## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Shared Concepts and Topics</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>The HP4SCORE Procedure</td>
<td>39</td>
</tr>
<tr>
<td>4</td>
<td>The HPDECIDE Procedure</td>
<td>49</td>
</tr>
<tr>
<td>5</td>
<td>The HPFOREST Procedure</td>
<td>65</td>
</tr>
<tr>
<td>6</td>
<td>The HPNEURAL Procedure</td>
<td>119</td>
</tr>
<tr>
<td>7</td>
<td>The HPREDUCE Procedure</td>
<td>139</td>
</tr>
</tbody>
</table>

**Subject Index**

**Syntax Index**
Credits and Acknowledgments

Credits

Documentation
Editing Anne Baxter, Ed Huddleston
Documentation Support Tim Arnold

Software
The procedures in this book 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.

- HP4SCORE Padraic Neville
- HPDECIDE Tao Wang, Scott Pope
- HPFOREST Padraic Neville
- HPNEURAL Larry Lewis
- HPREDUCE Alan (Zheng) Zhao
- 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 contributed with their leadership and support: Chris Bailey, Tanya Balan, David Pope, Oliver Schabenberger, Renee Sciortino.

Testing
Internationalization Testing

Jacky(Chen) Dong, Feng Gao, Alex(Wenqi) He, David Li, Frank(Jidong) Wang, Lina Xu, Catherine Yang.

Technical Support

Phil Gibbs, Doug Wielenga.

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.
Overview of the SAS Enterprise Miner High-Performance Procedures

The high-performance data mining procedures provides tools that have been specially developed to take advantage of parallel processing in both multithreaded single-machine mode and distributed multiple-machine mode. Data Mining methods include neural networks and ensemble tree models as well as variables selection techniques and applying decision maricies to make optimal decisions. The software is constantly being updated to reflect new methodology and advances in high-performance analytics.

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

About This Book

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

This book is organized as follows:

Chapter 1, this chapter, provides an overview of high-performance data mining procedures.

Chapter 2, “Shared Concepts and Topics,” describes the modes in which high-performance data mining procedures can execute.

Subsequent chapters describe the individual 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 other topics, such as ODS tables.
- The “Examples” section contains examples that use the procedure.
- The “References” section contains references for the methodology.

Typographical Conventions

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

- **roman** is the standard type style used for most text.
- **UPPERCASE ROMAN** is used for SAS statements, options, and other SAS language elements when they appear in 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 in the syntax definitions and in text to represent arguments for which you supply a value.
- **VariableName** is used for the names of variables and data sets when they appear in the text.
- **bold** is used to for 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

Most of the output shown in this book is produced with the following SAS System options:

```
options linesize=80 pagesize=500 nonumber nodate;
```

The HTMLBLUE style is used to create the HTML output and graphs that appear in the online documentation. A style template controls stylistic elements such as colors, fonts, and presentation attributes. The style template is specified in the ODS HTML statement as follows:

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

If you run the examples, your output might be slightly different, because of the SAS System options you use 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](http://support.sas.com/documentation).

SAS Technical Support Services

The SAS Technical Support staff is available to respond to problems and answer technical questions regarding the use of high-performance procedures. Go to [http://support.sas.com/techsup](http://support.sas.com/techsup) for more information.
Chapter 2
Shared Concepts and Topics

Contents

Overview ................................................................. 5
Processing Modes .................................................... 6
  Single-Machine Mode ............................................. 6
  Distributed Mode .................................................. 6
  Symmetric and Asymmetric Distributed Modes .............. 7
Controlling the Execution Mode with Environment Variables and Performance Statement Options .......................... 7
Determining Single-Machine Mode or Distributed Mode ................................................................. 9
Alongside-the-Database Execution ................................ 13
Alongside-LASR Distributed Execution ............................ 16
Running High-Performance Analytical Procedures Alongside a SAS LASR Analytic Server in Distributed Mode .................................................. 17
  Starting a SAS LASR Analytic Server Instance ............ 17
  Associating a SAS Libref with the SAS LASR Analytic Server Instance ........................................... 18
  Running a High-Performance Analytical Procedure Alongside the SAS LASR Analytic Server Instance .......... 18
  Terminating a SAS LASR Analytic Server Instance .......... 19
Alongside-LASR Distributed Execution on a Subset of the Appliance Nodes ........................................ 19
Running High-Performance Analytical Procedures in Asymmetric Mode ................................................. 19
  Running in Symmetric Mode ...................................... 20
  Running in Asymmetric Mode on One Appliance ............ 21
  Running in Asymmetric Mode on Distinct Appliances ........ 22
Alongside-HDFS Execution ........................................... 25
  Alongside-HDFS Execution by Using the SASHDAT Engine ................................................................. 25
  Alongside-HDFS Execution by Using the Hadoop Engine ................................................................. 27
Output Data Sets ......................................................... 31
Working with Formats ............................................... 32
PERFORMANCE Statement ........................................... 34

Overview

This chapter describes the modes of execution in which SAS high-performance analytical procedures can execute. If you have SAS Enterprise 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 Data Mining 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

Distributed mode has several variations:

- Client-data (or local-data) mode: The input data for the analytic task are not stored on the appliance or cluster but are distributed to the distributed computing environment by the SAS High-Performance Analytics infrastructure when the procedure runs.
- Alongside-the-database mode: The data are stored in the distributed database and are read from the DBMS in parallel into a high-performance analytical procedure that runs on the database appliance.
• Alongside-HDFS mode: The data are stored in the Hadoop Distributed File System (HDFS) and are read in parallel from the HDFS. This mode is available if you install the SAS High-Performance Deployment of Hadoop on the appliance or when you configure a Cloudera 4 Hadoop deployment on the appliance to operate with the SAS High-Performance Analytics infrastructure. For more information about installing the SAS High-Performance Deployment of Hadoop, see the *SAS High-Performance Analytics Infrastructure: Installation and Configuration Guide*.

• Alongside-LASR mode: The data are loaded from a SAS LASR Analytic Server that runs on the appliance.

---

**Symmetric and Asymmetric Distributed Modes**

SAS high-performance analytical procedures can run alongside the database or alongside HDFS in asymmetric mode. The primary reason for providing the asymmetric mode is to enable you to manage and house data on one appliance (the data appliance) and to run the high-performance analytical procedure on a second appliance (the computing appliance). You can also run in asymmetric mode on a single appliance that functions as both the data appliance and the computing appliance. This enables you to run alongside the database or alongside HDFS, where computations are done on a different set of nodes from the nodes that contain the data. The following subsections provide more details.

**Symmetric Mode**

When SAS high-performance analytical procedures run in symmetric distributed mode, the data appliance and the computing appliance must be the same appliance. Both the SAS Embedded Process and the high-performance analytical procedures execute in a SAS process that runs on the same hardware where the DBMS process executes. This is called symmetric mode because the number of nodes on which the DBMS executes is the same as the number of nodes on which the high-performance analytical procedures execute. The initial data movement from the DBMS to the high-performance analytical procedure does not cross node boundaries.

**Asymmetric Mode**

When SAS high-performance analytical procedures run in asymmetric distributed mode, the data appliance and computing appliance are usually distinct appliances. The high-performance analytical procedures execute in a SAS process that runs on the computing appliance. The DBMS and a SAS Embedded Process run on the 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 is called asymmetric mode because the number of nodes on the data appliance does not need to be the same as the number of nodes on the computing appliance.

---

**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.
Chapter 2: Shared Concepts and Topics

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 Data Mining software connects to run in distributed mode.

- **installation location** identifies the directory where the SAS High-Performance Data Mining software is installed on the appliance.

- **data server** identifies the database server on Teradata appliances as defined in the hosts file on the client. This data server is the same entry that you usually specify in the SERVER= entry of a LIBNAME statement for Teradata. For more information about specifying LIBNAME statements for Teradata and other engines, see the DBMS-specific section of SAS/ACCESS for Relational Databases: Reference for your engine.

- **grid mode** specifies whether the high-performance analytical procedures execute in symmetric or asymmetric mode. Valid values for this variable are 'sym' for symmetric mode and 'asym' for asymmetric mode. The default is symmetric mode.

You can set an environment variable directly from the SAS program by using the OPTION SET= command. For example, the following statements define three variables for a Teradata appliance (the grid mode is the default symmetric mode):

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

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

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

The following statements define three variables that are needed to run asymmetrically on a computing appliance.

```sas
option set=GRIDHOST ="compute_appliance.sas.com";
option set=GRIDINSTALLLOC="/opt/TKGrid";
option set=GRIDMODE ="asym";
```

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

```sas
performance host ="compute_appliance.sas.com"
    install =="/opt/TKGrid"
    gridmode ="asym"
```

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.
Specifying a data server is necessary only on Teradata systems when you do not explicitly set the *gridmode* environment variable or specify the GRIDMODE= option in the PERFORMANCE statement. The data server specification depends on the entries in the (client) *hosts* file. The file specifies the server (suffixed by *cop* and a number) and an IP address. For example:

```
myservercop1 33.44.55.66
```

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 and data server are 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 and data server (if necessary) have 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:

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

Figure 2.1 shows the results from the analysis.

**Figure 2.1** Results from Logistic Regression in Single-Machine Mode

<table>
<thead>
<tr>
<th>The HPLOGISTIC Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Information</td>
</tr>
<tr>
<td>Execution Mode</td>
</tr>
<tr>
<td>Number of Threads</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source</td>
</tr>
<tr>
<td>Response Variable</td>
</tr>
<tr>
<td>Class Parameterization</td>
</tr>
<tr>
<td>Distribution</td>
</tr>
<tr>
<td>Link Function</td>
</tr>
<tr>
<td>Optimization Technique</td>
</tr>
</tbody>
</table>
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 on the client.

The following statements use 10 nodes (in distributed mode) to analyze the data on the appliance; results appear in Figure 2.2:

```sas
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 2.2 Results from Logistic Regression in Distributed Mode

| Parameter | Estimate | 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|
Chapter 2: Shared Concepts and Topics

**Figure 2.2 continued**

<table>
<thead>
<tr>
<th>Data Source</th>
<th>WORK.SIMDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Variable</td>
<td>y</td>
</tr>
<tr>
<td>Class Parameterization</td>
<td>GLM</td>
</tr>
<tr>
<td>Distribution</td>
<td>Binary</td>
</tr>
<tr>
<td>Link Function</td>
<td>Logit</td>
</tr>
<tr>
<td>Optimization Technique</td>
<td>Newton-Raphson with Ridging</td>
</tr>
</tbody>
</table>

**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:

```sas
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 2.3** shows the “Performance Information” table. The numeric results are not reproduced here, but they agree with the previous analyses, which are shown in **Figure 2.1** and **Figure 2.2**.
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 on the client.

In the analysis shown previously in Figure 2.2, 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.

**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 co-located 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.

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

proc datasets lib=appliance 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;
  enddata;
run;

If the output table appliance.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 the previous section, which executed in single-machine mode.

```sas
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 2.4 shows the results from this analysis. The “Performance Information” table shows that the execution was in distributed mode. In this case the execution was alongside the Greenplum database. The numeric results agree with the previous analyses, which are shown in Figure 2.1 and Figure 2.2.

**Figure 2.4** Alongside-the-Database Execution on Greenplum

<table>
<thead>
<tr>
<th>The HPLOGISTIC Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Information</td>
</tr>
<tr>
<td>Host Node</td>
</tr>
<tr>
<td>Execution Mode</td>
</tr>
<tr>
<td>Grid Mode</td>
</tr>
<tr>
<td>Number of Compute Nodes</td>
</tr>
<tr>
<td>Number of Threads per Node</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source</td>
</tr>
<tr>
<td>Response Variable</td>
</tr>
<tr>
<td>Class Parameterization</td>
</tr>
<tr>
<td>Distribution</td>
</tr>
<tr>
<td>Link Function</td>
</tr>
<tr>
<td>Optimization Technique</td>
</tr>
</tbody>
</table>
When high-performance analytical procedures execute symmetrically alongside the database, any nonzero specification of the NODES= option in the PERFORMANCE statement is ignored. If the data are read alongside the database, the number of compute nodes is determined by the layout of the database and cannot be modified. In this example, the appliance contains 16 nodes. (See the “Performance Information” table.)

However, when high-performance analytical procedures execute asymmetrically alongside the database, the number of compute nodes that you specify in the PERFORMANCE statement can differ from the number of nodes across which the data are partitioned. For an example, see the section “Running High-Performance Analytical Procedures in Asymmetric Mode” on page 19.

**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 execute 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.
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=12345
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: Administration 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 19.

Figure 2.5 shows the “Performance Information” table, which shows that the LASR procedure executes in distributed mode on 16 nodes.

Figure 2.5 Performance Information

<table>
<thead>
<tr>
<th>The LASR Procedure</th>
<th>Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Node</td>
<td>hpa.sas.com</td>
</tr>
<tr>
<td>Execution Mode</td>
<td>Distributed</td>
</tr>
<tr>
<td>Grid Mode</td>
<td>Symmetric</td>
</tr>
<tr>
<td>Number of Compute Nodes</td>
<td>8</td>
</tr>
</tbody>
</table>
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:

```sas
libname MyLasr sasiola port=12345;
```

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: Administration 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:

```sas
proc hplogistic data=MyLasr.simData;
    class a b c;
    model y = a b c x1 x2 x3;
    output out=MyLasr.simulateScores pred=PredictedProbability;
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 2.6.

**Figure 2.6 Performance Information**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Node</td>
<td>hpa.sas.com</td>
</tr>
<tr>
<td>Execution Mode</td>
<td>Distributed</td>
</tr>
<tr>
<td>Grid Mode</td>
<td>Symmetric</td>
</tr>
<tr>
<td>Number of Compute Nodes</td>
<td>8</td>
</tr>
<tr>
<td>Number of Threads per Node</td>
<td>24</td>
</tr>
</tbody>
</table>

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 alongside 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:

```plaintext
proc lasr term port=12345;
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. It also includes examples that run in symmetric mode to highlight differences between the modes. For a description of asymmetric mode, see the section “Symmetric and Asymmetric Distributed Modes” on page 7.

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 Data Mining 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 Data Mining 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. Because SAS High-Performance Data Mining is also installed on this appliance, it can be used to run high-performance analytical procedures in both symmetric and asymmetric modes.
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 Data Mining software is not installed on the appliance.

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

### Running in Symmetric Mode

The following statements run the HPLOGISTIC procedure in symmetric mode on the data appliance:

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

Because you explicitly specified the GRIDMODE= option, you do not need to also specify the DATASERVER= option in the PERFORMANCE statement. **Figure 2.7** shows the results of this analysis.

**Figure 2.7** Alongside-the-Database Execution in Symmetric Mode on Teradata

<table>
<thead>
<tr>
<th>The HPLOGISTIC Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Information</td>
</tr>
<tr>
<td>Host Node</td>
</tr>
<tr>
<td>Execution Mode</td>
</tr>
<tr>
<td>Grid Mode</td>
</tr>
<tr>
<td>Number of Compute Nodes</td>
</tr>
<tr>
<td>Number of Threads per Node</td>
</tr>
</tbody>
</table>
Running in Asymmetric Mode on One Appliance

The “Performance Information” table shows that the execution occurs in symmetric mode on the 24 nodes of the data appliance. In this case, the NODES=10 option in the PERFORMANCE statement is ignored because the number of nodes that are used is determined by the number of nodes across which the data are distributed, as indicated in the following warning message in the SAS log:

WARNING: The NODES=10 option in the PERFORMANCE statement is ignored because you are running alongside the distributed data source DATALIB.simData.DATA. The number of compute nodes is determined by the configuration of the distributed DBMS.

Running in Asymmetric Mode on One Appliance

You can switch to running the HPLOGISTIC procedure in asymmetric mode by specifying the GRID-MODE=ASYM option in the PERFORMANCE statement as follows:

```sas
proc hplogistic data=dataLib.simData;
  class a b c;
  model y = a b c x1 x2 x3;
  performance host = "data_appliance.sas.com"
    nodes = 10
    gridmode = asym;
run;
```
Figure 2.8 shows the “Performance Information” table.

**Figure 2.8** Alongside Teradata Execution in Asymmetric Mode

<table>
<thead>
<tr>
<th>The HPLOGISTIC Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Information</td>
</tr>
<tr>
<td>Host Node</td>
</tr>
<tr>
<td>Execution Mode</td>
</tr>
<tr>
<td>Grid Mode</td>
</tr>
<tr>
<td>Number of Compute Nodes</td>
</tr>
<tr>
<td>Number of Threads per Node</td>
</tr>
</tbody>
</table>

You can see that now the grid mode is asymmetric. Furthermore, the NODES=10 option that you specified in the PERFORMANCE statement is honored. The data are moved in parallel from the 24 nodes on which the data are stored to the 10 nodes on which the execution occurs. The numeric results are not reproduced here, but they agree with the previous analyses.

**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 computing appliance that has 15 compute nodes, and it is a different appliance 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:

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

Figure 2.9 shows the “Performance Information” table.
PROC HPLOGISTIC ran on the 15 nodes of the computing appliance, even though the data are 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 2.1 and Figure 2.2.

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:

```sas
proc lasr port=54321
   data=dataLib.simData
   path="/tmp/
   performance host 
      ="compute_appliance.sas.com"
   gridmode = asym;
run;
```

```sas
libname MyLasr sasiola tag="dataLib" port=54321 host="compute_appliance.sas.com" ;
```

Figure 2.10 show the “Performance Information” table.

PROC LASR ran on the 15 nodes of the computing appliance, which has 15 compute nodes. In this mode, the data are loaded in parallel from the 24 data appliance nodes to the 15 compute nodes on the
computing appliance. By default, all the nodes on the computing appliance are used. You can use the NODES= option in the PERFORMANCE statement to run the LASR Analytic Server on a subset of the nodes on the computing appliance. If you omit the GRIDMODE=ASYM option from the PERFORMANCE statement, PROC LASR still runs successfully but much less efficiently. The Teradata access engine transfers the simData data set to a temporary table on the client, and the High-Performance Analytics infrastructure then transfers these data from the temporary table on the client to the grid nodes on the computing appliance.

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. Because these procedures run on the same computing appliance where the LASR Analytic Server is running, it is best to run these procedures in symmetric mode, which is the default or can be explicitly specified in the GRIDMODE=SYM option in the PERFORMANCE statement. The following statements provide an example. The OUTPUT statement creates an output data set that is held in memory by the LASR Analytic Server. The data appliance has no role in executing these 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 54321. 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. In order for this output to be directly written in parallel from the nodes of the computing appliance to the nodes of the data appliance, you need to run the HPLOGISTIC procedure in asymmetric mode by specifying the GRIDMODE=ASYM option in the PERFORMANCE statement as follows:

```
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";
    gridmode = asym;
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, it is important to run the high-performance analytical procedures in asymmetric mode so that the Read and Write operations take place in parallel without any movement of data to and from the SAS client. If you omit running the preceding PROC HPLOGISTIC step in asymmetric mode, then the output data set would be created much less efficiently: the output data would be moved sequentially to a temporary table on the client, after which the Teradata access engine sequentially would write this table to the data appliance.
When you no longer need the data in the SAS LASR Analytic Server, you should terminate the server instance as follows:

```sas
proc lasr term port=54321;
   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 co-located 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:

```sas
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: Administration 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.

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

Figure 2.11 shows the “Performance Information” table. You see that the procedure ran in distributed mode. The numeric results shown in Figure 2.12 agree with the previous analyses shown in Figure 2.1, Figure 2.2, and Figure 2.4.

**Figure 2.11** Alongside-HDFS Execution Performance Information

<table>
<thead>
<tr>
<th>Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Node</td>
</tr>
<tr>
<td>Execution Mode</td>
</tr>
<tr>
<td>Grid Mode</td>
</tr>
<tr>
<td>Number of Compute Nodes</td>
</tr>
<tr>
<td>Number of Threads per Node</td>
</tr>
</tbody>
</table>

**Figure 2.12** Alongside-HDFS Execution Model Information

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source</td>
</tr>
<tr>
<td>Response Variable</td>
</tr>
<tr>
<td>Class Parameterization</td>
</tr>
<tr>
<td>Distribution</td>
</tr>
<tr>
<td>Link Function</td>
</tr>
<tr>
<td>Optimization Technique</td>
</tr>
</tbody>
</table>
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 XML shows the contents of the file demo.xml that is used in this example:

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://hpa.sas.com:8020</value>
  </property>
  <property>
    <name>mapred.job.tracker</name>
    <value>hpa.sas.com:8021</value>
  </property>
  <property>
    <name>dfs.replication</name>
  </property>
</configuration>
```
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.

```sas
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 analytics 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. However, whenever you use the Hadoop engine, you must execute the analysis in asymmetric mode to cause the execution to occur alongside HDFS.

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

Figure 2.13 shows the “Performance Information” table. You see that the procedure ran asymmetrically in distributed mode. The numeric results shown in Figure 2.14 agree with the previous analyses.
Figure 2.14 Alongside-HDFS Execution by Using the Hadoop Engine

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source</td>
</tr>
<tr>
<td>Response Variable</td>
</tr>
<tr>
<td>Class Parameterization</td>
</tr>
<tr>
<td>Distribution</td>
</tr>
<tr>
<td>Link Function</td>
</tr>
<tr>
<td>Optimization Technique</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>Intercept</td>
</tr>
<tr>
<td>a 0</td>
</tr>
<tr>
<td>a 1</td>
</tr>
<tr>
<td>b 0</td>
</tr>
<tr>
<td>b 1</td>
</tr>
<tr>
<td>c 0</td>
</tr>
<tr>
<td>c 1</td>
</tr>
<tr>
<td>x1</td>
</tr>
<tr>
<td>x2</td>
</tr>
<tr>
<td>x3</td>
</tr>
</tbody>
</table>

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
- JavaScript Object Notation (JSON)
- 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 analytics 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:

```sas
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 analytics 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.

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

Figure 2.15 shows the results of this analysis. You see that the procedure ran asymmetrically in distributed mode. The metadata that you created by using the HDMD procedure have been used successfully in executing this analysis.
Figure 2.15  Alongside-HDFS Execution with CSV Data

The HPLOGISTIC Procedure

Performance Information

<table>
<thead>
<tr>
<th>Host Node</th>
<th>compute_appliance.sas.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Mode</td>
<td>Distributed</td>
</tr>
<tr>
<td>Grid Mode</td>
<td>Asymmetric</td>
</tr>
<tr>
<td>Number of Compute Nodes</td>
<td>15</td>
</tr>
<tr>
<td>Number of Threads per Node</td>
<td>24</td>
</tr>
</tbody>
</table>

Model Information

<table>
<thead>
<tr>
<th>Data Source</th>
<th>GRIDLIB.CSVEXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Variable</td>
<td>Success</td>
</tr>
<tr>
<td>Class Parameterization</td>
<td>GLM</td>
</tr>
<tr>
<td>Distribution</td>
<td>Binary</td>
</tr>
<tr>
<td>Link Function</td>
<td>Logit</td>
</tr>
<tr>
<td>Optimization Technique</td>
<td>Newton-Raphson with Ridging</td>
</tr>
</tbody>
</table>

Class Level Information

<table>
<thead>
<tr>
<th>Class</th>
<th>Levels</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>3</td>
<td>group1 group2 group3</td>
</tr>
</tbody>
</table>

Number of Observations Read 1000
Number of Observations Used 1000

Parameter Estimates

| Parameter | Estimate | 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
- 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:

```sas
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:

```sas
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:

```sas
%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.
Chapter 2: Shared Concepts and Topics

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:

```sas
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.

**DATASERVER=“name”**

specifies the name of the server on Teradata systems as defined through the *hosts* file and as used in the LIBNAME statement for Teradata. For example, assume that the *hosts* file defines the server for Teradata as follows:

```plaintext
myservercop1 33.44.55.66
```

Then a LIBNAME specification would be as follows:
libname TDLib teradata server=myserver user= password= database= ;

A PERFORMANCE statement to induce running alongside the Teradata server would specify the following:

    performance dataserver="myserver";

The DATASERVER= option is not required if you specify the GRIDMODE=option in the PERFORMANCE statement or if you set the GRIDMODE environment variable.

Specifying the DATASERVER= option overrides the GRIDDATASERVER environment variable.

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
specifies whether the high-performance analytical procedure runs in symmetric (SYM) mode or asymmetric (ASYM) mode. The default is GRIDMODE=SYM. For more information about these modes, see the section “Symmetric and Asymmetric Distributed Modes” on page 7.

If this option is specified, it 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 16 and the SAS LASR Analytic Server: Administration 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:

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

```sas
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:

```sas
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 Data Mining 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:

```sas
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:

```sas
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 execute 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 3
The HP4SCORE Procedure

Contents

Overview: HP4SCORE Procedure ................................................. 39
PROC HP4SCORE Features ...................................................... 40
PROC HP4SCORE Contrasted with Other SAS Procedures ............... 40
Getting Started: HP4SCORE Procedure .................................... 40
Syntax: HP4SCORE Procedure .................................................. 42
PROC HP4SCORE Statement ................................................... 42
ID Statement ................................................................. 42
PERFORMANCE Statement ................................................... 43
SCORE Statement ............................................................ 43
Details: HP4SCORE Procedure ................................................. 43
Displayed Output ............................................................. 44
Output Data Set ............................................................... 44
Examples: HP4SCORE Procedure ............................................. 44
Example 3.1: Running PROC HP4SCORE ................................. 45

Overview: HP4SCORE Procedure

The HP4SCORE procedure is a high-performance procedure that scores a data set with a forest predictive model that was previously trained by the HPFOREST procedure. See Chapter 5, “The HPFOREST Procedure.”

The forest predictive model is an ensemble of hundreds of decision trees that are used to predict a target. The target can have either an interval or a nominal measurement level. Each decision tree consists of a sequence of rules that are applied to the observation to arrive at the prediction. The final prediction is either an average of the individual predictions for a target that has an interval measurement level or is derived from the average of the individual posterior probabilities for a target that has a nominal measurement level.

The HP4SCORE procedure can score the data in concurrent threads run in parallel when executed in single-machine (SMP) or distributed (MPP) mode. See the section “Processing Modes” on page 6 in Chapter 2, “Shared Concepts and Topics,” for details about how to configure the execution mode of SAS High-Performance Analytics procedures.
PROC HP4SCORE Features

The HP4SCORE procedure is designed to be used subsequent to the training performed by the HPFOREST procedure. PROC HP4SCORE requires the binary model file that is created by the HPFOREST procedure and that encapsulates the full forest model. PROC HP4SCORE then applies the model for scoring a specified data set. For successful scoring, the variables in the data set that is specified in the input to PROC HP4SCORE must have the same attributes as the data set on which the model was trained (that is, the data set that was specified as input to PROC HPFOREST). If the attributes do not match, the HP4SCORE procedure stops with an error.

PROC HP4SCORE Contrasted with Other SAS Procedures

No other SAS procedure scores a forest model. SAS high-performance procedures create DATA step programs that incorporate the scoring logic in the model. The decision tree procedures in SAS Enterprise Miner (the ARBOR, DMSPLIT, and SPLIT procedures) also output the score code directly. Because the forest model is an ensemble of hundreds of decision trees and each decision tree can contain hundreds or thousands of decision rules for scoring, the DATA step program can become extremely large. In addition to slow scoring performance, a very large DATA step program also poses additional challenges. PROC HP4SCORE overcomes these difficulties by reading the binary model file and scoring the observations directly.

The HP4SCORE procedure takes full advantage of concurrent threads and distributed data. For general contrasts between SAS high-performance analytical procedures and other SAS procedures, see the section “Output Data Sets” on page 31.

Getting Started: HP4SCORE Procedure

This example shows the usage of the HP4SCORE procedure in conjunction with the HPFOREST procedure. The HPFOREST procedure first trains a model on the training data and saves the model as a binary file. The HP4SCORE procedure then uses the trained model to score a different data set.

The hypothetical data set contains the ratings by three different volunteers on six different proportions of fruits. The following DATA step creates the SAS data set PunchTrain with the proportions of watermelon, pineapple, orange, and the numerical ratings for each combination of fruit mix and the volunteer:

```
data PunchTrain;
  input watermelon pineapple orange rating;
datalines;
1.0 0.0 0.0 4.3
1.0 0.0 0.0 4.7
1.0 0.0 0.0 4.8
0.0 1.0 0.0 6.2
0.0 1.0 0.0 6.5
0.0 1.0 0.0 6.3
```
The following statements train the forest model on the data set with the rating as the target and different fruit mix proportions as the independent variables. The SAVE statement saves the model to a binary file `punchModel.sav` in the current directory. For the full details of the HPFOREST procedure options, see Chapter 5, “The HPFOREST Procedure.”

```sas
proc hpforest data=PunchTrain maxtrees=10;
  input watermelon pineapple orange;
  target rating;
  save file="punchModel.sav";
run;
```

The following DATA step creates the SAS data set `PunchScore`, which contains only the proportions of watermelon, pineapple, orange fruit:

```sas
data PunchScore;
  input watermelon pineapple orange;
  datalines;
  0.6 0.4 0.0
  0.9 0.1 0.0
  0.8 0.0 0.2
  0.5 0.3 0.2
  0.3 0.1 0.6
; run;
```

The following statements invoke HP4SCORE to score this data set:

```sas
proc hp4score data=PunchScore;
  score file="punchModel.sav" out=scoreout;
run;
```

The SAS data set `scoreout` contains the ratings for each input observation as predicted by the model. Output 3.1 shows the scoring results.
Syntax: HP4SCORE Procedure

The following statements are available in the HP4SCORE procedure:

PROC HP4SCORE data-options ;
   ID variables ;
   SCORE score-options ;
   PERFORMANCE performance-options ;

The PROC HP4SCORE statement and SCORE statements are required.

PROC HP4SCORE Statement

PROC HP4SCORE DATA=< libref. >SAS-data-set ;

The PROC HP4SCORE statement invokes the procedure.

DATA=< libref. >SAS-data-set

    names the SAS data set to be used for scoring by PROC HP4SCORE. This argument is required.

If the data are already distributed, PROC HP4SCORE reads the data alongside the distributed database. The different nodes then independently read the data rows, score them and write them back. See the section “Processing Modes” on page 6 for information about the various execution modes and the section “Alongside-the-Database Execution” on page 13 for information about the alongside-the-database model.

ID Statement

ID variables ;

The ID statement lists one or more variables from the input data set that are transferred to the output data set that is specified in the SCORE statement. The ID statement accepts numeric and character variables. By default, high-performance analytical procedures do not include all variables from the input data set in output data sets.
The ID statement is optional. However, when running in distributed mode or with concurrent threads, the SCORE statement rearranges the observations. An ID variable is needed to correctly merge the output data with other variables from the input data set.

**PERFORMANCE Statement**

```latex
PERFORMANCE < performance-options > ;
```

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of the HP4SCORE procedure. You can also use the PERFORMANCE statement to control whether the HPFOREST procedure executes in single-machine or distributed mode.

The PERFORMANCE statement is documented further in the section “PERFORMANCE Statement” on page 34 of Chapter 2, “Shared Concepts and Topics.”

**SCORE Statement**

```latex
SCORE score-options ;
```

The SCORE statement specifies the name of the model file and the output data set name.

When running in distributed mode or with concurrent threads, the SCORE statement rearranges the observations. An ID variable is needed to correctly merge the output data with other variables from the input data set.

You can specify the following **score-options**:

- **FILE=model-file-name**
  - specifies either the file reference or the full path and member name of the valid model file that was created by PROC HPFOREST.

- **MAXDEPTH=< n >**
  - produces predictions from trees pruned to a depth of \( n \). The trees are not pruned by default.

- **NTREES=< n >**
  - produces predictions from the first \( n \) trees only. Scoring with fewer trees can sometimes increase the speed without significantly reducing the accuracy.

- **OUT=< libref.>SAS-data-set**
  - names the output SAS data set to contain the scored results.

**Details: HP4SCORE Procedure**

When PROC HPFOREST saves the model, it saves only the variables that were used in the model. These might be all the input variables or a subset of those variables. The data set to be scored must contain the variables that are in the model. These variables must have same attributes as the training data. They include
variable names (differences in character case are ignored), the data type, and the length. If a variable that has a nominal measurement level has a level that was not present in the training data set, that level is treated as a missing value. The HP4SCORE procedure does not ignore observations with missing values while scoring. The full model is applied on all the observations, and the prediction is computed.

Displayed Output

The HP4SCORE procedure displays the following information tables:

- 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.

- The “Number of Observations” table displays the total number of observations that are read from the data set and the total number of observations scored.

- If you specify the DETAILS option in the PERFORMANCE statement, the procedure also produces a “Timing” table in which elapsed times for the main tasks of the procedure are displayed.

Output Data Set

The HP4SCORE procedure writes the scored results as a separate output data set, which contains one observation for each input observation. Only the ID variables specified in the ID statement are copied from the input data set to the output data set. The prediction variables depend on the measurement type of the target variable in the model. For a target that has an interval measurement level, a single prediction variable is generated. For each level of the target that has a nominal measurement level, a posterior probability variable is generated in addition to the final predicted level. The names of the variables are constructed using the rules that are explained in the SAS Enterprise Miner product documentation.

If the input data set is read in alongside-the-database mode from the SAS appliance, the output data set is written back in parallel. In this case, the output records are distributed across the processing nodes such that the input row and the corresponding output row are colocated.

Examples: HP4SCORE Procedure

All examples use the same training data set and the model file, which is created by PROC HPFOREST in the following statements:

```sas
data hmeq;
  set sampsio.hmeq;
  id = _n_;
run;
```
Example 3.1: Running PROC HP4SCORE

When running in distributed mode or with concurrent threads, the order of the observations in the PROC HP4SCORE output might differ from the original order. The observations must be sorted before they are merged with the original data. This example runs PROC HP4SCORE and computes the misclassification rate of the scored output twice: once without sorting, and once with sorting. Without sorting, the misclassification rate is incorrect.

```plaintext
filename outmodel "C:\Temp\HPForestModel";
proc hp4score data=hmeq;
   id id;
   score file=outmodel out=scoreout3;
run;
```

The ID statement in PROC HP4SCORE specifies a variable to include in the output that contains the original observation number. The output does not contain the original target variable. We need to add it in order to compute the misclassification rate. The following data step incorrectly merges the target variable with the predictions output from PROC HP4SCORE. It is incorrect because the predictions are not sorted. The data step also creates a variable to detect whether the observations are mismatched and another variable to detect whether the prediction equals the actual target.

