in the term-by-document matrix are weighted. The available cell weights are as follows:

LOG	specifies that cells are to be weighted by using the LOG formulation.
NONE	specifies that no cell weight be applied.

For information about the LOG formulation for cell weighting, see the section "Term and Cell Weighting" on page 90.

ENTITIES=STD | NONE

determines whether to use the standard LITI file for entity extraction, or not. You can specify the following values:

STD	uses the standard LITI file for entity extraction. Terms such as "George W. Bush" are 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. The LITILIST= option enables you to extract custom entities. When you specify the LITILIST= option and you specify ENTITIES=NONE, the HPTMINE procedure extracts only custom entitites by using the LITI files that are specified in the LITILIST= option; the default standard LITI file is ignored.

INDEXNAME=*"string"*

specifies the prefix of the files in the index. By default, INDEXNAME="tgindex".

INDEXPATH=*"string"*

specifies the file system location where index files are to be written. This directory must already exist. By default, the value for this option is the SAS working directory.

LITILIST=(*OS-file-reference***,** *OS-file-reference***, . . .)**

specifies the LITI files to use for entity extraction. LITI files contain custom entities. You can create as many LITI files as you need for customized entities extraction. You must specify the absolute path of each LITI file (for example, **`c:\litifiles\enities.li`**) as a separate *OS-file-reference* and use space between them.

The order of the *OS-file-references* defines precedence. The last *OS-file-reference* has the highest precedence, and the first *OS-file-reference* has the lowest precedence. If more than one match is found, the *OS-file-reference* that has the highest precedence is used. The default standard LITI file has the lowest prority compared to other LITI files that you specify in this option.

When you specify this option and you also specify ENTITIES=NONE, the HPTMINE procedure extracts only custom entitites by using the LITI files that are specified in the LITILIST= option; the default standard LITI file is ignored.

MULTITERM=*OS-file-reference*

specifies the path to a file that contains a list of multiword terms. These terms are case-sensitive and are treated as a single entry by the HPTMINE procedure. Thus, the terms "Thank You" and "thank you" are processed differently. Consequently, you must convert all text strings to lowercase or add each of the multiterm's case variations to the list before using the HPTMINE procedure to create consistent multiword terms. The multiterm file can be any file type as long as it is formatted in the following manner:

```
multiterm: 3: pos
```

Specifically, the first item is the multiword term itself followed by a colon, the second item is a number that represents the token type followed by a colon, and the third item is the part of speech that the multiword term represents. **NOTE:** The token type 3 is the most common token type for multiterm lists and represents compound words.

NAMEDFILE

indicates that the variable that is specified in the VAR statement refers to an external file that contains the text to parse[1]. Unlike most of the fields in the OUTCONFIG= data set, its value can change from training time to scoring time. In the current release this option is supported only in single-machine mode.

NONOUNGROUPS

NONG

suppresses standard noun group extraction. By default, noun groups are extracted and the HPTMINE procedure returns noun phrase without determiners or prepositions. Unless the NOSTEMMING option is specified, noun group elements are also stemmed.

[1]By default, the HPTMINE procedure assumes that this variable contains the text that is parsed.

There are two types of noun group extraction, standard and custom. Custom noun groups are still extracted when NONOUNGROUPS is specified and user-supplied LITI files are referenced in the LITILIST= option and contain the NOUN_GROUP entity type.

NOSTEMMING

requests that words not be stemmed. By default, words are stemmed; that is, terms such as "advises" and "advising" are mapped to the parent term "advise." The HPTMINE procedure uses dictionary-based stemming (also known as lemmatization).

NOTAGGING

requests that terms not be tagged. By default, terms are tagged and the HPTMINE 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 set.

OUTCHILD=*SAS-data-set*

specifies the data set to contain a compressed representation of the sparse term-by-document matrix. The term counts are not weighted. The data set saves only the kept, representative terms. The child frequencies are not attributed to their corresponding parent (as they are in the OUTPARENT= data set). For more information about the compressed representation of the sparse term-by-document matrix, see the section "Output Data Sets" on page 95.

OUTCONFIG=*SAS-data-set*

specifies the data set to contain configuration information that is used for the current run of PROC HPTMINE. The primary purpose of this data set is to relay the configuration information from the HPTMINE procedure to the HPTMSCORE procedure. The HPTMSCORE procedure is forced to use options that are consistent with the HPTMINE procedure. Thus, the data set that is created by using the OUTCONFIG= option becomes an input data set for PROC HPTMSCORE and ensures that the parsing options are consistent between the two runs. For more information about this data set, see the section "Output Data Sets" on page 95.

OUTPARENT=*SAS-data-set*

specifies the data set to contain a compressed representation of the sparse term-by-document matrix. The term counts can be weighted, if requested. The data set 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 "Output Data Sets" on page 95.

OUTTERMS=*SAS-data-set*

specifies the data set to contain the summary information about the terms in the document collection. For more information about this data set, see the section "Output Data Sets" on page 95.

OUTPOS=*SAS-data-set*

specifies the data set to contain the position information about the child terms' occurrences in the document collection. For more information about this data set, see the section "Output Data Sets" on page 95.

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 set and the OUTCHILD= data set.

START=*SAS-data-set*

specifies the data set that contains the terms that are to be kept for the analysis. These terms are displayed in the OUTTERMS= data set with a keep status of Y. All other terms have a keep status of N if the SHOWDROPPEDTERMS option is specified in the PARSE statement. The START data set must have a Term variable and can also have a Role variable. You cannot specify both the START and the STOP options.

STOP=*SAS-data-set*

specifies the data set that contains the terms to exclude from the analysis. These terms are displayed in the OUTTERMS= data set with a keep status of N, if the SHOWDROPPEDTERMS option is used in the PARSE statement. They are not identified as parents or children. The STOP data set must have a Term variable and can also have a Role variable. You cannot specify both the START and the STOP options.

SYNONYM=*SAS-data-set*

SYN=*SAS-data-set*

specifies the data set that contains user-defined synonyms to be used in the analysis. The data set specifies parent-child relationships that enable you to map child terms to a representative parent. The synonym relationship is indicated in the data set that is specified in the OUTTERMS= option and is also reflected in the term-by-document data set that is specified in the OUTPARENT= option. The input synonym data set must have either the two variables Term and Parent or the four variables Term, Parent, Termrole, and Parentrole. When stemming is turned on, this list overrides any relationships that are identified when terms are stemmed.

TERMWGT=ENTROPY | MI | NONE

specifies how terms are weighted. The available term weights are as follows:

ENTROPY	requests that terms be weighted using the entropy formulation.
MI	requests that terms be weighted using the mutual information formulation (requires 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 90.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables that describe the distributed computing environment, and requests detailed results about the performance characteristics of the HPTMINE procedure. You can also use the PERFORMANCE statement to control whether the HPTMINE procedure executes in single-machine or distributed mode. The

PERFORMANCE statement is documented further in the section "PERFORMANCE Statement" on page 33 of Chapter 3, "Shared Concepts and Topics."

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 field in the OUTTERMS= data set. Terms that are excluded by the SELECT statement cannot be included in the OUTTERMS= data set, whereas terms that have _keep=N can be included in the OUTTERMS= data set if the SHOWDROPPEDTERMS option is specified. Terms excluded by the SELECT statement are excluded from the OUTPOS= data set, but terms that have _keep=N are included in OUTPOS= data set. Table 5.3 summarizes the options you can specify in the SELECT statement. The options are then described fully in syntactic order.

Table 5.3 SELECT Statement Options

Option	Description
label-list	Specifies one or more labels of terms that are to be ignored or kept in your analysis
GROUP=	Specifies whether the labels are parts of speech, entities, or attributes
IGNORE	Ignores the terms whose labels are specified in the *label-list*
KEEP	Keeps the terms whose labels are specified in the *label-list*

You must specify a *label-list* and either the IGNORE or KEEP option:

label-list

specifies one or more labels that are either parts of speech or entities or attributes. Each label must be surrounded by double quotation marks and separated by spaces from other labels. Labels are case-insensitive. Terms that have these labels are either ignored during parsing (when the IGNORE option is specified) or kept in the parsing results in the OUTPOS= and OUTTERMS= data sets (when the KEEP option is specified). Table 5.5 shows all possible part-of-speech tags. Table 5.6 shows all valid English entities. The attribute field in Table 5.11 shows all possible attributes.

IGNORE

ignores during parsing all terms whose labels are specified in the *label-list*, but keeps all other terms in the parsing results (the OUTPOS= and OUTTERMS= data sets).

KEEP

keeps only the terms whose labels are specified in the *label-list* in the parsing results (the OUTPOS= and OUTTERMS= data sets).

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 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 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 was created during the parsing phase of PROC HPTMINE. Table 5.4 summarizes the *svd-options* in the statement by function. The *svd-options* are then described fully in alphabetical order.

Table 5.4 SVD Statement Options

svd-option	**Description**
Input Options	
COL=	Specifies the column variable, which contains the column indices of the term-by-document matrix that is stored in COO format
ROW=	Specifies the row variable, which contains the row indices of the term-by-document matrix that is stored in COO format
ENTRY=	Specifies the entry variable, which contains the entries of the term-by-document matrix that is stored in COO format
SVD Computation Options	
K=	Specifies the number of dimensions to be extracted
MAX_K=	Specifies the maximum number of dimensions to be extracted
TOL=	Specifies the maximum allowable tolerance for the singular value
RESOLUTION | RES=	Specifies the recommended number of dimensions (resolution) to be extracted by SVD, when the MAX_K= option is used
Topic Discovery Options	
TOPICLABELSIZE=	Specifies the number of terms to be used in the descriptive label for each topic
ROTATION=	Specifies the type of rotation to be used for topic discovery
IN_TERMS=	Specifies the data set that contains the terms for topic discovery in SVD-only mode
Output Options	
SVDU=	Specifies the U matrix, which contains the left singular vectors
SVDV=	Specifies the V matrix, which contains the right singular vectors
SVDS=	Specifies the S matrix, whose diagonal elements are the singular values

Table 5.4 *continued*

svd-option	**Description**
OUTDOCPRO=	Specifies the data set to contain the projections of the documents
OUTTOPICS=	Specifies the data set to contain the topics that have been discovered

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 HPTMINE in SVD-only mode (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 HPTMINE in SVD-only mode (that is, when you specify the SVD statement but not the PARSE statement).

IN_TERMS=*SAS-data-set*

specifies the data set that contains information about the terms in the document collection. The data set should have the fields described in Table 5.11. The terms are required to generate topic names in the OUTTOPICS= data set. 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 set after the SVD is performed. If the value of *k* is too large, then the HPTMINE procedure runs for an unnecessarily long time. You cannot specify both this option and the MAX_K= option. This option also controls the number of topics that are extracted from the text corpus when the ROTATION= option is specified.

