THE POWER TO KNOW.

SAS® Enterprise Miner™ High-Performance Data Mining Procedures and Macro Reference for SAS® 9.3

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2011. *SAS® Enterprise Miner™ High-Performance Data Mining Procedures and Macro Reference for SAS® 9.3*. Cary, NC: SAS Institute Inc.

SAS® Enterprise Miner™ High-Performance Data Mining Procedures and Macro Reference for SAS® 9.3

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, December 2011

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

Contents

Part 1

High-Performance Procedures

Chapter 1

HP4SCORE Procedure

Overview: HP4SCORE Procedure

The HP4SCORE procedure scores a previously trained random forest model produced by the HPFOREST procedure.

Syntax: HP4SCORE Procedure

PROC HP4SCORE DATA=<*libref.*>*SAS-data-set*;
ID *variables-list*;
PERFORMANCE <*performance-options*>;
SAVE FILE=*file-name*;
SCORE FILE=*file-name* OUT=*SAS-data-set*;

PROC HP4SCORE Statement

The PROC HP4SCORE statement invokes the procedure.

Syntax

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

Details

Required Arguments

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

Specifies the input data set that is used by the HPFOREST procedure to generate the random forest model.

Note: Due to restrictions on the length of variable names, it is possible that unique names are not generated by HPFOREST in the OUT= data set. The current versions of the HPFOREST procedure and the HP4SCORE procedure do not check for unique variable names and will terminate with an error if all variable names are not unique.

ID Statement

The ID statement lists one or more variables from the input data set that are transferred to the output data sets. The ID statement accepts numeric and character variables. For more information about the ID statement, see “Shared Concepts and Topics” in the *SAS High-Performance Analytics User’s Guide*.

Syntax

ID *variables-list*;

Details

Required Arguments

variables-list

Specifies the variables that you want to transfer from the input data set to the output data sets.

PERFORMANCE Statement

The PERFORMANCE statement defines performance parameters for multi-threaded and distributed computing, communicates variable information about the distributed computing environment, and requests detailed results about the performance characteristics of the HP4SCORE procedure. With the PERFORMANCE statement, you can control whether the HP4SCORE procedure executes in symmetric multiprocessing or massively parallel mode. For more information about the PERFORMANCE statement, see “Shared Concepts and Topics” in the *SAS High-Performance Analytics User’s Guide*.

Syntax

PERFORMANCE <*performance-options*>;

Details

Optional Arguments

COMMIT=*n*

Specifies the minimum number of observations that must transfer from the client to the appliance before the SAS log is updated. For example, if you specify **`COMMIT=5000`**, then every time the number of observations sent exceeds an integer multiple of 5000, a log message is produced. This message indicates the actual number of observations distributed, not the COMMIT= value that triggered the message.

CPUCOUNT= ACTUAL | *number*

Specifies how many processors that PROC HP4SCORE assumes are available on each host in the computing environment. Valid values for *number* are integers between 1 and 256, inclusive. Setting CPUCOUNT= to a value greater than the actual number of available CPUs can result in reduced performance.

Specify **`CPUCOUNT=ACTUAL`** to set CPUCOUNT= to the number of processors physically available. This number can be less than the physical number of CPUs if the SAS process has been restricted by system administration tools.

This option overrides the CPUCOUNT= SAS system option.

If PROC HP4SCORE executes in SMP mode, then this option refers to the client machine of the SAS session. If PROC HP4SCORE executes in MPP mode, then this option applies the nodes on the appliance.

DATASERVER=*"name"*

Specifies the server name on Teradata systems as defined through the *hosts* file and as used in the LIBNAME statement for Teradata. For example, if the *hosts* file defines **`myservercop1 33.44.55.66`** as the server for Teradata, then a LIBNAME statement would be as follows:

```
libname TDLIB terdata server=myserver user= password= database= ;
```

To induce PROC HP4SCORE to run alongside the Teradata server, specify the following performance statement:

```
performance dataserver="myserver";
```

DETAILS

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

TIMEOUT=*s*

Specifies the length of time, in seconds, that PROC HP4SCORE should wait for a connection to the appliance and should wait before establishing a connection back to the client. The default value for *s* 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.