```plaintext
data score;
   merge hmeq(keep=JOB) scoreout3;
   if id ne _n_ then unequalobs=1;
   else unequalobs=0;
   if upcase(JOB) ne upcase(I_JOB) then misclass=1;
   else misclass=0;
run;

proc means data=score;
   var unequalobs misclass;
run;
```

Now we merge the data again, this time sorting predictions before merging them with the original data.

```plaintext
proc sort data=scoreout3 out=scoreout3;
   by id;
run;
```
data score;  
  merge hmeq(keep=JOB id) scoreout3;  
  by id;  
  if id ne _n_ then unequalobs=1;  
  else unequalobs=0;  
  if upcase(JOB) ne upcase(I_JOB) then misclass=1;  
  else misclass=0;  
run;

proc means data=score;  
  var unequalobs misclass;  
run;

Output 3.1.1 shows that HP4SCORE was run on a single-machine with 4 threads.

Output 3.1.1  Performance Information

The HPFOREST Procedure
Performance Information
Execution Mode Single-Machine
Number of Threads 4

Output 3.1.2 shows the output from PROC MEANS when the output from PROC HP4SCORE is not sorted. The table shows that 75 percent of the observations are mismatched: the actual target is from a different observation than the predicted target. The apparent misclassification rate is 59 percent. This result is incorrect.

Output 3.1.2  Output from PROC MEANS with Mismatched Data

<table>
<thead>
<tr>
<th>Variable</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>unequalobs</td>
<td>5960</td>
<td>0.75000</td>
<td>0.433049</td>
<td>0</td>
<td>1.000000</td>
</tr>
<tr>
<td>misclass</td>
<td>5960</td>
<td>0.58943</td>
<td>0.491979</td>
<td>0</td>
<td>1.000000</td>
</tr>
</tbody>
</table>
Output 3.1.3 shows the output from PROC MEANS when the output from PROC HP4SCORE is correctly sorted. The table shows that the target variable is matched to the correct target prediction, and that the true misclassification rate is 29 percent.

**Output 3.1.3** Output from PROC MEANS with Correctly Matched Data

<table>
<thead>
<tr>
<th>Variable</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>unequalobs</td>
<td>5960</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>misclass</td>
<td>5960</td>
<td>0.2934564</td>
<td>0.4553839</td>
<td>0</td>
<td>1.0000000</td>
</tr>
</tbody>
</table>

The MEANS Procedure
Overview: HPDECIDE Procedure

The HPDECIDE procedure creates optimal decisions that are based on a decision matrix that you specify, on prior probabilities, and on output from a modeling procedure. This output can be either posterior probabilities for a categorical target variable or predicted values for an interval target variable. The HPDECIDE procedure can also adjust the posterior probabilities for changes in the prior probabilities.

Some modeling procedures assume that the prior probabilities for categorical variable level membership either are all equal or are proportional to the relative frequency of the corresponding response level in the data set. PROC HPDECIDE enables you to specify other prior probabilities. Thus, you can conduct a sensitivity analysis without running the modeling procedure again.

The HPDECIDE procedure runs in either single-machine mode or distributed mode.

NOTE: Distributed mode requires SAS High-Performance Data Mining.
Because the HPDECIDE procedure is a high-performance analytical procedure, it also does the following:

- enables you to run in distributed mode on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode on the server where SAS is installed
-exploits all the available cores and concurrent threads, regardless of execution mode

For more information, see the section “Processing Modes” on page 6 in Chapter 2, “Shared Concepts and Topics.”

### Getting Started: HPDECIDE Procedure

The HPDECIDE procedure can adjust posterior probabilities from a modeling procedure to make decisions. This fictitious example shows how to use the HPDECIDE procedure to adjust posterior probabilities from the DISCRIM procedure, and how to use a revenue matrix and cost constants to make decisions.

In a population of men who consult urologists for prostate problems, 70% have benign enlargement of the prostate, 25% have an infection, and 5% have cancer. A sample of 100 men is taken, and two new diagnostic measures, X and Y, are taken on each patient. The training set also includes the diagnosis that is made by reliable, conventional methods. For each patient, three treatments are available: antibiotics, surgery, or no treatment. Antibiotics are effective against infection, but they might have moderately bad side effects. Antibiotics have no effect on benign enlargement or cancer. Surgery is effective for all diseases but has potentially severe side effects such as impotence.

The first step is to create the sample of 100 men. To simulate the measurements of diagnostics X and Y, this example uses the SAS random number generator. Because you specify the initial seed to the random number generator, all your results will be identical to those presented in this example.

The following statements create the Prostate data set. The first 70 observations represent benign tumors, the next 25 represent infections, and the final 5 represent cancer.

```sas
data Prostate;
  length dx $14;
  dx='Benign';
  mx=30; sx=10;
  my=30; sy=10;
  n=70;
  link generate;
  dx='Infection';
  mx=70; sx=20;
  my=35; sy=15;
  n=25;
  link generate;
  dx='Cancer';
  mx=50; sx=10;
  my=50; sy=15;
  n=5;
  link generate;
```
The following statements run the DISCRIM procedure, which assumes that all prior probabilities are equal (1/3 for this example). In this example, the DISCRIM procedure misidentifies some of the benign tumors as cancer or an infection. Also, it misidentifies some of the infections as benign tumors.

```sas
proc discrim data=prostate out=outdis short;
class dx;
var x y;
run;
```

Because PROC DISCRIM misidentifies some of the data, you want to create a data set that contains prior probabilities and revenue information. The revenue information indicates the benefit of each treatment. The cost of each treatment, such as bad side effects, will be specified later in a DECISION statement. The following DATA step creates the revenue matrix:

```sas
data rx(type=revenue);
  input dx $14. eqprior prior nothing antibiot surgery;
datalines;
  Benign  0.3333 70 0 0 5
  Infection  0.3333 25 0 10 10
  Cancer  0.3333 5 0 0 100
;
```

The variable `eqprior` defines an equal prior probability for each diagnosis, and the variable `prior` uses information that is known from the sample data set. The other variables define the revenue of each treatment option. The revenue (benefit) of doing nothing in either case is 0, and the benefit of taking antibiotics is relevant only if the patient has an infection. Surgery can remove a benign tumor, but it has very little benefit because it is not necessary. Surgery completely removes an infection, so it has the same value as antibiotics. Finally, surgery can remove a cancerous tumor and therefore is an immense benefit to the patient.

The following statements assign a treatment to each patient. In the DECISION statement, you specify the costs of treatment. The cost of doing nothing is 0, the cost of antibiotics is 5, and the cost of surgery is 20.

```sas
proc hpdecide data=outdis out=decOut outstat=decSum;
target dx;
posteriors benign infection cancer;
decision decdata=rx
  oldpriorvar=eqprior priorvar=prior
decvars=nothing antibiot surgery
cost= 0 5 20;
run;
```

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

Output 4.1 shows the fit statistics information.
The data set `decOut` indicates that only one benign tumor was misidentified, but the number of infections that were misidentified as benign is similar to the results from the DISCRIM procedure. All the cancerous tumors were identified and assigned the treatment of surgery, as was the lone misidentified benign tumor. The total profit for all patients, identified in the data set `decSum`, is 470.

Because medical decisions are personal, the costs that are associated with each treatment can vary considerably from patient to patient. Some patients regard the side effects of surgery as more severe than other patients. Likewise, the costs of antibiotics might vary because of the patients’ insurance plans. The following statements assume a higher cost for surgery and leave the other costs constant:

```r
proc hpdecide data=outdis out=decOut2 outstat=decSum2;
  target dx;
  posteriors benign infection cancer;
  decision decdata=rx
    oldpriorvar=eqprior priorvar=prior
    decvars=nothing antibiot surgery
    cost= 0 5 50;
run;
```

```r
proc print data=decSum2;
run;
```

Output 4.2 shows the fit statistics information that results from the higher cost of surgery.

```
Obs  _PROF_  _APROF_
1    285     2.85
```

Notice that the misclassified benign tumor is now correctly classified. However, one of the cancer cases is identified as benign; this is a costly mistake. Notice in `decOut` that the total profit has been reduced from 470 to 285.
Syntax: HPDECIDE Procedure

The following statements are available in the HPDECIDE procedure:

```sas
PROC HPDECIDE <options>;
   ID variables;
   FREQ variable;
   PERFORMANCE performance-options;
   POSTERIORS variables;
   PREDICTED variables;
   TARGET variable;
   DECISION DECDATA=<libref.>SAS-data-set <options>;
   CODE <options>;
```

PROC HPDECIDE Statement

```sas
PROC HPDECIDE <options>;
```

The PROC HPDECIDE statement invokes the procedure and identifies the input and output data sets. You also need the following statements:

- DECISION statement
- either a POSTERIORS or a PREDICTED statement
- TARGET statement

You can specify the following `options` in the PROC HPDECIDE statement:

**DATA=<libref.>SAS-data-set**

specifies the input data set that contains the output from a modeling procedure. The default is the most recently created data set. If the data are already distributed, PROC HPDECIDE reads the data alongside the distributed database. For information about the various execution modes and about the alongside-the-database model, see the sections “Processing Modes” on page 6 and “Alongside-the-Database Execution” on page 13 in Chapter 2, “Shared Concepts and Topics.”

**OUT=<libref.>SAS-data-set**

specifies the output data set, which always contains any variables from the input data set that is specified in the ID statement, the chosen decision (with a prefix of D_), and the expected consequence of the chosen decision (with a prefix of either EL_ or EP_).

If the target value is in the input data set, then the output data set also contains the following variables:
- the consequence of the chosen decision (which is computed from the target value and has a prefix of either CL_ or CP_) and the consequence of the best possible decision when the target value is known (this variable has a prefix of either BL_ or BP_).

If the profit matrix is revenue, then the output data set also contains the following variables:
- the investment cost (which has a prefix of IC_) and the return on investment (which has a prefix of ROI_).

Additionally, if the PRIORVAR= and OLDPRIORVAR= variables are specified, then this data set contains the recalculated posterior probabilities.
OUTFIT=libref.SAS-data-set
specifies an output data set to contain fit statistics. These statistics include the total and average profit or loss. You cannot specify this option when ROLE=SCORE. By default, this data set is not created.

ROLE=TRAIN | VALID | VALIDATION | TEST | SCORE
specifies the role of the input data set. This option affects the variables that are created in the DATA= data set. The default value is TEST. You can specify the following values:

- TRAIN specifies that the role of the input data set is training.
- VALIDATION specifies that the role of the input data set is validation.
- TEST specifies that the role of the input data set is testing.
- SCORE specifies that the role of the input data set is scoring.

CODE Statement

CODE < options > ;
The CODE statement generates SAS DATA step code that can be used to score data sets. If neither the FILE= option nor the METABASE= option is specified, then the SAS code is written to the SAS log. You can specify both the FILE= option and the METABASE= option to write code to both locations.

- FILE=file-name
  names the file into which score code is saved.

- METABASE=catalog-spec
  specifies a catalog entry to contain the SAS score code. For example, you can specify METABASE=myLibrary.myCatalog.catalog-entry.

- RESIDUAL
  computes the variables that depend on the target variable in the score code.

DECISION Statement

DECISION DECDATA=libref.SAS-data-set < options > ;

Required Argument

- DECDATA=libref.SAS-data-set< (type)>
  names the input data set that contains the decision matrix or the prior probabilities, or both. This argument is required.

The named data set must contain the target variable that is specified in the TARGET statement. It might also contain decision variables that are specified in the DECVARS= option and prior probability variables that are specified in the PRIORVAR= option or the OLDPRIORVAR= option or both.
For a categorical target variable, there should be one observation for each class. Each entry $d_{ij}$ in the decision matrix indicates the consequence of selecting target value $i$ for variable $j$. If any class appears more than once in this data set, an error message is printed and the PROC HPDECIDE terminates. Any class value in the input data set that is not found in this data set is treated as a missing class value. The classes in this data set must correspond exactly to the variables in the POSTERIORS statement.

For an interval target variable, each row defines a knot in a piecewise linear spline function. The consequence of making a decision is computed by interpolation in the corresponding column of the decision matrix. If the predicted target value is outside the range of knots in the decision matrix, the consequence is computed by linear extrapolation. If the target values are monotonically increasing or decreasing, any interior target value is allowed to appear twice in data set. This enables you to specify discontinuities in the data. No target value is allowed to appear more than twice. If the target values are not monotonic, then they are sorted by PROC HPDECIDE and are not allowed to appear more than once.

The data set option type is specified in parentheses after the data set name when the data set is created or used. The possible values of type are LOSS, PROFIT, and REVENUE; the default is PROFIT.

Optional Arguments

You can specify the following options:

DECVARS=variables
specifies the numeric decision variables in the DECDATA= data set that contain the target-specific consequences for each decision. The decision variables cannot contain any missing values.

COST= list-of-costs
specifies the numeric constants that give the cost of a decision, the numeric variables in the input data set that contain case-specific costs, or any combination of constants and variables.

The number of cost constants and variables must match the number of decision variables in the DECVARS= option. You cannot use abbreviated variable lists. For any observation in which a cost variable is missing, the results for that observation are considered missing. By default, all costs are assumed to be 0. You can specify this option only when type is REVENUE.

PRIORVAR=variable
specifies the number variable in the DECDATA= data set that contains the prior probabilities that are used to make decisions. Prior probabilities are also used to adjust the total and average profit or loss. Prior probabilities cannot be missing or negative, and there must be at least one positive prior probability. The prior probabilities are not required to sum to 1. But if they do not sum to 1, then they are scaled by some constant so that they do sum to 1. If you do not specify this option, then no adjustment for prior probabilities is applied to the posterior probabilities.

OLDPRIORVAR=variable
specifies the numeric variable in the DECDATA= data set that contains the prior probabilities that were used the first time the model was fit. If you specify this option, then you must also specify the PRIORVAR= option.
**FREQ Statement**

FREQ variable;

The FREQ statement specifies a single numeric variable whose value represents the frequency of each observation. If you use the FREQ statement, the HPDECIDE procedure treats the data set as if each observation appeared \( n \) times, where \( n \) is the value of the FREQ variable. The FREQ variable has no effect on decisions of the adjustment for prior probabilities. It affects only the summary statistics in the OUTFIT= data set. If a value of the FREQ variable is not an integer, then the fractional part is not truncated. If a value of the FREQ variable is less than or equal to 0, then the observation does not contribute to the summary statistics. However, all the variables in the OUT= data set are processed as if the FREQ variable were positive.

**ID Statement**

ID variables;

The ID statement lists one or more variables from the input data set that are transferred to the output data set created by the HPDECIDE procedure. By default, to avoid data duplication for large data sets, the HPDECIDE procedure does not include any variables from the input data set in the output data sets. Therefore, the ID statement can be used to copy variables from the input data set to the output data set.

**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 HPDECIDE procedure.

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

The PERFORMANCE statement is documented further in the section “PERFORMANCE Statement” on page 34 of Chapter 2, “Shared Concepts and Topics.”

**POSTERIORS Statement**

POSTERIORS variable-list ;

The POSTERIORS statement specifies a list of the numeric variables in the input data set that contain the estimated posterior probabilities that correspond to the categories of the target variable. If one of the following conditions is met, then an observation is set to missing and the variable _WARN_ contains the flag \( P \):

- The posterior probability is missing, negative, or greater than 1.
• There is a nonzero posterior that corresponds to a zero posterior.
• There is not at least one valid positive posterior probability.

The order of the variables in the variable-list must correspond exactly to the order of the classes in the data set that is specified in the DECDATA= option in the DECISION statement.

PREDICTED Statement

PREDICTED variable;

The PREDICTED statement specifies the numeric variable in the input data set that contains the predicted values of an interval target variable. You can specify only an interval target variable in the PREDICTED statement. You cannot use both the POSTERIORS statement and the PREDICTED statement.

TARGET Statement

TARGET variable;

The TARGET statement specifies which variable is the target variable in the data set that is specified in the DECDATA= option in the DECISION statement. The TARGET statement is required.

The HPDECIDE procedure searches for a target variable that has the same name in the input data set. If none is found, then the HPDECIDE procedure assumes that actual target values are unknown. For a categorical variable, the target variables in the data sets that are specified in the DATA= option in the PROC HPDECIDE statement and in the DECDATA= option in the DECISION statement do not need to be the same type because only the formatted values are used for comparisons. For an interval target, both variables must be numeric. If scoring code is generated by the CODE statement, the code formats the target variable by using the format and length from the DATA= data set.

Details: HPDECIDE Procedure

Decision Matrix

The decision matrix contains columns (decision variables) that correspond to each decision and rows (observations) that correspond to target values. The values of the decision variables represent target-specific consequences, which might be profit, loss, or revenue. These consequences are the same for all cases that are scored.

For each decision, there might also be either a cost variable or a numeric constant. The values of these variables represent case-specific consequences, which are always costs. These consequences do not depend on the target values of the cases that are scored. Costs are used for computing return on investment as

\[
\frac{\text{revenue} - \text{cost}}{\text{cost}}
\]
Variables

Cost variables might be specified only if the decision data set contains revenue, not profit or loss. Therefore, if revenues and costs are specified, profits are computed as revenue minus cost. If revenues are specified without costs, the costs are assumed to be 0. The interpretation of consequences as profits, losses, revenues, and costs is needed only to compute return on investment. You can specify values in the decision data set that are target-specific consequences but that might have some practical interpretation other than profit, loss, or revenue. Likewise, you can specify values for the cost variables that are case-specific consequences but that might have some practical interpretation other than costs. If the revenue/cost interpretation is not applicable, the values that are computed for return on investment might not be meaningful.

The HPDECIDE procedure chooses the optimal decision for each observation. If the type of decision data set (as specified in the DECISION statement) is PROFIT or REVENUE, PROC HPDECIDE chooses the decision that produces the maximum expected or estimated profit. If type is LOSS, PROC HPDECIDE chooses the decision that produces the minimum expected or estimated loss.

If the actual value of the target variable is known, the HPDECIDE procedure calculates the following:

- the consequence of the chosen decision for the actual target value for each case
- the best possible consequence for each case
- summary statistics that give the total and average profit or loss

Displayed Output

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.

ODS Table Names

Table 4.1 lists the names of the ODS tables that are created by the HPDECIDE procedure. You must use these names in ODS statements.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Required Statement or Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>PerformanceInfo</td>
<td>Performance information</td>
<td>Default output</td>
</tr>
<tr>
<td>Timing</td>
<td>Timing</td>
<td>PERFORMANCE statement with DETAILS option</td>
</tr>
</tbody>
</table>
Example 4.1: Using a Revenue Matrix and Cost Variables to Make Decisions

This fictitious example demonstrates how to use PROC HPDECIDE to adjust posterior probabilities and how to use a revenue matrix and cost variables to make decisions. In the following DATA steps, the categorical target variable tar has two levels in the data set data1: a and b.

```plaintext
data data1(drop=i);
  do i=1 to 5;
    tar="b";
    if i<3 then tar="a";
    p_a=abs(ranuni(81923));
    p_b=abs(1-p_a);
    c1=ranpoi(38192,5);
    c2=ranpoi(28131,7);
    output;
  end;
run;

data decdata1(type=revenue);
  input dv1-dv2 op np tar $;
  cards;
  5 3 .5 .2 a
  6 3 .5 .8 b
;```

The estimated posterior probabilities that correspond to the categories of the target variable are denoted by p_a and p_b. The two cost variables, c1 and c2, represent the target-specific consequences for the decision variables dv1 and dv2, respectively. The variable op contains the “old” prior probabilities that were used the first time the model was fictitiously fit. The variable np contains the “new” prior probabilities that are used to make decisions.

The following statements take the data set data1 and the data set decdata1, and output the data set out1 and the data set outstat1. The target variable is tar. Because the input data set resides on the client and no PERFORMANCE statement is specified, the client performs all computations.

```plaintext
proc hpdecide data=data1 out=out1 outstat=outstat1;
  decision decdata=decdata1 decvars=dv1-dv2
    oldpriorvar=op priorvar=np cost=c1 c2;
  posteriors p_a p_b;
  target tar;
  performance details nthreads=2;
run;

proc print data=out1;
  var p_a p_b I_tar F_tar dv1 dv2 D_DECDATA1 EP_DECDATA1 CP_DECDATA1;
run;

proc print data=outstat1;
run;```
Output 4.1.1 shows the out1 data set, which displays the decision for each observation. The adjusted posterior probabilities are also shown in the out1 data set.

**Output 4.1.1  out1 Data Set**

<table>
<thead>
<tr>
<th>Obs</th>
<th>p_a</th>
<th>p_b</th>
<th>I_tar</th>
<th>F_tar</th>
<th>dv1</th>
<th>dv2</th>
<th>D_DECDATA1</th>
<th>EP_DECDATA1</th>
<th>CP_DECDATA1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.35602</td>
<td>0.64398</td>
<td>B</td>
<td>A</td>
<td>-6</td>
<td>dv1</td>
<td>0.64398</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.38571</td>
<td>0.61429</td>
<td>B</td>
<td>A</td>
<td>-1</td>
<td>dv1</td>
<td>0.61429</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.18922</td>
<td>0.81078</td>
<td>B</td>
<td>B</td>
<td>-2</td>
<td>dv1</td>
<td>-1.18922</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.11035</td>
<td>0.88965</td>
<td>B</td>
<td>B</td>
<td>0</td>
<td>dv1</td>
<td>4.88965</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.29158</td>
<td>0.70842</td>
<td>B</td>
<td>B</td>
<td>-2</td>
<td>dv2</td>
<td>-3.29158</td>
<td>-2</td>
<td></td>
</tr>
</tbody>
</table>

Output 4.1.2 shows the outstat1 data set, which shows that the total profit is 2.6666666667 and the average profit is 0.5333333333, based on the decisions from the out1 data set.

**Output 4.1.2  outstat1 Data Set**

<table>
<thead>
<tr>
<th>Obs</th>
<th>PROF</th>
<th>APROF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.66667</td>
<td>0.53333</td>
</tr>
</tbody>
</table>

Output 4.1.3 shows the performance information.

**Output 4.1.3  Performance Information**

The HPDECIDE Procedure

Performance Information

Execution Mode       Single-Machine
Number of Threads    2
Example 4.2: Running with Client Data in Distributed Mode

This example demonstrates how to use PROC HPDECIDE to make decisions when the type of the decision data set is LOSS and the target is continuous. When the input data set resides on the client and a PERFORMANCE statement with a NODES= option is specified, as in the following statements, PROC HPDECIDE copies the data set to the SAS appliance, where the computation is performed:

```sas
data data2;
  input tar p ;
cards;
  1  6
  2  2
  3  1
  3  4
  4  2
  5  0
  3  3
  6  2
  7  1
  3  5
;

data decdata2(type=loss);
  input dv1 dv2 dv3 tar ;
cards;
  0  0  3  0
  1  0  2  1
  0  2  1  2
  0  3  0  3
;
```

The following statements take as input the data2 and decdata2 data sets (which reside on the client) and output the out2 and outstat2 data sets. The decision variables are dv1, dv2, and dv3, and the target variable is tar. Because five grid nodes are specified in the PERFORMANCE statement, the input data sets are distributed from the client to the five grid nodes, which perform the computations and then write the output data sets back to the client.

```sas
proc hpdecide data=data2 out=out2 outstat=outstat2;
  decision decdata=decdata2 decvars=dv1-dv3;
  predicted p;
  target tar;
  performance details nodes=5 nthreads=8
    host="&GRIDHOST" install="&GRIDINSTALLLOC";
run;
```

```sas
proc print data=out2;
run;
```

```sas
proc print data=outstat2;
run;
```
Output 4.2.1 shows the out2 data set, which displays the decision for each observation. For each observation, the continuous target variable was interpolated.

**Output 4.2.1 out2 Data Set**

<table>
<thead>
<tr>
<th>Obs</th>
<th>p</th>
<th>dv1</th>
<th>dv2</th>
<th>dv3</th>
<th>D_DECCDATA2</th>
<th>EL_DECCDATA2</th>
<th>CL_DECCDATA2</th>
<th>BL_DECCDATA2</th>
<th><em>WARN</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>-3</td>
<td>dv3</td>
<td>-3</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>dv1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>dv2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>-1</td>
<td>dv3</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>dv1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>dv1</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>dv1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>dv1</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>dv2</td>
<td>0</td>
<td>7</td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>-2</td>
<td>dv3</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Output 4.2.2 shows the outstat2 data set, which shows that the total loss is 0 and the average loss is 0, based on the decisions from the out2 data set.

**Output 4.2.2 outstat2 Data Set**

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>LOSS</em></th>
<th><em>ALOSS</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12.0</td>
<td>1.2</td>
</tr>
</tbody>
</table>

Output 4.2.3 shows the performance information.

**Output 4.2.3 Performance Information**

<table>
<thead>
<tr>
<th>Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Node</td>
</tr>
<tr>
<td>Install Location</td>
</tr>
<tr>
<td>Execution Mode</td>
</tr>
<tr>
<td>Grid Mode</td>
</tr>
<tr>
<td>Number of Compute Nodes</td>
</tr>
<tr>
<td>Number of Threads per Node</td>
</tr>
</tbody>
</table>
Output 4.2.4 shows the timing information.

**Output 4.2.4** Timing Information

<table>
<thead>
<tr>
<th>Task</th>
<th>Seconds</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup of Distributed Environment</td>
<td>1.05</td>
<td>62.68%</td>
</tr>
<tr>
<td>Data Transfer from Client</td>
<td>0.00</td>
<td>0.14%</td>
</tr>
<tr>
<td>Computation</td>
<td>0.00</td>
<td>0.09%</td>
</tr>
<tr>
<td>Writing Output</td>
<td>0.62</td>
<td>37.09%</td>
</tr>
</tbody>
</table>
# Chapter 5
## The HPFOREST Procedure

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: HPFOREST Procedure</td>
<td>66</td>
</tr>
<tr>
<td>PROC HPFOREST Features</td>
<td>67</td>
</tr>
<tr>
<td>PROC HPFOREST Contrasted with Other SAS Procedures</td>
<td>67</td>
</tr>
<tr>
<td>Getting Started: HPFOREST Procedure</td>
<td>67</td>
</tr>
<tr>
<td>Syntax: HPFOREST Procedure</td>
<td>73</td>
</tr>
<tr>
<td>PROC HPFOREST Statement</td>
<td>73</td>
</tr>
<tr>
<td>FREQ Statement</td>
<td>76</td>
</tr>
<tr>
<td>INPUT Statement</td>
<td>76</td>
</tr>
<tr>
<td>ID Statement</td>
<td>77</td>
</tr>
<tr>
<td>PERFORMANCE Statement</td>
<td>77</td>
</tr>
<tr>
<td>SAVE Statement</td>
<td>78</td>
</tr>
<tr>
<td>SCORE Statement</td>
<td>78</td>
</tr>
<tr>
<td>TARGET Statement</td>
<td>78</td>
</tr>
<tr>
<td>Details: HPFOREST Procedure</td>
<td>79</td>
</tr>
<tr>
<td>Bagging the Data</td>
<td>79</td>
</tr>
<tr>
<td>Training a Decision Tree</td>
<td>79</td>
</tr>
<tr>
<td>Controlling for Variable Selection Bias</td>
<td>80</td>
</tr>
<tr>
<td>Selecting a Splitting Variable</td>
<td>83</td>
</tr>
<tr>
<td>Searching for a Splitting Rule</td>
<td>85</td>
</tr>
<tr>
<td>Rules</td>
<td>85</td>
</tr>
<tr>
<td>Criteria</td>
<td>85</td>
</tr>
<tr>
<td>Algorithm</td>
<td>86</td>
</tr>
<tr>
<td>Predicting an Observation</td>
<td>86</td>
</tr>
<tr>
<td>Computing the Average Square Error and Misclassification Rate</td>
<td>87</td>
</tr>
<tr>
<td>Adjusting Statistics When Sampling Target Classes Unevenly</td>
<td>87</td>
</tr>
<tr>
<td>Formulas for Adjusting the Predictions and Fit Statistics</td>
<td>88</td>
</tr>
<tr>
<td>Why the Adjustments Matter</td>
<td>89</td>
</tr>
<tr>
<td>Technical Derivations of Adjustment Formulas</td>
<td>93</td>
</tr>
<tr>
<td>Handling Missing Values</td>
<td>94</td>
</tr>
<tr>
<td>Strategies</td>
<td>94</td>
</tr>
<tr>
<td>Specifics</td>
<td>95</td>
</tr>
<tr>
<td>Handling Values That Are Absent from Training Data</td>
<td>96</td>
</tr>
<tr>
<td>Measuring Variable Importance</td>
<td>96</td>
</tr>
<tr>
<td>Loss Reduction</td>
<td>96</td>
</tr>
<tr>
<td>Breiman’s Method</td>
<td>98</td>
</tr>
<tr>
<td>Bias and Correlation</td>
<td>99</td>
</tr>
</tbody>
</table>
Overview: HPFOREST Procedure

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

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

The HPFOREST procedure creates decision trees that differ from each other in two ways. First, the training data for a tree is a sample, without replacement, from the original training data of the forest. Second, the input variables considered for splitting a node are randomly selected from all available inputs. Among these variables, the HPFOREST procedure considers only a single variable when forming a splitting rule. The chosen variable is the one that is most associated with the target.

PROC HPFOREST runs in either single-machine mode or distributed mode. In distributed mode, PROC HPFOREST trains decision trees in parallel, and accesses all the data for every tree. Note: Distributed mode requires SAS High-Performance Data Mining.
PROC HPFOREST Features

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

The HPFOREST procedure deletes from the training data any observation that has a missing target value or a FREQ variable whose value is less than or equal to 0.

Because the HPFOREST procedure is a high-performance analytical procedure, it also does the following:

- enables you to run in distributed mode on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode on the server where SAS is installed
- exploits all the available cores and concurrent threads, regardless of execution mode

However, the current release of PROC HPFOREST copies all the data to all cores in distributed mode and limits the use of concurrent threads to reading and scoring data. For more information, see the section “Processing Modes” on page 6 in Chapter 2, “Shared Concepts and Topics.”

PROC HPFOREST Contrasted with Other SAS Procedures

No SAS procedure other than PROC HPFOREST creates a forest of decision trees for predictive modeling. PROC HPSPLIT in SAS HPSTAT creates a single decision tree, as does PROC ARBORETUM in SAS Enterprise Miner. These procedures search for a split on every variable in every node; the HPFOREST procedure searches for a split on only one variable in a node: the variable that has the largest association with the target among candidates randomly selected in that node. Consequently, the HPFOREST procedure creates different trees than the other procedures.

The ARBORETUM procedure distinguishes between split-based and observation-based variable importance measures. The HPFOREST procedure calls these measures loss reduction and Breiman’s method of variable importance, respectively.

The HPFOREST and HPSPLIT procedures are high-performance analytical procedures; the ARBORETUM procedure is not.

Getting Started: HPFOREST Procedure

This example uses diabetes data to illustrate PROC HPFOREST. Diabetes is a major American disease. The American Diabetes Association estimates that over 8% of Americans have diabetes, and diabetes costs Americans over $175 billion a year. The National Institute of Diabetes and Digestive and Kidney Diseases (NIDDK) has been studying diabetes and obesity in the Pima Indians in Arizona for over 30 years. Smith et al. (1988) prepared some of the NIDDK data for forecasting the onset of diabetes mellitus, and then donated
the data for community use. Since then the data has been applied to dozens of experimental algorithms for predicting the onset of diabetes.

The Pima Indians diabetes data are available from the UCI Machine Learning Repository (Asuncion and Newman 2007) http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes. The following SAS statements create a SAS data set from the data downloaded into a file called c:\diabetes_data.txt:

```sas
data diabetes;
  infile 'c:\diabetes_data.txt' delimiter=',
  input NumPregnancies plasmaGlucose diastolicBloodPr tricepsSkinfold hrSerumInsulin BodyMassIndex DiabetesPedigreeFn Age diabetes $;
run;
```

The variable diabetes has values 0 and 1, 1 indicating the presence of diabetes. The other variables are raw measures on interval scales, except for DiabetesPedigreeFn which is an interval variable created by Smith et al. (1988) to capture the family history of diabetes. PROC HPFOREST uses an interval scale for numeric variables and a nominal scale for categorical variables unless the scale is specified. The following statements run PROC HPFOREST and saves the model in a binary file.