MAX_K=*n*

specifies the maximum value that the HPTMINE procedure should return as the recommended value of *n*. If the RESOLUTION= option is specified (to recommend the value of *k*), this option limits that value to at most *n*. The HPTMINE procedure attempts to calculate (as opposed to recommend) *k* dimensions when it performs SVD. You cannot specify both this option and the K= option. This option also controls the number of topics that are extracted from the text corpus when the ROTATION= option is specified.

OUTDOCPRO=*SAS-data-set* **<KEEPVARIABLES=***variable-list***><NONORMDOC>**

OUTDOCPRO=*SAS-data-set* **<KEEPVARS=***variable-list***><NONORMDOC>**

specifies the data set 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.

You can copy the variables from the data set that is specified in the DATA= option in the PROC HPTMINE statement to the *SAS-data-set* that is specified in this option. When you specify KEEP-VARIABLES=*variable-list*, the content of the variables that is specified in the *variable-list* is attached to the output. These variables must appear in the data set that is specified in the DATA= option in the PROC HPTMINE statement.

When you specify NONORMDOC, the columns that contain the projections of documents are not normalized to have a unit norm.

OUTTOPICS=*SAS-data-set*
specifies the data set to contain the topics that are discovered.

RESOLUTION=LOW | MED | HIGH
RES=LOW | MED | HIGH
specifies 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 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 $(5/6) * t\%$ of the total variance.
LOW	recommends a k that explains $(2/3) * t\%$ 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 use 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 HPTMINE in SVD-only mode (when you specify the SVD statement but not the PARSE statement).

SVDS=*SAS-data-set*
specifies the data set to contain the calculated singular values.

SVDU=*SAS-data-set*
specifies the data set to contain the calculated left singular vectors.

SVDV=*SAS-data-set*
specifies the data set to contain the calculated right singular vectors.

TOL=ϵ

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

TOPICLABELSIZE=*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 set. By default, TOPICLABELSIZE=5.

TARGET Statement

TARGET *variable* **;**

This statement specifies the *variable* that contains the information about the category that a document belongs to. A target variable can be any nominal or ordinal variable. It is used in calculating mutual information term weighting.

VARIABLES Statement

VARIABLES *variable* **;**

VAR *variable* **;**

This statement specifies the *variable* that contains the text to be processed.

When you specify the SVD statement but not the PARSE statement, PROC HPTMINE runs in SVD-only mode. In this case, the DATA= option in the PROC HPTMINE statement names the input SAS data set that contains the term-by-document matrix, which is generated by the OUTPARENT= option in the PARSE statement. If the term-by-document-matrix is stored in the BESR format, you need to use the VARIABLE statement to specify the variables that contain strings that are in the BESR format. The variables can be specified in any order. This functionality is supported in either single-machine mode or distributed mode.

Details: HPTMINE Procedure

PROC HPTMINE integrates the functionalities from both natural language processing and statistical analysis to provide essential support for high-performance text mining. This section provides details about various aspects of the HPTMINE 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 HPTMINE 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 HPTMINE 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, the stem selection process restricts the stem to be of the same part-of-speech as the original term.

Part-of-Speech Tagging

Using SAS linguistic technologies, tagging identifies, or disambiguates, 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 5.5 shows all possible part-of-speech tags.

Table 5.5 All Part-of-Speech Tags

Part-of-Speech Tag	**Description**
ABBR	Abbreviation
ADJ	Adjective
ADV	Adverb
AUX	Auxiliary or modal term
CONJ	Conjunction
DET	Determiner
INTERJ	Interjection
NOUN	Noun
NUM	Number or numeric expression
PART	Infinitive marker, negative participle, or possessive marker
PREF (Korean only)	Prefix
PREP	Preposition
PRON	Pronoun
PROP	Proper noun
PUNCT	Punctuation
VERB	Verb
VERBADJ	Verbal adjective

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 5.6 shows all valid entities for English. Not all languages support all entities. Table 5.7, Table 5.8, and Table 5.9 indicate the languages that are available for each entity.

Table 5.6 All Valid English Entities

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

Table 5.7 Supported Language-Entity Pairs, Part 1

Language	Address	Company	Currency	Date	Internet	Location
Chinese						✓
Dutch	✓	✓	✓	✓	✓	✓
English	✓	✓	✓	✓	✓	✓
Finnish						
French	✓	✓	✓	✓	✓	✓
German	✓	✓	✓	✓	✓	✓
Italian	✓	✓	✓	✓	✓	✓
Japanese	✓		✓	✓	✓	✓
Korean						✓
Portuguese	✓	✓	✓	✓	✓	✓
Russian						
Spanish	✓	✓	✓	✓	✓	✓
Turkish						

Table 5.8 Supported Language-Entity Pairs, Part 2

Language	Measure	Noun_Group	Organization	Percent	Person	Phone
Chinese			✓		✓	
Dutch	✓	✓	✓	✓	✓	✓
English	✓	✓	✓	✓	✓	✓
Finnish		✓				
French	✓	✓	✓	✓	✓	✓
German	✓	✓	✓	✓	✓	✓
Italian	✓	✓		✓	✓	✓
Japanese	✓		✓	✓	✓	✓
Korean			✓		✓	
Portuguese	✓	✓		✓	✓	✓
Russian		✓				
Spanish	✓	✓	✓	✓	✓	✓
Turkish		✓				

Table 5.9 Supported Language-Entity Pairs, Part 3

Language	Prop_Misc	SSN	Time	Time_Period	Title	Vehicle
Chinese						
Dutch	✓	✓	✓	✓	✓	✓
English	✓	✓	✓	✓	✓	✓
Finnish						
French	✓	✓	✓	✓	✓	
German	✓	✓	✓	✓	✓	
Italian	✓	✓	✓	✓	✓	
Japanese			✓	✓	✓	
Korean						
Portuguese	✓	✓	✓	✓	✓	
Russian						
Spanish	✓	✓	✓	✓	✓	
Turkish						

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 HPTMINE 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 ith row and jth 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:

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

In this equation, n is the number of documents, and $p_{i,j}$ is the probability that term i appears in document j, which can be estimated by $p_{i,j} = \frac{f_{i,j}}{g_i}$, where g_i is the global term frequency for term i.

When the TERMWGT=MI option is specified, the following equation is used to weight terms:

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

In this equation, C_k is the set of documents that belong to category k, $P\,(C_k)$ is the percentage of documents that belong to category k, and $P\,(t_i, C_k)$ is the percentage of documents that contain term t_i and belong to category k. Let d_i be the number of documents that term i appears in. Then $P\,(t_i) = \frac{d_i}{n}$.

Sparse Format

A matrix is sparse when most of its elements are 0. The term-by-document matrix that the HPTMINE procedure generates is a sparse matrix. To save storage space, the HPTMINE procedure supports two sparse formats for storing a sparse matrix: the COO format and the BESR format.

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. Storing this data structure in a relational database is straightforward because the database table needs to have only three columns and the number of rows is equal to the total nonzero elements in the matrix. In distributed computing, most distributed matrix computation operations have a grouping requirement. That is, they require that elements from the same row of the matrix be distributed to the same node of the grid and to be grouped together. When a sparse matrix is stored in the COO format in distributed mode, the grouping requirement might not be met when the data are distributed to the grid. To enforce the grouping, the row indices should be used as distribution keys to distribute the data.

The following example shows how to use the HPDS2 procedure to distribute sparse data in the COO format to a grid that runs a Teradata database system:

```
option set=GRIDHOST      ="&your_host";
option set=GRIDINSTALLLOC="&your_grid_instloc";
option set=GRIDDATASERVER="&your_data_server";

libname TDLib teradata
    server  =&your_data_server
    user    =&usr
    password=&pwd
    database=&db;

/* load data using the HPDS2 procedure */
proc hpds2 data= matrix_coo
    out=TDLib.matrix_coo(dbcreate_table_opts='PRIMARY INDEX (row_id)');
    data DS2GTF.out;
        method run();
            set DS2GTF.in;
            output;
        end;
    enddata;
run;
```

The DBCREATE_TABLE_OPTS= option in the DATA step specifies the distribution key. For the Teradata database system, the value of this option is “PRIMARY INDEX (row_id),” which tells the Teradata system to use the row_id column for distributing the data. This ensures that all elements that have the same row_id value are distributed to the same node. You can use similar code to load your data to a grid system that runs a Greenplum database system by specifying DBCREATE_TABLE_OPTS=“DISTRIBUTED BY (row_id).” For the SASHDAT and the SASIOLA engine, you can specify the distribution key by using the PARTITION option as follows:

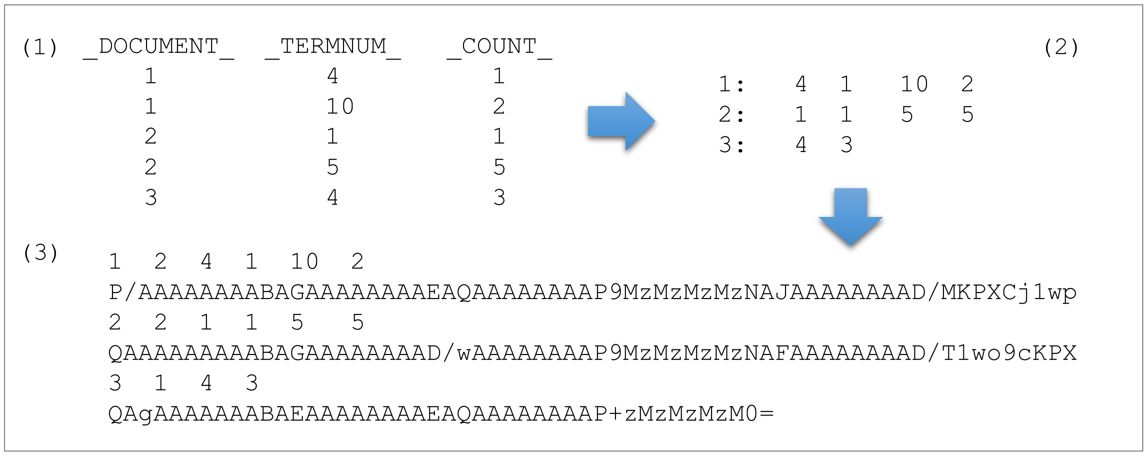
```
data lasr.docs_by_terms (partition=(row_id));
 set docs_by_terms;
run;
```

Not all distributed database systems can distribute data by using a distribution key. For those situations you should use BESR format, which has one observation per row.

Base64-Encoded Sparse Row (BESR) Format

This format encodes each row of the sparse design matrix as a string and stores the string in one or a few text variables in a data table. This sparse format enables you to store your sparse design matrix in any relational database. Because each row of the sparse matrix is stored in one row of the data table, distributing data to the grid cannot break the grouping requirment. The following figure illustrates an example of how three rows of a sparse matrix are converted to three strings by using the BESR format.