HOST=*"name"*
GRIDHOST=*"name"*

Specifies the name of the appliance host. The HOST= option overrides the value of the GRIDHOST environment variable.

INSTALL=*"name"*
INSTALLOC=*"name"*

Specifies the directory where the SAS High-Performance Analytics shared libraries are installed on the appliance. Specifying the INSTALL= option overrides the GRIDINSTALLLOC environment variable.

NODES=*n*
NNODES=*n*

Specifies the number of nodes in the distributed computing environment, provided that the data is not processed alongside the database. Specify **NODES=0** to indicate that you want to process the data in SMP mode on the client machine. If the input data is not alongside the database, this is the default setting. The HP4SCORE procedure then performs multi-threaded analysis on the client.

If the data is not read alongside the database, the NODES= option specifies the number of nodes on the appliance that are involved in the analysis.

If the number of nodes can be modified by the application, you can specify a NODES= option where *n* exceeds the number of physical nodes on the appliance. The SAS High-Performance Analytics software then over-subscribes the nodes and associates nodes with multiple units of work. For example, on a system with 16 appliance nodes, the following statement would over-subscribe the system by a factor of 3:

```
performance nodes=48 host="hpa.sas.com";
```

Generally, it is not advisable to over-subscribe the system because the analytic code is optimized for a certain level of multi-threading on the nodes that depend on the CPU count.

If the data is 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.

NTHREADS=*n*

Specifies the number of threads used for analytic computations and overrides the SAS system option THREADS | NOTHREADS. If you do not specify the NTHREADS= option, then the number of threads is determined based on the number of CPUs on the host machine where the analytic computations execute.

By default, SAS High-Performance Analytics procedures execute in multiple concurrent threads, unless you disable this behavior with the NOTHREADS system option or you specify NTHREADS=1 to force single-threaded execution. The value specified here must not exceed 256.

Note: The SAS system option THREADS | NOTHREADS applies to the current machine where the SAS High-Performance Analytics procedures execute. This option does not apply to the compute nodes in a distributed environment.

SAVE Statement

Syntax

SAVE FILE=*file-name*;

Details

Required Argument

FILE=*file-name*
Specifies the location where PROC HP4SCORE will save the scoring model. This version of PROC HP4SCORE requires a fully formed, physical path to the filename. File references are not supported.

SCORE Statement

The SCORE statement identifies the model created by the HPFOREST procedure and outputs the scoring information.

Syntax

SCORE FILE=*file-name* OUT=*SAS-data-set*;

Details

Required Arguments

FILE=*file-name*
Specifies the filename created by the FILE= argument in the HPFOREST procedure. This version of PROC HP4SCORE requires a fully formed, physical path to the file name. File references are not supported.

OUT=*SAS-data-set*
Specifies the output data set that contains scored model.

Chapter 2

HPBIN Procedure

Overview: HPBIN Procedure

The HPBIN Procedure

Binning is a common step in the data preparation stage of the model building process. You can use binning to classify missing variables, reduce the impact of outliers, or generate multiple effects. The generated effects are useful in modeling nonlinear processes.

The HPBIN procedure conducts high-performance binning that uses either bucket binning or pseudo-quantile binning. Like other high-performance procedures, the HPBIN procedure can read and write data in distributed form. Also, PROC HPBIN can perform analyses in parallel in either symmetric multiprocessing (SMP) mode or massively parallel processing (MPP) mode.

Features

The HPBIN Procedure has the following features:

- performs analysis on a massively parallel SAS high-performance appliance
- reads input data in parallel and writes output data in parallel when the data source is on the appliance database
- is highly multithreaded during all phases of analytic execution
- provides a bucket (or equal length) binning method
- provides a pseudo-quantile binning method, which is similar to quantile binning
- provides a mapping table for the selected binning method
- provides different output tables according to user preferences.