```sas
proc hpforest data=diabetes ;
  input NumPregnancies plasmaGlucose diastolicBloodPr tricepsSkinfold hrSerumInsulin BodyMassIndex DiabetesPedigreeFn Age ;
  target diabetes;
  save file="model";
run;
```
Getting Started: HPFOREST Procedure

Output 5.1 shows that the program ran locally and that four threads were available. The listing also shows the values of the training parameters and the number of observations (768). No parameters are specified in the PROC HPFOREST statement; therefore, all the values are default. The maximum number of decision trees to create is 50. The VARS_TO_TRY= option equals 3, indicating that 3 of the 8 input variables are randomly selected to be considered for a splitting rule.

**Figure 5.1** HPFOREST Getting Started Example Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category Bins</td>
<td>30 (Default)</td>
</tr>
<tr>
<td>Leaf Size</td>
<td>5 (Default)</td>
</tr>
<tr>
<td>Maximum Depth</td>
<td>50 (Default)</td>
</tr>
<tr>
<td>Maximum Trees</td>
<td>50 (Default)</td>
</tr>
<tr>
<td>Minimum Category Size</td>
<td>5 (Default)</td>
</tr>
<tr>
<td>Variables to Try</td>
<td>3 (Default)</td>
</tr>
<tr>
<td>Alpha</td>
<td>0.2 (Default)</td>
</tr>
<tr>
<td>Exhaustive</td>
<td>5000 (Default)</td>
</tr>
<tr>
<td>Leaf Fraction</td>
<td>0.001 (Default)</td>
</tr>
<tr>
<td>Train Fraction</td>
<td>0.6 (Default)</td>
</tr>
<tr>
<td>Split Criterion</td>
<td>6 Gini</td>
</tr>
<tr>
<td>Missing Value Handling</td>
<td>1 Valid value</td>
</tr>
<tr>
<td>Number of Observations</td>
<td>N</td>
</tr>
<tr>
<td>Number of Observations Read</td>
<td>768</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>768</td>
</tr>
</tbody>
</table>
Figure 5.2 shows the Baseline Fit Statistics. PROC HPFOREST first computes baseline statistics without using a model. The listing shows a baseline misclassification rate of 0.349 because that is the proportion of observations with diabetes equal to 1.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Square Error</td>
<td>0.227</td>
</tr>
<tr>
<td>Misclassification Rate</td>
<td>0.349</td>
</tr>
</tbody>
</table>

Figure 5.3 shows the Fit Statistics. When not run on a grid, PROC HPFOREST computes fit statistics for a sequence of forests with an increasing number of trees. Typically the fit statistics improve (decrease) with the number of trees until reaching a rough bottom. Forest models provide an alternative estimate of average square error and misclassification rate, called the out-of-bag (OOB) estimate. It is a convenient substitute for an estimate based on test data, a less biased estimate of how the model will perform on future data. For more information, see the section “Bagging the Data” on page 79. The listing shows that the out-of-bag error estimate is worse (larger) than the estimate that evaluates all observations on all trees. This is usual. The out-of-bag misclassification rate for the model is 0.230, which is much less than the baseline rate of rate of 0.349. We can conclude that the model is good. In fact, 0.230 is the best misclassification rate (or close to it) in the literature for these data.
### Fit Statistics

<table>
<thead>
<tr>
<th>Number of Trees</th>
<th>Number of Leaves</th>
<th>Average Square Error (Full Data)</th>
<th>Average Square Error (OOB)</th>
<th>Average Misclassification Rate (Full Data)</th>
<th>Average Misclassification Rate (OOB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>0.165</td>
<td>0.215</td>
<td>0.240</td>
<td>0.302</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
<td>0.146</td>
<td>0.190</td>
<td>0.227</td>
<td>0.279</td>
</tr>
<tr>
<td>3</td>
<td>47</td>
<td>0.135</td>
<td>0.189</td>
<td>0.197</td>
<td>0.274</td>
</tr>
<tr>
<td>4</td>
<td>69</td>
<td>0.130</td>
<td>0.179</td>
<td>0.193</td>
<td>0.253</td>
</tr>
<tr>
<td>5</td>
<td>89</td>
<td>0.130</td>
<td>0.177</td>
<td>0.189</td>
<td>0.250</td>
</tr>
<tr>
<td>6</td>
<td>101</td>
<td>0.131</td>
<td>0.174</td>
<td>0.191</td>
<td>0.253</td>
</tr>
<tr>
<td>7</td>
<td>121</td>
<td>0.128</td>
<td>0.171</td>
<td>0.184</td>
<td>0.250</td>
</tr>
<tr>
<td>8</td>
<td>135</td>
<td>0.127</td>
<td>0.167</td>
<td>0.180</td>
<td>0.248</td>
</tr>
<tr>
<td>9</td>
<td>154</td>
<td>0.128</td>
<td>0.167</td>
<td>0.182</td>
<td>0.246</td>
</tr>
<tr>
<td>10</td>
<td>171</td>
<td>0.127</td>
<td>0.167</td>
<td>0.176</td>
<td>0.246</td>
</tr>
<tr>
<td>11</td>
<td>192</td>
<td>0.126</td>
<td>0.165</td>
<td>0.181</td>
<td>0.238</td>
</tr>
<tr>
<td>12</td>
<td>201</td>
<td>0.127</td>
<td>0.164</td>
<td>0.184</td>
<td>0.243</td>
</tr>
<tr>
<td>13</td>
<td>215</td>
<td>0.127</td>
<td>0.163</td>
<td>0.182</td>
<td>0.245</td>
</tr>
<tr>
<td>14</td>
<td>228</td>
<td>0.128</td>
<td>0.162</td>
<td>0.182</td>
<td>0.245</td>
</tr>
<tr>
<td>15</td>
<td>238</td>
<td>0.128</td>
<td>0.162</td>
<td>0.182</td>
<td>0.242</td>
</tr>
<tr>
<td>16</td>
<td>254</td>
<td>0.128</td>
<td>0.162</td>
<td>0.181</td>
<td>0.240</td>
</tr>
<tr>
<td>17</td>
<td>268</td>
<td>0.128</td>
<td>0.162</td>
<td>0.184</td>
<td>0.236</td>
</tr>
<tr>
<td>18</td>
<td>293</td>
<td>0.127</td>
<td>0.162</td>
<td>0.180</td>
<td>0.241</td>
</tr>
<tr>
<td>19</td>
<td>313</td>
<td>0.127</td>
<td>0.162</td>
<td>0.177</td>
<td>0.240</td>
</tr>
<tr>
<td>20</td>
<td>321</td>
<td>0.128</td>
<td>0.162</td>
<td>0.177</td>
<td>0.240</td>
</tr>
<tr>
<td>21</td>
<td>342</td>
<td>0.128</td>
<td>0.161</td>
<td>0.174</td>
<td>0.241</td>
</tr>
<tr>
<td>22</td>
<td>351</td>
<td>0.128</td>
<td>0.162</td>
<td>0.178</td>
<td>0.240</td>
</tr>
<tr>
<td>23</td>
<td>376</td>
<td>0.128</td>
<td>0.162</td>
<td>0.176</td>
<td>0.237</td>
</tr>
<tr>
<td>24</td>
<td>390</td>
<td>0.128</td>
<td>0.162</td>
<td>0.182</td>
<td>0.237</td>
</tr>
<tr>
<td>25</td>
<td>411</td>
<td>0.127</td>
<td>0.161</td>
<td>0.177</td>
<td>0.240</td>
</tr>
<tr>
<td>26</td>
<td>427</td>
<td>0.127</td>
<td>0.161</td>
<td>0.178</td>
<td>0.238</td>
</tr>
<tr>
<td>27</td>
<td>439</td>
<td>0.127</td>
<td>0.161</td>
<td>0.184</td>
<td>0.240</td>
</tr>
<tr>
<td>28</td>
<td>454</td>
<td>0.127</td>
<td>0.161</td>
<td>0.181</td>
<td>0.238</td>
</tr>
<tr>
<td>29</td>
<td>471</td>
<td>0.127</td>
<td>0.161</td>
<td>0.184</td>
<td>0.236</td>
</tr>
<tr>
<td>30</td>
<td>491</td>
<td>0.127</td>
<td>0.160</td>
<td>0.184</td>
<td>0.237</td>
</tr>
<tr>
<td>31</td>
<td>510</td>
<td>0.127</td>
<td>0.160</td>
<td>0.184</td>
<td>0.237</td>
</tr>
<tr>
<td>32</td>
<td>529</td>
<td>0.127</td>
<td>0.160</td>
<td>0.182</td>
<td>0.242</td>
</tr>
<tr>
<td>33</td>
<td>545</td>
<td>0.127</td>
<td>0.160</td>
<td>0.182</td>
<td>0.243</td>
</tr>
<tr>
<td>34</td>
<td>561</td>
<td>0.127</td>
<td>0.160</td>
<td>0.186</td>
<td>0.245</td>
</tr>
<tr>
<td>35</td>
<td>574</td>
<td>0.128</td>
<td>0.160</td>
<td>0.186</td>
<td>0.242</td>
</tr>
<tr>
<td>36</td>
<td>590</td>
<td>0.127</td>
<td>0.160</td>
<td>0.184</td>
<td>0.241</td>
</tr>
<tr>
<td>37</td>
<td>611</td>
<td>0.127</td>
<td>0.161</td>
<td>0.182</td>
<td>0.242</td>
</tr>
<tr>
<td>38</td>
<td>631</td>
<td>0.127</td>
<td>0.161</td>
<td>0.182</td>
<td>0.243</td>
</tr>
<tr>
<td>39</td>
<td>641</td>
<td>0.127</td>
<td>0.161</td>
<td>0.182</td>
<td>0.242</td>
</tr>
<tr>
<td>40</td>
<td>654</td>
<td>0.127</td>
<td>0.161</td>
<td>0.178</td>
<td>0.241</td>
</tr>
<tr>
<td>41</td>
<td>670</td>
<td>0.127</td>
<td>0.160</td>
<td>0.184</td>
<td>0.238</td>
</tr>
<tr>
<td>42</td>
<td>681</td>
<td>0.128</td>
<td>0.161</td>
<td>0.188</td>
<td>0.237</td>
</tr>
<tr>
<td>43</td>
<td>695</td>
<td>0.128</td>
<td>0.161</td>
<td>0.188</td>
<td>0.237</td>
</tr>
</tbody>
</table>
Estimates of variable importance appear after the fit statistics. The NRules column in Figure 5.4 shows the number of splitting rules that use a variable. Section “Measuring Variable Importance” on page 96 explains the measures of importance. Each measure is computed twice: once on training data and once on out-of-bag data. As with fit statistics, the out-of-bag estimates are less biased. The GiniOOB column is negative for four variables. The splitting rules involving these variables are, on average, spurious. The worst is DiabetesPedigreeFn, suggesting that family history of diabetes is a misleading predictor, at least when measured with DiabetesPedigreeFn. The MarinOOB statistic gives DiabetesPedigreeFn more importance. The main conclusion from fitting the forest model to these data is that plasmaGlucose is the most important predictor of future onset of diabetes. Three other variables also contribute to the prediction.
Syntax: HPFOREST Procedure

The following statements are available in the HPFOREST procedure:

```
PROC HPFOREST < option(s) > ;
  FREQ variable ;
  INPUT variable(s) < option(s) > ;
  ID variable(s) ;
  PERFORMANCE performance-options ;
  SAVE < options > ;
  SCORE < score-options > ;
  TARGET variable < option(s) > ;
```

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

---

PROC HPFOREST Statement

```
PROC HPFOREST < options > ;
```

The PROC HPFOREST statement invokes the procedure. You can specify one or more of the following optional arguments.

**DATA=< libref. >SAS-data-set**

names the SAS data set to be used by PROC HPFOREST for training the model. The default is the most recently created data set.

If the data are already distributed, the procedure reads the data alongside the distributed database. See the section “Processing Modes” on page 6 for the various execution modes and the section “Alongside-the-Database Execution” on page 13 for the alongside-the-database model. Data from all the computer grid nodes are combined into a structure that is optimized for model training and redistributed to the nodes. The different nodes then proceed independently with identical data to create decision trees.

**ALPHA=number**

specifies a threshold p-value for the significance level of a test of association of a candidate variable with the target. If no association meets this threshold, the node is not split. The default value is 0.2.

**CATBINS=k**

specifies the maximum number of categories of a nominal candidate variable to use in the association test. k refers only to the categories that are present in the training data in the node and that satisfy the MINCATSIZE= option. The categories are counted independently in each node. If more than k categories are present, then the least frequent categories are removed from the association test. Many infrequent categories can dilute a strong predictive ability of common categories. The search for a splitting rule uses all categories that satisfy the MINCATSIZE= options. The value of k must be a positive integer. The default value is 30.
EXHAUSTIVE=number
specifies the maximum number of splits to examine in a complete enumeration of all possible splits when the input variable is nominal and the target has more than two nominal categories. The exhaustive method of searching for a split examines all possible splits. If the number of possible splits is greater than number, then a heuristic search is done instead of an exhaustive search. The default value of number is 5,000.

IMPORTANCE=YES | NO
specifies whether to calculate loss reduction variable importance. Avoiding the calculation can save some memory resources. The default action is YES: calculate loss reduction variable importance.

LEAFFRACTION=f
specifies the smallest number of training observations that a new branch can have, expressed as the fraction of the number N of available observations in the DATA= data set. N might be less than the total number of observations in the data set because observations with a missing target value or non positive value of the variable specified in the FREQ statement are excluded from N. If you specify a number in the LEAFSIZE= option that implies a larger number than that specified in the LEAFFRACTION= option, f is ignored. The value f must be larger than 0 and less than 1. The default value is 0.001.

LEAFSIZE=n
specifies the smallest number of training observations a new branch can have. If you specify a value for the LEAFFRACTION= option that implies a larger value than n, the LEAFSIZE= option is ignored. The default value is 5.

MAXDEPTH=d
specifies the maximum depth of a node in any tree that PROC HPFOREST creates. The depth of a node equals the number of splitting rules needed to define the node. The root node has depth 0. The children of the root have depth 1, and on. The smallest acceptable value of d is 1. The default value of d is 50.

MAXTREES=n
specifies the number of trees in the forest. n is a positive integer. The number of trees in the resulting forest can be less than n when the HPFOREST procedure fails to split the training data for a tree. Up to two times n trees are attempted. If the procedure fails to split the training data for more than n trees, then less than n trees are created. The ALPHA=, LEAFSIZE=, and MINCATSIZE= options constrain the split search to form trees that are more likely to predict well using new data. Setting all of these options to 1 generally frees the search algorithm to find a split and train a tree, although the tree might not help the forest predict well. The default value of n is 50.

MINCATSIZE=n
specifies the minimum number of observations that a given nominal input category must have in order to use the category in a split search. Categorical values that appear in fewer than n observations are handled as if they were missing. The categories that occur in fewer than n observations are merged into the pseudo category for missing values for the purpose of finding a split. The policy for assigning such observations to a branch is the same as the policy for assigning missing values to a branch. The default value of n is 5.
MINUSEINSEARCH=n
specifies a threshold for utilizing missing values in the split search when MISSING=USEINSEARCH is specified as the missing value policy. If the number of observations in which the splitting variable has missing values in a node is greater than or equal to n, then PROC HPFOREST initiates the USEINSEARCH policy for missing values. See the section “Handling Missing Values” on page 94 for a more complete explanation. The default value of n is 1.

MISSING=USEINSEARCH | DISTRIBUTE
specifies how the training procedure handles an observation with missing values. If MISSING=USEINSEARCH and the number of training observations in the node is more than n, where n is the value of the MINUSEINSEARCH= option, then the missing value is used as a separate, legitimate value in the test of association and the split search. If MISSING=DISTRIBUTE, observations with a missing value of the candidate variable are omitted from the test of association and split search in that node. A splitting rule distributes such an observation to all branches. See the section “Handling Missing Values” on page 94 for a more complete explanation. By default, MISSING=USEINSEARCH.

NODESIZE=n | ALL
specifies the number of training observations to use for association tests and split searches. NODESIZE=ALL requests to use all the observations. The acceptable range is from two to two billion on most machines. The default value of n is 100,000.

The procedure counts the number of training observations in a node without adjusting the number with the values of the variable specified in the FREQ statement. If the count is larger than n, then the split search for that node is based on a random sample of size n. For categorical targets, the sample uses as many observations with less frequent target values as possible. The calculations for the association measures and split worth adjust the category counts to the category proportions in the node before sampling.

SEED=n
specifies the seed for generating random numbers. The HPFOREST procedure uses random numbers to select training observations for each tree and to select candidate variables in each node to split on. n is a nonnegative integer. Set n to 0 to use the internal default. The default value of the seed is 8,976,153.

SKIP_SEQ_ROWS=n
specifies the number of rows to skip in the “Fit Statistics” table in distributed mode. After every n trees that are trained on a grid node, the fit statistics on the node are updated, consolidated with statistics from other nodes, and eventually output in the “Fit Statistics” table. Each row in the table contains statistics for a specific number of trees in a forest. The table has gaps of up to n rows. The default value of n is 5.

SPLITSIZE=n
specifies the requisite number of training observations a node must have for the HPFOREST procedure to consider splitting it. By default, n is twice the value of the LEAFSIZE= option (or n is the value implied by LEAFFRACTION= option if the procedure ignores the LEAFSIZE= option). The procedure counts the number of observations in a node without adjusting the number with the values of the variable specified in the FREQ statement when it interprets the value specified in the LEAFFRACTION=, LEAFSIZE=, MINCATSIZE=, and SPLITSIZE= options.
TRAINFRACTION= \( f \)
specifies the fraction of training observations to train a tree with. Using less than all the available data often improves the generalization error. A different training sample is taken for each tree. \( f \) can be any number greater than 0 and at most 1. The default value of \( f \) is 0.6. PROC HPFOREST uses at least four observations in the training data regardless of how small \( f \) is (assuming four observations exist). If \( f \) is too small to accommodate the LEAFSIZE=, LEAFFRACTION=, and SPLITSIZE options then no tree is made. The TRAINN= option accepts an absolute number instead of a fraction to specify the same quantity. Specifying both the TRAINN= and TRAINFRACTION= options is an error.

TRAINN= \( n \)
specifies how many observations to use to train each tree. The observations are counted without regard to the variable specified in the FREQ statement. Using less than all the available data often improves the generalization error. A different training sample is taken for each tree. \( n \) can be any positive integer. If \( n \) is greater than the number of observations in the data set specified in the DATA= option, then all the available data are used. \( n \) must be at least 3 and large enough to accommodate the values of the LEAFSIZE=, LEAFFRACTION=, and SPLITSIZE options. The default value is 0.6 times the number of available observations in DATA= data set. The TRAINFRACTION= option accepts a fraction instead of an absolute number to specify the same quantity as the TRAINN= option. Specifying both the TRAINN= and TRAINFRACTION= is an error.

VARS_TO_TRY= \( m \) | ALL
specifies the number of input variables to consider splitting on in a node. \( m \) ranges from 1 to the number of input variables, \( v \). The default value of \( m \) is \( \sqrt{v} \). Specify VARS_TO_TRY=ALL to use all the inputs as candidates in a node.

---

**FREQ Statement**

```
FREQ variable ;
```

The variable in the FREQ statement identifies a numeric variable in the data set that contains the frequency of occurrence for each observation. PROC HPFOREST accepts any positive value of a frequency variable without converting the value to an integer. If the frequency value is missing or less than or equal to 0, the observation is not used in the analysis. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

---

**INPUT Statement**

```
INPUT variable(s) < option(s) > ;
```

The INPUT statement names input variables with common options. The INPUT statement can be repeated. You can specify the following options:

- **LEVEL=level**
  specifies the level of measurement of the variables. Accepted values of level are: BINARY, NOMINAL, ORDINAL, and INTERVAL.
**ORDER=order**
specifies the sorting order of the values of an ordinal input variable. Table 5.1 provides recognized values of order.

<table>
<thead>
<tr>
<th>Value of ORDER=</th>
<th>Variable Values Sorted By</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCENDING</td>
<td>Ascending order of unformatted values (default)</td>
</tr>
<tr>
<td>ASCFORMATTED</td>
<td>Ascending order of formatted values</td>
</tr>
<tr>
<td>DESCENDING</td>
<td>Descending order of unformatted values</td>
</tr>
<tr>
<td>DESFORMATTED</td>
<td>Descending order of formatted values</td>
</tr>
<tr>
<td>DSORDER</td>
<td>Order of appearance in the input data set</td>
</tr>
</tbody>
</table>

**NOTE:** The DSORDER sort option is not supported for input data sets stored on the SAS appliance.

**ID Statement**

```
ID variables;
```

The ID statement lists one or more variables from the input data set that are transferred to the output data set that is specified in the SCORE statement. By default, high-performance analytical procedures do not include all variables from the input data set in output data sets.

The ID statement is optional. However, when you are running in distributed mode or with concurrent threads, the SCORE statement rearranges the observations. An ID variable is needed to correctly merge the output data with other variables from the input data set.

**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 HPFOREST procedure.

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

The PERFORMANCE statement is documented further in the section “PERFORMANCE Statement” on page 34 of Chapter 2, “Shared Concepts and Topics.”
SAVE Statement

```
SAVE < option > ;
```

The SAVE statement outputs the forest model information into a binary file. You can specify the following option:

**FILE=filename**

names the file into which tree information is saved. The filename can be either a SAS file reference or the full path and member name of the binary file.

You can score new data sets against the forest model by specifying the name of this binary file in the FILE= option in the SCORE statement in the HP4SCORE procedure.

SCORE Statement

```
SCORE < score-options > ;
```

The SCORE statement applies the forest model to the training data and outputs a data set containing the ID variables that are specified in the ID statement, predictions, residuals, and decisions. The prediction variables depend on the measurement type of the target variable in the model. For a target that has an interval measurement level, a single prediction variable is generated. For each level of the target that has a nominal measurement level, a posterior probability variable is generated in addition to the final predicted level. The names of the variables are constructed using the rules that are explained in the SAS Enterprise Miner product documentation.

When you are running in distributed mode or with concurrent threads, the SCORE statement rearranges the observations. An ID variable is needed to correctly merge the output data with other variables from the input data set.

You can specify one or more of the following optional arguments.

**MAXDEPTH=< n >**

produces predictions from trees pruned to a depth of n. The trees are not truncated by default.

**NTREES=< n >**

produces predictions from the first n trees only. Scoring with fewer trees can sometimes increase the speed without significantly reducing the accuracy.

**OUT=libref>SAS-data-set**

names the output data set to contain the scored data.

TARGET Statement