Figure 5.7 Base64-Encoded Sparse Row Format



```
(1)  _DOCUMENT_   _TERMNUM_   _COUNT_                                    (2)
         1            4          1
         1            10         2               1:   4  1   10  2
         2            1          1               2:   1  1   5   5
         2            5          5               3:   4  3
         3            4          3

(3)    1  2  4  1  10  2
       P/AAAAAAAABAGAAAAAAAAAEAQAAAAAAAAP9MzMzMzMzNAJAAAAAAAAAD/MKPXCj1wp
       2  2  1  1  5   5
       QAAAAAAAAABAGAAAAAAAAAD/wAAAAAAAAP9MzMzMzMzNAFAAAAAAAAAD/T1wo9cKPX
       3  1  4  3
       QAgAAAAAAABAEAAAAAAAAAEAQAAAAAAAAP+zMzMzMzM0=
```

Given a sparse design matrix, the BESR format first forms an array for each row of the sparse matrix. Each array starts with its row index, which is followed by the column index and value pairs. Before encoding, some meta-information is added to each array after the row index. In this example, the meta-information is the number of column and value pairs, which are equal to the number of nonzero elements in each row of the sparse design matrix. To make the BESR representation platform-independent, the array is then converted to an array that contains doubles in the IEEE 754 format. The resulting double arrays are finally encoded using the Base64 encoding.

The BESR format has the following advantages:

1. Strings are easy to store in SAS and database systems:
   a) In SAS, the maximum length for text variable is 32 KB, which can be used to store more than 3,000 doubles. And in database systems, the maximum length is usually longer than 32 KB.
   b) If the text string is too large, you can use several text variables to store it.
   c) SAS and most database systems can compress text variables.
2. The BESR format is independent of platform:
   a) Doubles in the array are converted from their native format to the portable IEEE 754 format before they are encoded to a string by using the Base64 encoding.
   b) Base64 encoding uses 64 characters that are common to all encoding systems. Therefore, there are no encoding compatibility issues when data are transported and stored in different systems.
   c) All 64 characters appear on a regular keyboard; thus they can be easily viewed.

3. There is no data integrity issue when data are distributed in a grid environment:
    a) Elements from the same sparse row are always grouped together.

The following example uses the HPTMINE procedure to process a text data set and store the sparse term-by-document matrix in the BESR format. It then reads in the stored term-by-document matrix and applies singular value decomposition in SVD-only mode.

```
option set=GRIDHOST      ="&your_host";
option set=GRIDINSTALLLOC="&your_grid_instloc";

proc hptmine data=&your_text spfmt=BESR;
    doc_id &doc_id; var &text_var;
    parse termwgt=entropy cellwgt=log stop=sashelp.engstop
      outterms=keys outparent=docs_by_terms_besr (compress=yes);
    performance nodes=4 details;
run;

proc hptmine data=docs_by_terms_besr spfmt=BESR;
    doc_id &doc_id; var sparse_row:;
    svd k=10 res=med outdocpro=docpro_svdonly;
    performance nodes=2 details;
run;
```

In the first PROC HPTMINE call, the SPFMT= option specifies the sparse format for storing the sparse term-by-document matrix that is generated by the HPTMINE procedure. In the docs_by_terms_besr data set that is generated by the procedure, each Base64-encoded sparse row is stored in one or a few text variables whose names include the prefix SPARSE_ROW_. In the second HPTMINE call, the docs_by_terms_besr data set is delivered to the procedure as input. The VAR statement defines the variables that are used to store the Base64-encoded sparse rows.

Singular Value Decomposition

Singular value decomposition (SVD) of a matrix $\mathbf{A}$ factors $\mathbf{A}$ into three matrices such that $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top$. The singular value decomposition also requires that the columns of $\mathbf{U}$ and $\mathbf{V}$ be orthogonal and that Σ be a real-valued diagonal matrix with monotonically decreasing, nonnegative entries. The entries of Σ are called singular values. The columns of $\mathbf{U}$ and $\mathbf{V}$ 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 $\mathbf{A} \in \mathbb{R}^{m \times n}$ 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 $\mathbf{A}$, and to project the columns of $\mathbf{A}$ 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 $\mathbf{A}$, for which you specify the parameter k to define how many singular values and singular vectors to compute. A 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 or accept a value that is recommended by the HPTMINE 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 $\mathbf{A}$ 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 $\mathbf{A}$ directly.

For example, consider the columns of $\mathbf{A}$, which represent the document space. By construction, the columns of $\mathbf{U}$ also reside in m-dimensional space. If $\mathbf{U}$ 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 $\mathbf{U}$ has two columns, then these columns would form the best fit plane to the original document space. In general, the first k columns of $\mathbf{U}$ form the best fit k-dimensional subspace for the document space. Thus, you can project the columns of $\mathbf{A}$ onto the first k columns of $\mathbf{U}$ in order to optimally reduce the dimension of the document space from m to k.

The projection of a document $\mathbf{d}$ (one column of $\mathbf{A}$) onto $\mathbf{U}$ results in k real numbers that are defined by the inner product $\mathbf{d}$ with each column of $\mathbf{U}$. That is, $p_i = \mathbf{d}^\top \mathbf{u}_i$. 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 $\mathbf{U}$ 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 $\mathbf{A}$ and the first k columns of $\mathbf{V}$. 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 documents into meaningful representations of the entire document collection.

Computation

The computation of the singular vector decomposition is fully parallelized in PROC HPTMINE to support both single-machine mode via multithreading and distributed model via 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 HPTMINE without a PARSE statement, PROC HPTMINE 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 HPTMINE in SVD-only mode, the DATA= option in the PROC HPTMINE statement names the data set that contains the term-by-document matrix.

When you specify SPFMT=COO in the PROC HPTMINE statement, you must specify the COL=, ROW=, and ENTRY= options in the SVD statement to name the variables that contain the column indices, the row indices, and the entries of the term-by-document matrix, respectively. The following statements run the HPTMINE procedure in the SVD-only model and specify SPFMT=COO:

```
proc hptmine data=docs_by_terms;
    svd k=10 row=_termnum_ col=_document_ entry=_count_ outdocpro=docpro;
    performance details;
run;
```

When SPFMT=BESR, you must use the VAR statement to specify the variables that contain the string for the BESR format. You can specify these variables in any order in the VAR statement. The following statements run the HPTMINE procedure in the SVD-only model and specify SPFMT=BESR:

```
proc hptmine data=docs_by_terms_besr spfmt=BESR;
    doc_id &doc_id;
    var sparse_row:;
    svd k=10 outdocpro=docpro_svdonly;
    performance nodes=2 details;
run;
```

In the preceding HPTMINE call, the VAR statement specifies that all variables with **sparse_row** as a prefix contain the string for the BESR format.

Topic Discovery

You can use the HPTMINE procedure to discover topics that exist in your collection. In PROC HPTMINE, 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 **V** matrix that has been rotated to maximize the explanatory power of each topic. The columns of the **U** matrix characterize the strength of the association of each document with each topic. Finally, the HPTMINE procedure can output a topic table that contains the best set of descriptor terms for each topic.

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

Output Data Sets

This section describes the output data sets that PROC HPTMINE produces when you specify the corresponding option.

The OUTCONFIG= Data Set

The OUTCONFIG= option specifies a SAS data set to contain the configuration that PROC HPTMINE uses in the current run. The primary purpose of this data set is to relay the configuration information from the HPTMINE procedure to the HPTMSCORE procedure. Thus, the HPTMSCORE procedure can use the options that are consistent with the HPTMINE procedure during scoring.

Table 5.10 shows the configuration information that is contained in this data set.

Table 5.10 Fields in the OUTCONFIG= Data Set

Field	Indicates
namedfile	Whether the parsed variable points to an external file or contains the input text
language	The source language of the documents
encoding	The encoding of the documents (this field is not applicable in the current release)
plugin	The plug-in that is used to parse the document collection ("Teragram" is the only supported plug-in)
stemming	Whether stemming is used: "Y" indicates that stemming is used, and "N" indicates that it is not used
tagging	Whether tagging is used: "Y" indicates that tagging is used, and "N" indicates that it is not used
NG	Whether noun group is used: "Y" indicates that noun group is used, and "N" indicates that it is not used
entities	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.
entList	The entities that are kept or ignored
attributes	Whether attributes are kept or ignored: "K" indicates that attributes are kept, "D" indicates that attributes are ignored, and "Y" indicates that no SELECT statement is specified for attributes
attrList	The attributes that are kept or ignored
pos	Whether parts of speech are kept or ignored: "K" indicates that parts of speech are kept, "D" indicates that parts of speech are ignored, and "Y" indicates that no SELECT statement is specified for parts of speech
posList	The parts of speech that are kept or ignored
multiterm	The path to the multiterm list file
litiList	The path to your custom entities
cellwgt	How the cells of the term-by-document matrix are weighted
nonormdoc	Whether document projections are normalized: "Y" indicates document projects are normalized to have unit norm
defaultentitiespriority	The priority of the default standard LITI file when both custom and default standard LITI files are used for entity extraction. In the current release, "1" is the only valid value for this field, meaning that the default standard LITI file has the lowest priority compared to other LITI files that you specify in the LITILIST= option.

The content of this data set is case-sensitive.

The OUTDOCPRO= Data Set

The OUTDOCPRO= option in the SVD statement specifies a SAS data set 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 with much lower dimensionality. If the K= option in the SVD statement is set to k and the input data set 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 Set

The OUTPARENT= option in the PARSE statement specifies a SAS data set to contain a compressed representation of the sparse term-by-document matrix. The term-by-document matrix is usually very sparse.[2] To save space, this matrix is stored in either COO format or BESR format.

When the data set that is specified in the OUTPARENT= option is stored in the COO format, it 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 set 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 set saves only the terms that are marked as kept in the data set that is specified in the OUTTERMS= option of the PARSE statement. In the data set the child frequencies are attributed to the corresponding parent. For example, if "said" has term ID `t1` and appears eight times in the document `d1`, "say" has term ID `t2` and appears one time in the document `d1`. Assume that "say" is the parent of "said" and that neither cell weighting nor term weighting is applied. Then the data set that is specified in the OUTPARENT= option will have an entry

```
t2    d1    9
```

The term count of "said" in `d1` is attributed to its parent, "say."

When the data set that is specified in the OUTPARENT= option is stored in BESR format, it contains the variable that is specified in the DOC_ID statement and one or more text variables for storing the strings that are generated by using the BESR format. These variables use "SPARSE_ROW_" as the prefix of their name.

The OUTCHILD= Data Set

The OUTCHILD= option specifies the data set 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 either COO format or BESR format in the same way as described in the previous section.

If you do not specify the SHOWDROPPEDTERMS option in the PARSE statement, the data set saves only the kept terms[3].

[2]Many elements of the matrix are 0.

[3]Terms that are marked as kept in the data set that is specified in the OUTTERMS= option in the PARSE statement.