Bucket Binning

Those familiar with SAS Enterprise Miner should recall a special variable named _AOV16_, which is used heavily during the data preparation and variable selection stages. This variable is a class variable with maximum level 16, derived from an interval variable that you want to bin. Bucket binning creates equal length bins and assigns the data to one of these bins. The bucket lengths of the equal length bins are calculated according to (maximum — minimum)/(number of valid observations).

Pseudo-Quantile Binning

Quantile binning requires a particular data sorting, and the sorting process typically consumes a significant amount of CPU time and memory usage. When the input data set is larger than the available memory, it is nearly impossible to sort the data in any amount of time. For distributed computing, data communications overhead increases the data sorting challenge.

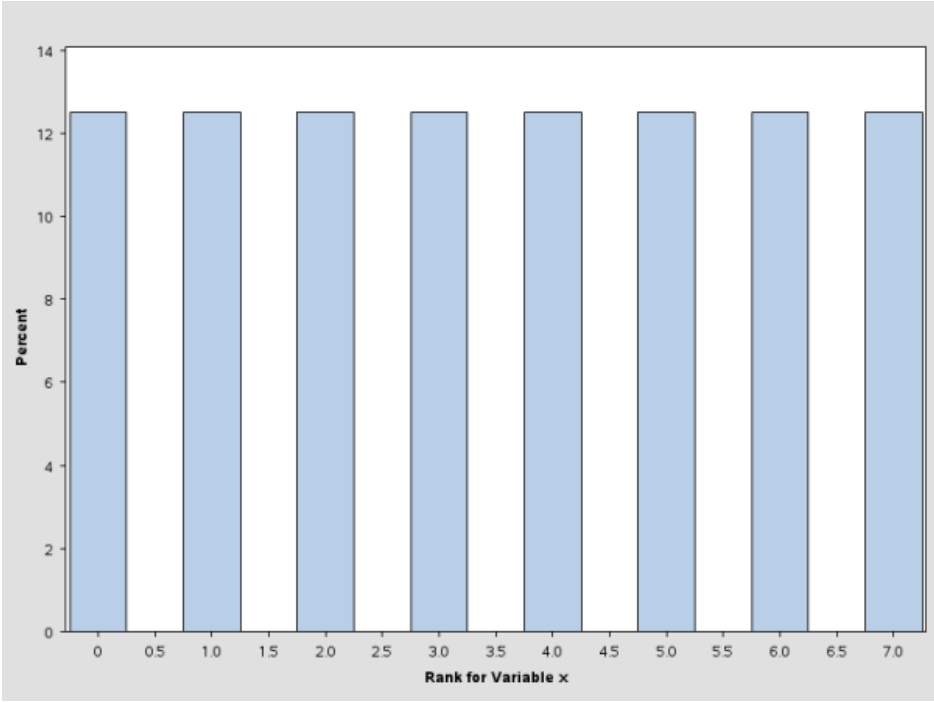
To address these issues, the HPBIN procedure contains a novel approach to quantile binning, named pseudo-quantile binning. The pseudo-quantile method is very efficient, and the results mimic those of the quantile binning method. For example, consider the code below:

```
data bindata;
    do i=1 to 1000;
    x=rannorm(1);
    output;
    end;
run;

proc rank data=bindata out=rankout group=8;
    var x;
    ranks rank_x;
run;

proc univariate data=rankout plot;
    var rank_x;
    histogram;
run;
```

This code creates a data set with 1000 observations, each generated by a random normal distribution. The histogram for this data set is shown below.