```
TARGET variable < LEVEL=level > ;
```

The TARGET statement names the variable whose values PROC HPFOREST tries to predict. You can specify the following optional argument:
**LEVEL=** level

specifies the level of measurement. Accepted values of level are: BINARY, NOMINAL, and INTERVAL. Note that level cannot be ORDINAL.

---

**Details: HPFOREST Procedure**

**Bagging the Data**

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

The HPFOREST procedure samples the original data without replacement to create the training data for an individual tree. Most forest algorithms sample with replacement. The convention of sampling with replacement originated with Leo Breiman’s bagging algorithm (Breiman 1996, 2001). The word bagging stems from “bootstrap aggregating,” where “bootstrap” refers to a procedure that uses sampling with replacement. Breiman refers to the observations that are excluded from the sample as out-of-bag (OOB) observations. Therefore, observations in the training sample are called the bagged observations, and the training data for a specific decision tree are called the bagged data. Subsequently, Freedman and Popescu (2003) argued that sampling without replacement can provide more variability between the trees, especially with larger training sets.

The TRAINN= and TRAINFRACTION= options in the PROC HPFOREST statement specify the number of observations to sample without replacement into a bagged data set.

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

**Training a Decision Tree**

The HPFOREST procedure trains a decision tree by forming a binary split of the bagged data, then forming a binary split of each of the segments, and so on recursively until some constraint is met.

Creating a binary split involves a few subtasks:

1. selecting candidate inputs
2. reducing the number of nominal input categories
3. computing the association of each input with the target
4. searching for the best split that uses the most highly associated input
PROC HPFOREST selects candidate inputs independently in every node. The purpose of preselecting candidate inputs is to increase the differences between the trees, thereby decreasing the correlation and theoretically increasing the quality of the forest predictions. The selection is random, and each input has the same chance. The VARS_TO_TRY= option specifies the number of candidates to select. The quality of the forest often depends on the number of candidates. Unfortunately, a good value for the VARS_TO_TRY= option is generally not known in advance. Data with more irrelevant variables generally warrant a larger value.

The reason for searching only one input variable for a splitting rule instead of searching all inputs and choosing the best split is to improve prediction on new data. An input that offers more splitting possibilities provides the search routine more chances to find a spurious split. Loh and Shih (1997) demonstrate the bias towards spurious splits that result. They also demonstrate that preselecting the input variable and then searching only on that one input reduces the bias. The HPFOREST procedure preselects the input with the largest $p$-value of an asymptotic permutation distribution of an association statistic. Hothorn, Hornik, and Zeileis (2006) originated the idea and describe the statistic.

The HPFOREST procedure sometimes reduces the number of categories of a nominal input. Nominal inputs with fewer categories in the node than the number specified in the CATBINS= option are not modified. For nominal inputs with more categories, PROC HPFOREST ignores observations with the least frequent category values. Limiting the number of categories in a nominal input can strengthen the association of that input with the target by eliminating categories that have less predictive potential. PROC HPFOREST reduces the categories independently in every node.

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

---

**Controlling for Variable Selection Bias**

Split-search algorithms generally inflate the worth of variables that offer many split possibilities beyond the predictive ability of the variable. Nominal variables are especially troublesome because they offer $2^{(k-1)}$ possible binary splits of the data, where $k$ is the number of categories. A nominal variable that has many categories and no predictive power can produce a split of greater apparent worth simply by chance than a predictive variable that offers fewer split choices.

The problem motivated Gordon Kass to invent CHAID (Kass 1980), an algorithm that penalizes variables that produce more split candidates. The problem is mentioned in Breiman et al. 1984, p. 42, as one of the weaknesses of their algorithm. The problem is worse in Ross Quinlan’s C5.0 algorithm (1993) because that algorithm creates many branches for a categorical input and only two for an interval input. Each branch provides an estimate of the target. Allowing some variables more estimates of a target than others gives those variables an unfair competitive advantage.

The HPFOREST procedure avoids the problem by not using the worth of a split to select the variable to split on. Instead, the inputs compete on a test of association with the target. The test adjusts for the different number of input categories. Only the winning variable is eligible for a splitting rule. Loh and Shih (1997) first proposed this in their QUEST algorithm. PROC HPFOREST uses the version in Hothorn, Hornik, and Zeileis (2006).

This section compares the different methods of selecting the variable to split the data. The data contain a purely random input $X_j$ with $j$ nominal categories and a second input $Z_k$ with $k$ categories that is predictive of the target. The plots show the proportions of samples in which a method selects $X_j$ instead of the predictive input.
input, $Z_k$. The larger the proportion, the greater the bias of the method towards variables that offer more splitting possibilities than predictive power.

The data contain 500 observations. The target $Y$ has values 0 and 1 with equal probabilities. Nominal input $Z_k$ has equally probable integer values from 0 through $k - 1$. The probability that $Y = 1$ given $Z_k$ is greater than or equal to $k/2$ is 0.6 (and therefore $P(Y = 1|Z_k < k/2) = 0.4$). With enough data, a splitting algorithm that uses $Z_k$ assigns values that are less than $k/2$ to one branch, and the rest to the other branch. $X_j$ is equally distributed without reference to $Y$. With enough data, the best split on $X_j$ has negligible worth. A variable selection method chooses between two variables. After repeating this for 1,000 samples, the proportion of times $X_j$ is selected is recorded and plotted for several combinations of $j$ and $k$.

Figure 5.5 shows the proportion of samples for which $X_j$ is selected instead of $Z_k$ when the split-search algorithm uses the reduction in Gini impurity as the splitting criterion. As shown in the figure, Gini reduction selects $X_j$ more often as the number of categories $j$ increases. When $j$ is similar to $k$, Gini reduction selects $X_j$ between 15% and 30% of the time. For $j - k = 10$, $X_j$ is selected about 75% of the time. Gini reduction selects the wrong variable more often than not for these values of $j$ and $k$.

**Figure 5.5 Proportion of Samples for Which Gini Reduction Selects $X_j$**
Figure 5.6 shows the proportion of samples for which $X_j$ is selected using the CHAID split-search algorithm. The figure shows that CHAID selects $X_j$ less often as the number of categories $j$ increases. CHAID penalizes variables with many categories, and the penalty is larger than necessary. For $j - k = 10$, the proportion of times CHAID selects $X_j$ decreases from about 70% ($j = 12$) to less than 10% ($j = 38$). CHAID selects the wrong variable more often than not for $j$ much less than $k$.

**Figure 5.6** Proportion of Samples for Which CHAID Selects $X_j$

Figure 5.7 shows the proportion of samples for which PROC HPFOREST selects $X_j$ by using a test of association. As shown in the figure, given $Z_k$, PROC HPFOREST selects $X_j$ the same proportion of times regardless of the number of categories $j$. When the number of categories $k$ of $Z_k$ increases, PROC HPFOREST selects $X_j$ more often. However, the proportion never reaches 40%. PROC HPFOREST never selects the wrong variable more often than not for any values of $j$ and $k$ examined.
Selecting a Splitting Variable

PROC HPFOREST considers only one input variable when it searches for a splitting rule in a node; it selects the variable with the highest association with the target. The measure of association is adapted from Hothorn, Hornik, and Zeileis (2006) and presented here.

Let $Y$ and $X$ denote the target variable and input variable, respectively. Let $Y_i$ and $X_i$ denote their values in observation $i$. The formulas for the association depend on whether $Y$ and $X$ are categorical. If $Y$ is categorical, let $J$ denote the number of values, and let $Y_{ij}$ equal 1 if $Y_i$ equals $j$ and 0 otherwise. Similarly, if $X$ is categorical, let $K$ denote the number of values, and let $X_{ik}$ equal 1 if $X_i$ equals $k$ and 0 otherwise. Let $T$ denote the statistic defined in Table 5.2.

The material in this section is part of a fuller discussion in de Ville and Neville of variable selection bias.
The test statistic is

\[ C_{\text{quad}} = (T - \hat{\mu})^T \hat{\Sigma}^{-1} (T - \hat{\mu}) \]

where

\[ \hat{\mu} = \text{estimate of the expected value of } T \]
\[ \hat{\Sigma} = \text{estimate of the covariance of } T \]
\[ \hat{\Sigma}^{-1} = \text{generalized inverse of } \hat{\Sigma} \]

PROC HPFOREST selects the variable with the smallest \( p \)-value, the integral from \( C_{\text{quad}} \) to infinity of the density of a chi-square distribution with degrees of freedom equal to the rank of \( \hat{\Sigma} \).

Table 5.3 contains formulas for \( \hat{\mu} \).

### Table 5.2 Definition of \( T \) for Different Types of Variables

<table>
<thead>
<tr>
<th>Type of Variable</th>
<th>Dimension of ( Y )</th>
<th>Dimension of ( X )</th>
<th>Dimension of ( T )</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
<td>Interval</td>
<td>1</td>
<td>( \sum Y_i X_i )</td>
<td></td>
</tr>
<tr>
<td>( J ) classes</td>
<td>Interval</td>
<td>( J )</td>
<td>( \sum Y_{ij} X_{i} )</td>
<td></td>
</tr>
<tr>
<td>Interval</td>
<td>( K ) categories</td>
<td>( K )</td>
<td>( \sum Y_{ij} X_{ik} )</td>
<td></td>
</tr>
<tr>
<td>( J ) classes</td>
<td>( K ) categories</td>
<td>( J \times K )</td>
<td>( \sum Y_{ij} X_{ik} )</td>
<td></td>
</tr>
</tbody>
</table>

To make the formulas more concise, let \( J = 1 \) and \( Y_{ij} = Y_i \) when \( Y \) has an interval measurement level. Now \( J \) and \( Y_{ij} \) are defined for all types of \( Y \). Similarly, let \( K = 1 \) and \( X_{ik} = X_i \) when \( X \) has an interval measurement level. The multiple definitions of \( T \) in Table 5.2 now reduce to the single formula,

\[ T_{jk} = \sum_{i=1}^{N} Y_{ij} X_{ik} \]

and the concise definition of \( \hat{\mu} \) becomes

\[ \hat{\mu}_{jk} = \frac{(\sum_{i=1}^{N} Y_{ij})(\sum_{i=1}^{N} X_{ik})}{N} \]

The dimension of \( \hat{\Sigma} \) equals \((J \times K)^2\), the dimension of \( T \) squared. If \( X \) has a large number of categories, then \( K \) is large, and storing and inverting \( \hat{\Sigma} \) can impede performance.
The formula for $\hat{\Sigma}$ is built as follows from the expectation and covariance of $Y$ and a factor that depends on $X$:

$$E(Y)_j = \sum_{i=1}^{N} Y_{ij} / N$$

$$\text{Cov}(Y)_{hj} = \sum_{i=1}^{N} (Y_{ih} - E(Y)_h)(Y_{ij} - E(Y)_j) / N$$

$$\Xi_{kl} = 1(k = l)(\sum_{i=1}^{N} X_{ik} X_{ik}) - (\sum_{i=1}^{N} X_{ik})(\sum_{i=1}^{N} X_{il})$$

$$\hat{\Sigma}_{hjkl} = \text{Cov}(Y)_{hj} \Xi_{kl} N / (N - 1)$$

where $1(k = l)$ equals 1 when $k = l$ and 0 otherwise.

---

**Searching for a Splitting Rule**

**Rules**

A PROC HPFOREST splitting rule uses the value of a single input variable to assign an observation to one of two branches. If the split-search algorithm uses missing values, then the rule includes an assignment of missing values to a branch.

A rule might assign all observations with a nonmissing value of the splitting variable to one branch, and all observations with a missing value to the other. In this case, the missing values go the second branch.

If the split-search algorithm does not use missing values, then the rule assigns an observation with a missing value to both branches and adds to each copy a fractional weight that is proportional to the number of training observations in each branch. This is the policy when the splitting variable contains no missing variables in the node during training, even if the MISSING=USEINSEARCH option is specified.

**Criteria**

The HPFOREST procedure searches for rules that maximize the measure of worth that is associated with the splitting criterion. Formally, the worth of a split $s$ is the reduction in node impurity,

$$\Delta i(s, \omega) = i(\omega) - \sum_{b=1}^{B} p(\omega_b | \omega) i(\omega_b)$$

where $i(\omega)$ is the impurity of the node $\omega$, and $p(\omega_b | \omega)$ is the proportion of training observations in branch $b$.

Generally, $p(\omega)$ is a nonnegative number that equals 0 if all observations in $\omega$ have the same target value, and $p(\omega)$ is large if the target values in $\omega$ are very different.

The impurity function for the Gini index is

$$i(\omega) = 1 - \sum_{j=1}^{J} p_j^2$$

where $p_j$ is the probability of target value $j$. The impurity function for variance reduction is

$$i(\omega) = \frac{1}{N(\omega)} \sum_{i=1}^{N(\omega)} (Y_i - \bar{Y})^2$$
where \( N(\omega) \) is the number of observations in node \( \omega \), \( Y_i \) is the target value of observation \( i \), and \( \bar{Y} \) is the average of \( Y_i \) in \( \omega \).

**Algorithm**

PROC HPFOREST searches for a splitting rule as follows:

1. It sorts the values in one of the following ways:
   - For an interval or ordinal input, it sorts by nonmissing values of the input.
   - For a nominal input with interval target, it sorts the categories by the average target value in the category.
   - For a nominal input with binary target, it sorts the categories by the proportion of one of the target values in the categories.

2. It walks from the lowest to the highest values, and it evaluates the split at every permissible position. A permissible position is one that does not separate two observations with the same target value and that satisfies any constraints, such as the LEAFSIZE= option.

If the algorithm allows missing input values, then the algorithm evaluates two splits at every permissible position of an interval or ordinal input: one that assigns the missing values to the left branch, and another that assigns missing values to the right branch.

For a nominal input, sorting reduces the number of candidate splits to \( m - 1 \) from \( 2^{(m-1)} \), where \( m \) is the number of nominal categories. Fisher (1958) proved it works and that the best split is among the \( m - 1 \) examined. Breiman et al. (1984) applied their theorem to decision trees.

No similar reduction is known for a nominal input with a nominal target (with more than two categories). However, PROC HPFOREST uses the following simple extension which usually works: For each nominal target category \( y \), it sorts the input categories by the proportion of \( y \) and finds the best permissible split among the \( m - 1 \) sorted positions. A total of \( km \) splits are considered, where \( k \) denotes the number of target values. PROC HPFOREST uses this approach unless the value of the EXHAUSTIVE= option is greater than or equal to \( 2^{(m-1)} \), in which case PROC HPFOREST examines all permissible splits.

Before PROC HPFOREST applies the algorithm, it might combine small nominal input categories to satisfy the CATBINS= and MINCATSIZE= options. The algorithm uses the combined category only if MISSING=USEINSEARCH and the number of missing observations in the node is greater than or equal to the value of the MINUSEINSEARCH= option.

**Predicting an Observation**

To predict an observation, the HPFOREST procedure first assigns the observation to a single leaf in each decision tree in the forest, then uses that leaf to make a prediction based on the tree that contains the leaf, and finally simply averages the predictions over the trees. For an interval target, the prediction in a leaf equals the average of the target values among the bagged training observations in that leaf. For a nominal target, the posterior probability of a target category equals the proportion of that category among the bagged training observations in that leaf. The predicted nominal target category is the category with the largest posterior probability. In case of a tie, the first category that occurs in the training data is the prediction.
The HPFOREST procedure also computes out-of-bag predictions. The out-of-bag prediction of an observation uses only trees for which the observation is out of bag (that is, not selected as part of the training data for that tree).

A model is worthless if its predictions are no better than predictions without a model. For an interval target, the no-model prediction of an observation is the average of the target among training observations. For a nominal target, the no-model posterior probabilities are the class proportions in the training data. The no-model predictions are the same for every observation.

## Computing the Average Square Error and Misclassification Rate

The HPFOREST procedure computes the average square error and the misclassification rate to assess the model’s goodness of fit. The average square error applies to all types of targets. The misclassification rate applies only to nominal targets.

The average square error for an interval, the average square error for a nominal target, and the misclassification rate for a nominal target are defined respectively as

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

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

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

where \(\hat{y}_i\) is the target prediction of observation \(i\), \(\delta_{ij}\) equals 1 or 0 if the nominal target value \(j\) does or does not occur in observation \(i\), respectively, \(\hat{p}_{ij}\) is the predicted probability of nominal target value \(j\) for observation \(i\), \(N\) is the number of observations, and \(J\) is the number of nominal target values (classes).

The definitions are valid whether \(\hat{y}_i\) is the usual model prediction, the out-of-bag prediction, or the no-model prediction. The three predictions result in three different estimates of \(\text{ASE}_{\text{int}}\). The model has some predictive ability if the out-of-bag estimate of fit is smaller than the no-model estimate. The \(\text{ASE}_{\text{int}}\) that is based on the usual model predictions of the original training data is usually optimistic, smaller than what its value will be on future data.

## Adjusting Statistics When Sampling Target Classes Unevenly

When the proportions of target classes in the training data differ from the proportions in the population to which the model is applied, predictions and fit statistics need to be adjusted for the population of interest. Consider fraud detection as an example. A random sample of 100,000 transactions might have too few fraudulent observations for training a good model. One solution is to preferentially include fraudulent cases in the sample. Adding extra observations of a rare target class into the training data is called oversampling. Another solution is to randomly sample enough transactions (several million perhaps) to obtain enough fraudulent cases. In this situation, the number of non-fraudulent cases might be more than are necessary for
training and might be a burden for data processing. Removing observations of a common target class from the training data is called *undersampling*.

Typically, the greater the class proportion in the training data, the greater the class posterior probability from a model. When the class proportions in the training data differ from those in the population of interest, the model inflates the predictions of the classes that are overrepresented in the training data. The following sections explain how to adjust the predictions and the fit statistics that estimate how well the model will perform when applied, illustrate why the adjustments matter, and present a technical derivation of the adjustment formulas.

### Formulas for Adjusting the Predictions and Fit Statistics

The formulas for adjusting the probabilities and statistics assume that the distribution of the inputs for a target class is the same in the training data as in the population of interest. The class proportions can differ, but the distribution of input values within a class must be the same.

The formula for converting probabilities from the training sample to the population of interest is

\[
\hat{p}_{\pi j} = \frac{p_{\tau j} \gamma_j}{\sum_k p_{\tau k} \gamma_k}
\]

where

- \(\hat{p}_{\pi j}\) = estimate of \(p_{\pi j}\) that is computed with the training data
- \(p_{\pi j}\) = adjusted probability of class \(j\) for the population of interest
- \(p_{\tau j}\) = unadjusted probability of class \(j\) for the training data
- \(\gamma_j = \pi_j / \tau_j\)
- \(\pi_j = \) proportion of class \(j\) in the population of interest
- \(\tau_j = \) proportion of class \(j\) in the training data

The circumflex above a population statistic such as \(p_{\pi j}\) indicates that the statistic is estimated from training data. A population statistic without a circumflex indicates that the statistic is computed from population data.

Let \(S\) denote a statistic that can be expressed as the average of a loss metric, \(Loss(y, p(y))\), between the actual target value and the predicted probabilities. Average square error and misclassification rate are examples.

The probability \(p(y)\) might be adjusted or unadjusted, and the average might be taken over the training data or the population of interest. An optional argument to \(S\) indicates whether the probabilities are adjusted, and an optional suffix indicates the sample for evaluating the statistic. Thus,

\[
S_{\tau}(p_{\tau}) = \sum_{i=1}^{N_{\tau}} \frac{Loss(y_i, p_{\tau i})}{N_{\tau}}
\]

\[
S_{\tau}(p_{\pi}) = \sum_{i=1}^{N_{\tau}} \frac{Loss(y_i, p_{\pi i})}{N_{\pi}}
\]

\[
S_{\pi}(p_{\tau}) = \sum_{i=1}^{N_{\pi}} \frac{Loss(y_i, p_{\tau i})}{N_{\pi}}
\]

\[
S_{\pi}(p_{\pi}) = \sum_{i=1}^{N_{\pi}} \frac{Loss(y_i, p_{\pi i})}{N_{\pi}}
\]
To compute the estimate of a population statistic $S_\pi(p_\pi)$, multiply the terms for target class $j$ by the proportion of class $j$ observations in the population as

$$\hat{S}_\pi = \sum_j S_{\tau j}(\hat{p}_\pi)\pi_j$$

where

$$S_{\tau j}(\hat{p}_\pi) = \sum_{i : y_i = j} \frac{\text{Loss}(y_i, p_{\pi ij})}{N_{\tau j}}$$

$N_{\tau j}$ = number of class $j$ observations in the training data

The class $j$ formulas for the average square error and misclassification rate are

$$\text{ASE}_{\tau j} = \sum_{i : y_i = j} \sum_{k = 1}^J \frac{(\delta_{jk} - \hat{p}_{\pi ik})^2}{JN_{\tau j}}$$

$$\text{MISC}_{\tau j} = \sum_{i : y_i = j} \frac{1(\hat{y}_i \neq \hat{y}_i)}{N_{\tau j}}$$

where

$$\delta_{jk} = 1 \text{ if } j = k \text{ and } 0 \text{ otherwise}$$

$\hat{y}_i$ = the class $k$ with the largest value of $\hat{p}_{\pi ik}$

**Why the Adjustments Matter**

This section illustrates the need to adjust probabilities and statistics. The data consist of a binary target and two independent input variables from a normal distribution with standard deviation of 0.25. The mean value of an input is 0.25 for target class 0, and is 0.75 for class 1.

The test data set has 20,000 class 0 observations and 80,000 class 1 observations. All runs use the same test data. The training and validation data sets have 5,000 observations. The percentage of class 0 observations varies from 5 to 95. For each percentage, 20 training and validation data sets are generated. A decision tree is fit to each of the 20 training data sets and applied to the test data.

Figure 5.8 shows the misclassification rate of each tree, which is evaluated on the test data. A smooth curve passes near the average misclassification rate at each percentage of class 0 observations. (Technically, the curve is a cubic spline with $\lambda$, equal to 0.05.) The rate is computed twice: once with the probabilities adjusted for the class proportions in the test data, and once without adjusting. A separate curve is drawn for each. The two rates are the same when the proportion of class 0 observations in the training data is 20 percent, which is the same as in the test data. The misclassification rates increase (get worse) as the class proportions in the training data differ more from the proportions in the test data. If you could choose the proportions of the target classes in the training data, this example suggests that the best choice would be to choose the same proportions as in the population of interest. However, if one target value is rare, oversampling would still be necessary to ensure that the training data have enough rare observations for the algorithm to work with.
In this example, adjusting probabilities is better because the rates that are computed after adjusting probabilities are generally smaller (better) than those computed without adjusting. The difference is larger when the class proportions in the training data differ more from those in the test data.

**Figure 5.8** Test Set Misclassification Rates Using Adjusted and Unadjusted Predictions

![Figure 5.8](image)

**Figure 5.9** shows the average square error (ASE) for each tree, which is evaluated on the test data. A smooth curve passes near the average ASE for each percentage of class 0 observations. When the proportion of class 0 observations in the training data is 0.5 or less, adjusting the probabilities makes no difference to the ASE. For larger proportions of class 0, adjusting the probabilities results in a slightly larger (worse) ASE in this example.
Figure 5.10 shows the misclassification rate that is computed using validation data and adjusted probabilities. The rate is estimated in two ways: one estimate adjusts only the probabilities, and the other adjusts both the statistics and the probabilities. The two are defined respectively as

\[
MISC_{\text{unadjusted}} = \sum_{j} \sum_{i \, \forall y_i = j} \frac{1(j \neq \hat{y}_i)}{N_{\tau j} \pi_j} \\frac{N_{\tau j}}{N_{\tau}}
\]

\[
MISC_{\text{adjusted}} = \sum_{j} \sum_{i \, \forall y_i = j} \frac{1(j \neq \hat{y}_i)}{N_{\tau j} \pi_j}
\]

Only the last factor in each term is different. In Figure 5.10, the curve for MISC_{\text{adjusted}} is indistinguishable from the curve for the test set misclassification rate. MISC_{\text{adjusted}} predicts the test set rate perfectly in this example. On the other hand, the curve for MISC_{\text{unadjusted}} is completely different and unreliable. Adjusting the misclassification rate is necessary in this example to get a good estimate of the rate in the test set.
Figure 5.10 Estimates of Test Set Misclassification Using Adjusted and Unadjusted Statistics

Figure 5.11 is the corresponding plot for ASE. The solid gray curve passes near the average ASE that is computed on the test data. The dashed blue curve that approximately follows the gray curve is the adjusted statistic that is computed on the validation data. The dashed green hill-shaped curve is the unadjusted statistic that is computed on the validation data. The evidence for adjusting the statistics for ASE is less compelling in this example than for the misclassification rate. For class 0 proportions between 45 and 65%, the unadjusted estimates of ASE are closer to the test set ASE than the adjusted estimates. For other class 0 proportions, the adjusted estimates are closer.
Technical Derivations of Adjustment Formulas

The formulas for adjusting the probabilities and statistics rely on statistical theory that assumes that the training data and the population of interest are both samples drawn from two infinite, ideal populations. For each target value, the distributions of the inputs are assumed to be the same in the two ideal populations: $P(X|Y = j) = P(X_\pi|Y_\pi = j)$, where $X_\tau$ and $Y_\tau$ designate the inputs and target random variables in the population of the training sample and $X_\pi$ and $Y_\pi$ designate the inputs and target variables in the population of interest.

The formulas also assume that the model predictions converge to the population probabilities $P(y = j|x)$ as the training sample becomes large. The formula for adjusting the probabilities follows from Bayes’ theorem, where $\alpha$ is a constant that is independent of the target class:
The sum of the class probabilities equals 1:

\[ \sum_j P(Y_\pi = j | X_\pi) = 1 \]

Therefore,

\[ \alpha = \frac{1}{\sum_j P(Y_\tau = j | X_\tau) P(Y_\pi = j | X_\pi) / P(Y_\tau = j)} \]

The formula works well when the model predictions are close to the population probabilities, \( P(y = j | x) \). Similarly, the formula for adjusting a statistic \( S \) works well when \( S \) is close to its expected value. \( S_\pi (p_\pi) \) equals the average of a loss function over a sample from the population of interest, and \( S_\tau (p_\pi) \) equals the average of a loss function over the training population. The central limit theorem asserts that \( S_\pi (p_\pi) \) converges to its expected value, \( E_\pi (\text{Loss}(Y, p_\pi)) \), as the sample size increases. Similarly, \( S_\tau (p_\pi) \rightarrow E_\tau (\text{Loss}(Y, p_\pi)) \). The same is true when the average is restricted to observations with the same target value, \( S_{\tau j} (p_\pi) \rightarrow E_{\tau j} (\text{Loss}(Y, p_\pi)) \). The formula that uses \( S_\tau \) to estimate \( S_\pi \) assumes the averages are close to their expected values.

\[ E_\pi (\text{Loss}(Y_\pi, p_\pi)) = \int \int \text{Loss}(Y_\pi, p_\pi) p(X_\pi, Y_\pi) dX_\pi dY_\pi \\
= \sum_j \int \text{Loss}(Y_\pi, p_\pi) p(X_\pi | Y_\pi = j) dX_\pi P(Y_\pi = j) \\
= \sum_j \int \text{Loss}(Y_\tau, p_\pi) p(X_\tau | Y_\tau = j) dX_\tau P(Y_\tau = j) \\
= \sum_j E_{\tau j} (\text{Loss}(Y_\tau, p_\pi)) P(Y_\pi = j) \]

King and Zeng (2001, Appendix B) discuss the adjustments of probabilities in more detail. Discussions of the adjustments of the statistics are hard to find.

**Handling Missing Values**

**Strategies**

Tree-based models use observations with missing input values. The HPFOREST procedure offers two strategies for handling missing values. The simplest strategy is to regard a missing value as a special nonmissing value. For a nominal input, a missing value simply constitutes a new categorical value. For an input whose values are ordered, each missing value constitutes a special value that is assigned a place in the ordering that yields the best split. The place is generally different in different nodes of the tree.

This strategy is beneficial when missing values are predictive of certain target values. For example, people with large incomes might be more reluctant to disclose their income than people with ordinary incomes. If income were predictive of a target, then missing income would be predictive of the target and the missing
values would be regarded as a special large income value. The strategy seems harmless when the distribution of missing values is uncorrelated with the target because no choice of branch for the missing values would help predict the target.

A linear regression could use the same strategy by adding binary indicator variables to designate whether a value is missing. Alternatively, and much more commonly, a linear regression could simply remove observations in which any input is missing. Let \( p \) denote the probability that a variable value is missing, and let \( v \) denote the number of input variables. The probability that an observation has one or more missing values is \( \left(1 - p\right)^v \) (assuming missingness is independent and identically distributed among the inputs). If \( p = 0.1 \) and \( v = 10 \), then 65% of the observations would have missing values and be removed from linear regression.

The alternative strategy for decision trees is to exclude from the search algorithm observations that have a missing value in the single input variable that defines the splitting rule. If \( p = 0.1 \) and \( v = 10 \), then only 10% instead of 65% of the observations are excluded. Although this compares favorably with common linear regression, using observations with missing values might still be better. PROC HPFOREST is designed to run faster by using observations with missing values than by not using them.

When missing values are excluded from the split search, a new policy is needed for assigning an observation with missing values to a branch. Assigning all the observations with missing values to a single branch is likely to reduce the purity of the branch, thereby degrading the split. Instead, the HPFOREST procedure assigns the observation to both branches by replacing the single observation with two copies and assigning each copy a fractional frequency that is proportional to the size of the branches. The prediction for an observation is the weighted average of predictions of the derived fractional observations. The prediction that uses the two child nodes is the same as the prediction that would be obtained from not splitting the node. (This can be proved as follows: The prediction based on the two child nodes is the weighted average of the predictions of the two child nodes. The prediction in a node is the average of some variable. The weighted average of the averages in the child nodes is the average in the parent, which is the prediction based on the parent.)

**Specifics**

If the value of a target variable is missing, the observation is excluded from training and from evaluating the model. If the value of an input variable is missing, PROC HPFOREST uses the missing value as a legitimate value by default or if `MISSING=USEINSEARCH` and the number of observations in which the splitting variable has missing values is at least as large as the value of the `MINUSEINSEACH=` option.

Specifying `MISSING=DISTRIBUTE` forces every splitting rule to distribute an observation to the branches when the value of the splitting variable is missing. Specifying `MISSING=USEINSEARCH` also produces rules that distribute observations if the splitting variable has no missing values in the training data or when the value specified in the `MINUSEINSEACH=` option prevents using missing values in the search.

Observations that are distributed into multiple branches might slow down training noticeably. Values of distributed observations in a leaf are stored in a linked list and passed to the association and split-search routines individually. Values of observations that are not distributed (that is, observations that reside entirely within one leaf) are passed together in a single vector. Processing a single vector of values is much faster than plodding through a linked list and calling an accumulation routine separately for each value.

The discussion in this section applies to each candidate variable and each node separately. For example, the test of association that uses input variable \( X \) might use all observations, and the test that uses input variable \( Z \) might ignore some observations because of missing values. The test that uses \( X \) might use all observations in one node but not all observations in another node.
Handling Values That Are Absent from Training Data

A splitting rule that uses a categorical variable might not recognize all possible values of the variable. Some categories might not exist in the training data. Others might be so infrequent in the training sample in the node that the procedure excludes them. The MINCATSIZE= option specifies the minimum number of occurrences required for a categorical value to participate in the search for a splitting rule. Splitting rules handle unseen categorical values the same way they handle missing values.

Measuring Variable Importance

The importance of a variable is the contribution it makes to the success of the model. For a predictive model, success means good prediction. Often the prediction relies mainly on a few variables. A good measure of importance reveals those variables. The better the prediction, the closer the model represents reality, and the more plausible it is that the important variables represent the true cause of prediction. Some people prefer a simple model so they can understand it. However, a simple model usually relinquishes details of reality. Sometimes it is better to first find a good model and ask what variables are important than to first ask what model is good for variable importance and train that model.

M. J. van der Laan (2006) asks whether a predictive model is appropriate at all. He believes that if variable importance is your goal, then you should predict importance directly instead of fitting a model. If your goal is to select suspicious genes for further study in a lab or to find variables in an industrial process that might influence the quality of the product, then his argument is persuasive. However, the purpose of many predictive models is to make predictions. In these cases, gaining insight to causes can be useful.

Variable importance is also useful for selecting variables for a subsequent model. The comparative importance between the selected variables does not matter. Researchers often seek speed and simplicity from the first model and seek accuracy from the subsequent model. Despite this tendency, a forest is often more useful than a simpler regression as a first model when interactions are wanted because variables contribute to the forest model through interactions.

Several authors have demonstrated that using a forest to select variables, then using only those variables in a subsequent forest, and then repeating the process produces a final forest with better prediction than the original.

Leo Breiman’s seminal publication (2001) gives one measure of importance, which is called Breiman’s method here. Breiman and Cutler (2003) introduce another method, which they call Gini increase but which is called loss reduction here for reasons discussed in the following section. Several modifications to Breiman’s method have been proposed. Strobl’s method (2008) assigns less importance to correlated variables than Breiman’s method, which in turn assigns less than loss reduction. Strobl’s method is also the most complex and takes the longest time to compute. Breiman’s method is again in the middle. Running time is the main reason Breiman introduced loss reduction.

Loss Reduction

Loss reduction is also called Gini increase, Gini importance, or impurity reduction. It was introduced in Breiman et al. (1984) for decision trees, later modified for gradient boosting machines (Friedman 2001), and later used in forests (Breiman and Cutler 2003).
The importance of variable $v$ is proportional to the sum of the reduction in node impurity, summed over nodes that $v$ splits. Breiman et al. (1984) and Breiman and Cutler (2003) introduce the impurity measure with the Gini splitting criterion, hence the name Gini importance. However, Gini impurity is defined only for a categorical target. For an interval target, the most common node impurity measure is the sum of square errors. Friedman (2001) uses a square root at the end of the calculation to revert back to the scale of the target. This can fail when you use validation data because the impurity reduction can be negative. Therefore, the HPFOREST procedure computes both the reduction in absolute error and the reduction in square error.

PROC HPFOREST uses the word loss instead of impurity to associate the measure of importance with the reduction in loss from using the model. A loss function is a statistic that measures how well a model fits data. Average square error is a common loss function. Given a loss function, the next equation defines an associated measure of variable importance. The sum over variables of the associated variable importance equals the total loss when a model is not used minus the loss when a model is used. In other words, the loss reduction variable importance assigns shares to the variables of the total reduction in the loss that the model achieves.

The loss reduction variable importance for input $v$ in tree $T$ is computed as

$$I_{\text{loss}}(v; T) \propto \sum_{\omega \in T} 1(v \text{ splits } \omega) \Delta \text{Loss}(\omega)$$

where the sum is over internal nodes $\omega$ in $T$ and where $1(v \text{ splits } \omega)$ is 1 if $v$ is the splitting variable in $\omega$ and 0 otherwise. $\Delta \text{Loss}(\omega)$ is the reduction in loss from splitting $\omega$. A loss function maps a response value and a prediction to a number that represents how bad the prediction is. Square error loss is most common,

$$\Delta \text{Loss}(\omega) = \text{SSE}(\omega) - \sum_{b \in B(\omega)} \text{SSE}(\omega_b)$$

$$\text{SSE}(\omega) = \begin{cases} 
\frac{N(\omega)}{\sum_{i=1}^{N(\omega)}} (Y_i - \hat{Y}(\omega))^2 & \text{for interval target } Y \\
\frac{N(\omega)}{\sum_{i=1}^{N(\omega)}} \sum_{j=1}^{J} (\delta_{ij} - \hat{p}_j(\omega))^2 & \text{for target with } J \text{ categories}
\end{cases}$$

where

- $B(\omega)$ = set of branches from $\omega$
- $\omega_b$ = child node of $\omega$ in branch $b$
- $N(\omega)$ = number of observations in $\omega$
- $\hat{Y}(\omega)$ = average $Y$ in training data in $\omega$
- $\delta_{ij}$ = 1 if $Y_i = j$, 0 otherwise
- $\hat{p}_j(\omega)$ = average $\delta_{ij}$ in training data in $\omega$

For an interval target, PROC HPFOREST also computes absolute error loss,

$$\Delta \text{Loss}(\omega) = \text{SAE}(\omega) - \sum_{b \in B(\omega)} \text{SAE}(\omega_b)$$

where

$$\text{SAE}(\omega) = \sum_{i=1}^{N(\omega)} |Y_i - \hat{Y}(\omega)|$$
For a categorical target, the formula for \( \text{SSE}(\omega) \) reduces to

\[
\text{SSE}(\omega) = \begin{cases} 
N(1 - \sum_{j=1}^{J} \hat{p}_{j}^2) & \text{for training data} \\
N(1 - \sum_{j=1}^{J} (2p_{j} - \hat{p}_{j}) \hat{p}_{j}) & \text{for validation data}
\end{cases}
\]

where \( p_{j} \) is the proportion of the validation data with target value \( j \), and \( N, p_{j}, \) and \( \hat{p}_{j} \) are evaluated in node \( \omega \). \( \text{SSE}(\omega) \) for training data equals the Gini impurity index. Loss reduction variable importance is commonly called Gini importance for this reason.

Another measure of importance for a categorical target is based on the margin, the probability of the true class minus the maximum probability of the other classes. A good model increases the margin. Therefore, loss reduction variable importance uses the negative of margin.

\[
\Delta \text{Loss}(\omega) = \text{SNM}(\omega) - \sum_{b \in B(\omega)} \text{SNM}(\omega_b)
\]

where

\[
\text{SNM}(\omega) = -\sum_{j=1}^{J} N_j (\hat{p}_{j} - \max_{k \neq j} \hat{p}_{k})
\]

and \( N_j \) is the number of class \( j \) observations in \( \omega \) in the data set being used to evaluate the variable importance.

When the target is binary, variable importance based on the margin equals twice that of the variable importance based on the Gini index.

**Breiman’s Method**

Breiman’s method is also called a permutation-based or randomization method. Breiman’s method calculates importance as

\[
I_{\text{Breiman}}(v; T) \propto \sum_{i=1}^{n} \text{Loss}(y_i, \hat{y}_i(\pi(v))) - \sum_{i=1}^{n} \text{Loss}(y_i, \hat{y}_i)
\]

where \( \hat{y}_i \) is the prediction for observation \( i \) and \( \hat{y}_i(\pi(v)) \) is the prediction for observation \( i \) after randomizing the values of input \( v \). In Breiman’s writings, the sum uses only out-of-bag observations, and randomizing is done by permuting the out-of-bag values of \( v \). Originally Breiman (2001) uses misclassification as the loss function. Breiman and Cutler (2003) retracted that, saying misclassification loss is too volatile with many variables. Instead they recommend the margin for a nominal target: the probability of the true class minus the maximum probability among the other classes. Breiman (2001) bases the loss on the entire forest, not a single tree. Today authors generally compute the importance for each tree and then average these (Berk 2008; Grömping 2009).
Bias and Correlation

The loss reduction method uses only the variables that PROC HPFOREST selects for splitting rules. Selecting the wrong variable among candidates that compete for a rule reduces the importance of the correct variable in the final tally. Variable selection bias produces bias in variable importance as a consequence. PROC HPFOREST uses a variable selection method that is widely agreed to be free from bias for practical purposes (Strobl et al. 2008).

Even so, only one of possibly several deserving variables is selected in a node for splitting. Only one gets credit for loss reduction in a node. Ideally, a forest handles this by letting the variables compete many times in many trees with slightly different data and in nodes in which some variables are randomly excluded. The process should result in a fair distribution of the use of the variables, without correlated inputs masking each other, and a fair representation in the final variable importance.

What variable importance measures actually do with correlated variables is subtle. Suppose $X$ and $Z$ are correlated, and $Y = D_X$. Should $Z$ be assigned any importance? One answer is no: $Z$ is not even in the formula that generates $Y$. The other answer is yes: observing $Z$ provides information about $Y$, and therefore $Z$ is helpful in explaining the variation of $Y$. An input $W$ has conditional importance if it is needed for prediction even after values of the other variables are given. $Z$ has no conditional importance. An input $W$ has marginal importance if it is predictive of $Y$ by itself. $Z$ has marginal importance.

The degree to which correlations determine the final importance values depends both on the algorithm for importance and on the algorithmic parameters for the model. In the current example, if VARS_TO_TRY=2, then $X$ and $Z$ compete in every node, PROC HPFOREST selects $X$ to split almost all the nodes, and all importance measures assign $Z$ negligible importance. If VARS_TO_TRY=1, then PROC HPFOREST must use $Z$ to split some nodes, and it assigns some importance to $Z$. In general, loss reduction assigns more importance to correlated variables than Breiman’s method, and Breiman’s method assigns more importance than a conditional permutation method discussed in Strobl et al. (2008).

To illustrate, you can generate 500 bivariate correlated normal observations, run PROC HPFOREST with VARS_TO_TRY=1, and plot the proportion of total importance that is assigned to $Z$. Figure 5.12 shows the result from generating 25 samples at each of several correlation values that range from 0 to 1. The figure has a point for each of three measures of importance that are evaluated on each sample. Unless the correlation equals 1, Breiman’s method and Strobl’s method assign little importance to $Z$. (Strobl’s method extends Breiman’s method by first segregating the observations by quantiles of $X$, then permuting $Z$ within the quantile limits, and then scoring the result to compute the importance of $Z$.) All methods pick $X$ as the most important variable unless they are perfectly correlated.
Archer and Kimes (2008) present a simulation that compares the variable importance methods. To emulate genomic data, correlated variables appear in groups, and at most one variable in a correlated group is in the regression equation that generates the target. Both forest methods are about as good as using a method of variable importance from regression. This is consistent with Figure 5.12, which deems $X$ the most important variable by every method and every correlation value except a correlation of 1.

Preferences

Loss reduction applies to tree-based models. Breiman’s method can apply to any predictive model by using hold-out data instead of out-of-bag data. However, it is generally not used in practice. One reason is the long running time needed to score every observation for every variable that is evaluated. A more important reason is the lack of a convincing example where Breiman’s method succeeds and others fail. The comparison in the previous section applies only to forests because only in forests can the VARS_TO_TRY= option be set. No comparison of importance measures is published with the VARS_TO_TRY= option equal to all the variables.

Some authors insist on using Breiman’s method. Berk (2008) says that importance must be measured outside of the procedure that is used to measure it. Otherwise, it is not a practical measure. It simply restates a part of the model itself without reference to the practical reason for creating the model. Nicodemus and Malley...
(2009) present plots that clearly show loss reduction as hopelessly biased, while Breiman’s method gives the correct results on the same data. Actually, they are not the same data. All these authors use training data with loss reduction and hold-out data with Breiman’s method. Berk even says loss reduction is a fit measure, and as such should be used with the training data. This is misguided. Using hold-out data to evaluate a predictive model is generally recommended. Computing loss reduction with both training and validation data can reveal which inputs are fooling the training algorithm, and corrective action can be taken.

---

**Displaying the Output**

The HPFOREST procedure displays the parameters that are used to train the model, fit statistics of the trained model, and other information. The output is organized into various tables, which are discussed here in 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 for the main tasks of the procedure are displayed.

**Model Information**

The “Model Information” table contains the settings of the training parameters. The table is produced by default.

**Number of Observations**

The “Number of Observations” table contains the number of observations that are read from the input data set and the number of observations that are used in the analysis.

**Baseline Fit Statistics**

The “Baseline Fit Statistics” table contains fit statistics that are calculated without a model. Compare the baseline statistics with the model fit statistics to determine how beneficial the model is. Fit statistics are described in the section “Computing the Average Square Error and Misclassification Rate” on page 87. The table is produced by default.

**Fit Statistics**

The “Fit Statistics” table contains statistics that measure the model’s goodness of fit. The fit of the model to the data improves with the number of trees in the forest. Successive rows in the table contain fit statistics for a forest that has more trees. The SKIP_SEQ_ROWS= option controls how many more trees successive rows represent. Compare these fit statistics with the baseline fit statistics to determine how beneficial the model is. Fit statistics are described in the section “Computing the Average Square Error and Misclassification Rate” on page 87. The table is produced by default.
Loss Reduction Variable Importance

The “Variable Importance” table displays variable importance based on loss reduction, which is explained in the section “Loss Reduction” on page 96. The table is produced by default. If the table is not wanted, then specifying IMPORTANCE=NO turns off the calculation and conserves some memory resources.

ODS Table Names

Table 5.4 lists the names of the data tables created by the HPFOREST procedure. Use these names in ODS statements.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Required Statement and Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>PerformanceInfo</td>
<td>Performance information</td>
<td>Default output</td>
</tr>
<tr>
<td>NObs</td>
<td>Number of observations</td>
<td>Default output</td>
</tr>
<tr>
<td>Baseline</td>
<td>Fit statistics without a model</td>
<td>Default output</td>
</tr>
<tr>
<td>FitStatistics</td>
<td>Fit statistics from the model</td>
<td>Default output</td>
</tr>
<tr>
<td>VariableImportance</td>
<td>Loss reduction variable importance</td>
<td>Default output</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>Default output</td>
</tr>
<tr>
<td>Timing</td>
<td>Absolute and relative times for tasks</td>
<td>PERFORMANCE DETAILS</td>
</tr>
<tr>
<td></td>
<td>performed by the procedure</td>
<td></td>
</tr>
</tbody>
</table>

Examples: HPFOREST Procedure

The following examples illustrate the basic concepts of forests. The first three examples use the spambase data available from the UCI Machine Learning Repository (Asuncion and Newman 2007) http://archive.ics.uci.edu/ml/datasets/Spambase.

Example 5.1: Out-Of-Bag Estimate of Misclassification Rate

Using the original training data to evaluate a forest model is poor practice because the forest predicts the training data much better than it predicts similar data withheld from training. Using the out-of-bag data is better practice because, with enough trees, the fit of a forest to the out-of-bag data converges to what the fit would be on similar data withheld from training. With only a few trees, the fit to the out-of-bag data is worse than what the fit would be on withheld data. Consequently, the training and out-of-bag data provide lower and upper bounds to what the error rate will be when the forest is applied to new data.
Example 5.1: Out-Of-Bag Estimate of Misclassification Rate

This example illustrates the difference between the misclassification rates estimated from the training and out-of-bag data. The HPFOREST procedure is run on the spambase data. The target, SPAM, has two values: 0 indicates a legitimate e-mail, 1 indicates spam. The number of trees is set large enough for the out-of-bag misclassification error rates to converge (MAXTREES=200 or 500).

The following SAS statements create a SAS data set from the data downloaded into a file called `c:\spambase_data.txt`:

```sas
data spambase;
  infile 'c:\spambase_data.txt' delimiter = ',';
  input wf_make wf_adress wf_all wf_3d wf_our
       wf_over wf_remove wf_internet wf_order wf_mail
       wf_receive wf_will wf_people wf_report wf_addresses
       wf_free wf_business wf_email wf_you wf_credit
       wf_your wf_font wf_000 wf_money wf_hp
       wf_hpl wf_george wf_650 wf_lab wf_labs
       wf_telnet wf_857 wf_data wf_415 wf_85
       wf_technology wf_1999 wf_parts wf_pm wf_direct
       wf_cs wf_meeting wf_original wf_project wf_re
       wf_edu wf_table wf_conference
       cf_semicolon cf_parenthese cf_bracket cf_exclamation
       cf_dollar cf_pound
  average longest total spam;
run;

proc hpforest data=spambase maxtrees=200;
  input w: c: average longest total/level=interval;
  target spam/level=binary;
  ods output FitStatistics=fitstats(rename=(Ntrees=Trees));
run;

data fitstats;
  set fitstats;
  label Trees = 'Number of Trees';
  label MiscAll = 'Full Data';
  label Miscoob = 'OOB';
run;
proc sgplot data=fitstats;
  title "OOB vs Training";
  series x=Trees y=MiscAll;
  series x=Trees y=Miscoob/lineattrs=(pattern=shortdash thickness=2);
  yaxis label='Misclassification Rate';
run;
title;
```
Output 5.1.1 Plot of OOB versus Training Misclassification Rate

Figure 5.1.1 shows the misclassification rate is worse (larger) based on the out-of-bag (OOB) data, and more trees are needed for the out-of-bag rates to level off. Both characteristics are typical of a forest.
Example 5.2: Number of Variables to Try When Splitting a Node

This example illustrates the effect of changing the number of variables to randomly select as candidate splitting variables in a node. In each node in each tree, \( m \) variables are randomly selected to be candidates to split on. Use the `VARS_TO_TRY=` option to specify \( m \). Specifying \( m \) less than the number of available inputs is one way to reduce the correlation between the trees in the forest. Broadly speaking, the predictions of a forest improve when the trees are less correlated. On the other hand, the predictions of the forest improve when the predictions of the trees improve (without changing the correlations). When the number of useful inputs are much less than the total number of inputs, smaller values of \( m \) produce weaker trees because fewer nodes consider useful inputs for defining a splitting rule. Try several values of \( m \) to find a good one for the data.

The following SAS statements create a SAS data set from the data downloaded into a file called `c:\spambase_data.txt`:

```sas
data spambase;
  infile 'c:\spambase_data.txt' delimiter = ',';
  input wf_make  wf_adress  wf_all  wf_3d  wf_order  wf_mail
            wf_remove  wf_internet  wf_our
            wf_will  wf_people  wf_report  wf_addresses
            wf_free  wf_business  wf_email  wf_you  wf_credit
            wf_our
  wf_hpl  wf_george  wf_650  wf_lab  wf_labs
  wf_telnet  wf_857  wf_data  wf_415  wf_85
  wf_technology  wf_1999  wf_parts  wf_pm  wf_direct
  wf_cs  wf_meeting  wf_original  wf_project  wf_re
  wf_edu  wf_table  wf_conference
  cf_semicolon  cf_parenthese  cf_bracket  cf_exclamation
  cf_dollar  cf_pound
  average  longest  total
  spam;
run;
```
%macro hpforest(Vars=);
proc hpforest data=spambase maxtrees=200
   vars_to_try=&Vars.;
   input w: c: average longest total/level=interval;
   target spam/level=binary;
   ods output
      FitStatistics = fitstats_vars&Vars.(rename=(Miscoob=VarsToTry&Vars.));
run;
%mend;

%hpforest(vars=all);
%hpforest(vars=40);
%hpforest(vars=26);
%hpforest(vars=7);
%hpforest(vars=2);

data fitstats;
   merge
      fitstats_varsall
      fitstats_vars40
      fitstats_vars26
      fitstats_vars7
      fitstats_vars2;
   rename Ntrees=Trees;
   label VarsToTryAll = "Vars=All";
   label VarsToTry40 = "Vars=40";
   label VarsToTry26 = "Vars=26";
   label VarsToTry7 = "Vars=7";
   label VarsToTry2 = "Vars=2";
run;

proc sgplot data=fitstats;
   title "Misclassification Rate for Various VarsToTry Values";
   series x=Trees y = VarsToTryAll/lineattrs=(Color=black);
   series x=Trees y=VarsToTry40/lineattrs=(Pattern=ShortDash Thickness=2);
   series x=Trees y=VarsToTry26/lineattrs=(Pattern=ShortDash Thickness=2);
   series x=Trees y=VarsToTry7/lineattrs=(Pattern=MediumDashDotDot Thickness=2);
   series x=Trees y=VarsToTry2/lineattrs=(Pattern=LongDash Thickness=2);
   yaxis label='OOB Misclassification Rate';
run;
title;
Example 5.2: Number of Variables to Try When Splitting a Node

Output 5.2.1 Effect of the VARS_TOTRY= Option on the Misclassification Rate

Specifying a value of 40 or 26 for the VARS_TO_TRY= option results in a slightly more accurate forest than would occur without random selection of variables (VARS_TO_TRY=ALL). Specifying VARS_TO_TRY=2 is much worse than specifying VARS_TO_TRY=ALL. A good value for VARS_TO_TRY= depends on the data. In this example, the HPFOREST procedure uses a default value of $\sqrt{58} \approx 7$. 
Example 5.3: Fraction of Training Data to Train a Tree

This example illustrates the effect of changing the fraction of original training observations used to train an individual tree. Use the TRAINFRACTION= option to specify $f$. Specifying $f$ less than 1 is one way to reduce the correlation between the trees in the forest.

The following SAS statements create a SAS data set from the data downloaded into a file called c:\spambase_data.txt:

```sas
data spambase;
  infile 'c:\spambase_data.txt' delimiter = ',';
  input wf_make  wf_adress  wf_all  wf_3d  wf_our
  wf_over  wf_remove  wf_internet  wf_order  wf_mail
  wf_receive  wf_will  wf_people  wf_report  wf_addresses
  wf_free  wf_business  wf_email  wf_you  wf_credit
  wf_your  wf_font  wf_000  wf_money  wf_hp
  wf_hpl  wf_george  wf_650  wf_lab  wf_labs
  wf_telnet  wf_857  wf_data  wf_415  wf_85
  wf_technology  wf_1999  wf_parts  wf_pm  wf_direct
  wf_cs  wf_meeting  wf_original  wf_project  wf_re
  wf_edu  wf_table  wf_conference
  cf_semicolon  cf_parenthese  cf_bracket  cf_exclamation
  cf_dollar  cf_pound
  average  longest  total
  spam;
run;

%macro hpforest(f=, output_suffix=);
  proc hpforest data=spambase maxtrees=500 vars_to_try=26
     trainfraction=&f;
     input w: c: average longest total/level=interval;
     target spam/level=binary;
     ods output FitStatistics = fitstats_f&output_suffix.(rename=(Miscoob=fraction&output_suffix.));
   run;
%mend;

%hpforest(f=0.8, output_suffix=08);
%hpforest(f=0.6, output_suffix=06);
%hpforest(f=0.4, output_suffix=04);

data fitstats;
  merge
    fitstats_f08
    fitstats_f06
    fitstats_f04;
  rename Ntrees=Trees;
  label fraction08 = "Fraction=0.8";
  label fraction06 = "Fraction=0.6";
  label fraction04 = "Fraction=0.4";
run;
```
Using the default number of trees (MAXTREES=50) in this example, the default value of TRAINFRACTION=0.6 results in the best OOB misclassification rate. Using more trees, TRAINFRACTION=0.8 is best.
Example 5.4: Loss Reduction Variable Importance

This example compares the loss reduction variable importance measure on uncorrelated and correlated variables. The data have eight inputs that are generated from a standard normal distribution. The first four inputs are independent; the last four have a correlation of 0.9. The target $Y$ is computed as

$$Y = X_1 + X_2 + 2X_3 + X_5 + X_6 + 2X_7$$

The following SAS statements create a SAS data set and run PROC HPFOREST:

```sas
data output;
    call streaminit(54321);
    do i=1 to 10000;
        x1 = rand('normal', 0, 1);
        x2 = rand('normal', 0, 1);
        x3 = rand('normal', 0, 1);
        output;
    end;
run;

data cov;
    input x5-x8;
    datalines;
    1 0.9 0.9 0.9
    0.9 1 0.9 0.9
    0.9 0.9 1 0.9
    0.9 0.9 0.9 1
run;

proc simnormal data=cov(type=cov) out = osim(drop=Rnum) numreal = 10000 seed = 54321; var x5-x8; run;

data output;
    merge output osim;
    y = x1 + x2 + 2*x3 + x5 + x6 + 2*x7;
run;

proc hpforest data=output vars_to_try=all;
    input x:/level=interval;
    target y/level=interval;
    ods select VariableImportance;
run;
```

Output 5.4.1 shows the PROC HPFOREST variable importance table. The NRules column contains the number of splitting rules that use each variable. The next four columns are loss reduction measures of variable importance. The mean square error and the absolute error are computed with the training data. The OOB columns contain the same measures computed with out-of-bag data. In this example, the relative importance of any pair of variables is similar in every measure.
Example 5.5: Missing Values and Imputed Values

This example uses the Home Equity data from the SAS sample library to illustrate the difference between using missing values and using imputed values. A nonrandom pattern of missingness in the data can help predict the target. PROC HPFOREST cannot use this pattern when missing values are replaced by imputed values in the training data. The following statements illustrate this by running PROC HPFOREST twice: once on the original data, and once on the data after missing nominal values have been replaced by the mode of the variable and missing interval values have been replaced by the mean of the variable.

The Sampsio.Hmeq data set contains fictitious mortgage data in which each case represents an applicant for a home equity loan. All applicants have an existing mortgage. The binary target BAD equals 1 for an applicant who eventually defaulted or was ever seriously delinquent. Nine interval inputs are available for modeling. JOB and REASON are the only nominal inputs. The modes for JOB and REASON are OTHER and DEBTCON, respectively.
Chapter 5: The HPFOREST Procedure

```plaintext
proc hpimpute data=sampsio.hmeq out=imout;
  input mortdue value yoj clage ninq clno debtinc derog delinq;
  impute mortdue value yoj clage ninq clno debtinc derog delinq/method=mean;
run;

data job_reason;
  set sampsio.hmeq;
  if job='' then job="Other";
  if reason='' then reason="DebtCon";
run;

data imout;
  merge imout job_reason;
run;

proc hpforest data=imout vars_to_try=all;
  input im:/level=interval;
  input reason job/level=nominal;
  target bad/level=binary;
  ods output
    VariableImportance=imvi
    FitStatistics=imfit(rename=(Ntrees=Trees Miscall=ImMiscall Miscoob=ImMiscoob));
run;

proc hpforest data=sampsio.hmeq vars_to_try=all;
  input mortdue value yoj clage ninq clno debtinc derog delinq/level=interval;
  input reason job/level=nominal;
  target bad/level=binary;
  ods output
    Baseline=bs
    VariableImportance=vi
    FitStatistics=fit(rename=(Ntrees=Trees));
run;

proc sql noprint;
  select value into :MiscBaseline trimmed from bs where Statistic='Misclassification Rate';
quit;

data fitstats;
  merge imfit fit;
  MiscBaseline = &MiscBaseline;
  label Trees = 'Number of Trees';
  label MiscAll = 'Full Data';
  label Miscoob = 'OOB';
  label ImMiscAll = 'Full Data - Impute';
  label ImMiscoob = 'OOB - Impute';
  label Miscbaseline = 'Baseline';
run;

proc sgplot data=fitstats;
  title "Misclassification Rate With and Without Imputed Values";
  series x=trees y=Miscbaseline/lineattrs=(Pattern=Solid Color=black);
  series x=Trees y=MiscAll/lineattrs=(Pattern=Solid Thickness=2);
```
Example 5.5: Missing Values and Imputed Values

```plaintext
series x=Trees y=Miscoob/lineattrs=(Pattern=ShortDash Thickness=2);
series x=Trees y=ImMiscAll/lineattrs=(Pattern=ShortDash Thickness=2);
series x=Trees y=ImMiscoob/lineattrs=(Pattern=MediumDashDotDot Thickness=2);
yaxis label='Misclassification Rate';
run;

data vi;
  set vi;
  keep Variable NRules Gini GiniOOB Rank;
  Rank = _n_;
run;

proc sort data=vi;
  by Variable;
run;

data imvi;
  set imvi;
  keep Variable RankImputed NRules Gini GiniOOB;
  if substr(Variable,1,3)='IM_' then Variable=substr(Variable, 4);
  RankImputed=_n_;
  label RankImputed="Rank (Imput)";
  rename NRules=RulesImputed;
  label NRules="Rules (Imputed)";
  rename Gini=GiniImputed;
  label Gini="Gini (Imputed)";
  rename GiniOOB=GiniOOBImputed;
  label GiniOOB="OOB Gini Reduction (Impute)";
run;

proc sort data=imvi;
  by Variable;
run;

data vi;
  set vi;
  merge vi imvi;
  by Variable;
  rename NRules=Rules;
run;

proc sort data=vi;
  by rank;
run;

data t1(keep=Variable Rules RulesImputed RankImputed)
  t2(keep=Variable Gini GiniImputed GiniOOB GiniOOBImputed);
  set vi;
run;

proc print data=t1; run;
proc print data=t2; run;
```

data debtinc_miss;
set sampsio.hmeq;
if debtinc =. then debtinc_is_missing='MISSING';
else debtinc_is_missing='NOT MISSING';
run;

proc freq data=debtinc_miss;
    tables debtinc_is_missing*bad/nocol;
run;

Figure 5.5.1 shows the misclassification rate and the out-of-bag misclassification rate with and without imputed missing values. Without any model, the misclassification rate equals 0.1995. The out-of-bag rate from the model trained with imputed values is not much better. The out-of-bag rate with the original data is much better, close to half the baseline rate.

**Output 5.5.1** The Effect of Imputing Missing Values

Output 5.5.2 shows the number of times each model uses each variable. The Rules column shows the number of times by using the original data; the RankImputed column shows the number of times by using the imputed data. The numbers in the Rules column vary much more than those in the RuleImputed column, which suggests that variables with missing values have more distinctive information than variables with imputed values.
The order of the variables is the order of importance from the model that uses the original data. The RankImpute column shows the order of importance from the model that uses imputed values. DEBTINC is the most important variable when the original data are used, and among the last in importance when imputed values are used. Imputing changes the characteristics of these data dramatically.

**Output 5.5.2 Variable Importance Ranking**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Variable</th>
<th>Rules</th>
<th>Imputed</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEBTINC</td>
<td>862</td>
<td>526</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>DELINQ</td>
<td>748</td>
<td>414</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>DEROG</td>
<td>440</td>
<td>427</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>CLAGE</td>
<td>1130</td>
<td>534</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>VALUE</td>
<td>4140</td>
<td>519</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>NINQ</td>
<td>1034</td>
<td>518</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>CLNO</td>
<td>573</td>
<td>608</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>REASON</td>
<td>223</td>
<td>310</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>JOB</td>
<td>498</td>
<td>534</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>MORTDUE</td>
<td>716</td>
<td>398</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>YOJ</td>
<td>980</td>
<td>610</td>
<td>10</td>
</tr>
</tbody>
</table>

**Output 5.5.3 Variable Importance Ranking**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Variable</th>
<th>Gini</th>
<th>Gini OOB</th>
<th>Gini Imputed</th>
<th>Gini OOB Imputed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEBTINC</td>
<td>0.114505</td>
<td>0.07181</td>
<td>0.005911</td>
<td>-0.00159</td>
</tr>
<tr>
<td>2</td>
<td>DELINQ</td>
<td>0.022592</td>
<td>0.01198</td>
<td>0.011046</td>
<td>0.00406</td>
</tr>
<tr>
<td>3</td>
<td>DEROG</td>
<td>0.010230</td>
<td>0.00471</td>
<td>0.004509</td>
<td>-0.00020</td>
</tr>
<tr>
<td>4</td>
<td>CLAGE</td>
<td>0.015952</td>
<td>0.00167</td>
<td>0.007823</td>
<td>-0.00237</td>
</tr>
<tr>
<td>5</td>
<td>VALUE</td>
<td>0.017810</td>
<td>0.00134</td>
<td>0.008441</td>
<td>-0.00101</td>
</tr>
<tr>
<td>6</td>
<td>NINQ</td>
<td>0.005895</td>
<td>-0.00061</td>
<td>0.004520</td>
<td>-0.00199</td>
</tr>
<tr>
<td>7</td>
<td>CLNO</td>
<td>0.005783</td>
<td>-0.00067</td>
<td>0.008768</td>
<td>-0.00120</td>
</tr>
<tr>
<td>8</td>
<td>REASON</td>
<td>0.001477</td>
<td>-0.00068</td>
<td>0.002987</td>
<td>-0.00010</td>
</tr>
<tr>
<td>9</td>
<td>JOB</td>
<td>0.003181</td>
<td>-0.00086</td>
<td>0.006935</td>
<td>0.00081</td>
</tr>
<tr>
<td>10</td>
<td>MORTDUE</td>
<td>0.003799</td>
<td>-0.00175</td>
<td>0.005901</td>
<td>-0.00151</td>
</tr>
<tr>
<td>11</td>
<td>YOJ</td>
<td>0.007206</td>
<td>-0.00197</td>
<td>0.006911</td>
<td>-0.00225</td>
</tr>
</tbody>
</table>
Output 5.5.4 shows the count of observations for which DEBTINC is or is not missing for each value of the target BAD. DEBTINC is missing in 21% of the observations. If DEBTINC is missing, then the proportion of observations with BAD equal to 1 (which indicates an applicant who becomes delinquent) is 62%. If DEBTINC is not missing, then the proportion is only 8.6%. Missing values in DEBTINC are highly predictive of BAD in this example. Imputing the missing values destroys the predictive power.

**Output 5.5.4** Contingency Table of BAD by DEBTINC_IS_MISSING

<table>
<thead>
<tr>
<th></th>
<th>Frequency Table of debtinc_is_missing by BAD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Percent Row Pct BAD</td>
</tr>
<tr>
<td>debtinc_is_missing</td>
<td>0 1 Total</td>
</tr>
<tr>
<td>MISSING</td>
<td>481 786 1267</td>
</tr>
<tr>
<td>8.07</td>
<td>13.19 21.26</td>
</tr>
<tr>
<td>37.96</td>
<td>62.04</td>
</tr>
<tr>
<td>NOT MISSING</td>
<td>4290 403 4693</td>
</tr>
<tr>
<td>71.98</td>
<td>6.76 78.74</td>
</tr>
<tr>
<td>91.41</td>
<td>8.59</td>
</tr>
<tr>
<td>Total</td>
<td>4771 1189 5960</td>
</tr>
<tr>
<td>80.05</td>
<td>19.95 100.00</td>
</tr>
</tbody>
</table>

**References**


Overview: HPNEURAL Procedure

The HPNEURAL procedure is a high-performance procedure that trains a multilayer perceptron neural network. For more information about multilayer perceptron neural networks, see Bishop (1995). PROC HPNEURAL can also use a previously trained network to score a data set (referred to as stand-alone scoring), or it can generate SAS DATA step statements that can be used to score a data set.
PROC HPNEURAL runs in either single-machine mode or distributed mode.

**NOTE:** Distributed mode requires SAS High-Performance Data Mining.

Because the HPNEURAL procedure is a high-performance analytical procedure, it also does the following:

- enables you to run in distributed mode on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode on the server where SAS is installed
- exploits all the available cores and concurrent threads, regardless of execution mode

For more information, see the section “Processing Modes” on page 6 in Chapter 2, “Shared Concepts and Topics.”

Training a multilayer perceptron neural network requires the unconstrained minimization of a nonlinear objective function. Because there are currently no practical methods to guarantee finding a global minimum of that objective function, one way to be reasonably sure of finding a good solution is to train the network multiple times using different sets of initial values for the weights. Thus, even problems with smaller numbers of variables and training observations can benefit from the use of distributed mode.

---

**PROC HPNEURAL Features**

The HPNEURAL procedure was designed with two goals in mind: to perform efficient, high-speed training of neural networks, and to be as easy to use as possible while still creating models that fit the training data well and generalize well. With these goals in mind, most parameters for the neural network are automatically selected.

The following list summarizes some basic features of PROC HPNEURAL:

- ability to train and score using distributed mode
- parallel read of input data and parallel write of output data when the data source is the appliance database
- high degree of multithreading during all phases of training and scoring
- automatic standardization of input and target variables
- intelligent defaults for most neural network parameters such as activation and error functions
- either automatic or manual selection and use of a validation data subset
- automatic termination of training when the validation error stops improving
- ability to weight individual observations or automatically use inverse prior probabilities as weights
Table 6.1 compares the HPNEURAL procedure with the SAS Enterprise Miner Neural Network Node.

<table>
<thead>
<tr>
<th>PROC HPNEURAL</th>
<th>Neural Network Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multithreaded</td>
<td>Single-threaded</td>
</tr>
<tr>
<td>Can execute in single-machine mode or</td>
<td>Can execute only in single-machine mode</td>
</tr>
<tr>
<td>distributed mode</td>
<td></td>
</tr>
<tr>
<td>Has few required user-specified parameters so that</td>
<td>Gives you fine control over the myriad of possible</td>
</tr>
<tr>
<td>users with minimal experience with neural</td>
<td>choices of parameters that control the training of a</td>
</tr>
<tr>
<td>networks can obtain good solutions to problems</td>
<td>neural network</td>
</tr>
<tr>
<td>that are amenable to supervised training</td>
<td></td>
</tr>
<tr>
<td>Automatically selects the data standardization</td>
<td>Can use many different user-specified data</td>
</tr>
<tr>
<td>methods, network architecture, activation</td>
<td>standardization methods, network architectures,</td>
</tr>
<tr>
<td>functions, error functions, and weight</td>
<td>activation functions, error functions, and ways of selecting initial</td>
</tr>
<tr>
<td>initialization method</td>
<td>weights</td>
</tr>
<tr>
<td>Uses the limited memory</td>
<td>Can use a variety of conjugate gradient or</td>
</tr>
<tr>
<td>Broyden–Fletcher–Goldfarb–Shanno (LBFGS)</td>
<td>quasi-Newton optimization methods, but not</td>
</tr>
<tr>
<td>optimization method (Nocedal and Liu 1989) with</td>
<td>LBFGS</td>
</tr>
<tr>
<td>proprietary enhancements. LBFGS was chosen</td>
<td></td>
</tr>
<tr>
<td>for the HPNEURAL procedure because of both its</td>
<td></td>
</tr>
<tr>
<td>speed of training and its limited use of memory,</td>
<td></td>
</tr>
<tr>
<td>which can be especially important for problems</td>
<td></td>
</tr>
<tr>
<td>with large amounts of training data.</td>
<td></td>
</tr>
</tbody>
</table>

**Getting Started: HPNEURAL Procedure**

**Training**

The HPNEURAL procedure can either train a neural network model or use a previously trained model to score a data set. We will first discuss training.

The HPNEURAL procedure does not have many parameters that you must specify. It simply needs to know where the training data are (the DATA= option in the PROC statement), the names and types of the input variables (the INPUT statement), the names and types of the target variables (the TARGET statement), the number of hidden neurons (the HIDDEN statement), the number of training tries with each using different
randomly generated initial weights (the \texttt{TRAIN} statement), and, optionally, where to write the score file that
contains targets from the input file and predicted targets from the trained network and where to write the
model file that contains the parameters of the trained network (the \texttt{SCORE} statement).

The single most important parameter you can specify is the number of hidden neurons in the network. A
good strategy is to start with a small number and slowly increase the number until the validation error stops
improving.

The next most important parameter you can specify is the number of times the network is to be retrained
using different sets of initial weights (the \texttt{NUMTRIES} option in the \texttt{TRAIN} statement). A good strategy is to
start with 5 (the default) and increase by 2 until the validation error stops improving.

The following small example trains a neural network to predict the type of iris plant, given several measure-
ments, and then scores the same data set that was used for training. The DATA step contains 150 observations
derived from the R. A. Fisher (1936) Iris data set:

\begin{verbatim}
  title 'Fisher (1936) Iris Data';
  proc format;
    value specname
      1='Setosa'
      2='Versicolor'
      3='Virginica';
  run;
  data iris;
    input SepalLength SepalWidth PetalLength PetalWidth Species @@;
    format Species specname.;
  datalines;
50 33 14 02 1 64 28 56 22 3 65 28 46 15 2 67 31 56 24 3 63 28 51 15 3 46 34 14 03 1 69 31 51 23 3 62 22 45 15 2 59 32 48 18 2 46 36 10 02 1 61 30 46 14 2 60 27 51 16 2 65 30 52 20 3 56 25 39 11 2 65 30 55 18 3 58 27 51 19 3 68 32 59 23 3 51 33 17 05 1 57 28 45 13 2 62 34 54 23 3 77 38 67 22 3 63 33 47 16 2 67 33 57 25 3 76 30 66 21 3 49 25 45 17 3 55 35 13 02 1 67 30 52 23 3 70 32 47 14 2 64 32 45 15 2 61 28 40 13 2 48 31 16 02 1 59 30 51 18 3 55 24 38 11 2 63 25 50 19 3 64 32 53 23 3 52 34 14 02 1 49 36 14 01 1 54 30 45 15 2 79 38 64 20 3 44 32 13 02 1 67 33 57 21 3 50 35 16 06 1 58 26 40 12 2 44 30 13 02 1 77 28 67 20 3 63 27 49 18 3 47 32 16 02 1 55 26 44 12 2 50 23 33 10 2 72 32 60 18 3 48 30 14 03 1 51 38 16 02 1 61 30 49 18 3 48 34 19 02 1 50 30 16 02 1 50 32 12 02 1 61 26 56 14 3 64 28 56 21 3 43 30 11 01 1 58 40 12 02 1 51 38 19 04 1 67 31 44 14 2 62 28 48 18 3 49 30 14 02 1 51 35 14 02 1 67 31 44 14 2 62 28 48 18 3 49 30 14 02 1 51 38 19 04 1 67 31 44 14 2 62 28 48 18 3 49 30 14 02 1 46 32 14 02 1 60 29 45 15 2 57 26 35 10 2 57 44 15 04 1 50 36 14 02 1 77 30 61 23 3 63 34 56 24 3 58 27 51 19 3 57 29 42 13 2 72 30 58 16 3 54 34 15 04 1 52 41 15 01 1 71 30 59 21 3 64 31 55 18 3 60 30 48 18 3 63 29 56 18 3 49 24 33 10 2 56 27 42 13 2 57 30 42 12 2 55 42 14 02 1 49 31 15 02 1 77 26 69 23 3 60 22 50 15 3 54 39 17 04 1
\end{verbatim}
proc hpneural data=iris;
  input SepalLength SepalWidth PetalLength PetalWidth;
  target Species / level=nom;
  hidden 2;
  train outmodel=model_iris;
  score out=scores_iris;
run;

Figure 6.1 displays the SAS log output, which shows the percentage of validation observations that were misclassified by the trained network. If there had been any interval targets, the log would have shown the absolute average percentage error and the absolute maximum percentage error for each interval target.

NOTE: The HPNEURAL procedure is executing on the client.
NOTE: Reading data...
NOTE: 150 usable observations in input data set.
NOTE: Training...
NOTE: Try 1 complete after 26 iterations. Reason for stopping: Training error=0.000000
NOTE: Try 2 complete after 27 iterations. Reason for stopping: Training error=0.000000
NOTE: Try 3 complete after 28 iterations. Reason for stopping: Training error=0.000000
NOTE: Try 4 complete after 26 iterations. Reason for stopping: Training error=0.000000
NOTE: Try 5 complete after 28 iterations. Reason for stopping: Training error=0.000000
NOTE: Scoring...
NOTE: Misclassification Error for target Species: 5.2632%
NOTE: There were 150 observations read from the data set WORK.IRIS.
NOTE: The data set WORK.SCORES_IRIS has 150 observations and 6 variables.
Figure 6.2 displays the “Model Information,” “Performance Information,” and “Number of Observations” tables. The HPNEURAL procedure creates a neural network model for the nominal variable Species. Of the 150 observations, 38 are used as a validation subset, which consists of the first observation and every fourth observation thereafter. The other 112 observations make up the training subset.

PROC HPNEURAL executes in single-machine mode. That is, the model is trained on the machine where the SAS session executes.

**Figure 6.2 Model Information, Performance Information, and Number of Observations Tables**

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source</td>
</tr>
<tr>
<td>Architecture</td>
</tr>
<tr>
<td>Optimization Technique</td>
</tr>
<tr>
<td>Number of Input Variables</td>
</tr>
<tr>
<td>Number of Target Variables</td>
</tr>
<tr>
<td>Number of Hidden Neurons</td>
</tr>
<tr>
<td>Number of Weights</td>
</tr>
<tr>
<td>Number of Observations Read</td>
</tr>
<tr>
<td>Number of Observations Used</td>
</tr>
<tr>
<td>Number Used for Training</td>
</tr>
<tr>
<td>Number Used for Validation</td>
</tr>
</tbody>
</table>

Figure 6.3 displays the “Misclassification Table.” It shows the results of scoring the validation subset by using the neural network model that is trained on the training subset. This example shows two incorrect classifications: two observations whose target value was “Virginica” were incorrectly classified as “Versicolor.”
Stand-Alone Scoring

The primary purpose for training a neural network model is to use the trained model to score new data that was not seen during training. It is very important that the new data have the same statistical characteristics as the data that were used for training. The following statements use the model that was trained in the preceding section to score some new observations:

```
title 'New Iris Data';

data new_iris;
   input SepalLength SepalWidth PetalLength PetalWidth;
   datalines;
   50 33 14 02
   64 28 56 22
   65 28 46 15
   67 31 56 24
   63 28 51 15
   46 34 14 03
   69 31 51 23
   62 22 45 15
   59 32 48 18
   46 36 10 02
   61 30 46 14
   60 27 51 16
   65 30 52 20
   56 25 39 11
   65 30 55 18
   58 27 51 19
;

proc hpneural data=new_iris;
   score model=model_iris out=scores_new_iris;
run;
```
### Syntax: HPNEURAL Procedure

The following statements are available in the HPNEURAL procedure:

```
PROC HPNEURAL < DATA=SAS-data-set> < DISTR=ALL | SPLIT > < NOPRINT > ;
PERFORMANCE performance-options ;
ARCHITECTURE architecture-option ;
ID variables ;
INPUT variables < / LEVEL=INT | LEVEL=NOM < MISSING=MAP > > ;
WEIGHT variable | _INVERSE_PRIORS_ ;
HIDDEN number ;
TARGET variables < / LEVEL=INT | LEVEL=NOM > ;
PARTITION ROLEVAR=variable( TRAIN=value | VALIDATE=value ) ;
PARTITION FRACTION( TRAIN=number | VALIDATE=number ) ;
TRAIN < NUMTRIES=number > < MAXITER=number >
< VALID=_NONE_ > < OUTMODEL=SAS-data-set > ;
SCORE OUT=SAS-data-set < MODEL=SAS-data-set > ;
CODE FILE='external-file' | fileref ;
```

When you train a neural network, the PROC HPNEURAL, INPUT, TARGET, and TRAIN statements are required. The HIDDEN statement is required unless you use the logistic architecture (in which case, the HIDDEN statement is not allowed).

When you use a previously trained neural network to score a data set, only the PROC HPNEURAL, SCORE, ID, PERFORMANCE, and CODE statements are allowed.

### PROC HPNEURAL Statement

```
PROC HPNEURAL < DATA=SAS-data-set> < DISTR=ALL | SPLIT > < NOPRINT > ;
```

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

- **DATA=SAS-data-set**
  
  names the SAS data set that contains the training and validation observations to be used by PROC HPNEURAL to train the neural network or that contains the observations to be scored when you are performing stand-alone scoring. The default input data set is the most recently created data set.

  When you use PROC HPNEURAL to train a neural network, each observation must contain the input, weight, validation, ID, and target variables that are specified in the associated INPUT, WEIGHT, PARTITION, ID, and TARGET statements.

  When you use PROC HPNEURAL to perform stand-alone scoring, the input data set must contain the input variables that were specified when the network was trained (as saved in the model data set) and optionally the target variables that were specified when the network was trained. In addition, if you applied formats to variables when training, the same formats must be applied when you do stand-alone scoring. Only the target variables (if they exist) and the network’s predictions are written to the output data set, which is specified in the SCORE statement.
For nominal variables of character type, levels are truncated to 32 bytes and converted to upper case. Also, when you train a network, if all observations that have a specific level for a nominal variable contain missing values in other input, target, weight, or validation variables, then that specific level is discarded and does not appear in the analysis.

If PROC HPNEURAL executes in distributed mode, the input data are distributed to memory on the appliance nodes and analyzed in parallel. For information about the alongside-the-database model, see the section “Alongside-the-Database Execution” on page 13.

When PROC HPNEURAL runs in single-machine mode, the input data set must fit into the available memory on the single machine. When PROC HPNEURAL runs in distributed mode, the input data set must fit into the total memory available across the distributed environment.

**DISTR=ALL | SPLIT**

specifies whether the input data set is to be replicated in the memory of each node in distributed mode. If this option is not specified, PROC HPNEURAL makes this decision automatically based on the size of the input data set. This option is ignored if PROC HPNEURAL is not running in distributed mode.

When PROC HPNEURAL runs in distributed mode, PROC HPNEURAL usually divides the input data set among all the nodes to minimize the time it takes to optimize each try. However, if the input data set is small, dividing the data in this way might be inefficient because of the interconnect delay (the time it takes to send partial results between nodes). It might be more efficient to have each node have a complete copy of the data and run each try in parallel on separate nodes. Each try might take longer because it uses only a single node, but it could take less time to finish all the tries because the tries are running in parallel.

You can force the data to be redistributed so that each node has a complete in-memory copy by specifying DISTR=ALL. You can prevent the data from being redistributed by specifying DISTR=SPLIT.

**NOPRINT**

specifies that no ODS tables be created.

---

**ARCHITECTURE Statement**

```
ARCHITECTURE architecture-option ;
```

The ARCHITECTURE statement specifies the architecture of the neural network to be trained. The `architecture-option` must be one of the following:

**LOGISTIC**

specifies a multilayer perceptron with no hidden units (which is equivalent to a logistic regression). If you specify this architecture, the HIDDEN statement is not allowed.

**LAYER1**

specifies a multilayer perceptron with a single hidden layer.

**LAYER1SKIP**

specifies a multilayer perceptron with a single hidden layer and additional connections between each input and each target neuron. Logistic regression is used to initialize the network for the first try.
LAYER2 specifies a multilayer perceptron with two hidden layers. The number of hidden neurons is split equally between the first and second layer. If the number of hidden neurons is odd, the first hidden layer has the extra neuron.

LAYER2SKIP specifies a multilayer perceptron with two hidden layers and additional connections between each input and each target neuron. The number of hidden neurons is split equally between the first and second layer. If the number of hidden neurons is odd, the first hidden layer has the extra neuron. Logistic regression is used to initialize the network for the first try.

When you use PROC HPNEURAL to train a neural network, the ARCHITECTURE statement is optional. The default is LAYER1. The ARCHITECTURE statement is not allowed when you use PROC HPNEURAL to perform stand-alone scoring.

---

**CODE Statement**

```sas
CODE FILE= 'external-file' | fileref ;
```

The CODE statement uses the current neural network model to generate SAS DATA step statements and save them in an external text file that can later be used to score a data set. The file does not contain the surrounding PROC and RUN statements. The DATA step statements can be used with the standard DATA step, PROC DS2, or PROC HPDS2.

The CODE statement is optional.

**FILE= 'external-file'**

specifies an external text file where the generated statements are saved.

**FILE= fileref**

specifies a fileref that refers to an external text file where the generated statements are saved.

---

**HIDDEN Statement**