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

```
t1    d1    8
t2    d1    1
```

The term count of "said" in `d1` is not attributed to its parent, "say." The data set that is specified in the OUTCHILD= option can be combined with the data set that is specified in the OUTTERMS= option to construct the data set that is specified in the OUTPARENT= option.

When you specify the SHOWDROPPEDTERMS option in the PARSE statement, the data set saves all the terms that appear in the data set that is specified in the OUTTERMS= option of the PARSE statement.

The OUTTERMS= Data Set

The data set specified in the OUTTERMS= option contains the summary information about the terms in the document collection. Table 5.11 shows the fields in this data set.

Table 5.11 Fields in the OUTTERMS= Data Set

Field	Description	
Term	A lowercase version of the term	
Role	The term's part of speech (this variable is empty if the NOTAGGING option is specified in the PARSE statement)	
Attribute	An indication of the characters that compose the term. Possible attributes are as follows:	
	Alpha	only alphabetic characters
	Mixed	a combination of attributes
	Num	only numbers
	Punct	punctuation characters
	Entity	an identified entity
Freq	The frequency of a term in the entire document collection	
numdocs	The number of documents that contain the term	
_keep	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 set is of a reasonable size, only terms that have _keep=Y are kept in the OUTTERMS= data set by default.	
Key	The assigned term number (each unique term in the parsed documents and each unique parent term has a unique Key value)	

Table 5.11 *continued*

Field	Description
Parent	The Key value of the term's parent or a "." (period): • If a term has a parent, this variable gives the term number of that parent. • If a term does not have a parent, this value is a "." (period). • If the values of Key, Parent, and Parent_id are identical, the parent occurs as itself. • If the values of Parent and Parent_id are identical but differ from Key, the observation is a child.
Parent_id	Another description of the term's parent: Parent gives the parent's term number if a term is a child, but Parent_id gives this value for all terms.
_ispar	An indication of term's status as a parent, child, or neither: • A "+" (plus sign) indicates that the term is a parent. • A "." (period) indicates that the term is a child. • A missing value indicates that the term is neither a parent nor a child.
Weight	The weights of the terms

If you do not specify the SHOWDROPPEDTERMS option in the PARSE statement, the data set saves only the terms that have _keep=Y. This helps ensure that the OUTTERMS= data set is of a reasonable size. When you specify the SHOWDROPPEDTERMS option, the data set also saves terms that have _keep=N.

The OUTPOS= Data Set

The data set that is specified in the OUTPOS= option in the PARSE statement contains the position information about the child terms' occurrences in the document collection. Table 5.12 shows the fields in this data set.

Table 5.12 Fields in the OUTPOS= Data Set

Field	Description
TERM	A lowercase version of the term
ROLE	The term's part of speech (this variable is empty if the NOTAGGING option is specified in the PARSE statement)
PARENT	A lowercase version of the parent term
START	The starting position of the term's occurrence (the first position is 0)
END	The ending position of the term's occurrence
SENTENCE	The sentence where the occurrence appears

Table 5.12 *continued*

Field	Description
PARAGRAPH	The paragraph where the occurrence appears (this has not been implemented in the current release, and the value is always set to 0)
DOCUMENT	The ID of the document where the occurrence appears

If you exclude terms by specifying the IGNORE option in the SELECT statement, then those terms are excluded from the OUTPOS= data set. No synonym lists, start lists, or stop lists are used when generating the OUTPOS= data set.

The OUTTOPICS= Data Set

If you specify the ROTATION= option in the SVD statement, the OUTTOPICS= option specifies the data set for storing the topics that have been discovered. This data set 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 range 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 set 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 with the highest term loadings once the rotation has been performed. The number of terms that are used in the generated name is determined by the TOPICLABELSIZE= option in the SVD statement.

Displayed Output

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

Performance Information

The "Performance Information" table is produced by default; it displays information about the grid host for distributed execution and information about whether the procedure executes in single-machine mode, distributed mode, or alongside-the-database mode. The numbers of computing nodes and threads are also displayed, depending on the environment.

Procedure Task Timing

When the DETAILS option is specified in the PERFORMANCE statement, PROC HPTMINE produces a "Procedure Task Timing" table, which displays the elapsed time (absolute and relative) for the main tasks.

ODS Table Names

Each table that is created by the HPTMINE procedure has a name associated with it, and you must use this name to refer to the table when you use ODS statements. These names are listed in Table 5.13.

Table 5.13 ODS Tables Produced by PROC HPTMINE

Table Name	Description	Required Statement / Option
PerformanceInfo	Information about the high-performance computing environment	Default output
Timing	Absolute and relative times for tasks performed by the procedure	DETAILS option in the PERFORMANCE statement

System Configuration

Prerequisites for Running PROC HPTMINE

To use the HPTMINE procedure, you must have a valid SAS Text Miner license, and the language binary files that are provided under that license must be available to PROC HPTMINE for parsing text. It is critical that you account for the encoding and text length of the character variables that you use for storing your text data, particularly for data sets that are predistributed on various databases. Different database systems might store character variables in different encodings, and the maximum length of character variables might also be different. Failure to configure your SAS client appropriately might result in character corruption or truncated documents. To avoid any potential transcoding issue, you need to use a SAS session encoding that is compatible with the encoding that your text data uses. For more information about proper configuration, see the appropriate SAS/ACCESS documents.

Accessing Input Data

When you run PROC HPTMINE in distributed mode, input data sets that are specified by the following options need to be accessible to the client machine:

- MULTITERM=
- START=
- STOP=
- SYNONYM=

When the HPTMINE procedure runs in distributed mode, it first reads these data sets on the client machine and then sends them to the grid.

Configuring for Distributed Mode

When PROC HPTMINE runs in distributed mode, it 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, and the GRID_TEXTANALYTICS_BIN_LOC macro variable must be specified to indicate the location of the binary files on the grid.

Deploying Language Binary Files on the Grid

The language binary files must be deployed to the grid in one of the following ways:

- For grid systems that consist of Linux nodes and run SAS Text Miner 13.1 or later, the SAS grid installation script can automatically install the language binary files.
- For earlier releases or other grid systems, 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 HPTMINE runs, and this can be time-consuming. Consult your grid administrator for a recommended binary location on the grid.

Language Binary Files

If you have licensed SAS Text Miner, the language binary files are originally installed in the following locations:

- In a Windows environment, the files are installed in the subdirectory **`tktg\sasmisc`** under **`$SASROOT`**. For example: **`C:\Program Files\SASHome\SASFoundation\9.4\tktg\sasmisc`**.
- In a UNIX environment, the files are installed in **`$SASROOT/misc/tktg`**.

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-AnaInfSinFas.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 file names begin with the corresponding two-letter language codes. For any language 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 HPTMINE procedure where to find these binary files. This macro is required only when you run PROC HPTMINE on the grid. When you run PROC HPTMINE in single-machine mode, the language binary files that were installed in the **$SASROOT** location during your SAS Text Miner installation are used for parsing text.

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 HPTMINE the location of the language binary files, insert the following statement before calling the procedure:

```
%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 HPTMINE procedure goes to the directory **/local_dir/tktg/misc** on each node to load the binary files:

```
%let GRID_TEXTANALYTICS_BIN_LOC=/local_dir/tktg/misc;
```

Examples: HPTMINE Procedure

Example 5.1: Parsing with No Options Turned On

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

```
/* 1)  create data set */

data CarNominations;
infile cards delimiter='|' missover;
length text $70 ;
input text$ i;
cards;
    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 HPTMINE to parse the documents. Because no host for distributed computing is specified and the NODES option is not specified in the PERFORMANCE statement, PROC HPTMINE runs automatically in single-machine mode.

```
/* 2) starting code */
proc hptmine data=work.CarNominations;
doc_id i;
var text;
parse
    nostemming notagging nonoungroups
    termwgt        = none
    cellwgt        = none
    reducef        = 0
    entities       = none
    outparent      = outparent
    outterms       = outterms
    outchild       = outchild
    outconfig      = outconfig
    ;
performance details;
run;

/* 3) print outterms data set */
proc print data=outterms; title "OUTTERMS Data Set"; run;
```

Output 5.1.1 shows the content of the outterms data set. In this example, none of the following parsing options are turned on: stemming, part-of-speech tagging, noun group extraction, entity identification, term and cell weighting, and term filtering. No synonym list, multiterms word list, or stop list is specified. As a result of this configuration, there is no child term in the outterms data set. Also, the outparent data set and the outchild data set are exactly the same. The HPTMINE procedure automatically drops punctuation and numbers. For example, in the outterms data set the "." (period) term has a "keep" value of "N".

Output 5.1.1 The outterms Data Set

OUTTERMS Data Set

Obs	Term	Role	Attribute	Freq	numdocs	_keep	Key	Parent	Parent_id	_ispar	Weight
1	all		Alpha	1	1	Y	1	.	1		1
2	toyota		Alpha	2	1	Y	2	.	2		1
3	ford		Alpha	2	2	Y	3	.	3		1
4	tacoma		Alpha	1	1	Y	4	.	4		1
5	year		Alpha	3	3	Y	5	.	5		1
6	taurus		Alpha	2	2	Y	6	.	6		1
7	won		Alpha	1	1	Y	7	.	7		1
8	honda		Alpha	1	1	Y	8	.	8		1
9	bright		Alpha	1	1	Y	9	.	9		1
10	sold		Alpha	2	2	Y	10	.	10		1
11	colors		Alpha	1	1	Y	11	.	11		1
12	lime		Alpha	1	1	Y	12	.	12		1
13	except		Alpha	1	1	Y	13	.	13		1
14	hyundai		Alpha	1	1	Y	14	.	14		1
15	in		Alpha	3	3	Y	15	.	15		1
16	is		Alpha	2	2	Y	16	.	16		1
17	for		Alpha	1	1	Y	17	.	17		1
18	world		Alpha	2	2	Y	18	.	18		1
19	green		Alpha	2	2	Y	19	.	19		1
20	the		Alpha	8	5	Y	20	.	20		1
21	of		Alpha	2	2	Y	21	.	21		1
22	award		Alpha	1	1	Y	22	.	22		1
23	was		Alpha	1	1	Y	23	.	23		1
24	car		Alpha	2	2	Y	24	.	24		1
25	insight		Alpha	1	1	Y	25	.	25		1
26	last		Alpha	1	1	Y	26	.	26		1

Example 5.2: Parsing with Stemming Turned On

This example uses the data set that is generated in Example 5.1. The following statements run PROC HPTMINE to parse the documents. Because the NOSTEMMING option is not specified in the PARSE statement, words are stemmed (the default).

```
proc hptmine data=work.CarNominations;
doc_id i;
var text;
parse
    notagging nonoungroups
    termwgt  = none
    cellwgt  = none
    reducef  = 0
    entities = none

    outparent= outparent
    outterms = outterms
    outchild = outchild
    outconfig= outconfig
    ;
performance details;
run;

proc print data = outterms; title "OUTTERMS Data Set"; run;
```

Output 5.2.1 shows the content of the outterms data set. In this example, words are stemmed. You can see that the term “sold” now stems to have the parent term “sell.” Also, the outparent data set and the outchild data set are different. The parent term “sell” shows up in outparent (key=11), but not the child term “sold” (key=26). Only “sold” appears in the outchild data set, and “sell” does not appear.