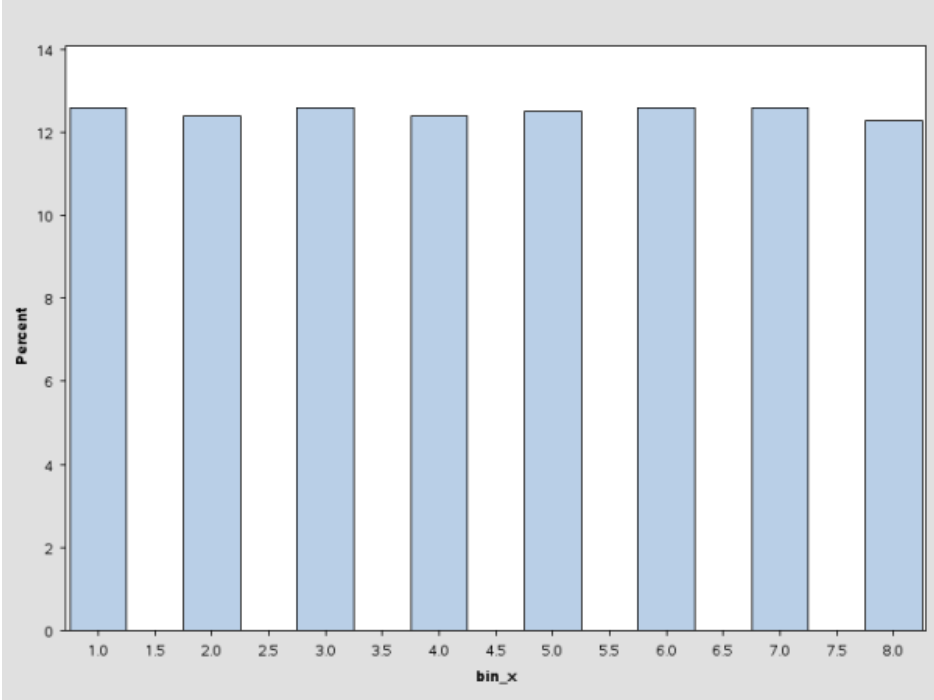


The pseudo-quantile method used by PROC HPBIN achieves similar results in far less computation time. In this case, the time complexity is C*O(n), where C is a constant and n is the number of observations. When the algorithm runs on the grid, the total computation time is much less. For example, if a cluster has 32 nodes and each node has 24 shared memory CPUs, then the time complexity is (C*O(n))/(32*24).

The code below bins the data using the PSEUDO_QUANTILE option. The histogram for the data is given below the code. Note that it is similar to the histogram shown above.

```
proc hpbin data=bindata output=binout numbin=8 pseudo_quantile;
    var x;
run;

proc univariate data=binout plot;
    var bin_x;
    histogram;
run;
```



The Output Data Set

The output data set that PROC HPBIN generates varies based on the statements and options that you include in your PROC HPBIN call. By default, the output data set includes the original input data and the binning variables. However, you can alter this data set with an ID statement, the REPLACE option, or certain other conditions. The following conditions affect the information in the output data set:

- if PROC HPBIN was run in solo mode
- if the output is transferred back to the client machine
- if the output is created on the grid alongside a database
- if NOPRINT is specified, suppressing all ODS output

Note: If the input variable value is missing, then the binning output level value is 0.

Variable Mapping Table

By default, the variable mapping table is provided in the PROC HPBIN output. This table provides the level mapping for the input variables. The level starts at 1 and continues to the value of NUMBINS. In the mapping table, a missing value for the lower bound indicates negative infinity, and a missing value for the upper bound indicates positive infinity.

Note that the final binning level might be less than NUMBINS, if the input data is small or the binning variable is discrete. In this case, a warning message is printed.

Performance Information

The Performance Information table is produced by default, and displays information about the grid host for distributed execution. Moreover, this table specifies if the

procedure was executed in client mode, distributed mode, or alongside-the-database mode. The number of compute nodes and threads are also provided, depending on the environment.

ODS Tables

Each table created by the HPBIN procedure has a name that is associated with it. You must use this name to refer to each table when you use ODS statements. These tables are described below.

Table Name	Description	Required Statements or Arguments
hpbininfo	This table provides basic binning information, including the method, number of bins, number of binning variables, and number of observations.	This table requires the PROC HPBIN statement, options, and observation count.
mappingTable	This table provides the level mapping table, calculated based on the binning option specified.	This table requires you to specify either BUCKET or PSEUDO-QUANTILE in the PROC HPBIN statement.
PerformanceInfo	This table contains information about the high-performance computing environment.	This table requires the OUTPUT= argument.