```sas
HIDDEN number ;
```

The HIDDEN statement specifies the number of hidden neurons in the network. The number must be an integer greater than or equal to 1 (2 for two-layer architectures). For two-layer architectures (LAYER2 and LAYER2SKIP in the ARCHITECTURE statement), the hidden neurons are split between the first and second layer. In this case, if the number of hidden neurons is odd, the first hidden layer has the extra neuron.

All hidden neurons use a hyperbolic-tangent activation function.

When training, you must include exactly one HIDDEN statement, unless you specify ARCHITECTURE LOGISTIC (in which case the HIDDEN statement is not allowed).

The HIDDEN statement is not allowed when you do stand-alone scoring.
ID Statement

**ID** variables ;

The ID statement lists one or more variables from the input data set that are transferred to the output data set, which is specified in the SCORE statement.


The ID statement is optional.

INPUT Statement

**INPUT** variables </LEVEL=INT | LEVEL=NOM <MISSING=MAP> ;

The INPUT statement identifies the variables in the input data set that are inputs to the neural network.

**LEVEL=INT**

specifies that the variables are interval variables, which must be numeric. The default for the LEVEL option is INT.

**LEVEL=NOM**

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

**MISSING=MAP**

specifies that the missing value for nominal variables should be treated as a valid level (mapped to level 0). This option is not allowed for interval variables.

When training, you must include one or more INPUT statements. You need more than one INPUT statement when you have both interval and nominal input variables. The INPUT statement is not allowed when you do stand-alone scoring.

All interval input variables are automatically standardized to the range $[-1, 1]$.

When you are training, any observation that has missing values for any variable is not used.

When you are performing stand-alone scoring, if an interval variable is missing, its mean (as observed during the training phase) is used. If a nominal variable is missing, all input neurons associated with the variable (one per class level, except for binary variables, which have a single neuron) are set to 0.

PARTITION Statement

**PARTITION ROLEVAR=variable** ( TRAIN=value | VALIDATE=value ) ;

**PARTITION FRACTION** ( TRAIN=number | VALIDATE=number ) ;

The PARTITION statement specifies how to divide the input data set into a training subset and a validation subset.
The statement implements two alternate methods of specifying the split between the training and validation data. Either you can explicitly specify training observations and validation observations by specifying **ROLEVAR=variable**, where **variable** is a variable in the input data set, or you can specify that an approximate fraction of the input data set be used for training observations or validation observations by specifying **FRACTION( TRAIN=number )** or **FRACTION( VALIDATE=number )**.

**ROLEVAR=variable( TRAIN=value | VALIDATE=value )** specifies that the **variable** in the input data set be used to decide whether an observation is used for training or for validation. You can either specify the value used to identify training observations or the value used to identify validation observations. If you specify **TRAIN=value**, then an observation is used for training if the value of **variable** equals **value**; otherwise the observation is used for validation. If you specify **VALIDATE=value**, then an observation is used for validation if the value of **variable** equals **value**; otherwise the observation is used for training.

**FRACTION( TRAIN=number | VALIDATE=number )** specifies the approximate fraction of the input data set to be used for training or validation. If you specify **TRAIN=number**, then approximately the fraction of the data set specified by **number** is used as training observations, and the rest are used for validation observations. If you specify **VALIDATE=number**, then approximately the fraction of the data set specified by **number** is used as validation observations. The split between training and validation observations can only approximate the requested fraction because that fraction is used as a cutoff value for a random number generator to determine the actual split. If you require a more accurate split or a split that is guaranteed to be identical across different distributed computing environments, you must use the **ROLEVAR** option to specify the split explicitly.

When you are training, the **PARTITION** statement is optional. If you do not include the **PARTITION** statement, every fourth observation (starting with the first observation) is used as a validation observation, unless you specify **VALID=_NONE_** in the **TRAIN** statement. In this case, no validation is performed. The **PARTITION** statement is not allowed when you are doing stand-alone scoring.

Fit statistics reported after training are only computed using validation observations. Fit statistics reported after stand-alone scoring are computed using all observations.

**PERFORMANCE Statement**

```
PERFORMANCE < performance-options > ;
```

The **PERFORMANCE** statement defines performance parameters for multithreaded and distributed computing. The **PERFORMANCE** statement is documented further in the section “Processing Modes” on page 6 in Chapter 2, “Shared Concepts and Topics.”

The **PERFORMANCE** statement is optional.
**SCORE Statement**

**SCORE OUT=SAS-data-set < MODEL=SAS-data-set> ;**  

The SCORE statement causes the HPNEURAL procedure to write the network’s target and predicted output for each observation in the input data set to the output data set that is specified by the OUT option, along with any variables from the input data set that are specified in the ID statement.

**OUT=SAS-data-set**  
specifies the data set to contain the predicted values of the target variables. For nominal variables, each observation also contains the computed probabilities of each class level. This keyword is required.

**MODEL=SAS-data-set**  
specifies the data set that contains the model parameters for a previously trained network. You can specify this keyword only when you are doing stand-alone scoring.

When you are training, the SCORE statement is optional but the MODEL= keyword is not allowed. When you are doing stand-alone scoring, the SCORE statement is required and the MODEL= keyword must be used.

---

**TARGET Statement**

**TARGET variables < / LEVEL=INT | LEVEL=NOM > ;**  

The TARGET statement identifies the variables in the input data set that the network is to be trained to predict. The default for the LEVEL= option is INT.

**LEVEL=INT**  
specifies that the variables are interval variables, which must be numeric.

**LEVEL=NOM**  
specifies that the variables are nominal variables, also known as classification variables, which can be numeric or character.

When training, you must include one or more TARGET statements. You need more than one TARGET statement when you have both interval and nominal target variables. The TARGET statement is not allowed when you do stand-alone scoring.

For interval variables, all target neurons use the identity activation function.

Nominal variables have one target neuron per class level. Each of these neurons uses the softmax activation function to ensure that the sum of the outputs for all neurons is 1.0. The output of each neuron can then be interpreted as the probability that the variable is the corresponding class level.

When you are training, any observation that has missing values for any variable is not used.

You cannot specify the same variable in both an INPUT statement and a TARGET statement.
Chapter 6: The HPNEURAL Procedure

TRAIN Statement

```
TRAIN < NUMTRIES=number > < MAXITER=number >
< VALID=_NONE_> < OUTMODEL=SAS-data-set > ;
```

The TRAIN statement causes the HPNEURAL procedure to use the training data that are specified in the PROC HPNEURAL statement to train a neural network model whose structure is specified in the ARCHITECTURE, INPUT, TARGET, and HIDDEN statements. The goal of training is to determine a set of network weights that best predicts the targets in the training data while still doing a good job of predicting targets of unseen data (that is, generalizing well and not overfitting).

Except for the first try when you specify LAYER1SKIP or LAYER2SKIP in the ARCHITECTURE statement, training starts with a pseudorandomly generated initial set of weights. For the first try in LAYER1SKIP or LAYER2SKIP architectures, the initial set of weights is generated by performing a logistic regression. PROC HPNEURAL then computes the objective function for the training subset (the sum of the squared differences between the target values of each observation and the outputs of the network), and the minimization algorithm adjusts the weights. This process is repeated until any one of the following conditions is met:

- The objective function that is computed using the training subset stops improving.
- The objective function that is computed using the validation subset stops improving.
- The process has been repeated the number of times specified in the MAXITER= option.

By default, every fourth observation that is read (starting with the first observation) is used as a validation observation. You can override this by specifying VALID=_NONE_ in the TRAIN statement or by using the PARTITION statement. Any observations with missing values for input, weight, validation, or target variables are not used for training or validation error calculation, although they are scored if a SCORE statement is present. Fit statistics reported after training are only computed using validation observations. Fit statistics reported after stand-alone scoring are computed using all observations.

In distributed mode, the order of the observations that are read is not guaranteed to be the same across different distributed environments. Therefore, different validation observations might be selected in different environments when the default method is used for choosing validation observations, causing results to be somewhat different between different distributed environments. You can avoid this by using the ROLEVAR= option in a PARTITION statement to explicitly specify which observations are validation observations.

The weights that result in the smallest value of the objective function for the validation subset are saved and used for calculating fit statistics and for scoring.

When you are training, you must include exactly one TRAIN statement. The TRAIN statement is not allowed when you are doing stand-alone scoring.

NUMTRIES=number

specifies the number of times the network is to be trained using a different starting points. Specifying this option helps ensure that the optimizer finds the set of weights that truly minimizes the objective function and does not return a local minimum. The value of number must be an integer between 1 and 99,999. The default is 5.
**MAXITER=**number

specifies the maximum number of iterations (weight adjustments) for the optimizer to make before terminating.

Setting number to a large value does not mean that the optimizer actually iterates that many times. Often, training or validation error stops improving much sooner, usually after a few hundred iterations.

When you are training using large data sets, you can do a training run with MAXITER=1 to determine approximately how long each iteration will take.

The default is 50.

**VALID=_NONE_**

specifies that a validation subset not be used to help determine when to stop training.

If you specify VALID=_NONE_ in the TRAIN statement, you cannot have a PARTITION statement.

**OUTMODEL=**SAS-data-set

specifies the data set to which to save the model parameters for the trained network. These parameters include the network architecture, input and target variable names and types, and trained weights.

You can use the model data set later to score a different input data set as long as the variable names and types of the variables in the new input data set match those of the training data set.

---

**WEIGHT Statement**

```
WEIGHT variable | _INVERSE_PRIORS_;
```

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

If, instead of specifying a variable, you specify the keyword _INVERSE_PRIORS_, the HPNEURAL procedure calculates the weight applied to the prediction error of each nominal target variable as the total number of observations divided by the number of observations whose target class is the same as the current observation (in other words, the inverse of the fraction of the number of times that the target class occurs in the input data set).

If variable is less than or equal to 0 or is missing, the observation is not used for training or for computing validation error. When validation error is computed during training, the weights on the validation observations are used even though weights are not used when scoring.

The WEIGHT statement is optional. If a WEIGHT statement is not included, all observations are assigned a weight of 1.
Details: HPNEURAL Procedure

Computational Method

PROC HPNEURAL trains a multilayer perceptron neural network with one or two hidden layers. For more information about multilayer perceptron neural networks, see Bishop (1995).

All continuous input variables are scaled to be in the range \([-1, 1]\).

For all nominal input variables, except binary variables (which have a single input neuron), there is one input neuron per level. The value assigned to each input neuron is 0 except for the neuron which represents the actual input level for an observation, which has an input value of 1.

All activation functions for hidden neurons are hyperbolic tangents.

For all continuous target variables, the output activation function is the identity function, and the error function is the squared difference between the scaled target value and the network output.

For all nominal target variables, the output activation function is the softmax function, and the error function is the cross entropy function. There is one output neuron per level, except for binary variables (which have a single output neuron). The target value for each output neuron is 0 except for the neuron that represents the actual target level for an observation, which has a target value of 1.

The error function for the network is a scalar function of the network weights. This function defines an error surface on which the optimization algorithm attempts to locate a minimum. Optimization is done in two parts: the limited memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS) algorithm (Nocedal and Liu 1989) computes a direction along the error surface; then the Moré Thuente line search algorithm (Moré and Thuente 1992) finds a new minimum of the error function on the surface along that direction. If a sufficient decrease in the error function can be found, the weights that generated the new minimum are then used by the LBFGS algorithm to calculate a new descent direction. The process is repeated until a sufficient decrease in the error function cannot be obtained.

Both the LBFGS search direction algorithm and the Moré Thuente line search algorithm need to know the gradient of the error surface at several different points (sets of weights). This gradient is calculated by using the algorithm described in Bishop (1995).

Besides terminating due to the inability to improve the error function, the optimization algorithm also stops if the validation error (which is calculated after each line search) lacks improvement 40 times in a row. The validation error is computed as the sum, over each validation observation, of the absolute difference between the target and the network output. By default, every fourth observation is used as a validation observation, starting with the first observation. This default can be changed by using the PARTITION statement. Validation observations are not used in any other way by the optimization algorithm. Fit statistics reported after training are only computed using validation observations. Fit statistics reported after stand-alone scoring are computed using all observations.
Multithreading

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

By default, the number of threads used by the HPNEURAL procedure is the number of CPUs on a machine. You can control the number of threads by specifying the NTHREADS= option in the PERFORMANCE statement.

The number of threads per machine is displayed in the “Performance Information” table, which is part of the default output.

The tasks multithreaded by the HPNEURAL procedure are primarily defined by dividing the data that are processed on a single machine among the threads; that is, the HPNEURAL procedure implements multithreading through a data-parallel model. For example, if the input data set on a machine has 1,000 observations and PROC HPNEURAL is running four threads in parallel, then 250 observations are processed in parallel by each thread.

Output Data Set

The output data set is specified by the OUT= option in the SCORE statement. If there is no SCORE statement, then no output data set is created, but fit statistics are still displayed in ODS tables.

Table 6.2 describes the columns of the output data set.

<table>
<thead>
<tr>
<th>Column</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID variables from the input data set</td>
<td>Name of each ID variable in the input data set</td>
</tr>
<tr>
<td>Predicted value of the variable from the trained network</td>
<td>A name of the form P_varname for interval variables and I_varname for nominal variables, where varname is the variable name from the input data set. The name is truncated to 32 bytes if necessary. Double-byte characters are converted to underscores unless the SAS system option VALIDVARNAME=any.</td>
</tr>
<tr>
<td>For nominal values, additional columns for the raw network output for each level of each nominal variable. Because nominal variables use the softmax activation function, the raw value for a specific level is usually interpreted as the probability that the target variable is that level.</td>
<td>P_varnameLevelname where varname is the variable name from the input data set and Levelname is the level name from the input data set. The name is truncated to 32 bytes if necessary. Double-byte characters are converted to underscores unless the SAS system option VALIDVARNAME=any.</td>
</tr>
</tbody>
</table>
**Displayed Output**

PROC HPNEURAL displays basic fit statistics in the SAS log and more detailed information in several ODS tables.

When you are training, the statistics are based on the network’s prediction accuracy on the validation subset, unless there is no validation subset, in which case the statistics are based on the entire input data set.

When you are performing stand-alone scoring, the statistics depend on whether the target variables exist in the input data set: if the target variables exist in the input data set, the statistics are based on the network’s prediction accuracy on the entire input data set; otherwise no fits statistics can be computed.

For interval variables, PROC HPNEURAL displays the average absolute percentage error and maximum absolute percentage error. The percentage is the percentage of the range of the variable across the entire input data set (not just the validation observations). For nominal variables, PROC HPNEURAL displays the percentage of observations that were misclassified.

In addition to information displayed in the SAS log, PROC HPNEURAL generates ODS tables that give detailed information about the model structure, input data, detailed training results, and timings.

Other fit statistics can be computed from information in the score data set.

**ODS Table Names**

Each table created by the HPNEURAL procedure has a name associated with it, and you must use this name to refer to the table when you use ODS statements. The names of each table and a short description of the contents are listed in Table 6.3.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Required Statement / Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassLevels</td>
<td>Level information for nominal input variables</td>
<td>INPUT with LEVEL=NOM</td>
</tr>
<tr>
<td>Details</td>
<td>Detailed real times for each phase of the procedure</td>
<td>PERFORMANCE with DETAILS option</td>
</tr>
<tr>
<td>ErrorSummary</td>
<td>Average and maximum errors for interval targets</td>
<td>TARGET with LEVEL=INT (the default)</td>
</tr>
<tr>
<td>Iteration</td>
<td>Training and validation error for each iteration of the best try</td>
<td>Default output</td>
</tr>
<tr>
<td>Misclassification</td>
<td>Misclassification matrix for nominal target variables</td>
<td>TARGET with LEVEL=NOM</td>
</tr>
<tr>
<td>ModelInformation</td>
<td>Information about the modeling environment</td>
<td>Default output</td>
</tr>
<tr>
<td>Nobs</td>
<td>Number of observations read and used; number of validation observations used</td>
<td>Default output</td>
</tr>
<tr>
<td>PerformanceInfo</td>
<td>Information about the high-performance computing environment</td>
<td>Default output</td>
</tr>
<tr>
<td>Training</td>
<td>Training error, validation error, and reason for stopping for each try</td>
<td>Default output</td>
</tr>
</tbody>
</table>
References


Chapter 7
The HPREDUCE Procedure

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: HPREDUCE Procedure</td>
<td>140</td>
</tr>
<tr>
<td>PROC HPREDUCE Features</td>
<td>140</td>
</tr>
<tr>
<td>PROC HPREDUCE Contrasted with Other SAS Procedures</td>
<td>141</td>
</tr>
<tr>
<td>Getting Started: HPREDUCE Procedure</td>
<td>141</td>
</tr>
<tr>
<td>Unsupervised Variable Selection with the HPREDUCE Procedure</td>
<td>143</td>
</tr>
<tr>
<td>Supervised Variable Selection with HPREDUCE Procedure</td>
<td>146</td>
</tr>
<tr>
<td>Syntax: HPREDUCE Procedure</td>
<td>149</td>
</tr>
<tr>
<td>PROC HPREDUCE Statement</td>
<td>149</td>
</tr>
<tr>
<td>CLASS Statement</td>
<td>151</td>
</tr>
<tr>
<td>PERFORMANCE Statement</td>
<td>152</td>
</tr>
<tr>
<td>REDUCE Statement</td>
<td>152</td>
</tr>
<tr>
<td>Details: HPREDUCE Procedure</td>
<td>153</td>
</tr>
<tr>
<td>Unsupervised Variable Selection</td>
<td>154</td>
</tr>
<tr>
<td>Supervised Variable Selection</td>
<td>154</td>
</tr>
<tr>
<td>Variable Selection for Regression</td>
<td>155</td>
</tr>
<tr>
<td>Variable Selection for Classification</td>
<td>155</td>
</tr>
<tr>
<td>Criteria Used in Model Selection</td>
<td>155</td>
</tr>
<tr>
<td>Computational Method</td>
<td>156</td>
</tr>
<tr>
<td>Displayed Output</td>
<td>157</td>
</tr>
<tr>
<td>Performance Information</td>
<td>158</td>
</tr>
<tr>
<td>Model Information</td>
<td>158</td>
</tr>
<tr>
<td>Number of Observations</td>
<td>158</td>
</tr>
<tr>
<td>Class Level Information</td>
<td>158</td>
</tr>
<tr>
<td>Selection Summary</td>
<td>158</td>
</tr>
<tr>
<td>Selected Variables</td>
<td>159</td>
</tr>
<tr>
<td>Procedure Task Timing</td>
<td>159</td>
</tr>
<tr>
<td>ODS Table Names</td>
<td>159</td>
</tr>
<tr>
<td>Examples: HPREDUCE Procedure</td>
<td>160</td>
</tr>
<tr>
<td>Example 7.1: Running in Single-Machine Mode</td>
<td>160</td>
</tr>
<tr>
<td>Example 7.2: Running in Distributed Mode</td>
<td>162</td>
</tr>
<tr>
<td>Example 7.3: Output a Correlation Matrix to a SAS Data File</td>
<td>164</td>
</tr>
<tr>
<td>Example 7.4: Output the Correlation Matrix in LIL Format</td>
<td>165</td>
</tr>
<tr>
<td>Example 7.5: Output an ODS Table as A Local Data File</td>
<td>166</td>
</tr>
<tr>
<td>References</td>
<td>167</td>
</tr>
</tbody>
</table>
Overview: HPREDUCE Procedure

The HPREDUCE procedure is a high-performance procedure that performs both supervised and unsupervised variable selection on the SAS appliance. You can use the HPREDUCE procedure to read data in distributed form and perform variable selection in parallel in single-machine mode or distributed mode. For more information about these modes, see the section “Processing Modes” on page 6 in Chapter 2, “Shared Concepts and Topics.”

**NOTE:** Distributed mode requires SAS High-Performance Data Mining.

The HPREDUCE procedure performs unsupervised variable selection by identifying a set of variables that jointly explain the maximum amount of data variance. Unlike principal component analysis (PCA), which reduces dimensionality by generating a set of new variables (variable extraction), the HPREDUCE procedure reduces dimensionality by selecting a subset of the original variables (variable selection). Thus, this technique preserves model interpretation.

The HPREDUCE procedure performs supervised variable selection by identifying a set of variables that jointly explain the maximum amount of variance contained in the response variables. The HPREDUCE procedure supports variable selection in both the regression setting and the classification (categorization) setting.

The HPREDUCE procedure can also be used to output the sums of squares and crossproducts (SSCP) matrix, the correlation (CORR) matrix, or the covariance (COV) matrix for exploratory data analysis and direct input to statistical procedures that accept that form. This step saves time by eliminating redundant matrix aggregations.

PROC HPREDUCE Features

The HPREDUCE procedure conducts a variance analysis and reduces dimensionality by selecting the variables that contribute the most to the overall variance of the data (or the dependent variables). The following list summarizes the basic features of the HPREDUCE procedure:

- Variable selection is based on covariance analysis.
- Analysis can be performed on a massively parallel SAS high-performance appliance.
- Input data can be read in parallel when the data source is the appliance database.
- Computation of the CORR, COV, or SSCP matrix is distributed.
- Computation of the variable selection steps is distributed.
- All phases of analytic execution use of high degree of multithreading.
- Both supervised and unsupervised variable selection are supported.
- Multiple response variables are supported in variable selection for regression.
- The CLASS statement supports categorical inputs.
• The REDUCE statement supports main and interaction effects.
• The OUTCP statement supports outputting a CORR, COV, or SSCP matrix.

PROC HPREDUCE Contrasted with Other SAS Procedures

This section compares the HPREDUCE procedure with the FACTOR, PRINCOMP, GLMSELECT, and DISCRIM procedures in SAS/STAT software.

When PROC HPREDUCE performs unsupervised variable selection, it conducts variance analysis and reduces dimensionality by forward selection of the variables that contribute the most to the overall data variance. The output lists the variables in order of their contribution to data variance and can be used directly for reporting or for selecting variables for model building procedures. In contrast, principal component analysis (PCA) conducts a variance analysis and then projects the data space to an orthogonal set of axes by a linear combination of the original variables. These new principal components best explain the data variance and can be used as input to model building procedures. In either case, the number of inputs to the modeling procedure has been reduced from the original set. PCA can be done through the SAS/STAT FACTOR and PRINCOMP procedures. The primary difference between PCA and PROC HPREDUCE is that PCA generates new variables, while PROC HPREDUCE reduces data dimensionality by selecting a subset of the original variables. This feature of PROC HPREDUCE is beneficial in applications where retaining the original variables is important for model exploration and interpretation.

When PROC HPREDUCE performs supervised variable selection, it conducts variance analysis and reduces dimensionality by forward selection of the variables that contribute the most to explaining the overall variance of the response variables (targets). The output lists the variables in order of their contribution to explaining response variance. The output can be used directly for reporting or for selecting variables for model building procedures. When PROC HPREDUCE is used to perform supervised variable selection, it most resembles the GLMSELECT procedure. However, PROC HPREDUCE allows multiple response variables, which is not supported by PROC GLMSELECT. When the response variable is a classification variable and its levelization is done in a special format, PROC HPREDUCE conducts variance analysis in the same way as linear discriminant analysis (LDA) does. LDA can be done through the SAS/STAT DISCRIM procedure. Like PCA, LDA generates new variables by linearly combining all original variables, while PROC HPREDUCE reduces data dimensionality by selecting a subset of the original variables.

Getting Started: HPREDUCE Procedure

The following DATA step contains 100 observations with one character variable (C), one classification variable (y), and 10 continuous variables (x1-x10). This data set is used for both of the getting-started examples in the following sections.

```plaintext
data getStarted;
  input C$ y x1-x10;
  datalines;
D 0 10.2 6 1.6 38 15 2.4 20 0.8 8.5 3.9
F 1 12.2 6 2.6 42 61 1.5 10 0.6 8.5 0.7
D 1 7.7 1 2.1 38 61 1 90 0.6 7.5 5.2
```
### Chapter 7: The HPREDUCE Procedure

| J  | 10.9 | 7   | 3.5 | 46  | 42   | 0.3 | 0   | 0.2 | 6   | 3.6 |
| E  | 17.3 | 6   | 3.8 | 26  | 47   | 0.9 | 10  | 0.4 | 1.5 | 4.7 |
| A  | 18.7 | 4   | 1.8 | 2   | 34   | 1.7 | 80  | 1   | 9.5 | 2.2 |
| B  | 7.2  | 1   | 0.3 | 48  | 61   | 1.1 | 10  | 0.8 | 3.5 | 4   |
| D  | 0.1  | 3   | 2.4 | 0   | 65   | 1.6 | 70  | 0.8 | 3.5 | 0.7 |
| H  | 2.4  | 4   | 0.7 | 38  | 22   | 0.2 | 20  | 0   | 3   | 4.2 |
| J  | 15.6 | 7   | 1.4 | 0   | 98   | 0.3 | 0   | 1   | 5   | 5.2 |
| J  | 11.1 | 3   | 2.4 | 42  | 55   | 2.2 | 60  | 0.6 | 4.5 | 0.7 |
| F  | 4.6  | 0.9 | 4   | 36  | 2.1  | 30  | 0.8 | 9   | 4.6 |    |
| A  | 6.2  | 2   | 1.8 | 14  | 79   | 1.1 | 70  | 0.2 | 0   | 5.1 |
| H  | 3.7  | 3   | 0.8 | 12  | 66   | 1.3 | 40  | 0.4 | 0.5 | 3.3 |
| A  | 9.2  | 3   | 2.3 | 48  | 51   | 2.3 | 50  | 0   | 6   | 5.4 |
| G  | 14   | 3   | 2   | 18  | 12   | 2.2 | 0   | 0   | 3   | 3.4 |
| E  | 19.5 | 6   | 3.7 | 26  | 81   | 0.1 | 30  | 0.6 | 5   | 4.8 |
| C  | 11   | 3   | 2.8 | 38  | 9    | 1.7 | 50  | 0.8 | 6.5 | 0.9 |
| I  | 15.3 | 7   | 2.2 | 20  | 98   | 2.7 | 100 | 0.4 | 7   | 0.8 |
| H  | 7.4  | 4   | 0.5 | 28  | 65   | 1.3 | 60  | 0.2 | 9.5 | 5.4 |
| F  | 11.4 | 2   | 1.6 | 42  | 12   | 2.4 | 10  | 0.4 | 1   | 4.5 |
| C  | 19.4 | 1   | 0.4 | 42  | 4.2  | 10  | 0   | 6.5 | 0.1 |    |
| G  | 5.9  | 4   | 2.6 | 12  | 57   | 0.8 | 50  | 0.4 | 2   | 5.8 |
| G  | 15.8 | 6   | 3.7 | 34  | 8    | 1.3 | 90  | 0.6 | 2.5 | 5.7 |
| I  | 10   | 3   | 1.9 | 16  | 80   | 3   | 90  | 0.4 | 9.5 | 1.9 |
| E  | 15.7 | 1   | 2.7 | 32  | 25   | 1.7 | 20  | 0.2 | 8.5 | 6   |
| G  | 11   | 5   | 2.9 | 48  | 53   | 0.1 | 50  | 1   | 3.5 | 1.2 |
| J  | 16.8 | 0   | 0.9 | 14  | 86   | 1.4 | 40  | 0.8 | 9   | 5   |
| D  | 11   | 4   | 3.2 | 48  | 63   | 2.8 | 90  | 0.6 | 0   | 2.2 |
| J  | 4.8  | 7   | 3.6 | 24  | 1    | 2.2 | 20  | 1   | 8.5 | 0.5 |
| J  | 10.4 | 5   | 2   | 42  | 56   | 1   | 20  | 0   | 3.5 | 4.2 |
| G  | 12.7 | 7   | 3.6 | 8   | 56   | 2.1 | 70  | 1   | 4.5 | 1.5 |
| G  | 6.8  | 1   | 3.2 | 30  | 27   | 0.6 | 0   | 0.8 | 2   | 5.6 |
| E  | 8.8  | 0   | 3.2 | 2   | 67   | 0.7 | 10  | 0.4 | 1   | 5   |
| I  | 0.2  | 0   | 2.9 | 10  | 41   | 2.3 | 60  | 0.2 | 9   | 0.3 |
| J  | 4.6  | 7   | 3.9 | 50  | 61   | 2.1 | 50  | 0.4 | 3   | 4.9 |
| J  | 2.3  | 3.2 | 36  | 98  | 0.1  | 40  | 0.6 | 4.5 | 4.3 |    |
| I  | 10.8 | 3   | 2.7 | 28  | 58   | 0.8 | 80  | 0.8 | 3   | 6   |
| B  | 9.3  | 2   | 3.3 | 44  | 44   | 0.3 | 50  | 0.8 | 5.5 | 0.4 |
| F  | 9.2  | 6   | 0.6 | 4   | 64   | 0.1 | 0   | 0.6 | 4.5 | 3.9 |
| D  | 7.4  | 0   | 2.9 | 14  | 0    | 0.2 | 30  | 0.8 | 7.5 | 4.5 |
| G  | 18.3 | 3   | 3.1 | 8   | 60   | 0.3 | 60  | 0.2 | 7   | 1.9 |
| F  | 5.3  | 4   | 0.2 | 48  | 63   | 2.3 | 80  | 0.2 | 8   | 5.2 |
| C  | 2.6  | 5   | 2.2 | 24  | 4    | 1.3 | 20  | 0   | 2   | 1.4 |
| F  | 13.8 | 4   | 3.6 | 4   | 7    | 1.1 | 10  | 0.4 | 3.5 | 1.9 |
| B  | 12.4 | 6   | 1.7 | 30  | 44   | 1.1 | 60  | 0.2 | 6   | 1.5 |
| I  | 1.3  | 1   | 1.3 | 8   | 53   | 1.1 | 70  | 0.6 | 7   | 0.8 |
| F  | 18.2 | 7   | 1.7 | 26  | 92   | 2.2 | 30  | 1   | 8.5 | 4.8 |
| J  | 5.2  | 2   | 2.2 | 18  | 12   | 1.4 | 90  | 0.8 | 4   | 4.9 |
| G  | 9.4  | 2   | 0.8 | 22  | 86   | 0.4 | 30  | 0.4 | 1   | 5.9 |
| J  | 10.4 | 2   | 1.7 | 26  | 31   | 2.4 | 10  | 0.2 | 7   | 1.6 |
| J  | 13   | 1   | 1.8 | 14  | 11   | 2.3 | 50  | 0.6 | 5.5 | 2.6 |
| A  | 17.9 | 4   | 3.1 | 46  | 58   | 2.6 | 90  | 0.6 | 1.5 | 3.2 |
| D  | 19.4 | 6   | 3   | 20  | 50   | 2.8 | 100 | 0.2 | 9   | 1.2 |
| I  | 19.6 | 3   | 3.6 | 22  | 19   | 1.2 | 0   | 0.6 | 5   | 4.1 |
| I  | 6   | 2   | 1.5 | 30  | 30   | 2.2 | 20  | 0.4 | 8.5 | 5.3 |
| G  | 13.8 | 1   | 2.7 | 0   | 52   | 2.4 | 20  | 0.8 | 6   | 2   |
Unsupervised Variable Selection with the HPREDUCE Procedure

The following statements use PROC HPREDUCE for unsupervised variable selection. The statements specify that the technique used for variable selection is variance analysis. The maximum number of variables to select is 5, and the maximum percentage of the total variance to explain is 95%. The procedure stops when either of two conditions is satisfied.