Output 5.2.1 The outterms Data Set with Stemming

OUTTERMS Data Set

Obs	Term	Role	Attribute	Freq	numdocs	_keep	Key	Parent	Parent_id	_ispar	Weight
1	all		Alpha	1	1	Y	1	.	1		1
2	win		Alpha	1	1	Y	2	2	2	+	1
3	toyota		Alpha	2	1	Y	3	.	3		1
4	ford		Alpha	2	2	Y	4	.	4		1
5	tacoma		Alpha	1	1	Y	5	.	5		1
6	year		Alpha	3	3	Y	6	.	6		1
7	taurus		Alpha	2	2	Y	7	.	7		1
8	won		Alpha	1	1	Y	26	2	2	.	1
9	honda		Alpha	1	1	Y	8	.	8		1
10	bright		Alpha	1	1	Y	9	.	9		1
11	be		Alpha	3	3	Y	10	10	10	+	1
12	sold		Alpha	2	2	Y	27	11	11	.	1
13	sell		Alpha	2	2	Y	11	11	11	+	1
14	colors		Alpha	1	1	Y	28	23	23	.	1
15	lime		Alpha	1	1	Y	12	.	12		1
16	except		Alpha	1	1	Y	13	.	13		1
17	hyundai		Alpha	1	1	Y	14	.	14		1
18	in		Alpha	3	3	Y	15	.	15		1
19	is		Alpha	2	2	Y	29	10	10	.	1
20	for		Alpha	1	1	Y	16	.	16		1
21	world		Alpha	2	2	Y	17	.	17		1
22	green		Alpha	2	2	Y	18	.	18		1
23	the		Alpha	8	5	Y	19	.	19		1
24	of		Alpha	2	2	Y	20	.	20		1
25	award		Alpha	1	1	Y	21	.	21		1
26	was		Alpha	1	1	Y	30	10	10	.	1
27	car		Alpha	2	2	Y	22	.	22		1
28	color		Alpha	1	1	Y	23	23	23	+	1
29	insight		Alpha	1	1	Y	24	.	24		1
30	last		Alpha	1	1	Y	25	.	25		1

Example 5.3: Adding Entities and Noun Groups

This example uses the data set that is generated in Example 5.1. The following statements run PROC HPTMINE to parse the documents. Because the NONOUNGROUPS option is not specified in the PARSE statement, noun group extraction is turned on, and because the ENTITIES option is not specified, entity identification is turned off.

```
proc hptmine data=work.CarNominations;
doc_id i;
var text;
PARSE
    notagging
    termwgt   = none
    cellwgt   = none
    reducef   = 0
    entities  = std
    outparent = outparent
    outterms  = outterms
    outchild  = outchild
    outconfig = outconfig
    ;
performance details;
run;

proc print data=outterms; title "OUTTERMS Data Set"; run;
```

Output 5.3.1 shows the content of the outterms data set. Compared to Output 5.2.1, the outterms data set is longer, because it contains entities and noun groups. For example, “honda insight” is included in the outterms data set as an entity with Role=Vehicle, and “bright green” is also included in the outterms data set as a noun group.

Output 5.3.1 The outterms Data Set with Noun Group Extraction and Entity Identification

OUTTERMS Data Set

Obs	Term	Role	Attribute	Freq	numdocs	_keep	Key	Parent	Parent_id	_ispar	Weight
1	lime green	NOUN_GROUP	Alpha	1	1	Y	1	.	1		1
2	all		Alpha	1	1	Y	2	.	2		1
3	win		Alpha	1	1	Y	3	3	3	+	1
4	toyota		Alpha	1	1	Y	4	.	4		1
5	award last year	NOUN_GROUP	Alpha	1	1	Y	5	.	5		1
6	ford		Alpha	2	2	Y	6	.	6		1
7	toyota tacoma	VEHICLE	Entity	1	1	Y	7	.	7		1
8	tacoma		Alpha	1	1	Y	8	.	8		1
9	year		Alpha	3	3	Y	9	.	9		1
10	taurus		Alpha	2	2	Y	10	.	10		1
11	won		Alpha	1	1	Y	34	3	3	.	1
12	honda		Alpha	1	1	Y	11	.	11		1
13	bright		Alpha	1	1	Y	12	.	12		1
14	be		Alpha	3	3	Y	13	13	13	+	1
15	sold		Alpha	2	2	Y	35	16	16	.	1
16	bright green	NOUN_GROUP	Alpha	1	1	Y	14	.	14		1
17	ford taurus	VEHICLE	Entity	2	2	Y	15	.	15		1
18	sell		Alpha	2	2	Y	16	16	16	+	1
19	hyundai	COMPANY	Entity	1	1	Y	17	.	17		1
20	colors		Alpha	1	1	Y	36	31	31	.	1
21	lime		Alpha	1	1	Y	18	.	18		1
22	except		Alpha	1	1	Y	19	.	19		1
23	honda insight	VEHICLE	Entity	1	1	Y	20	.	20		1
24	in		Alpha	3	3	Y	21	.	21		1
25	is		Alpha	2	2	Y	37	13	13	.	1
26	for		Alpha	1	1	Y	22	.	22		1
27	toyota	COMPANY	Entity	1	1	Y	23	.	23		1
28	world		Alpha	2	2	Y	24	.	24		1
29	green		Alpha	2	2	Y	25	.	25		1
30	the		Alpha	8	5	Y	26	.	26		1
31	of		Alpha	2	2	Y	27	.	27		1
32	world car	PROP_MISC	Entity	2	2	Y	28	.	28		1
33	award		Alpha	1	1	Y	29	.	29		1
34	was		Alpha	1	1	Y	38	13	13	.	1
35	car		Alpha	2	2	Y	30	.	30		1
36	color		Alpha	1	1	Y	31	31	31	+	1
37	insight		Alpha	1	1	Y	32	.	32		1
38	last		Alpha	1	1	Y	33	.	33		1

Example 5.4: Adding Part-of-Speech Tagging

This example uses the data set that is generated in Example 5.1. The following statements run PROC HPTMINE to parse the documents. Because the NOTAGGING option is not specified in the PARSE statement, PROC HPTMINE uses context clues to determine a term's part of speech.

```
proc hptmine data=work.CarNominations;
doc_id i;
var text;
PARSE
    termwgt   = none
    cellwgt   = none
    reducef   = 0
    entities  = std
    outparent = outparent
    outterms  = outterms
    outchild  = outchild
    outconfig = outconfig
    ;
performance details;
run;

proc print data=outterms; title "OUTTERMS Data Set"; run;
```

Output 5.4.1 shows the content of the outterms data set. Compared to Output 5.3.1, the outterms data set also contains the part-of-speech tag for the terms.

Output 5.4.1 The outterms Data Set with Part-of-Speech Tagging

OUTTERMS Data Set

Obs	Term	Role	Attribute	Freq	numdocs	_keep	Key	Parent	Parent_id	_ispar	Weight
1	car	Prop	Alpha	2	2	Y	1	.	1		1
2	lime green	NOUN_GROUP	Alpha	1	1	Y	2	.	2		1
3	insight	Prop	Alpha	1	1	Y	3	.	3		1
4	last	Adj	Alpha	1	1	Y	4	.	4		1
5	lime	Adj	Alpha	1	1	Y	5	.	5		1
6	award last year	NOUN_GROUP	Alpha	1	1	Y	6	.	6		1
7	sell	Verb	Alpha	2	2	Y	7	7	7	+	1
8	toyota tacoma	VEHICLE	Entity	1	1	Y	8	.	8		1
9	bright	Adj	Alpha	1	1	Y	9	.	9		1
10	honda	Prop	Alpha	1	1	Y	10	.	10		1
11	sold	Verb	Alpha	2	2	Y	34	7	7	.	1
12	colors	Noun	Alpha	1	1	Y	35	12	12	.	1
13	for	Prep	Alpha	1	1	Y	11	.	11		1
14	color	Noun	Alpha	1	1	Y	12	12	12	+	1
15	bright green	NOUN_GROUP	Alpha	1	1	Y	13	.	13		1
16	the	Det	Alpha	8	5	Y	14	.	14		1
17	is	Verb	Alpha	2	2	Y	36	24	24	.	1
18	ford taurus	VEHICLE	Entity	2	2	Y	15	.	15		1
19	hyundai	COMPANY	Entity	1	1	Y	16	.	16		1
20	toyota	Prop	Alpha	1	1	Y	17	.	17		1
21	except	Verb	Alpha	1	1	Y	18	.	18		1
22	in	Prep	Alpha	3	3	Y	19	.	19		1
23	was	Verb	Alpha	1	1	Y	37	24	24	.	1
24	honda insight	VEHICLE	Entity	1	1	Y	20	.	20		1
25	won	Verb	Alpha	1	1	Y	38	27	27	.	1
26	ford	Prop	Alpha	2	2	Y	21	.	21		1
27	world	Prop	Alpha	2	2	Y	22	.	22		1
28	year	Noun	Alpha	3	3	Y	23	.	23		1
29	be	Verb	Alpha	3	3	Y	24	24	24	+	1
30	all	Adj	Alpha	1	1	Y	25	.	25		1
31	toyota	COMPANY	Entity	1	1	Y	26	.	26		1
32	win	Verb	Alpha	1	1	Y	27	27	27	+	1
33	taurus	Noun	Alpha	2	2	Y	28	.	28		1
34	world car	PROP_MISC	Entity	2	2	Y	29	.	29		1
35	of	Prep	Alpha	2	2	Y	30	.	30		1
36	award	Noun	Alpha	1	1	Y	31	.	31		1
37	tacoma	Prop	Alpha	1	1	Y	32	.	32		1
38	green	Noun	Alpha	2	2	Y	33	.	33		1

Example 5.5: Adding Synonyms

This example uses the data set that is generated in Example 5.1. So far, by looking at the outterms data sets that are generated by Example 5.1 to Example 5.4, 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 to map each vehicle to the company that produces it. The following statements show this mapping.

```
data synds;
infile cards delimiter=',';
length TERM $13;
input TERM $ TERMROLE $ PARENT $ parentrole$;
cards;
    honda insight, VEHICLE , honda,  COMPANY,
    ford taurus,   VEHICLE,  ford,   COMPANY,
    toyota tacoma, VEHICLE,  toyota, COMPANY,
;
run;

proc hptmine data=work.CarNominations;
doc_id i;
var text;
parse
    termwgt    = none
    cellwgt    = none
    reducef    = 0
    entities   = std
    synonym    = synds
    outparent = outparent
    outterms   = outterms
    outchild   = outchild
    outconfig = outconfig
    ;
PERFORMANCE details;
RUN;

proc print data=outterms; title "OUTTERMS Data Set"; run;
```

Output 5.5.1 shows the content of the outterms data set. You can see that the term "honda insight" (key=39) is assigned the parent term "honda" (key=5). Only the term "honda" appears in the outparent data set.