Syntax: HPBIN Procedure

Requirement: The VAR statement is required for PROC HPBIN.

```
PROC HPBIN DATA=SAS-data-set <options>;
    FREQ variable;
    ID variables-list;
    PERFORMANCE <performance-options;>
    VAR variables-list;
RUN;
```

PROC HPBIN Statement

The PROC HBPIN statement invokes the procedure.

Syntax

```
PROC HPBIN DATA=data-set-name <options>;
```

Details

Required Arguments

DATA=<*libref.*>*SAS-data-set*
Specifies the data set that is used in the binning process. By default, the most recently created data set is used. If the data is already distributed, the procedure reads the data alongside the distributed database.

Optional Arguments

NOPRINT
Suppresses the generation of ODS output.

NUMBIN=*n*
Specifies the number of bins that are created. The value of *n* must be an integer between 2 and 1000, inclusive. The default value is 16.

OUTPUT=*SAS-data-set*
Specifies the binning output data set. By default, this data set contains the original data and the extra binning data.

REPLACE
Replaces the specified variables in the original data set with the binning variables. If the ID statement is present, the REPLACE option is ignored, because the output data set contains only the ID and binning variables.

Options

BUCKET | PSEUDO_QUANTILE
Specify BUCKET to apply equal-length binning. You can specify the binning method after you have specified your arguments. The PSEUDO_QUANTILE method approximates the results of quantile binning. The default binning method is BINNING.

FREQ Statement

The variable in the FREQ statement identifies a numeric variable in the input data set that contains the frequency of occurrence for each observation.

Note: For bucket binning, the FREQ statement has no effect.

Syntax

FREQ *variable*;

Details

Required Argument

variable
Specify the variable in the input data set that contains the frequency for each observation. The HPBIN procedure treats each observation as if it appeared *f* times, where *f* is the value of the frequency variable for that observation. If the frequency

value is not an integer, then it is truncated to an integer. If the frequency value is less than 1 (or missing), the observation is not used in the analysis. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

ID Statement

The ID statement lists one or more variables from the input data set that are transferred to the output data sets. The ID statement accepts numeric and character variables. If an ID statement is used, the ID variables and the binning output variables are included in the output data set. For more information about the ID statement, see “Shared Concepts and Topics” in the *SAS High-Performance Analytics User’s Guide*.

Note: The ID statement is optional.

Syntax

ID *variables-list*

Details

Required Argument

variables-list

Use the ID statement to specify the variables that you want to transfer from the input data set to the output data sets.

PERFORMANCE Statement

The PERFORMANCE statement defines performance parameters for multi-threaded and distributed computing, communicates variable information about the distributed computing environment, and requests detailed results about the performance characteristics of the HPBIN procedure. With the PERFORMANCE statement, you can control whether the HPBIN procedure executes in symmetric multiprocessing mode or massively parallel mode. For more information about the PERFORMANCE statement, see “Shared Concepts and Topics” in the *SAS High-Performance Analytics User’s Guide*.

Syntax

PERFORMANCE <*performance-options*>;

Details

Optional Arguments

COMMIT=*n*

Specifies the minimum number of observations transferred from the client to the appliance necessary to update the SAS Log. For example, if you specify `COMMIT=5000`, then every time the number of observations sent exceeds an integer multiple of 5000 a log message is produced. This message indicates the actual number of observations distributed, not the COMMIT= value that triggered the message.

CPUCOUNT= ACTUAL | *number*

Specifies how many processors PROC HPBIN assumes are available on each host in the computing environment. Valid values for *number* are integers between 1 and 256, inclusive. Setting CPUCOUNT= to a value greater than the actual number of available CPUs might results in reduced performance.

Specify **`CPUCOUNT=ACTUAL`** to set CPUCOUNT= to the number of processors physically available. This number can be less than the physical number of CPUs if the SAS process has been restricted by system administration tools.