```
; Unsupervised Variable Selection with the HPREDUCE Procedure

B 0 14.3 4 2.9 30 11 0.6 90 0.6 0.5 4.9
E 0 15.6 0 0.4 38 79 0.4 80 0.4 1 3.3
D 0 14 2 1 22 61 3 90 0.6 2 0.1
C 1 9.4 5 0.4 12 53 1.7 40 0 3 1.1
H 0 13.2 1 1.6 40 15 0.7 40 0.2 9 5.5
A 0 13.5 5 2.4 18 89 1.6 20 0.4 9.5 4.7
E 0 2.6 4 2.3 38 6 0.8 20 0.4 5 5.3
E 0 12.4 3 1.3 26 8 2.8 10 0.8 6 5.8
D 0 7.6 2 0.9 44 89 1.3 50 0.8 6 0.4
I 0 12.7 1 2.3 42 6 2.4 10 0.4 1 3
C 1 10.7 4 3.2 28 23 2.2 90 0.8 5.5 2.8
H 0 10.1 2 2.3 10 62 0.9 50 0.4 2.5 3.7
C 1 16.6 1 0.5 12 88 0.1 20 0.6 5.5 1.8
I 1 0.2 3 2.2 8 71 1.7 80 0.4 0.5 5.5
C 0 10.8 4 3.5 30 70 2.3 60 0.4 4.5 5.9
F 0 7.1 4 3 14 63 2.4 70 0 7 3.1
D 0 16.5 1 3.3 30 80 1.6 40 0 3.5 2.7
H 0 17.1 7 2.1 30 45 1.5 60 0.6 0.5 2.8
D 0 4.3 1 1.5 24 44 0 70 0 5 0.5
H 0 15 2 0.2 14 87 1.8 50 0 4.5 4.7
G 0 19.7 3 1.9 36 99 1.5 10 0.6 3 1.7
H 1 2.8 6 0.6 34 21 2 60 1 9 4.7
G 0 16.6 3 3.3 46 1 1.4 70 0.6 1.5 5.3
E 0 11.7 5 2.7 48 4 0.9 60 0.8 4.5 1.6
F 0 15.6 3 0.2 4 79 0.5 0 0.8 1.5 2.9
C 1 5.3 6 1.4 8 64 2 80 0.4 9 4.2
B 1 8.1 7 1.7 40 36 1.4 60 0.6 6 3.9
I 0 14.8 2 3.2 8 37 0.4 10 0 4.5 3
D 0 7.4 4 3 12 3 0.6 60 0.6 7 0.7
D 0 4.8 3 2.3 44 41 1.9 60 0.2 3 3.1
A 0 4.5 0 0.2 4 48 1.7 80 0.8 9 4.2
D 0 6.9 6 3.3 14 92 0.5 40 0.4 7.5 5
B 0 4.7 4 0.9 14 99 2.4 80 1 0.5 0.7
I 1 7.5 4 2.1 20 79 0.4 40 0.4 2.5 0.7
C 0 6.1 0 1.4 38 18 2.3 60 0.8 4.5 0.7
C 0 18.3 1 1 26 98 2.7 20 1 8.5 0.5
F 0 16.4 7 1.2 32 94 2.9 40 0.4 5.5 2.1
I 0 9.4 2 2.3 32 42 0.2 70 0.4 8.5 0.3
F 1 17.9 4 1.3 32 42 2 40 0.2 1 5.4
H 0 14.9 3 1.6 36 74 2.6 60 0.2 2 1.3
C 0 12.7 0 2.6 0 88 1.1 80 0.8 0.5 2.1
F 0 5.4 4 1.5 2 1 1.8 70 0.4 5.5 3.6
J 1 12.1 4 1.8 20 59 1.3 60 0.4 3 3.8
```
The output from this analysis is presented in Figure 7.1 through Figure 7.5.

**Figure 7.1** Performance Information

<table>
<thead>
<tr>
<th>Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Mode: Single-Machine</td>
</tr>
<tr>
<td>Number of Threads: 2</td>
</tr>
</tbody>
</table>

**Figure 7.2** Model Information and Number of Observations

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source: WORK.GETSTARTED</td>
</tr>
<tr>
<td>Model Type: Unsupervised</td>
</tr>
<tr>
<td>Class Parameterization: GLM</td>
</tr>
<tr>
<td>Selection Technique: Variance Analysis</td>
</tr>
<tr>
<td>Number of Variables: 11</td>
</tr>
<tr>
<td>Number of Variables to Select: 5</td>
</tr>
<tr>
<td>Variance to Explain: 0.95</td>
</tr>
</tbody>
</table>

| Number of Observations Read: 100        |
| Number of Observations Used: 100        |

**Figure 7.3** Class Level Information

<table>
<thead>
<tr>
<th>Class Level Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS variable C has 10 unique formatted levels.</td>
</tr>
</tbody>
</table>
Figure 7.3 Class Level Information and Response Profile

Class Level Information

<table>
<thead>
<tr>
<th>Class</th>
<th>Levels</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>10</td>
<td>A B C D E F G H I J</td>
</tr>
</tbody>
</table>

Figure 7.4 shows the “Selection Summary” and “Selected Variables” tables. The “Selection Summary” table shows which variable (or level for CLASS variables) is selected in each step, in addition to the total variance that is explained by the variables selected so far. The “Selected Variables” table presents all the selected variables and their corresponding variable types.

Figure 7.4 Selection Summary and Selected Variables

Selection Summary

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Effect</th>
<th>Level</th>
<th>Selected Variance Explained</th>
<th>SSE</th>
<th>MSE</th>
<th>AIC</th>
<th>AICC</th>
<th>BIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x3</td>
<td>-</td>
<td>0.0681</td>
<td>18.6380</td>
<td>0.1883</td>
<td>7.1052</td>
<td>27.2163</td>
<td>2.9713</td>
</tr>
<tr>
<td>2</td>
<td>x7</td>
<td>-</td>
<td>0.1326</td>
<td>17.3484</td>
<td>0.1770</td>
<td>6.9935</td>
<td>26.0940</td>
<td>2.9456</td>
</tr>
<tr>
<td>3</td>
<td>x10</td>
<td>-</td>
<td>0.1933</td>
<td>16.1341</td>
<td>0.1663</td>
<td>6.8609</td>
<td>24.9455</td>
<td>2.9191</td>
</tr>
<tr>
<td>4</td>
<td>C I</td>
<td>-</td>
<td>0.2529</td>
<td>14.9423</td>
<td>0.1556</td>
<td>6.7042</td>
<td>23.7675</td>
<td>2.8884</td>
</tr>
<tr>
<td>5</td>
<td>C F</td>
<td>-</td>
<td>0.3120</td>
<td>13.7607</td>
<td>0.1448</td>
<td>6.5218</td>
<td>22.5585</td>
<td>2.8521</td>
</tr>
<tr>
<td>6</td>
<td>C J</td>
<td>-</td>
<td>0.3690</td>
<td>12.6210</td>
<td>0.1343</td>
<td>6.3154</td>
<td>21.3202</td>
<td>2.8117</td>
</tr>
<tr>
<td>7</td>
<td>C B</td>
<td>-</td>
<td>0.4257</td>
<td>11.4856</td>
<td>0.1235</td>
<td>6.0811</td>
<td>20.0487</td>
<td>2.7635</td>
</tr>
<tr>
<td>8</td>
<td>C G</td>
<td>-</td>
<td>0.4813</td>
<td>10.3749</td>
<td>0.1128</td>
<td>5.8194</td>
<td>18.7444</td>
<td>2.7078</td>
</tr>
<tr>
<td>9</td>
<td>C D</td>
<td>-</td>
<td>0.5351</td>
<td>9.2980</td>
<td>0.1022</td>
<td>5.5298</td>
<td>17.4070</td>
<td>2.6443</td>
</tr>
<tr>
<td>10</td>
<td>C H</td>
<td>-</td>
<td>0.5889</td>
<td>8.2218</td>
<td>0.0914</td>
<td>5.2068</td>
<td>16.0308</td>
<td>2.5673</td>
</tr>
<tr>
<td>11</td>
<td>C A</td>
<td>-</td>
<td>0.6407</td>
<td>7.1867</td>
<td>0.0807</td>
<td>4.8522</td>
<td>14.6178</td>
<td>2.4788</td>
</tr>
<tr>
<td>12</td>
<td>x5</td>
<td>-</td>
<td>0.6916</td>
<td>6.1678</td>
<td>0.0701</td>
<td>4.4593</td>
<td>13.1611</td>
<td>2.3720</td>
</tr>
</tbody>
</table>

Selected Variables

<table>
<thead>
<tr>
<th>Number</th>
<th>Selected Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x3</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>2</td>
<td>x7</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>3</td>
<td>x10</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>CLASS</td>
</tr>
<tr>
<td>5</td>
<td>x5</td>
<td>INTERVAL</td>
</tr>
</tbody>
</table>
Figure 7.5 shows the “Procedure Task Timing” table, which provides details about how much time is used by each processing step.

**Figure 7.5** Procedure Task Timing

<table>
<thead>
<tr>
<th>Task</th>
<th>Time (sec.)</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data read and variable levelization</td>
<td>0.03</td>
<td>91.2%</td>
</tr>
<tr>
<td>Effect Levelization</td>
<td>0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>Cross-product accumulation</td>
<td>0.00</td>
<td>5.88%</td>
</tr>
<tr>
<td>Variable selection</td>
<td>0.00</td>
<td>2.94%</td>
</tr>
</tbody>
</table>

**Supervised Variable Selection with HPREDUCE Procedure**

The following statements use PROC HPREDUCE for supervised variable selection. The statements specify that \( y \) is the response variable, the technique used for variable selection is discriminant analysis, and the maximum number of variables to select is 5.

```sas
proc hpreduce data=getstarted technique=DiscriminantAnalysis;
  class C y;
  reduce supervised y = C x1-x10 / maxeffects=5;
  performance details;
run;
```

The output from this analysis is presented in Figure 7.6 through Figure 7.10.

Figure 7.6 shows the “Performance Information” table, which indicates that the procedure executes in single-machine mode. That is, the procedure runs on the machine where the SAS system is running. The table also shows that two threads are used for computing.

**Figure 7.6** Performance Information

<table>
<thead>
<tr>
<th>Execution Mode</th>
<th>Single-Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Threads</td>
<td>2</td>
</tr>
</tbody>
</table>
Figure 7.7 displays the “Model Information” and “Number of Observations” tables. The “Model Information” table shows that HPREDUCE procedure performed supervised variable selection. The CLASS variables are parameterized in the general linear model (GLM) parameterization, which is the default. The total number of variables is 12. The technique used for variable selection is discriminant analysis, and the maximum number of variables to select is 5. The “Number of Observations” table shows that all 100 observations in the data set are used in the analysis.

Figure 7.7 Model Information and Number of Observations

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source</td>
</tr>
<tr>
<td>Model Type</td>
</tr>
<tr>
<td>Class Parameterization</td>
</tr>
<tr>
<td>Selection Technique</td>
</tr>
<tr>
<td>Number of Variables</td>
</tr>
<tr>
<td>Number of Variables to Select</td>
</tr>
</tbody>
</table>

| Number of Observations Read | 100 |
| Number of Observations Used  | 100 |

Figure 7.8 shows the “Class Level Information” table, which indicates that the CLASS variable C has 10 unique formatted levels and the CLASS variable y has two levels.

Figure 7.8 Class Level Information and Response Profile

<table>
<thead>
<tr>
<th>Class Level Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>y</td>
</tr>
</tbody>
</table>

Figure 7.9 shows the “Selection Summary” and “Selected Variable” tables. The “Selection Summary” table shows which variable (or level for CLASS variables) is selected in each step, in addition to the total variance that is explained by the variables selected so far. The “Selected Variable” table presents all the selected variables and their corresponding variable types.
Figure 7.9  Selection Summary and Selected Variable

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Effect</th>
<th>Selection</th>
<th>Proportion of Selected Variance</th>
<th>Level Explained</th>
<th>SSE</th>
<th>MSE</th>
<th>AIC</th>
<th>AICC</th>
<th>BIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>J</td>
<td>0.0811</td>
<td>0.9189</td>
<td>0.00928</td>
<td>0.0154</td>
<td>2.0196</td>
<td>-0.0385</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x8</td>
<td>-</td>
<td>0.1323</td>
<td>0.8677</td>
<td>0.00885</td>
<td>-0.0019</td>
<td>2.0055</td>
<td>-0.0498</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>x2</td>
<td>-</td>
<td>0.1687</td>
<td>0.8313</td>
<td>0.00857</td>
<td>-0.0048</td>
<td>2.0067</td>
<td>-0.0467</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>C</td>
<td>0.1992</td>
<td>0.8008</td>
<td>0.00834</td>
<td>-0.0021</td>
<td>2.0145</td>
<td>-0.0379</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>x4</td>
<td>-</td>
<td>0.2184</td>
<td>0.7816</td>
<td>0.00823</td>
<td>0.0136</td>
<td>2.0362</td>
<td>-0.0161</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>x9</td>
<td>-</td>
<td>0.2369</td>
<td>0.7631</td>
<td>0.00812</td>
<td>0.0297</td>
<td>2.0593</td>
<td>0.0060</td>
<td></td>
</tr>
</tbody>
</table>

Selected Variables

<table>
<thead>
<tr>
<th>Number</th>
<th>Selected</th>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x8</td>
<td>INTERVAL</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>x2</td>
<td>INTERVAL</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>x4</td>
<td>INTERVAL</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>x9</td>
<td>INTERVAL</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.10 shows the “Procedure Task Timing” table, which provides details about how much time is used by each processing step.

Figure 7.10  Procedure Task Timing

<table>
<thead>
<tr>
<th>Task</th>
<th>Time (sec.)</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data read and variable levelization</td>
<td>0.04</td>
<td>70.7%</td>
</tr>
<tr>
<td>Effect Levelization</td>
<td>0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>Data preparation for discriminant analysis</td>
<td>0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>Cross-product accumulation</td>
<td>0.02</td>
<td>27.6%</td>
</tr>
<tr>
<td>Variable selection</td>
<td>0.00</td>
<td>1.72%</td>
</tr>
</tbody>
</table>
Syntax: HPREDUCE Procedure

The following statements are available in the HPREDUCE procedure:

```
PROC HPREDUCE < options > ;
   CLASS variable < (options) > . . . < variable < (options) > > </ global-options > ;
   REDUCE UNSUPERVISED effects </ reduce-options > ;
   REDUCE SUPERVISED response . . . < response > = effects </ reduce-options > ;
   PERFORMANCE performance-options ;
```

The PROC HPREDUCE statement and the REDUCE statement are required. The CLASS statement can appear multiple times. If a CLASS statement is specified, it must precede the REDUCE statement.

PROC HPREDUCE Statement

```
PROC HPREDUCE < options > ;
```

The PROC HPREDUCE statement invokes the procedure. Table 7.1 summarizes the important options in the PROC HPREDUCE statement by function. The options are then described fully in alphabetical order.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic Options</strong></td>
<td></td>
</tr>
<tr>
<td>DATA=</td>
<td>Specifies the input data set</td>
</tr>
<tr>
<td>NAMELEN=</td>
<td>Limits the length of effect names</td>
</tr>
<tr>
<td>TECHNIQUE=</td>
<td>Selects the variable selection technique</td>
</tr>
<tr>
<td><strong>Options Related to Output</strong></td>
<td></td>
</tr>
<tr>
<td>NOPRINT</td>
<td>Suppresses ODS output</td>
</tr>
<tr>
<td>NOCLPRINT</td>
<td>Limits or suppresses the display of CLASS levels</td>
</tr>
<tr>
<td>NOSUMPRINT</td>
<td>Suppresses generation of the selection summary table</td>
</tr>
<tr>
<td>TIMEPRINT</td>
<td>Prints the time used by each variable selection iteration</td>
</tr>
<tr>
<td>OUTCP=</td>
<td>Outputs the CORR, COV, or SSCP matrix</td>
</tr>
<tr>
<td><strong>Pearson Correlation Statistics</strong></td>
<td></td>
</tr>
<tr>
<td>COV</td>
<td>Computes covariances</td>
</tr>
<tr>
<td>CORR</td>
<td>Computes correlations (default)</td>
</tr>
<tr>
<td>SSCP</td>
<td>Computes sums of squares and crossproducts</td>
</tr>
<tr>
<td><strong>User-Defined Formats</strong></td>
<td></td>
</tr>
<tr>
<td>FMTLIBXML=</td>
<td>Specifies the file reference for a format stream</td>
</tr>
</tbody>
</table>
You can specify the following options in the PROC HPREDUCE statement.

**CORR**

selects variables based on the correlation matrix. Assuming that $X$ and $Y$ are two variables, the correlation between $X$ and $Y$ is computed by:

$$\text{Corr}(X, Y) = \frac{E((X - E(X))(Y - E(Y)))}{\sqrt{E((X - E(X))^2) E((Y - E(Y))^2)}}$$

This is the default option for computing the Pearson correlation statistics in PROC HPREDUCE.

**COV**

selects variables based on the covariance matrix. Assuming that $X$ and $Y$ are two variables, the covariance between $X$ and $Y$ is computed by:

$$\text{Cov}(X, Y) = E((X - E(X))(Y - E(Y)))$$

**DATA=SAS-data-set**

names the input SAS data set to be used by PROC HPREDUCE. The default is the most recently created data set. If the procedure 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, the procedure reads the data alongside the distributed database. See the sections “Processing Modes” on page 6 and “Alongside-the-Database Execution” on page 13 in Chapter 2, “Shared Concepts and Topics.”

**FMTLIBXML=file-ref**

specifies the file reference for the XML stream that contains the user-defined format definitions. In a distributed computing environment, user-defined formats are handled differently than they are in other SAS products. For information about how to generate an XML stream for your formats, see the section “Working with Formats” on page 32 in Chapter 2, “Shared Concepts and Topics.”

**NAMELEN=number**

specifies the length to which long effect names are shortened ($20 \leq \text{number} \leq 128$). The default and minimum value is 64.

**NOCLPRINT=number**

suppresses the display of the “Class Level Information” table if you do not specify number. If you specify number, the values of the classification variables are displayed for only those variables whose number of levels is less than number. Specifying a number helps to reduce the size of the “Class Level Information” table if some classification variables have a large number of levels.

**NOPRINT**

suppresses the generation of ODS output.

**NOSUMPRINT**

suppresses the generation of the “Selection Summary” table.

**OUTCP=SAS-data-set< /LIST< (EPS = number) >>**

creates both a data set that contains a symmetric matrix that depicts the relationships among variables and also a set of statistics about the input data set and variables. Depending on the Pearson correlation statistics option specified in the PROC HPREDUCE statement, the symmetric matrix can be a correlation (CORR) matrix, a covariance (COV) matrix, or a sums of squares and crossproducts (SSCP) matrix.
When the LIST option is specified, the symmetric matrix is output in the list-of-list (LIL) format. In this format, the matrix is represented as a set of tuples \((i, j, x)\), where \(x\) is an entry in the matrix and \(i\) and \(j\) denote its row and column indices, respectively. LIL format can be used when the output contains too many columns to fit in a data set. For example, in most database systems the maximum number of columns in a table is usually limited to several thousand. If an output matrix contains more columns than the limit, you must use the LIST option to avoid errors that would arise from writing too many columns to the table. When LIL format is used, all 0 entries in the matrix are ignored in the output.

When \(\text{EPS=} \text{number}\) is specified in the LIST option, matrix entries that have an absolute value smaller than \(\text{number}\) are ignored in the output. This feature helps omit unreliable estimations and generate a compact representation for the matrix. When the \(\text{EPS=}\) option is not specified, only the 0 entries in the matrix are ignored in the output.

\section*{SSCP}

selects variables based on the sums of squares and crossproducts matrix. Assuming that \(X\) and \(Y\) are two variables and that \(x\) and \(y\) are their corresponding variable vectors, the SSCP between \(X\) and \(Y\) is computed by:

\[
SSCP(X, Y) = x^T y
\]

\section*{TECHNIQUE=keyword}

\section*{TECH=keyword}

specifies the variable selection technique. You can specify the following \textbf{keywords}:

- \texttt{VARIANCEANALYSIS | VAR} performs variance analysis for variable selection.
- \texttt{DISCRIMINANTANALYSIS | DSC} performs discriminant analysis for variable selection.

The default value is TECHNIQUE=VAR.

You can use variance analysis for both supervised and unsupervised variable selection. You can use discriminant analysis only for supervised variable selection with one classification variable as the response. For more information, see the section “Variable Selection for Classification” on page 155.

\section*{TIMEPRINT}

prints the time (in seconds) used by each variable selection iteration in the “Selection Summary” table. If this option does not appear in the PROC HPREDUCE statement, the time information is not printed.
PERFORMANCE Statement

PERFORMANCE < performance-options> ;

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

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

The PERFORMANCE statement is documented further in the section “PERFORMANCE Statement” on page 34 of Chapter 2, “Shared Concepts and Topics.”

REDUCE Statement

REDUCE UNSUPERVISED effects < / reduce-options> ;

REDUCE SUPERVISED response . . . < response > = effects < / reduce-options> ;

PROC HPREDUCE can be used for both supervised and unsupervised variable selection. In unsupervised case, the REDUCE statement specifies the effects to be considered in the variable selection process. An effect can be an original variable in the input data set or a variable constructed from the original variables. In the supervised case, you need to specify both the effects and the response variables. A response variable can be an original variable in the input data set or a variable constructed from the original variables. For the regression case, you can specify more than one response variable.

Table 7.2 summarizes the reduce-options, which control the number of variables to be selected.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIC</td>
<td>Performs model selection by using Akaike’s information criterion</td>
</tr>
<tr>
<td>AICC</td>
<td>Performs model selection by using the corrected Akaike’s information criterion</td>
</tr>
<tr>
<td>BIC</td>
<td>Perform model selection by using Schwarz Bayesian information criterion</td>
</tr>
<tr>
<td>MAXSTEPS=</td>
<td>Specifies the maximum number of steps to take; the number must be greater than or equal to 1</td>
</tr>
<tr>
<td>MAXEFFECTS=</td>
<td>Specifies the number of effects to select; the number must be greater than or equal to 1.</td>
</tr>
<tr>
<td>VARIANCEEXPLAINED</td>
<td>VAREXP= Specifies the fraction of the total variance to be explained; the value must be between 0 and 1.</td>
</tr>
<tr>
<td>MINVARIANCEINCREMENT</td>
<td>VARINC= Specifies the minimum increment of explained variance (in a fraction of the total variance); the value must be between 0 and 1.</td>
</tr>
</tbody>
</table>
The *reduce-options* determine the number of variables to be selected. You can specify the following *reduce-options* as stopping criteria for the HPREDUCE procedure. When you specify more than one option, PROC HPREDUCE stops whenever one of the specified options is satisfied, or when the explained variance equals to the total variance. In the latter case, the procedure prints the following message in the log: “Early stop: the proportion of the explained variance to the total variance equals 1.”

**AIC**

stops PROC HPREDUCE if the Akaike’s information criterion (AIC) value fails to decrease in three contiguous steps.

**AICC**

stops PROC HPREDUCE if the corrected Akaike’s information (AICC) value fails to decrease in three contiguous steps.

**BIC**

stops PROC HPREDUCE if the Schwarz Bayesian information criterion (BIC) value fails to decrease in three contiguous steps.

**MAXSTEPS=\(n\)**

stops PROC HPREDUCE after it runs \(n\) steps.

**MAXEFFECTS=\(n\)**

stops PROC HPREDUCE after \(n\) effects have been selected. Because individual levels of one classification variable can be selected in different steps of the variable selection process, PROC HPREDUCE might take more than \(n\) steps to select \(n\) effects.

**VARIANCEEXPLAINED=\(\text{fraction}\)**

**VAREXP=\(\text{fraction}\)**

stops PROC HPREDUCE when the *fraction* of the total variance can be explained by the selected variables.

**MINVARIANCEINCREMENT=\(\text{fraction}\)**

**VARINC=\(\text{fraction}\)**

stops PROC HPREDUCE when the minimum increment of the explained variance is less than *fraction* of the total variance.

---

**Details: HPREDUCE Procedure**

The performance of a learning model usually decreases in terms of accuracy and efficiency when the dimensionality of the input data is high. The problem is known as the “curse of dimensionality.” Variable selection techniques can reduce the dimensionality of a data set by removing irrelevant and redundant variables (Liu and Motoda 1998).

The HPREDUCE procedure performs both supervised and unsupervised variable selection. It selects variables by identifying a set of variables that can jointly explain the maximum amount of data variance.
Unsupervised Variable Selection

When no response variable is specified, PROC HPREDUCE conducts unsupervised variable selection. Assume that \( k \) variables need to be selected. Let \( X \in \mathbb{R}^{n \times m} \) be a data set that contains \( n \) samples and \( m \) variables; let \( X = (X_1, X_2) \), where \( X_1 \in \mathbb{R}^{n \times k} \) is the data set that contains the \( k \) selected variables and \( X_2 \in \mathbb{R}^{n \times (m-k)} \) contains the remaining \( m - k \) variables. PROC HPREDUCE selects the variables by minimizing the equation:

\[
\min \text{Trace} \left( X_2^T \left( I - X_1 \left( X_1^T X_1 \right)^{-1} X_1^T \right) X_2 \right)
\]

\( \left( I - X_1 \left( X_1^T X_1 \right)^{-1} X_1^T \right)^{\frac{1}{2}} \) projects \( X_2 \) to the null space of \( X_1 \). Therefore, the above equation measures the data variance that resides in the null space of \( X_1 \), which is the data variance that cannot be explained by the variables in \( X_1 \). Minimizing this equation leads to the selection of the variables that jointly explain the maximum amount of the variance in the original data.

Let \( C_{11} = X_1^T X_1 \), \( C_{12} = X_1^T X_2 \), and \( C_{21} = X_2^T X_1 \). The following equations hold:

\[
C = X^T X = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}
\]

\[
X_2^T \left( I - X_1 \left( X_1^T X_1 \right)^{-1} X_1^T \right) X_2 = C_{22} - C_{21} C_{11}^{-1} C_{21}
\]

When all the variables are centralized to have a zero mean, \( C \) is the covariance matrix. This corresponds to setting the \texttt{COV} option in the PROC HPREDUCE statement, which specifies that the covariance matrix be used for variable selection. Similarly, if variables need to be both centralized and normalized to have unit length, you should specify the \texttt{CORR} option in the PROC HPREDUCE statement, which leads to the use of the correlation matrix for variable selection. If neither centralization nor normalization should be applied, you need to specify the \texttt{SSCP} option in the PROC HPREDUCE statement.

Principal component analysis (PCA) (Jolliffe 2002) also reduces dimensionality by preserving data variance. The key difference between PCA and PROC HPREDUCE is that PCA generates a small set of new variables (variable extraction) by linearly combining the original variables, while PROC HPREDUCE selects a small set of the original variables (variable selection). The variables returned by PROC HPREDUCE are the original variables. This feature is very important in applications where retaining the original variables is important for model exploration or interpretation (for example, genetic analysis, and text mining).

Supervised Variable Selection

When response variables are specified in a REDUCE statement, PROC HPREDUCE conducts supervised variable selection. The procedure supports variable selection both in a regression context and in a classification (categorization) context.
Variable Selection for Regression

In a regression setting, all response variables should be numerical. When a classification variable is in the response, this variable needs to be levelized to multiple dummy variables, with each dummy variable corresponding to a level of the classification variable. You can achieve this levelization by adding this variable to the variable list of the CLASS statement.

Let $Y \in \mathbb{R}^{n \times t}$ be the response data that contain $t$ response variables. Assume that $k$ variables need to be selected. Let $X \in \mathbb{R}^{n \times m}$ be a data set that contains $n$ samples and $m$ variables; let $X = (X_1, X_2)$, where $X_1 \in \mathbb{R}^{n \times k}$ is the data set that contains the $k$ selected variables and $X_2 \in \mathbb{R}^{n \times (m-k)}$ contains the remaining $m-k$ variables. PROC HPREDUCE selects the variables by minimizing the following equation:

$$\min \text{Trace} \left(Y^T \left(I - X_1 \left(X_1^T X_1 \right)^{-1} X_1^T \right) Y \right)$$

$\left(I - X_1 \left(X_1^T X_1 \right)^{-1} X_1^T \right)^{1/2}$ $Y$ projects $Y$ to the null space of $X_1$. Therefore, the equation measures the response variance that resides in the null space of $X_1$, which is the variance of the response variables that cannot be explained by the variables in $X_1$. Minimizing the equation leads to the selection of the variables that jointly explain the maximum amount of the variance of the response variables.

Variable Selection for Classification

In a classification setting, one classification variable is specified as the response, with each of its levels corresponding to a category of the classification problem. Let the classification variable be $y$ with $c$ levels $\{1, \ldots, c\}$. Then $y$ can be levelized in a special way to generate a response data $Y \in \mathbb{R}^{n \times c}$ as:

$$Y_{i,j} = \begin{cases} \frac{1}{\sqrt{n}} \left( \sqrt{\frac{n}{n_j}} - \sqrt{\frac{n}{n}} \right), & y_i = j \\ -\frac{1}{\sqrt{n}} \sqrt{\frac{n_j}{n}}, & y_i \neq j \end{cases}$$

By using this $Y$ in the variance analysis, PROC HPREDUCE selects variables by using the discriminant criterion specified in linear discriminant analysis (LDA) (Fisher 1936; Cooley and Lohnes 1971). LDA also reduces dimensionality. The key difference between LDA and PROC HPREDUCE is that LDA generates a small set of new variables (variable extraction) by linearly combining the original variables, while PROC HPREDUCE selects a small set of the original variables (variable selection).

Criteria Used in Model Selection

The HPREDUCE procedure supports the following three fit statistics that you can specify as stopping criteria in the REDUCE statement:

- **AIC** Akaike’s information criterion (Akaike 1969; Judge et al. 1985)
- **AICC** Corrected Akaike’s information criterion (Hurvich and Tsai 1989)
- **BIC** Schwarz Bayesian information criterion (Schwarz 1978; Judge et al. 1985)
The HPREDUCE procedure supports multiple response variables; therefore, it computes the AIC, AICC, and BIC that are defined for multivariate regression. Besides the three criteria, it also computes the error sum of squares (SSE) and residual mean square error (MSE).

Table 7.3 provides formulas and definitions for these fit statistics.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition or Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>Number of observations</td>
</tr>
<tr>
<td>( p )</td>
<td>Number of parameters</td>
</tr>
<tr>
<td>( t )</td>
<td>Number of response variables</td>
</tr>
<tr>
<td>SSE</td>
<td>Error sum of squares</td>
</tr>
<tr>
<td>MSE</td>
<td>( \frac{SSE}{n-p} )</td>
</tr>
<tr>
<td>AIC</td>
<td>( \ln\left(\frac{SSE}{n}\right) + \frac{2pt + t(t+1)}{n} )</td>
</tr>
<tr>
<td>AICC</td>
<td>( \ln\left(\frac{SSE}{n}\right) + \frac{(n+p)t}{n-p-t-1} )</td>
</tr>
<tr>
<td>BIC</td>
<td>( \ln\left(\frac{SSE}{n}\right) + \frac{p \ln(n)}{n} )</td>
</tr>
</tbody>
</table>

**Computational Method**

Given \( m \) variables, finding the \( k \) variables that minimize the proposed equations is a combinatorial problem, which is NP-hard (nondeterministic polynomial-time hard). To select \( k \) variables, PROC HPREDUCE applies \( k \) steps of a greedy search to generate a suboptimal solution for the problem.

Assume that \( q \) features have been selected, that \( X_1 \) contains the \( q \) selected variables, and that \( X_2 \) contains the remaining variables. PROC HPREDUCE selects the \( q + 1 \) variable, \( F \), by minimizing the equation

\[
\arg \min_F \text{Trace} \left( \hat{X}_2^T \left( I - \hat{X}_1 \left( \hat{X}_1^T \hat{X}_1 \right)^{-1} \hat{X}_1^T \right) \hat{X}_2 \right)
\]

where \( \hat{X}_1 \) is the data set that contains the feature \( F \) and the \( q \) selected variables, and \( \hat{X}_2 \) is the data set that contains the remaining variables. Minimizing the preceding problem is equivalent to maximizing the following problem:

\[
\left\| X_2^T \left( I - X_1 \left( X_1^T X_1 \right)^{-1} X_1^T \right) f \right\|_2^2 - \left\| \left( I - X_1 \left( X_1^T X_1 \right)^{-1} X_1^T \right)^{1/2} f \right\|_2^2
\]

In the preceding equation, \( \left\| X_2^T \left( I - X_1 \left( X_1^T X_1 \right)^{-1} X_1^T \right) f \right\|_2^2 \) is the summation of the squares of the covariance between the variable \( f \) and all the unselected variables in the null space of \( X_1 \). And
$$\left\| \left(I - X_1 (X_1^T X_1)^{-1} X_1^T\right)^{1/2} f \right\|_2^2$$ is the square of the variance of \( f \) in the null space of \( X_1 \), which is used as a normalization factor.

This problem can be solved efficiently. Assuming that \( m \gg k \), the time complexity for solving it is

$$O \left(m^2 \left(n + k^2\right)\right)$$

where \( m \) is the number of variables, \( n \) is the number of samples, and \( k \) is the number of selected variables. In the equation, \( m^2 n \) corresponds to the time for computing the covariance (or correlation, or SSCP) matrix. And \( m^2 k^2 \) corresponds to the time for selecting \( k \) variables out of \( m \).

Similar analysis also applies to supervised variable selection with PROC HPRDUCE. In this case, the following problem is maximized for variable selection:

$$\left\| Y^T \left(I - X_1 (X_1^T X_1)^{-1} X_1^T\right)^{1/2} f \right\|_2^2$$

Here, \( Y \) is the response data. Let \( c \) be the number of columns in \( Y \). The time complexity for selecting \( k \) variables by solving the preceding problem is

$$O \left(k^2 \left(c + k\right) m + m^2 n\right)$$

Note that for most data of very high dimensionality, \( c + k \ll m \).

PROC HPRDUCE is fully threaded and distributed. When there are \( p \) machines used for computing, the time complexity for unsupervised variable selection is

$$CPU \left(\frac{m^2 (n + k^2)}{p} + m^2 \log p\right) + NET \left(m^2 \log p\right)$$

And the time complexity for supervised variable selection is

$$CPU \left(\frac{k^2 (c + k) m + m^2 n}{p} + m^2 \log p\right) + NET \left(m^2 \log p\right)$$

where CPU corresponds to the time used for computing and NET corresponds to the time used for communication among computers.

**Displayed Output**

The following sections describe the output that PROC HPRDUCE produces by default. The output is organized into various tables, which are discussed in the 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.

Model Information

The “Model Information” table displays basic information about the model, such as the data source, the selection technique, the number of selected variables, and the execution mode that the HPREDUCE procedure determines based on your input and options. If you want to know whether the procedure executed on the client machine or in distributed form or whether data were sent from the client or processed alongside the database, check the execution mode entry of this table.

Number of Observations

The “Number of Observations” table displays the number of observations read from the input data set and the number of observations used in the analysis.

Class Level Information

The “Class Level Information” table lists the levels of every variable specified in the CLASS statement. You should check this information to make sure that the data are correct. You can adjust the order of the CLASS variable levels with the ORDER= option in the CLASS statement.

If the classification variables are in reference parameterization, the “Class Level Information” table also displays the reference value for each variable.

Selection Summary

The “Selection Summary” table displays for each iteration the name of the selected effect, the name of the selected level, and the total variance explained after the iteration. If you specify the TIMEPRINT option in the PROC HPREDUCE statement, information about the time used by each iteration is added to the “Selection Summary” table.
Selected Variables

The “Selected Variables” table summarizes which variables were selected in the selection process. It also provides information about the variable type of each selected variable.

Procedure Task Timing

If you specify the DETAILS option in the PERFORMANCE statement, the procedure produces a “Procedure Task Timing” table, which displays the elapsed time (absolute and relative) for the main tasks of the procedure.

ODS Table Names

Each table created by the HPREDUCE 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 7.4.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Required Statement / Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassLevels</td>
<td>Level information from the CLASS statement</td>
<td>CLASS</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Information about modeling environment</td>
<td>Default output</td>
</tr>
<tr>
<td>NObs</td>
<td>Number of observations read and used; number of events and trials, if applicable</td>
<td>Default output</td>
</tr>
<tr>
<td>SelectedEffects</td>
<td>Summary of selected variables</td>
<td>Default output</td>
</tr>
<tr>
<td>SelectionSummary</td>
<td>Selection summary</td>
<td>Default output</td>
</tr>
<tr>
<td>Timing</td>
<td>Absolute and relative times for tasks performed by the procedure</td>
<td>DETAILS option in the PERFORMANCE statement</td>
</tr>
</tbody>
</table>
Examples: HPREDUCE Procedure

Example 7.1: Running in Single-Machine Mode

This example first generates a data set, which has 2,000 observations and contains both interval variables ($x_1$–$x_{10}$) and CLASS variables ($b_1$–$b_3$, $c_1$–$c_{10}$). Then PROC HPREDUCE is run to select variables. When a host for distributed computing is not specified or the NODES option in the PERFORMANCE statement is not specified, PROC HPREDUCE automatically runs in single-machine mode.