Output 5.5.1 The outterms Data Set with Synonym Mapping

OUTTERMS Data Set

Obs	Term	Role	Attribute	Freq	numdocs	_keep	Key	Parent	Parent_id	_ispar	Weight
1	car	Prop	Alpha	2	2	Y	1	.	1		1
2	lime green	NOUN_GROUP	Alpha	1	1	Y	2	.	2		1
3	insight	Prop	Alpha	1	1	Y	3	.	3		1
4	last	Adj	Alpha	1	1	Y	4	.	4		1
5	lime	Adj	Alpha	1	1	Y	5	.	5		1
6	honda	COMPANY	Entity	1	1	Y	6	6	6	+	1
7	award last year	NOUN_GROUP	Alpha	1	1	Y	7	.	7		1
8	sell	Verb	Alpha	2	2	Y	8	8	8	+	1
9	toyota tacoma	VEHICLE	Entity	1	1	Y	33	24	24	.	1
10	bright	Adj	Alpha	1	1	Y	9	.	9		1
11	honda	Prop	Alpha	1	1	Y	10	.	10		1
12	sold	Verb	Alpha	2	2	Y	34	8	8	.	1
13	colors	Noun	Alpha	1	1	Y	35	12	12	.	1
14	for	Prep	Alpha	1	1	Y	11	.	11		1
15	color	Noun	Alpha	1	1	Y	12	12	12	+	1
16	bright green	NOUN_GROUP	Alpha	1	1	Y	13	.	13		1
17	the	Det	Alpha	8	5	Y	14	.	14		1
18	is	Verb	Alpha	2	2	Y	36	22	22	.	1
19	ford taurus	VEHICLE	Entity	2	2	Y	37	26	26	.	1
20	hyundai	COMPANY	Entity	1	1	Y	15	.	15		1
21	toyota	Prop	Alpha	1	1	Y	16	.	16		1
22	except	Verb	Alpha	1	1	Y	17	.	17		1
23	in	Prep	Alpha	3	3	Y	18	.	18		1
24	was	Verb	Alpha	1	1	Y	38	22	22	.	1
25	honda insight	VEHICLE	Entity	1	1	Y	39	6	6	.	1
26	won	Verb	Alpha	1	1	Y	40	25	25	.	1
27	ford	Prop	Alpha	2	2	Y	19	.	19		1
28	world	Prop	Alpha	2	2	Y	20	.	20		1
29	year	Noun	Alpha	3	3	Y	21	.	21		1
30	be	Verb	Alpha	3	3	Y	22	22	22	+	1
31	all	Adj	Alpha	1	1	Y	23	.	23		1
32	toyota	COMPANY	Entity	2	1	Y	24	24	24	+	1
33	toyota	COMPANY	Entity	1	1	Y	24	.	24	.	1
34	win	Verb	Alpha	1	1	Y	25	25	25	+	1
35	ford	COMPANY	Entity	2	2	Y	26	26	26	+	1
36	taurus	Noun	Alpha	2	2	Y	27	.	27		1
37	world car	PROP_MISC	Entity	2	2	Y	28	.	28		1
38	of	Prep	Alpha	2	2	Y	29	.	29		1
39	award	Noun	Alpha	1	1	Y	30	.	30		1
40	tacoma	Prop	Alpha	1	1	Y	31	.	31		1
41	green	Noun	Alpha	2	2	Y	32	.	32		1

Example 5.6: Adding a Stop List

This example uses the data set that is generated in Example 5.1. If you want to eliminate 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 HPTMIME drops the term "Toyota" (role=COMPANY) and all its children terms in the outterms data set by marking Keep=N for these terms.

```
data synds;
infile cards delimiter=',';
length TERM $13;
input TERM $ TERMROLE $ PARENT $ parentrole$;
cards;
    honda insight, VEHICLE , honda,  COMPANY,
    ford taurus,   VEHICLE,  ford,   COMPANY,
    toyota tacoma, VEHICLE,  toyota, COMPANY,
;
run;

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

proc hptmine data=work.CarNominations;
doc_id i;
var text;
parse
    termwgt    = none
    cellwgt    = none
    reducef    = 0
    entities   = std
    synonym    = synds
    stop       = stopList
    outparent  = outparent
    outterms   = outterms
    outchild   = outchild
    outconfig  = outconfig
    ;
performance details;
run;

proc print data=outterms; title "OUTTERMS Data Set"; run;
```

Output 5.6.1 shows the content of the outterms data set. You can see that the term “Toyota” (key=40) and the term “Toyota Tacoma” (key=41) are assigned Keep=N. The outparent data set is shorter than the one generated in Example 5.5.

Output 5.6.1 The outterms Data Set Filtered Using Stop List

OUTTERMS Data Set

Obs	Term	Role	Attribute	Freq	numdocs	_keep	Key	Parent	Parent_id	_ispar	Weight
1	car	Prop	Alpha	2	2	Y	1	.	1		1
2	lime green	NOUN_GROUP	Alpha	1	1	Y	2	.	2		1
3	insight	Prop	Alpha	1	1	Y	3	.	3		1
4	last	Adj	Alpha	1	1	Y	4	.	4		1
5	lime	Adj	Alpha	1	1	Y	5	.	5		1
6	honda	COMPANY	Entity	1	1	Y	6	6	6	+	1
7	award last year	NOUN_GROUP	Alpha	1	1	Y	7	.	7		1
8	sell	Verb	Alpha	2	2	Y	8	8	8	+	1
9	bright	Adj	Alpha	1	1	Y	9	.	9		1
10	honda	Prop	Alpha	1	1	Y	10	.	10		1
11	sold	Verb	Alpha	2	2	Y	32	8	8	.	1
12	colors	Noun	Alpha	1	1	Y	33	12	12	.	1
13	for	Prep	Alpha	1	1	Y	11	.	11		1
14	color	Noun	Alpha	1	1	Y	12	12	12	+	1
15	bright green	NOUN_GROUP	Alpha	1	1	Y	13	.	13		1
16	the	Det	Alpha	8	5	Y	14	.	14		1
17	is	Verb	Alpha	2	2	Y	34	22	22	.	1
18	ford taurus	VEHICLE	Entity	2	2	Y	35	25	25	.	1
19	hyundai	COMPANY	Entity	1	1	Y	15	.	15		1
20	toyota	Prop	Alpha	1	1	Y	16	.	16		1
21	except	Verb	Alpha	1	1	Y	17	.	17		1
22	in	Prep	Alpha	3	3	Y	18	.	18		1
23	was	Verb	Alpha	1	1	Y	36	22	22	.	1
24	honda insight	VEHICLE	Entity	1	1	Y	37	6	6	.	1
25	won	Verb	Alpha	1	1	Y	38	24	24	.	1
26	ford	Prop	Alpha	2	2	Y	19	.	19		1
27	world	Prop	Alpha	2	2	Y	20	.	20		1
28	year	Noun	Alpha	3	3	Y	21	.	21		1
29	be	Verb	Alpha	3	3	Y	22	22	22	+	1
30	all	Adj	Alpha	1	1	Y	23	.	23		1
31	win	Verb	Alpha	1	1	Y	24	24	24	+	1
32	ford	COMPANY	Entity	2	2	Y	25	25	25	+	1
33	taurus	Noun	Alpha	2	2	Y	26	.	26		1
34	world car	PROP_MISC	Entity	2	2	Y	27	.	27		1
35	of	Prep	Alpha	2	2	Y	28	.	28		1
36	award	Noun	Alpha	1	1	Y	29	.	29		1
37	tacoma	Prop	Alpha	1	1	Y	30	.	30		1
38	green	Noun	Alpha	2	2	Y	31	.	31		1

Example 5.7: Adding a Multiterm List

A multiterm list can be used to define terms that consist of multiple words. This example uses the data set that is generated in Example 5.1 to show how to use MULTITERM option.

```
data synds;
infile cards delimiter=',';
length TERM $13;
input TERM $ TERMROLE $ PARENT $ parentrole$;
cards;
    honda insight, VEHICLE , honda,  COMPANY,
    ford taurus,   VEHICLE,  ford,   COMPANY,
    toyota tacoma, VEHICLE,  toyota, COMPANY,
;
run;

data stopList;
infile cards delimiter='|' missover;
length term $25 role $40;
input term$ role$ ;
    cards;
        toyota| COMPANY
    ;
run;

/* create multiterms list*/
data _null_ ;
    FILE  'encomp.txt' ;
    PUT 'except for :3:Prep';
run ;

proc hptmine data=work.CarNominations;
doc_id i;
var text;
parse
    termwgt    = none
    cellwgt    = none
    reducef    = 0
    entities   = std
    synonym    = synds
    stop       = stopList
    multiterm  ="encomp.txt"
    outparent  = outparent
    outterms   = outterms
    outchild   = outchild
    outconfig  = outconfig
    ;
performance details;
run;

proc print data=outterms; title "OUTTERMS Data Set"; run;
```

Output 5.7.1 shows the content of the outterms data set. In the preceding statements, “except for” is defined as an individual term. In the outterms data set, you can see that the two terms “except” and “for” have become one term, “except for.”