This option overrides the CPUCOUNT= SAS system option.

If PROC HPBIN executes in SMP mode, then this option refers to the client machine of the SAS session. If PROC HPBIN executes in MPP mode, then this option applies the nodes on the appliance.

DATASERVER=*"name"*

Specifies the server name on Teradata systems as defined through the *hosts* file and as used in the LIBNAME statement for Teradata. For example, if the *hosts* file defines **`myservercop1 33.44.55.66`** as the server for Teradata, then a LIBNAME statement would be as follows:

```
libname TDLIB terdata server=myserver user= password= database= ;
```

To induce PROC HPBIN to run alongside the Teradata server, specify the following performance statement:

```
performance dataserver="myserver";
```

DETAILS

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

TIMEOUT=*s*

Specifies the length of time, in seconds, that PROC HPBIN should wait for a connection to the appliance and to establish a connection back to the client. The default value for *s* 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.

HOST=*"name"*
GRIDHOST=*"name"*

Specifies the name of the appliance host. The HOST= option overrides the value of the GRIDHOST environment variable.

INSTALL=*"name"*
INSTALLOC=*"name"*

Specifies the directory where the High-Performance Analytics shared libraries are installed on the appliance. Specifying the INSTALL= option overrides the GRIDINSTALLLOC environment variable.

NODES=*n*
NNODES=*n*

Specifies the number of nodes in the distributed computing environment, provided that the data is not processed alongside the database. Specify **`NODES=0`** to indicate that you want to process the data in SMP mode on the client machine. If the input data is not alongside the database, this is the default setting. The HPBIN procedure then performs multithreaded analysis on the client.

If the data is not read alongside the database, the NODES= option specifies the number of nodes on the appliance that are involved in the analysis.

If the number of nodes can be modified by the application, you can specify a NODES= option where *n* exceeds the number of physical nodes on the appliance.

The High-Performance Analytics software then over-subscribes the nodes and associates nodes with multiple units of work. For example, on a system with 16 appliance nodes, the following statement would over-subscribe the system by a factor of 3:

```
performance nodes=48 host="hpa.sas.com";
```

Generally, it is not advisable to over-subscribe the system because the analytic code is optimized for a certain level of multithreading on the nodes that depend on the CPU count.

If the data is 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.

NTHREADS=*n*

Specifies the number of threads used for analytic computations and overrides the SAS system option THREADS | NOTHREADS. If you do not specify the NTHREADS= option, then the number of threads is determined based on the number of CPUs on the host machine where the analytic computations execute.

By default, High-Performance Analytics procedures execute in multiple concurrent threads, unless you disable this behavior with the NOTHREADS system option or you specify NTHREADS=1 to force single-threaded execution. The value specified here must not exceed 256.

Note: The SAS system option THREADS | NOTHREADS applies to the current machine where the SAS High-Performance Analytics procedures execute. This option does not apply to the compute nodes in a distributed environment.

VAR Statement

The VAR statement identifies which variables the HPBIN procedure will bin. PROC HPBIN does not support duplicated variables, and the VAR statement terminates with an error if duplicated variables exist.

Requirement: The variables specified here must be interval variables. PROC HPBIN terminates with an error if class variables are specified

Syntax

VAR *variables-list*;

Details

Required Arguments

variables-list

Lists the variables that the HPBIN procedure will bin. You cannot specify the same variable more than once, or PROC HPBIN terminates in an error. If you specify a variable in the VAR statement, then you cannot specify that variable in either the ID or the FREQ statement.