```sas
data one;
array x{10};
array c{10};
do i=1 to 2000;
   do j=1 to 10;
      x{j}=ranuni(1);
      c{j}=int(ranuni(1)*4);
   end;
   if c{1} eq 0 then b1 = 'aa';
   if c{1} eq 1 then b1 = 'bb';
   if c{1} eq 2 then b1 = 'cc';
   if c{1} eq 3 then b1 = 'dd';
   if c{1} eq 4 then b1 = 'ee';
   if c{2} eq 0 then b2 = 'ff';
   if c{2} eq 1 then b2 = 'gg';
   if c{2} eq 2 then b2 = 'hh';
   if c{2} eq 3 then b2 = 'ii';
   if c{2} eq 4 then b2 = 'jj';
   if c{3} eq 0 then b3 = 'kk';
   if c{3} eq 1 then b3 = 'll';
   if c{3} eq 2 then b3 = 'mm';
   if c{3} eq 3 then b3 = 'nn';
   if c{3} eq 4 then b3 = 'oo';
   output;
end;
r
run;

proc hprreduce data=one;
   class b1-b3 c1-c3;
   reduce unsupervised b1 b1*b2 b3 c1-c3 x1-x10/maxsteps=5;
   performance details;
run;
```

Output 7.1.1 shows the results for PROC HPREDUCE running in single-machine mode. Notice that the “Performance Information” table shows that the “Execution mode” is “On client.”
### Output 7.1.1 PROC HPREDUCE Running in Single-Machine Mode

#### Performance Information

- **Execution Mode**: Single-Machine
- **Number of Threads**: 2

#### Model Information

- **Data Source**: WORK.ONE
- **Model Type**: Unsupervised
- **Class Parameterization**: GLM
- **Selection Technique**: Variance Analysis
- **Number of Variables**: 16
- **Maximal Number of Steps**: 5
- **Number of Observations Read**: 2000
- **Number of Observations Used**: 2000

#### Class Level Information

<table>
<thead>
<tr>
<th>Class</th>
<th>Levels</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td>4</td>
<td>aa bb cc dd</td>
</tr>
<tr>
<td>b2</td>
<td>4</td>
<td>ff gg hh ii</td>
</tr>
<tr>
<td>b3</td>
<td>4</td>
<td>kk ll mm nn</td>
</tr>
<tr>
<td>c1</td>
<td>4</td>
<td>0 1 2 3</td>
</tr>
<tr>
<td>c2</td>
<td>4</td>
<td>0 1 2 3</td>
</tr>
<tr>
<td>c3</td>
<td>4</td>
<td>0 1 2 3</td>
</tr>
</tbody>
</table>

#### Selection Summary

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Effect</th>
<th>Level</th>
<th>Proportion of Variance</th>
<th>SSE</th>
<th>MSE</th>
<th>AIC</th>
<th>AICC</th>
<th>BIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b1</td>
<td>cc</td>
<td>0.0821</td>
<td>42.2236</td>
<td>0.0211</td>
<td>4.8230</td>
<td>49.8490</td>
<td>3.7468</td>
</tr>
<tr>
<td>2</td>
<td>b1</td>
<td>bb</td>
<td>0.1637</td>
<td>38.4694</td>
<td>0.0193</td>
<td>4.7279</td>
<td>48.7538</td>
<td>3.6575</td>
</tr>
<tr>
<td>3</td>
<td>b1</td>
<td>aa</td>
<td>0.2443</td>
<td>34.7635</td>
<td>0.0174</td>
<td>4.6236</td>
<td>47.6494</td>
<td>3.5600</td>
</tr>
<tr>
<td>4</td>
<td>b3</td>
<td>kk</td>
<td>0.3036</td>
<td>32.0323</td>
<td>0.0160</td>
<td>4.5377</td>
<td>46.5635</td>
<td>3.4819</td>
</tr>
<tr>
<td>5</td>
<td>b3</td>
<td>mm</td>
<td>0.3619</td>
<td>29.3519</td>
<td>0.0147</td>
<td>4.4454</td>
<td>45.4710</td>
<td>3.3984</td>
</tr>
</tbody>
</table>

#### Selected Variable

<table>
<thead>
<tr>
<th>Number</th>
<th>Effect</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b1</td>
<td>CLASS</td>
</tr>
<tr>
<td>2</td>
<td>b3</td>
<td>CLASS</td>
</tr>
</tbody>
</table>

#### Procedure Task Timing

<table>
<thead>
<tr>
<th>Task</th>
<th>Time (sec.)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Data read and variable levelization</td>
<td>0.03</td>
<td>56.5%</td>
</tr>
<tr>
<td>Effect Levelization</td>
<td>0.00</td>
<td>8.70%</td>
</tr>
<tr>
<td>Cross-product accumulation</td>
<td>0.02</td>
<td>34.8%</td>
</tr>
<tr>
<td>Variable selection</td>
<td>0.00</td>
<td>0.00%</td>
</tr>
</tbody>
</table>
Example 7.2: Running in Distributed Mode

When a host for distributed computing is specified and the NODES option in the PERFORMANCE statement is specified, PROC HPREDUCE uses the specified host for computing and runs in distributed mode.

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

data one;
  array x{10};
  array c{10};
  do i=1 to 2000;
    do j=1 to 10;
      x(j)=ranuni(1);
      c(j)=int(ranuni(1)*4);
    end;
    y=int(ranuni(1)*2);
    output;
  end;
run;

proc hpreduce data=one tech=var;
  class c1 c2 c3;
  reduce supervised y = c1-c3 x1-x10/maxsteps=5;
  performance nodes=2;
run;
```

To run the preceding example successfully, you need to set the macro variables GRIDHOST and GRIDINSTALLLOC to resolve to appropriate values, or you can replace the references to the macro variables in the example with the appropriate values.

Output 7.2.1 shows the results for PROC HPREDUCE running in distributed mode. Notice that the “Performance Information” table shows that the “Execution mode” is “Distributed.”
### Output 7.2.1 PROC HPREDUCE Running in Distributed Mode

**Performance Information**

- Host Node: << your grid host >>
- Install Location: << your grid install location >>
- Execution Mode: Distributed
- Number of Compute Nodes: 2
- Number of Threads per Node: 8

**Model Information**

- Data Source: WORK.ONE
- Model Type: Supervised
- Class Parameterization: GLM
- Selection Technique: Variance Analysis
- Number of Variables: 14
- Maximal Number of Steps: 5

- Number of Observations Read: 2000
- Number of Observations Used: 2000

**Class Level Information**

<table>
<thead>
<tr>
<th>Class</th>
<th>Levels</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>4</td>
<td>0 1 2 3</td>
</tr>
<tr>
<td>c2</td>
<td>4</td>
<td>0 1 2 3</td>
</tr>
<tr>
<td>c3</td>
<td>4</td>
<td>0 1 2 3</td>
</tr>
</tbody>
</table>

**Selection Summary**

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Effect</th>
<th>Level</th>
<th>Variance Explained</th>
<th>SSE</th>
<th>MSE</th>
<th>AIC</th>
<th>AICC</th>
<th>BIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c2</td>
<td>0</td>
<td>0.0017</td>
<td>0.9983</td>
<td>0.000499</td>
<td>0.0003</td>
<td>1.0003</td>
<td>0.0021</td>
</tr>
<tr>
<td>2</td>
<td>c3</td>
<td>0</td>
<td>0.0026</td>
<td>0.9974</td>
<td>0.000499</td>
<td>0.0004</td>
<td>1.0004</td>
<td>0.0050</td>
</tr>
<tr>
<td>3</td>
<td>c3</td>
<td>3</td>
<td>0.0034</td>
<td>0.9966</td>
<td>0.000499</td>
<td>0.0006</td>
<td>1.0006</td>
<td>0.0080</td>
</tr>
<tr>
<td>4</td>
<td>x2</td>
<td>–</td>
<td>0.0042</td>
<td>0.9958</td>
<td>0.000499</td>
<td>0.0008</td>
<td>1.0008</td>
<td>0.0110</td>
</tr>
<tr>
<td>5</td>
<td>x8</td>
<td>–</td>
<td>0.0045</td>
<td>0.9955</td>
<td>0.000499</td>
<td>0.0015</td>
<td>1.0015</td>
<td>0.0145</td>
</tr>
</tbody>
</table>

**Selected Variable**

<table>
<thead>
<tr>
<th>Number</th>
<th>Selected Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c2</td>
<td>CLASS</td>
</tr>
<tr>
<td>2</td>
<td>c3</td>
<td>CLASS</td>
</tr>
<tr>
<td>3</td>
<td>x2</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>4</td>
<td>x8</td>
<td>INTERVAL</td>
</tr>
</tbody>
</table>
Chapter 7: The HPREDUCE Procedure

Example 7.3: Output a Correlation Matrix to a SAS Data File

This example shows how to output a correlation matrix to a SAS data file. The OUTCP option creates an output data set named corr.

```sas
data one;
  array x{2};
  array c{2};
  do i=1 to 2000;
    do j=1 to 2;
      x{j}=ranuni(1);
      c{j}=int(ranuni(1)*2);
    end;
    output;
  end;
run;

title "Output the Correlation Matrix";
proc hpreduce data=one corr outcp=corr;
  class a;
  reduce unsupervised a x1-x2 /maxsteps=4;
run;
proc print data=corr;
run;
```

Output 7.3.1 shows the content of the data file generated by PROC HPREDUCE.

Output 7.3.1  Output the Correlation Matrix

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>ID</em></th>
<th><em>TYPE</em></th>
<th><em>VAR</em></th>
<th><em>LEV</em></th>
<th><em>vID</em></th>
<th>v1</th>
<th>v2</th>
<th>v3</th>
<th>v4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>MEAN/FREQ</td>
<td>979.00</td>
<td>1021.00</td>
<td>0.49</td>
<td>0.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>N</td>
<td>2000.00</td>
<td>2000.00</td>
<td>2000.00</td>
<td>2000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>CORR</td>
<td>a</td>
<td>0</td>
<td>v1</td>
<td>1.00</td>
<td>-1.00</td>
<td>0.03</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>CORR</td>
<td>a</td>
<td>1</td>
<td>v2</td>
<td>-1.00</td>
<td>1.00</td>
<td>-0.03</td>
<td>-0.00</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>CORR</td>
<td>x1</td>
<td>v3</td>
<td>0.03</td>
<td>-0.03</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>CORR</td>
<td>x2</td>
<td>v4</td>
<td>0.00</td>
<td>-0.00</td>
<td>0.00</td>
<td>1.00</td>
<td></td>
</tr>
</tbody>
</table>

The values in the column _VAR_ are the name of the variables. The _LEV_ column shows the name of a CLASS variable’s levels, but is empty for interval variables. Assuming that you have \( n \) effects (the total number of interval variables and the levels of CLASS variables), the _vID_ column contains \( n \) markers, \( v_1 \) to \( v_n \), where \( v_i \) denotes the \( i \)th effect. The column _TYPE_ defines the role of each row. When the _TYPE_ column shows MEAN/FREQ, the corresponding row contains either the mean for an interval variable or the frequency for a level of a CLASS variable. When the _TYPE_ column shows N, the corresponding row contains the number of samples. And when the _TYPE_ column shows CORR, COV, or SSCP, the corresponding row contains a row of the CORR, COV, or SSCP matrix. In this example, the CORR matrix is \( 4 \times 4 \), and it resides in the table in row 3 through row 6 and column 7 through column 10.
Example 7.4: Output the Correlation Matrix in LIL Format

By using the LIST option in the OUTCP option, you can output a correlation matrix in LIL format.

```sas
data one;
  array x{2};
  do i=1 to 2000;
    a=int(ranuni(1)*2);
    do j=1 to 2;
      x{j}=ranuni(1);
    end;
    output;
  end;
run;

title "Output the Correlation Matrix in LIL format";
proc hpreduce data=one corr outcp=corr_lil/list(eps=0.01);
  class a;
  reduce unsupervised a x1-x2 /maxsteps=4;
run;
proc print data=corr_lil;
run;
```

Output 7.4.1 shows the correlation matrix in LIL format.

Output 7.4.1  Output the Correlation Matrix in LIL Format

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>TYPE</em></th>
<th><em>ID</em></th>
<th><em>NAME1</em></th>
<th><em>NAME2</em></th>
<th><em>VAL</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S</td>
<td>1</td>
<td>samples</td>
<td></td>
<td>2000.00</td>
</tr>
<tr>
<td>2</td>
<td>S</td>
<td>2</td>
<td>nVar</td>
<td></td>
<td>3.00</td>
</tr>
<tr>
<td>3</td>
<td>S</td>
<td>3</td>
<td>nEff</td>
<td></td>
<td>4.00</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>1</td>
<td>a</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>2</td>
<td>a</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>M</td>
<td>3</td>
<td>x1</td>
<td></td>
<td>0.49</td>
</tr>
<tr>
<td>7</td>
<td>M</td>
<td>4</td>
<td>x2</td>
<td></td>
<td>0.50</td>
</tr>
<tr>
<td>8</td>
<td>R</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>9</td>
<td>R</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>-1.00</td>
</tr>
<tr>
<td>10</td>
<td>R</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1.00</td>
</tr>
<tr>
<td>11</td>
<td>R</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0.03</td>
</tr>
<tr>
<td>12</td>
<td>R</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>-0.03</td>
</tr>
<tr>
<td>13</td>
<td>R</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>1.00</td>
</tr>
<tr>
<td>14</td>
<td>R</td>
<td>10</td>
<td>4</td>
<td>4</td>
<td>1.00</td>
</tr>
</tbody>
</table>

The column _TYPE_ defines the type of each row:

- When the _TYPE_ column shows S, the corresponding row contains the statistics of the data set. More specifically, when the _TYPE_ column shows S and the _NAME1_ column shows samples, the _VAL_ column in the corresponding row contains the number of samples in the data set. Similarly, when the
_TYPE_ column shows S and the _NAME1_ column shows nVar, the _VAL_ column contains the number of variables. And when the _TYPE_ column shows S and the _NAME1_ column shows nEff, the _VAL_ column in the corresponding row contains the number of effects.

- When the _TYPE_ column shows F, the row contains the frequency of a level of a CLASS variable. In this case, the _NAME1_ column contains the name of the CLASS variable and the _NAME2_ column contains the name of the corresponding level.

- When the _TYPE_ column shows M, the row contains the mean of an interval variable. In this case, the _NAME1_ column contains the name of the variable and the _NAME2_ column is empty.

- When the _TYPE_ column shows R, the row contains an entry in the correlation matrix. In this case, the _NAME1_ column contains the row ID, the _NAME2_ column contains the column ID, and the _VAL_ column contains the value.

- When the _TYPE_ column shows V or P, the corresponding row contains an entry of a COV matrix or an SSCP matrix, respectively.

Only entries in the lower triangle of the correlation matrix are written to the file, because the correlation matrix is symmetric. Also any entry of the matrix that has a value smaller than 0.01 is ignored in the output (EPS = 0.01), which saves storage space.

---

**Example 7.5: Output an ODS Table as A Local Data File**

The ODS output of PROC HPREDUCE can be stored in a local data file. The following example shows how the “Iteration History” table can be stored as a local file named IterHist by using the ODS output statement:

```plaintext
data one;
   array x{10};
   array c{10};
   do i=1 to 2000;
      do j=1 to 10;
         x{j}=ranuni(1);
         c{j}=int(ranuni(1)*4);
      end;
      if c{1} eq 0 Then b1 = 'aa';
      if c{1} eq 1 Then b1 = 'bb';
      if c{1} eq 2 then b1 = 'cc';
      if c{1} eq 3 then b1 = 'dd';
      if c{1} eq 4 then b1 = 'ee';
      if c{2} eq 0 Then b2 = 'ff';
      if c{2} eq 1 Then b2 = 'gg';
      if c{2} eq 2 then b2 = 'hh';
      if c{2} eq 3 then b2 = 'ii';
      if c{2} eq 4 then b2 = 'jj';
      if c{3} eq 0 Then b3 = 'kk';
      if c{3} eq 1 Then b3 = 'll';
      if c{3} eq 2 then b3 = 'mm';
```


```markdown
if c(3) eq 3 then b3 = 'nn';
if c(3) eq 4 then b3 = 'oo';
output;
end;
run;

proc hpreduce data=one;
ods output SelectionSummary=Summary;
class b1-b3 c1-c3;
reduce unsupervised b1 b1*b2 b3 c1-c3 x1-x10 /maxsteps =5;
performance details;
run;
```

## References


### Subject Index

- **adjusting statistics when sampling target classes unevenly**
  - HPFOREST procedure, 87

- **algorithm**
  - HPFOREST procedure, 86

- **bagging the data**
  - HPFOREST procedure, 79

- **baseline fit statistics**
  - HPFOREST procedure, 101

- **bias and correlation**
  - HPFOREST procedure, 99

- **CLASS level**
  - HPREDUCE procedure, 150, 158

- **computational method**
  - HPNEURAL procedure, 134
  - HPREDUCE procedure, 156

- **controlling for variable selection bias**
  - HPFOREST procedure, 80

- **criteria**
  - HPFOREST procedure, 85

- **decision matrix**
  - HPDECRIDE procedure, 57

- **displayed output**
  - HP4SCORE procedure, 44
  - HPDECRIDE procedure, 58
  - HPFOREST procedure, 101
  - HPNEURAL procedure, 136
  - HPREDUCE procedure, 157

- **effect**
  - name length (HPREDUCE), 150

- **fit criteria**
  - HPREDUCE procedure, 155

- **fit statistics**
  - HPFOREST procedure, 101

- **formulas for adjusting statistics when sampling unevenly**
  - HPFOREST procedure, 88

- **frequency variable**
  - HPDECRIDE procedure, 56
  - HPFOREST procedure, 76

- **handling missing values**
  - HPFOREST procedure, 94

- **hpforest procedure**
  - 87

- **HP4SCORE procedure**
  - displayed output, 44
  - input data sets, 42
  - input model file, 43
  - output data set, 44

- **HPDECIDE procedure**
  - 49
  - code file, 54
  - code metabase, 54
  - code residual, 54
  - cost options, 55
  - decision data sets, 54
  - decision matrix, 57
  - decision variables, 55
  - displayed output, 58
  - input data sets, 53
  - ODS table name, 58
  - old prior variable, 55
  - outfit data sets, 54
  - output data sets, 53
  - prior variable, 55
  - role of data set, 54
  - variables, 58

- **HPFOREST procedure**
  - 66
    - adjusting statistics when sampling target classes unevenly, 87
    - algorithm, 86
    - association test number of categories, 73
    - association test significance level, 73
    - bagging the data, 79
    - baseline fit statistics, 101
    - bias and correlation, 99
    - candidate variables to try, 76
    - controlling for variable selection bias, 80
    - criteria, 85
    - displayed output, 101
    - fit statistics, 101
    - formulas for adjusting statistics when sampling unevenly, 88
    - handling missing values, 94
    - handling values that are absent from training data, 96
    - illustrations of adjusting when sampling unevenly, 89
    - input data sets, 73
    - loss reduction variable importance, 102
    - MAXDEPTH= option, 74
    - maximum number of trees, 74
measuring variable importance, 96
minimum missing values to use in split search, 75
missing values, 75
model information, 101
number of observations, 101
ODS table names, 102
performance information, 101
random number seed, 75
rules, 85
save file, 78
score out, 78
searching for a splitting rule, 85
selecting a splitting variable, 83
skip rows of displayed fit statistics, 75
split search exhaustive method, 74
split search LEAFFRACTION= option, 74
split search LEAFSIZE= option, 74
split search MINCATSIZE= option, 74
split search SPLITSIZE= option, 75
technical derivations of adjustments formulas, 93
train fraction, 76
train n, 76
training a decision tree, 79
turn variable importance calculations on or off, 74
within node sample, 75
HPNEURAL procedure, 119
computational method, 134
displayed output, 136
input data distribution, 127
input data sets, 126
multithreading, 130, 135
ODS table names, 136
output data set, 135
HPREDUCE procedure, 139
AIC, 153
AICC, 153
BIC, 153
CLASS level, 150, 158
computational method, 156
computes correlations, 150, 151
computes covariances, 150
displayed output, 157
effect name length, 150
fit criteria, 155
input data sets, 150
maximum effects, 153
maximum steps, 153
minimal explained variance increment, 153
model information, 158
multithreading, 152
number of observations, 158
ODS table names, 159
performance information, 158
prints the time used by each variable selection iteration, 151
procedure task timing, 159
reduce options, 152
reduce options summary, 152
selected variable, 159
selection summary, 158
user-defined formats, 158
variance explained, 153
XML input stream, 150
illustrations of adjusting when sampling unevenly
HPFOREST procedure, 89
input variables
HPNEURAL procedure, 129
loss reduction variable importance
HPFOREST procedure, 102
measuring variable importance
HPFOREST procedure, 96
model
information (HPFOREST), 101
information (HPREDUCE), 158
multithreading
HPNEURAL procedure, 130, 135
HPREDUCE procedure, 152
number of hidden neurons
HPNEURAL procedure, 128
number of observations
HPFOREST procedure, 101
HPREDUCE procedure, 158
options summary
PROC HPREDUCE statement, 149
output data set
HP4SCORE procedure, 44
HPNEURAL procedure, 135
performance information
HPFOREST procedure, 101
HPREDUCE procedure, 158
posterior variables
HPDECIDE procedure, 56
predicted variable
HPDECIDE procedure, 57
procedure task timing
HPDECIDE procedure, 159
reduce options
HPREDUCE procedure, 152
rules
HPFOREST procedure, 85
searching for a splitting rule
HPFOREST procedure, 85
selected variable
HPREDUCE procedure, 159
selecting a splitting variable
HPFOREST procedure, 83
selection summary
HPREDUCE procedure, 158
target variables
HPNEURAL procedure, 131
technical derivations of adjustments formulas
HPFOREST procedure, 93
training a decision tree
HPFOREST procedure, 79
variables
HPDECIDE procedure, 58
weight variable
HPNEURAL procedure, 133
Syntax Index

AIC option
  REDUCE statement, 153
AICC option
  REDUCE statement, 153
ALPHA= option
  PROC HPFOREST statement, 73
ARCHITECTURE statement
  HPNEURAL procedure, 127
BIC option
  REDUCE statement, 153
CATBINS= option
  PROC HPFOREST statement, 73
CLASS statement
  HPREDUCE procedure, 151
CODE statement
  HPDECIDE procedure, 54
  HPNEURAL procedure, 128
COMMIT= option
  PERFORMANCE statement (high-performance analytical procedures), 34
CORR
  PROC HPREDUCE statement, 150
COST= option
  PROC HPDECIDE statement, 55
COV
  PROC HPREDUCE statement, 150
DATA= option
  PROC HP4SCORE statement, 42
  PROC HPDECIDE statement, 53
  PROC HPFOREST statement, 73
  PROC HPNEURAL statement, 126
  PROC HPREDUCE statement, 150
DATASERVER= option
  PERFORMANCE statement (high-performance analytical procedures), 34
DECDATA= option
  PROC HPDECIDE statement, 54
DECISION statement
  HPDECIDE procedure, 54
DECVARS= option
  PROC HPDECIDE statement, 55
DETAILS option
  PERFORMANCE statement (high-performance analytical procedures), 35
DISTR option
  PROC HPNEURAL statement, 127
EXHAUSTIVE= option
  PROC HPFOREST statement, 74
FILE= option
  PROC HPDECIDE statement, 54
  SCORE statement, 43
FMTLIBXML= option
  PROC HPREDUCE statement, 150
FREQ statement
  HPDECIDE procedure, 56
  HPFOREST procedure, 76
GRIDHOST= option
  PERFORMANCE statement (high-performance analytical procedures), 35
GRIDMODE= option
  PERFORMANCE statement (high-performance analytical procedures), 35
GRIDTIMEOUT= option
  PERFORMANCE statement (high-performance analytical procedures), 35
HIDDEN statement
  HPNEURAL procedure, 128
high-performance analytical procedures,
  PERFORMANCE statement, 34
  COMMIT= option, 34
  DATASERVER= option, 34
  DETAILS option, 35
  GRIDHOST= option, 35
  GRIDMODE= option, 35
  GRIDTIMEOUT= option, 35
  HOST= option, 35
  INSTALL= option, 35
  INSTALLLOC= option, 35
  LASR= option, 35
  LASRSERVER= option, 35
  MODE= option, 35
  NNODES= option, 36
  NODES= option, 36
  NTHREADS= option, 37
  THREADS= option, 37
  TIMEOUT= option, 35
HOST= option
  PERFORMANCE statement (high-performance analytical procedures), 35
HP4SCORE procedure, 42
  syntax, 42
HP4SCORE procedure, PROC HP4SCORE statement
DATA= option, 42
HP4SCORE procedure, SCORE statement, 43
FILE= option, 43
MAXDEPTH= option, 43
NTREES= option, 43
OUT= option, 43
HP4SCOREC procedure, ID statement, 42
HP4SCOREC procedure, PROC HP4SCORE statement, 42
HPDECIDE procedure, 53
CODE statement, 54
DECISION statement, 54
FREQ statement, 56
ID statement, 56
PERFORMANCE statement, 56
POSTERIORS statement, 56
PREDICTED statement, 57
PROC HPDECIDE statement, 53
syntax, 53
TARGET statement, 57
HPDECIDE procedure, PROC HPDECIDE statement
COST= option, 55
DATA= option, 53
DECDATA= option, 54
DECVARs= option, 55
FILE= option, 54
METABASE= option, 54
OLDPRIORVAR= option, 55
OUT= option, 53
OUTFIT= option, 54
PRIORVAR= option, 55
RESIDUAL, 54
ROLE= option, 54
HPFOREST procedure, 73
FREQ statement, 76
ID statement, 77
INPUT statement, 76
PROC HPFOREST statement, 73
SAVE statement, 78
SCORE statement, 78
syntax, 73
TARGET statement, 78
HPFOREST procedure, FREQ statement, 76
HPFOREST procedure, ID statement, 77
HPFOREST procedure, PROC HPFOREST statement
ALPHA= option, 73
CATBINS= option, 73
DATA= option, 73
EXHAUSTIVE= option, 74
IMPORTANCE= option, 74
LEAFFRACTION= option, 74
LEAFSIZE= option, 74
MAXDEPTH= option, 74
MAXTREES= option, 74
MINCATSIZE= option, 74
MINUSENSEARCH= option, 75
MISSING= option, 75
NODESIZE= option, 75
SEED= option, 75
SKIP_SEQ_ROWS= option, 75
SPLITSIZE= option, 75
TRAINFRACTION= option, 76
TRAINN= option, 76
VARS_TO_TRY= option, 76
HPFOREST procedure, SCORE statement, 78
MAXDEPTH= option, 78
NTREES= option, 78
HPLOGISTIC procedure, PROC HPREDUCE statement
NOPRINT option, 150
HPNEURAL procedure, 126
ARCHITECTURE statement, 127
CODE statement, 128
HIDDEN statement, 128
ID statement, 129
INPUT statement, 129
PARTITION statement, 129
PERFORMANCE statement, 130
PROC HPNEURAL statement, 126
SCORE statement, 131
syntax, 126
TARGET statement, 131
TRAIN statement, 132
WEIGHT statement, 133
HPNEURAL procedure, ID statement, 129
HPNEURAL procedure, PERFORMANCE statement, 130
HPNEURAL procedure, PROC HPNEURAL statement
DATA= option, 126
DISTR= option, 127
NOPRINT option, 127
HPNEURAL procedure, TRAIN statement
MAXITER= option, 132
NUMTRIES= option, 132
OUTMODEL= option, 133
VALID= option, 133
HPNEURAL procedures, ARCHITECTURE statement, 127
HPNEURAL procedures, CODE statement, 128
HPNEURAL procedures, HIDDEN statement, 128
HPNEURAL procedures, INPUT statement, 129
HPNEURAL procedures, PARTITION statement, 129
HPNEURAL procedures, SCORE statement, 131
HPNEURAL procedures, TARGET statement, 131
HPNEURAL procedures, TRAIN statement, 132
HPNEURAL procedures, WEIGHT statement, 133
HPREDUCE procedure, 149
PERFORMANCE statement, 152
PROC HPREDUCE statement, 149
REDUCE statement, 152
syntax, 149

HPREDUCE procedure, CLASS statement, 151
HPREDUCE procedure, PERFORMANCE statement, 152
HPREDUCE procedure, PROC HPREDUCE statement
CORR, 150
COV, 150
DATA= option, 150
FMTLIBXML= option, 150
NAMELEN= option, 150
NOSUMPRINT option, 150
OUTCP= option, 150
SSCP, 151
TECHNIQUE= option, 151
TIMEPRINT, 151
HPREDUCE procedure, REDUCE statement, 152
AIC option, 153
AICC option, 153
BIC option, 153
MAXEFFECTS option, 153
MAXSTEPS option, 153
MINVARIANCEINCREMENT option, 153
VAREXPI option, 153
VARIANCEEXPLAINED option, 153
VARINC option, 153
HPREDUCEC procedure, PROC HPREDUCE statement, 149
ID statement
HPDECIDE procedure, 56
HPFOREST procedure, 77
HPNEURAL procedure, 129
IMPORTANCE= option
PROC HPFOREST statement, 74
INPUT statement
HPFOREST procedure, 76
HPNEURAL procedure, 129
INSTALL= option
PERFORMANCE statement (high-performance analytical procedures), 35
INSTALLLOC= option
PERFORMANCE statement (high-performance analytical procedures), 35
LEAFFRACTION= option
PROC HPFOREST statement, 74
LEAFSIZE= option
PROC HPFOREST statement, 74
MAXDEPTH= option
PROC HPFOREST statement, 74
SCORE statement, 43, 78
MAXEFFECTS option
REDUCE statement, 153
MAXITER= option
TRAIN statement, 132
MAXSTEPS option
REDUCE statement, 153
MAXTREES= option
PROC HPFOREST statement, 74
METABASE= option
PROC HPDECIDE statement, 54
MINCATSIZE= option
PROC HPFOREST statement, 74
MINUSEINSEARCH= option
PROC HPFOREST statement, 75
MINVARIANCEINCREMENT option
REDUCE statement, 153
MISSING= option
PROC HPFOREST statement, 75
MODE= option
PERFORMANCE statement (high-performance analytical procedures), 35
NAMELEN= option
PROC HPREDUCE statement, 150
NNODES= option
PERFORMANCE statement (high-performance analytical procedures), 36
NOCLPRINT option
PROC HPREDUCE statement, 150
NODES= option
PERFORMANCE statement (high-performance analytical procedures), 36
NODESIZE= option
PROC HPFOREST statement, 75
NOPRINT option
PROC HPNEURAL statement, 127
PROC HPREDUCE statement, 150
NOSUMPRINT option
PROC HPREDUCE statement, 150
NTHREADS= option
PERFORMANCE statement (high-performance analytical procedures), 37
NTREES= option
SCORE statement, 43, 78
NUMTRIES= option
TRAIN statement, 132
OLDPRIORVAR= option
   PROC HPDECIDE statement, 55
OUT= option
   PROC HPDECIDE statement, 53
   SCORE statement, 43
OUTCP= option
   PROC HPREDUCE statement, 150
OUTFIT= option
   PROC HPDECIDE statement, 54
OUTMODEL= option
   TRAIN statement, 133

PARTITION statement
   HPNEURAL procedure, 129
PERFORMANCE statement
   high-performance analytical procedures, 34
   HPDECIDE procedure, 56
   HPNEURAL procedure, 130
   HPREDUCE procedure, 152

POSTERIORS statement
   HPDECIDE procedure, 56
PREDICTED statement
   HPDECIDE procedure, 57
PRIORVAR= option
   PROC HPDECIDE statement, 55
PROC HPDECIDE statement
   HPDECIDE procedure, 53
PROC HPFOREST statement
   HPFOREST procedure, 73
PROC HPNEURAL statement, see HPNEURAL procedure
   HPNEURAL procedure, 126
PROC HPREDUCE statement
   HPREDUCE procedure, 149

REDUCE statement
   HPREDUCE procedure, 152
RESIDUAL
   PROC HPDECIDE statement, 54
ROLE= option
   PROC HPDECIDE statement, 54

SAVE statement
   HPFOREST procedure, 78
SCORE statement
   HPFOREST procedure, 78
   HPNEURAL procedure, 131
SEED= option
   PROC HPFOREST statement, 75
SKIP_SEQ_ROWS= option
   PROC HPFOREST statement, 75
SPLITSIZE= option
   PROC HPFOREST statement, 75
SSCP
   PROC HPREDUCE statement, 151

syntax
   HP4SCORE procedure, 42
   HPDECIDE procedure, 53
   HPFOREST procedure, 73
   HPREDUCE procedure, 149

TARGET statement
   HPDECIDE procedure, 57
   HPFOREST procedure, 78
   HPNEURAL procedure, 131

TECHNIQUE= option
   PROC HPREDUCE statement, 151
THREADS= option
   PERFORMANCE statement (high-performance analytical procedures), 37
TIMEOUT= option
   PERFORMANCE statement (high-performance analytical procedures), 35
TIMEPRINT
   PROC HPREDUCE statement, 151

TRAIN statement
   HPNEURAL procedure, 132
TRAINFRACTION= option
   PROC HPFOREST statement, 76
TRAINN= option
   PROC HPFOREST statement, 76

VALID= option
   TRAIN statement, 133
VAREXP option
   REDUCE statement, 153
VARIANCEEXPLAINED option
   REDUCE statement, 153
VARS_TO_TRY= option
   PROC HPFOREST statement, 76

WEIGHT statement
   HPNEURAL procedure, 133
We welcome your feedback.

- If you have comments about this book, please send them to yourturn@sas.com. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to suggest@sas.com.
SAS® Publishing Delivers!

Whether you are new to the work force or an experienced professional, you need to distinguish yourself in this rapidly changing and competitive job market. SAS® Publishing provides you with a wide range of resources to help you set yourself apart. Visit us online at support.sas.com/bookstore.

SAS® Press

Need to learn the basics? Struggling with a programming problem? You’ll find the expert answers that you need in example-rich books from SAS Press. Written by experienced SAS professionals from around the world, SAS Press books deliver real-world insights on a broad range of topics for all skill levels.

support.sas.com/saspress

SAS® Documentation

To successfully implement applications using SAS software, companies in every industry and on every continent all turn to the one source for accurate, timely, and reliable information: SAS documentation. We currently produce the following types of reference documentation to improve your work experience:

- Online help that is built into the software.
- Tutorials that are integrated into the product.
- Reference documentation delivered in HTML and PDF – free on the Web.
- Hard-copy books.

support.sas.com/publishing

SAS® Publishing News

Subscribe to SAS Publishing News to receive up-to-date information about all new SAS titles, author podcasts, and new Web site features via e-mail. Complete instructions on how to subscribe, as well as access to past issues, are available at our Web site.

support.sas.com/spn