Output 5.7.1 The outterms Data Set

OUTTERMS Data Set

Obs	Term	Role	Attribute	Freq	numdocs	_keep	Key	Parent	Parent_id	_ispar	Weight
1	car	Prop	Alpha	2	2	Y	1	.	1		1
2	lime green	NOUN_GROUP	Alpha	1	1	Y	2	.	2		1
3	insight	Prop	Alpha	1	1	Y	3	.	3		1
4	last	Adj	Alpha	1	1	Y	4	.	4		1
5	lime	Adj	Alpha	1	1	Y	5	.	5		1
6	honda	COMPANY	Entity	1	1	Y	6	6	6	+	1
7	award last year	NOUN_GROUP	Alpha	1	1	Y	7	.	7		1
8	sell	Verb	Alpha	2	2	Y	8	8	8	+	1
9	bright	Adj	Alpha	1	1	Y	9	.	9		1
10	honda	Prop	Alpha	1	1	Y	10	.	10		1
11	sold	Verb	Alpha	2	2	Y	31	8	8	.	1
12	colors	Noun	Alpha	1	1	Y	32	12	12	.	1
13	except for	Prep	Alpha	1	1	Y	11	.	11		1
14	color	Noun	Alpha	1	1	Y	12	12	12	+	1
15	bright green	NOUN_GROUP	Alpha	1	1	Y	13	.	13		1
16	the	Det	Alpha	8	5	Y	14	.	14		1
17	is	Verb	Alpha	2	2	Y	33	21	21	.	1
18	ford taurus	VEHICLE	Entity	2	2	Y	34	24	24	.	1
19	hyundai	COMPANY	Entity	1	1	Y	15	.	15		1
20	toyota	Prop	Alpha	1	1	Y	16	.	16		1
21	in	Prep	Alpha	3	3	Y	17	.	17		1
22	was	Verb	Alpha	1	1	Y	35	21	21	.	1
23	honda insight	VEHICLE	Entity	1	1	Y	36	6	6	.	1
24	won	Verb	Alpha	1	1	Y	37	23	23	.	1
25	ford	Prop	Alpha	2	2	Y	18	.	18		1
26	world	Prop	Alpha	2	2	Y	19	.	19		1
27	year	Noun	Alpha	3	3	Y	20	.	20		1
28	be	Verb	Alpha	3	3	Y	21	21	21	+	1
29	all	Adj	Alpha	1	1	Y	22	.	22		1
30	win	Verb	Alpha	1	1	Y	23	23	23	+	1
31	ford	COMPANY	Entity	2	2	Y	24	24	24	+	1
32	taurus	Noun	Alpha	2	2	Y	25	.	25		1
33	world car	PROP_MISC	Entity	2	2	Y	26	.	26		1
34	of	Prep	Alpha	2	2	Y	27	.	27		1
35	award	Noun	Alpha	1	1	Y	28	.	28		1
36	tacoma	Prop	Alpha	1	1	Y	29	.	29		1
37	green	Noun	Alpha	2	2	Y	30	.	30		1

Example 5.8: Selecting Parts of Speech and Entities to Ignore

This example uses the data set that is generated in Example 5.1. If you want to eliminate prepositions, determiners and proper nouns from your analysis, you can add a SELECT statement with these part-of-speech labels. If you also want to eliminate entities “PROP_MISC”, you can add another SELECT statement with the entity label.

```
proc hptmine data=work.CarNominations;
doc_id i;
var text;
parse
    termwgt    = none
    cellwgt    = none
    reducef    = 0
    entities   = std
    synonym    = synds
    stop       = stopList
    outparent  = outparent
    outterms   = outterms
    outchild   = outchild
    outconfig  = outconfig
    ;
select "prep" "det" "prop"/ignore;
select "prop_misc"/group="entities" ignore;
performance details;
run;

proc print data=outterms; title "OUTTERMS Data Set"; run;
```

Output 5.8.1 shows the content of the outterms data set. You can see that prepositions, determiners and proper nouns are excluded. Terms that are “PROP_MISC” are also excluded.

Output 5.8.1 The outterms Data Set

OUTTERMS Data Set

Obs	Term	Role	Attribute	Freq	numdocs	_keep	Key	Parent	Parent_id	_ispar	Weight
1	lime green	NOUN_GROUP	Alpha	1	1	Y	1	.	1		1
2	last	Adj	Alpha	1	1	Y	2	.	2		1
3	lime	Adj	Alpha	1	1	Y	3	.	3		1
4	award last year	NOUN_GROUP	Alpha	1	1	Y	4	.	4		1
5	sell	Verb	Alpha	2	2	Y	5	.	5	+	1
6	toyota tacoma	VEHICLE	Entity	1	1	Y	6	.	6		1
7	bright	Adj	Alpha	1	1	Y	7	.	7		1
8	sold	Verb	Alpha	2	2	Y	22	5	5	.	1
9	colors	Noun	Alpha	1	1	Y	23	8	8	.	1
10	color	Noun	Alpha	1	1	Y	8	.	8	+	1
11	bright green	NOUN_GROUP	Alpha	1	1	Y	9	.	9		1
12	is	Verb	Alpha	2	2	Y	24	15	15	.	1
13	ford taurus	VEHICLE	Entity	2	2	Y	10	.	10		1
14	hyundai	COMPANY	Entity	1	1	Y	11	.	11		1
15	except	Verb	Alpha	1	1	Y	12	.	12		1
16	was	Verb	Alpha	1	1	Y	25	15	15	.	1
17	honda insight	VEHICLE	Entity	1	1	Y	13	.	13		1
18	won	Verb	Alpha	1	1	Y	26	18	18	.	1
19	year	Noun	Alpha	3	3	Y	14	.	14		1
20	be	Verb	Alpha	3	3	Y	15	.	15	+	1
21	all	Adj	Alpha	1	1	Y	16	.	16		1
22	toyota	COMPANY	Entity	1	1	Y	17	.	17		1
23	win	Verb	Alpha	1	1	Y	18	.	18	+	1
24	taurus	Noun	Alpha	2	2	Y	19	.	19		1
25	award	Noun	Alpha	1	1	Y	20	.	20		1
26	green	Noun	Alpha	2	2	Y	21	.	21		1

Example 5.9: Running in Distributed Mode

This example uses the data set that is generated in the section “Getting Started: HPTMINE Procedure” on page 70. When a host for distributed computing is specified and the NODES option in the PERFORMANCE statement is specified as in the following statements, PROC HPTMINE uses the specified host for computing and runs in distributed mode.

```
%let GRID_TEXTANALYTICS_BIN_LOC=&GRIDTXTBIN;

option set=GRIDHOST="&GRIDHOST";
option set=GRIDINSTALLLOC="&GRIDINSTALLLOC";

proc hptmine data=getstart;
doc_id      did;
var         text;
parse
            outterms  = terms
            reducef   = 2;
svd
            k         = 10
            outdocpro = docpro;
performance nodes=2 details;
run;
```

In the preceding statements, the macro variable GRID_TEXTANALYTICS_BIN_LOC specifies the location on the grid where the language binary files are installed. These files are used by the HPTMINE procedure in parsing text.

Chapter 6

The HPTMSCORE Procedure

Contents

Overview: HPTMSCORE Procedure

For high-performance text mining, scoring is the process of applying the parsing and SVD projections to new textual data. The HPTMSCORE procedure performs this scoring of new documents, and its primary output are the Outparent data set, which holds the parsing results of the score collection; and the Outdocpro data set, which holds the reduced dimensional representation of the score collection. PROC HPTMSCORE uses some of the output data sets of the HPTMINE 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 HPTMINE to create three data sets, Outconfig, Outterms, and Svdu, respectively, and then you specify those three data sets as inputs to PROC HPTMSCORE. For more information about these data sets, see their respective options in the section "PROC HPTMSCORE Statement" on page 126.

PROC HPTMSCORE runs in either single-machine mode or distributed mode.

NOTE: Distributed mode requires SAS High-Performance Text Miner.

PROC HPTMSCORE Features

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

- 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.
- Analysis can be performed on a massively parallel SAS high-performance appliance.
- All phases of processing make use of a high degree of multithreading.
- Input data can be read in parallel when the data source is the appliance database.

PROC HPTMSCORE Contrasted with Other SAS Procedures

The high-performance PROC HPTMSCORE functionality replaces what has in the past been done by using the DATA step function TGSCORE followed by a call to the SPSVD procedure. The TGSCORE function was used to parse each document and to append each document's term counts to an Outparent data set. Then PROC SPSVD applied the document projections. PROC HPTMSCORE combines these tasks into a single parallel procedure. For PROC HPTMSCORE, you provide the necessary inputs to perform the parsing, term-by-document table creation, and document projections in a single call. The primary output data sets of PROC HPTMSCORE are the one that contains the parsing results that is specified in the OUTPARENT= option, and the one contains document projections that is specified in the SVDDOCPRO= option.

Getting Started: HPTMSCORE Procedure

The following DATA steps generate two data sets. The getstart data set contains 36 observations, and the getstart_score data set contains 31 observations. Both data sets have two variables. The text variable contains the input documents, and the did variable contains the ID of the documents. Each row in the data set represents a "document" for analysis.

```
data getstart;
    infile cards delimiter='|' missover;
    length text $150;
    input text$ did$;
    cards;
        High-performance analytics hold the key to |d01
        unlocking the unprecedented business value of big data.|d02
        Organizations looking for optimal ways to gain insights|d03
        from big data in shorter reporting windows are turning to SAS.|d04
        As the gold-standard leader in business analytics |d05
        for more than 36 years,|d06
        SAS frees enterprises from the limitations of |d07
        traditional computing and enables them |d08
        to draw instant benefits from big data.|d09
        Faster Time to Insight.|d10
        From banking to retail to health care to insurance, |d11
        SAS is helping industries glean insights from data |d12
        that once took days or weeks in just hours, minutes or seconds.|d13
        It's all about getting to and analyzing relevant data faster.|d14
        Revealing previously unseen patterns, sentiments and relationships.|d15
        Identifying unknown risks.|d16
        And speeding the time to insights.|d17
        High-Performance Analytics from SAS Combining industry-leading |d18
        analytics software with high-performance computing technologies|d19
        produces fast and precise answers to unsolvable problems|d20
        and enables our customers to gain greater competitive advantage.|d21
        SAS In-Memory Analytics eliminate the need for disk-based processing|d22
        allowing for much faster analysis.|d23
        SAS In-Database executes analytic logic into the database itself |d24
        for improved agility and governance.|d25
        SAS Grid Computing creates a centrally managed,|d26
        shared environment for processing large jobs|d27
        and supporting a growing number of users efficiently.|d28
        Together, the components of this integrated, |d29
        supercharged platform are changing the decision-making landscape|d30
        and redefining how the world solves big data business problems.|d31
        Big data is a popular term used to describe the exponential growth,|d32
        availability and use of information,|d33
        both structured and unstructured.|d34
        Much has been written on the big data trend and how it can |d35
        serve as the basis for innovation, differentiation and growth.|d36
run;

data getstart_score;
    infile cards delimiter='|' missover;
    length text $150;
    input text$ did$;
    cards;
        Big data according to SAS|d1
        At SAS, we consider two other dimensions|d2
        when thinking about big data:|d3
        Variability. In addition to the|d4
        increasing velocities and varieties of data, data|d5
        flows can be highly inconsistent with periodic peaks.|d6
        Is something big trending in the social media?|d7
        Perhaps there is a high-profile IPO looming.|d8
        Maybe swimming with pigs in the Bahamas is suddenly|d9
        the must-do vacation activity. Daily, seasonal and|d10
        event-triggered peak data loads can be challenging|d11
        to manage - especially with social media involved.|d12
        Complexity. When you deal with huge volumes of data,|d13
        it comes from multiple sources. It is quite an|d14
        undertaking to link, match, cleanse and|d15
```

```
            transform data across systems. However,|d16
            it is necessary to connect and correlate|d17
            relationships, hierarchies and multiple data|d18
            linkages or your data can quickly spiral out of|d19
            control. Data governance can help you determine|d20
            how disparate data relates to common definitions|d21
            and how to systematically integrate structured|d22
            and unstructured data assets to produce|d23
            high-quality information that is useful,|d24
            appropriate and up-to-date.|d25
            Ultimately, regardless of the factors involved,|d26
            we believe that the term big data is relative|d27
            it applies (per Gartner's assessment)|d28
            whenever an organization's ability|d29
            to handle, store and analyze data|d30
            exceeds its current capacity.|d31
    run;
```

The following statements use PROC HPTMINE for processing the input text data set getstart and create three data sets, outconfig, outterms, and svdu, which can be used in PROC HPTMSCORE for scoring. The statements then use PROC HPTMSCORE to score the input text data set getstart_score. The statements take the three data sets that are generated by PROC HPTMINE as input and create a SAS data set named docpro, which contains the projection of the documents in the input data set getstart_score.

```
proc hptmine data      = getstart;
doc_id       did;
variables    text;
parse
             outterms  = outterms
             outconfig = outconfig
             reducef   = 2;
svd
             k         = 10
             svdu      = svdu;
performance details;
run;

proc hptmscore
             data      = getstart_score
             terms     = outterms
             config    = outconfig
             svdu      = svdu
             svddocpro = docpro;
doc_id       did;
variable     text;
performance details;
run;
```

The output from this analysis is presented in Figure 6.1 through Figure 6.4.