Examples: HPBIN Procedure

Example 1: Bucket Binning

Create the Data Set

This example creates a sample data set and applies the bucket binning method to partition the data. First, you need to create the sample data set with the code below:

```
data ex12;
    length i 8;
    length cl1 $20;

    do i=1 to 1000;
        x1 = ranuni(1);
        x2 = 10*ranuni(2);
        x3 = 100*ranuni(3);
        x4 = 1000*ranuni(4);
        x5 = 5*ranuni(6);

        if x1 < 0.3 then do;
            cl1 = "East";
        end;
        else if x1 < 0.6 then do;
            cl1 = "West";
        end;
        else do;
            cl1 = "Middle";
        end;
        output;
    end;
run;
```

Run the HPBIN Procedure

Run the code below to bin the data set ex12.

```
proc hpbin data=ex12 output=out numbin=5 bucket;
    id x5;
    var x1-x3;
run;
```

The preceding code uses bucket binning on the variables x1, x2, and x3 to create 5 bins. Also, x5 is passed to the output data set, because it is included in the ID statement. The HPBIN Mapping Table, shown below, displays the variable assignments for each bin.

```
HPBIN Mapping Table
NAME BIN_NAME LB         UB         EM_BIN_LABEL                BIN
x1   bin_x1   .          0.203283   x1 < 0.203283               1
x1   bin_x1   0.203283   0.402403   0.203283 <= x1 < 0.402403   2
x1   bin_x1   0.402403   0.601524   0.402403 <= x1 < 0.601524   3
```

x1	bin_x1	0.601524	0.800644	0.601524 <= x1 < 0.800644	4
x1	bin_x1	0.800644	.	0.800644 <= x1	5
x2	bin_x2	.	2.009399	x2 < 2.009399	1
x2	bin_x2	2.009399	3.995784	2.009399 <= x2 < 3.995784	2
x2	bin_x2	3.995784	5.982170	3.995784 <= x2 < 5.982170	3
x2	bin_x2	5.982170	7.968556	5.982170 <= x2 < 7.968556	4
x2	bin_x2	7.968556	.	7.968556 <= x2	5
x3	bin_x3	.	19.946486	x3 < 19.946486	1
x3	bin_x3	19.946486	39.865621	19.946486 <= x3 < 39.865621	2
x3	bin_x3	39.865621	59.784756	39.865621 <= x3 < 59.784756	3
x3	bin_x3	59.784756	79.703890	59.784756 <= x3 < 79.703890	4
x3	bin_x3	79.703890	.	79.703890 <= x3	5

Note that because you specified bucket binning, each bin is the same length.

Example 2: Pseduo-Quantile Binning

Binning in Soloist Mode

This example uses the HMEQ data set, available in the SAMPSIO library. Consider the following code:

```
proc hpbin data=sampsio.hmeq pseudo_quantile;
    performance details;
    var loan mortdue;
    ods table mappingTable=binmap;
run;
```

This code bins the variables loan and mortdue into 16 bins. The Performance Information table provides information about the host, Execute mode, nodes, and threads used by the HPBIN procedure. The Binning Information table displays the method, number of bins, number of variables, and number of observations used by PROC HPBIN. Note in the following mapping table shown that these bins are not equal length.