Figure 6.1 shows the "Performance Information" table, which indicates that PROC HPTMSCORE executes in single-machine mode. That is, PROC HPTMSCORE runs on the machine where the SAS system is running. The table also shows that four threads are used for computing.

Figure 6.1 Performance Information

The HPTMSCORE Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Figure 6.2 shows the "Procedure Task Timing" table, which provides details about how much time is used by each processing step.

Figure 6.2 Procedure Task Timing

Task	Seconds	Percent
Parse documents	7.41	99.93%
Generate term-document matrix	0.00	0.03%
Compute SVD	0.00	0.03%
Output OUTDOCPRO table	0.00	0.01%

Figure 6.3 shows the "Data Access Information" table, which provides the information about the data sets that the HPTMSCORE procedure has accessed and generated.

Figure 6.3 Data Access Information

Data	Engine	Role	Path
WORK.GETSTART_SCORE	V9	Input	On Client
WORK.DOCPRO	V9	Output	On Client

The following statements use PROC PRINT to show the content of the first 10 rows of the docpro data set that is generated by the HPTMSCORE procedure.

```
proc print data = docpro (obs=10) round; run;
```

Figure 6.4 shows the output of PROC PRINT. For more information about the output of the OUTDOCPRO= option, see the description of the SVDDOCPRO= option.

Figure 6.4 The DOCPRO Data Set

Obs	did	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8	COL9	COL10
1	d1	0.90	0.07	0.07	0.35	0.06	-0.12	-0.12	-0.06	-0.02	0.16
2	d2	0.33	-0.54	0.11	0.62	-0.28	0.12	0.06	-0.05	-0.17	0.28
3	d3	0.84	0.34	0.28	0.20	0.08	-0.16	-0.07	-0.15	0.09	0.02
4	d4	0.48	-0.20	0.01	-0.18	-0.03	0.15	-0.51	0.44	0.11	0.47
5	d5	0.71	0.22	0.07	0.01	-0.05	0.01	0.04	-0.32	-0.58	0.06
6	d6	0.33	0.22	-0.37	-0.10	-0.09	-0.78	0.13	0.08	0.21	0.08
7	d7	0.71	0.15	0.11	-0.17	-0.14	-0.35	-0.23	0.24	0.25	0.35
8	d8	0.32	0.18	-0.27	-0.06	-0.25	-0.84	-0.01	-0.11	-0.12	0.00
9	d9	0.52	0.03	-0.02	-0.24	-0.19	-0.40	-0.27	0.36	0.31	0.41
10	d10	0.57	-0.03	-0.16	-0.50	-0.26	0.25	0.20	-0.25	-0.26	-0.31

Syntax: HPTMSCORE Procedure

The following statements are available in the HPTMSCORE procedure:

PROC HPTMSCORE < *options* > ;
  VARIABLES *variable* ;
  DOC_ID *variable* ;
  PERFORMANCE *performance-options* ;

PROC HPTMSCORE Statement

PROC HPTMSCORE < *options* > ;

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

Table 6.1 PROC HPTMSCORE Statement Options

option	**Description**
Basic Options	
DATA | DOC=	Specifies the input document data set
TERMS=	Specifies the data set that contains the summary information about the terms that are to be used for scoring
CONFIG=	Specifies the data set that contains the option settings that are to be used for scoring
SVDU=	Specifies the data set that contains the U matrix whose columns are the left singular vectors
Output Options	
NOPRINT	Suppresses ODS output
OUTPARENT=	Specifies the term-by-document frequency matrix that is used to model the document collection, in which the child terms are not represented and child terms' frequencies are attributed to their corresponding parents
SVDDOCPRO=	Specifies the projections of the documents

You can specify the following *options* in the PROC HPTMSCORE statement.

CONFIG=*SAS-data-set*

specifies the input SAS data set that contains configuration information for PROC HPTMSCORE. The data set that is used for this option should be the one that is generated by the OUTCONFIG= option in the PARSE statement of the HPTMINE procedure during training. For more information about this data set, see the section "The OUTCONFIG= Data Set" on page 96 of Chapter 5, "The HPTMINE Procedure."

DATA | DOC=*SAS-data-set*

specifies the input SAS data set of documents to be used by PROC HPTMSCORE. The default is the most recently created data set. If PROC HPTMSCORE executes in distributed mode, the input data are distributed to memory on the appliance nodes and analyzed in parallel, unless the data are already distributed in the appliance database. When data are already distributed, PROC HPTMSCORE reads the data alongside the distributed database. For more information, see the sections "Processing Modes" on page 8 and "Alongside-the-Database Execution" on page 16. Each row of the input data set must contain one text field and one ID field, which correspond to the text and the unique ID of a document, respectively.

NOPRINT

suppresses the generation of ODS output.

OUTPARENT=*SAS-data-set*

specifies the output data set to contain a compressed representation of the sparse term-by-document frequency matrix. The data set 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 Set" on page 97 of Chapter 5, "The HPTMINE Procedure."

SVDDOCPRO=*SAS-data-set* **<KEEPVARS | KEEPVARIABLES=***variable-list***>**

specifies the output data set to contain the reduced dimensional projections for each document. The contents of this data set are formed by multiplying the term-by-document frequency matrix by the input data set that is specified in the SVDU= option and then normalizing the result.

The variables in the data set that is specified by the DATA= option can be copied to the output of this option. When KEEPVARS | KEEPVARIABLES=*variable-list* is used, the content of the variables that is specified in the *variable-list* is attached to the output. These variables must appear in the data set that is specified by the DATA= option.

SVDU=*SAS-data-set*

specifies the input data set that contains the **U** matrix, which is created during training by PROC HPTMINE. The data set contains the information that is needed to project each document into the reduced dimensional space. For more information about the contents of this data set, see the SVDU= option in Chapter 5, "The HPTMINE Procedure."

TERMS=*SAS-data-set*

specifies the input SAS data set of terms to be used by PROC HPTMSCORE. The data set that is used for this option should be the one that is generated by the OUTTERMS= option in the PARSE statement of the HPTMINE procedure during training. This data set conveys to PROC HPTMSCORE which terms should be used in the analysis and whether they should be mapped to a parent. The data set also assigns them a key that corresponds to the key that is used in the input data set that is specified by the SVDU= option. For more information about this data set, see the section "The OUTTERMS= Data Set" on page 98 of Chapter 5, "The HPTMINE Procedure."

DOC_ID Statement

DOC_ID *variable* < *docid-options* > **;**

This statement specifies the variable that contains the ID of each document. In the input data set, 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.

You can specify the following *docid-option* in the DOC_ID statement:

DUPLICATIONCHECK | DUPCHK < (WARNING | STOP) >

checks whether the ID of each document is unique. When this option is not specified, no duplication check is performed. When you specify this option along with the WARNING keyword and duplicate document IDs are detected, the HPTMSCORE procedure outputs a warning message, which shows the number of duplicate document IDs that have been detected. When you specify this option along with the STOP keyword, the HPTMSCORE procedure outputs an error message and terminates. The error message shows the number of duplicate document IDs that have been detected. If you specify the DUPLICATIONCHECK option without the WARNING or STOP keyword, the HPTMSCORE procedure takes the WARNING keyword by default.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables that describe the distributed computing environment, and requests detailed results about the performance characteristics of the HPTMSCORE procedure.

You can also use the PERFORMANCE statement to control whether the HPTMSCORE procedure executes in single-machine or distributed mode.

The PERFORMANCE statement for is documented further in the section "PERFORMANCE Statement" on page 33 of Chapter 3, "Shared Concepts and Topics."

VARIABLE Statement

VARIABLES | VAR *variable* **;**

This statement specifies the variable that contains the text to be processed.

Details: HPTMSCORE Procedure

For information about the techniques that are used for nature language processing, term processing, and singular value decomposition, see the section "Details: HPTMINE Procedure" on page 86 of Chapter 5, "The HPTMINE Procedure."

Displayed Output

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

Performance Information

The "Performance Information" table is produced by default. It displays information about the execution mode. For single-machine mode, the table displays the number of threads used. For distributed mode, the table displays the grid mode (symmetric or asymmetric), the number of compute nodes, and the number of threads per node. If you specify the DETAILS option in the PERFORMANCE statement, the procedure also produces a "Timing" table in which elapsed times (absolute and relative) for the main tasks of the procedure are displayed.

Procedure Task Timing

When the DETAILS option is specified in the PERFORMANCE statement, the procedure produces a "Procedure Task Timing" table, which displays the elapsed time (absolute and relative) for the main tasks.

ODS Table Names

Each table that is created by the HPTMSCORE procedure has a name associated with it, and you must use this name to refer to the table when you use ODS statements. These names are listed in Table 6.2.

Table 6.2 ODS Tables Produced by PROC HPTMINE

Table Name	Description	Required Statement / Option
PerformanceInfo	Information about the high-performance computing environment	Default output
Timing	Absolute and relative times for tasks that are performed by the procedure	DETAILS option in the PERFORMANCE statement

System Configuration

Prerequisites for Running PROC HPTMSCORE

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

Input Data Accessibility

When you run PROC HPTMSCORE in distributed mode, the following input data sets need to be accessible to the client machine:

- TERMS= data set
- CONFIG= data set
- SVDU= data set

When the HPTMSCORE procedure runs in distributed mode, it first reads these data sets on the client machine and then sends them to the grid.

Configuration for Distributed Mode

When PROC HPTMSCORE runs in distributed mode, it needs to locate on the grid the language binary files that are used in parsing text. These binary files must be deployed to the grid. In addition, the GRID_TEXTANALYTICS_BIN_LOC macro must be specified to indicate to location of the binary files on your grid. For more information, see the section "Configuring for Distributed Mode" on page 102 of Chapter 5, "The HPTMINE Procedure."

Subject Index

Syntax Index