The Binning output

Obs	NAME	BIN_NAME	LB	UB	EM_BIN_LABEL	BIN
1	LOAN	bin_LOAN	.	6339.20	LOAN < 6339.20	1
2	LOAN	bin_LOAN	6339.20	8381.60	6339.20 <= LOAN < 8381.60	2
3	LOAN	bin_LOAN	8381.60	9980.00	8381.60 <= LOAN < 9980.00	3
4	LOAN	bin_LOAN	9980.00	11134	9980.00 <= LOAN < 11134.40	4
5	LOAN	bin_LOAN	11134	12466	11134.40 <= LOAN < 12466.40	5
6	LOAN	bin_LOAN	12466	13710	12466.40 <= LOAN < 13709.60	6
7	LOAN	bin_LOAN	13710	15042	13709.60 <= LOAN < 15041.60	7
8	LOAN	bin_LOAN	15042	16374	15041.60 <= LOAN < 16373.60	8
9	LOAN	bin_LOAN	16374	17883	16373.60 <= LOAN < 17883.20	9
10	LOAN	bin_LOAN	17883	19748	17883.20 <= LOAN < 19748.00	10
11	LOAN	bin_LOAN	19748	21435	19748.00 <= LOAN < 21435.20	11
12	LOAN	bin_LOAN	21435	23389	21435.20 <= LOAN < 23388.80	12
13	LOAN	bin_LOAN	23389	25431	23388.80 <= LOAN < 25431.20	13
14	LOAN	bin_LOAN	25431	28273	25431.20 <= LOAN < 28272.80	14
15	LOAN	bin_LOAN	28273	36620	28272.80 <= LOAN < 36620.00	15
16	LOAN	bin_LOAN	36620	.	36620.00 <= LOAN	16
17	MORTDUE	bin_MORTDUE	.	21142	MORTDUE < 21142.37	1
18	MORTDUE	bin_MORTDUE	21142	31477	21142.37 <= MORTDUE < 31477.03	2
19	MORTDUE	bin_MORTDUE	31477	40222	31477.03 <= MORTDUE < 40221.75	3
20	MORTDUE	bin_MORTDUE	40222	46582	40221.75 <= MORTDUE < 46581.54	4

```
21   MORTDUE  bin_MORTDUE  46582    50954    46581.54 <= MORTDUE < 50953.90    5
22   MORTDUE  bin_MORTDUE  50954    56121    50953.90 <= MORTDUE < 56121.23    6
23   MORTDUE  bin_MORTDUE  56121    60494    56121.23 <= MORTDUE < 60493.58    7
24   MORTDUE  bin_MORTDUE  60494    65263    60493.58 <= MORTDUE < 65263.43    8
25   MORTDUE  bin_MORTDUE  65263    70828    65263.43 <= MORTDUE < 70828.25    9
26   MORTDUE  bin_MORTDUE  70828    76791    70828.25 <= MORTDUE < 76790.55    10
27   MORTDUE  bin_MORTDUE  76791    83150    76790.55 <= MORTDUE < 83150.34    11
28   MORTDUE  bin_MORTDUE  83150    91498    83150.34 <= MORTDUE < 91497.57    12
29   MORTDUE  bin_MORTDUE  91498    101435   91497.57 <= MORTDUE < 101434.75   13
30   MORTDUE  bin_MORTDUE  101435   120117   101434.75 <= MORTDUE < 120116.63  14
31   MORTDUE  bin_MORTDUE  120117   145556   120116.63 <= MORTDUE < 145555.80  15
32   MORTDUE  bin_MORTDUE  145556   .        145555.80 <= MORTDUE              16
```

Note: Some of the values in the preceding table have been truncated to two decimal places.

Binning on the Grid

For this example, you create a data set similar to that in Example 1: Bucket Binning on page 18 . However, this data set is much larger. Consider the following code:

```
data ex12;
    length i 8;
    length cl1 $10;

    do i=1 to 1000000;
        x1 = ranuni(1);
        x2 = 10*ranuni(2);
        x3 = 100*ranuni(3);

        if x1 < 0.3 then do;
            cl1 = "East";
        end;
        else if x1 < 0.6 then do;
            cl1 = "West";
        end;
        else do;
            cl1 = "Middle";
        end;
        output;
    end;
run;
```

Note that this data set has 1,000,000 observations, compared to the 1,000 observations in the previous example. Next, this data is binned on a grid with 100 nodes, each having 8 processors.

Note: You must replace **<yourGridHostName>** and **<yourGridInstallLocation>** with your specific grid host name and installation location, respectively.

```
option set=GRIDHOST="<yourGridHostName>";
option set=GRIDINSTALLLOC="<yourGridInstallLocation>";

ods output hpbininfo=bininfo;
ods output mappingTable=mapTable;
ods output performanceinfo=perfTable;
ods listing close;
```

```
proc hpbin data=ex12 output=out numbin=10 pseudo_quantile ;
    var x1-x3;
    performance nodes=100 nthreads=8;
run;
ods listing;

proc print data=perfTable noobs;
    title "The Performance Information";
run;

proc print data=bininfo noobs;
    title "The Binning information";
run;

proc print data=mapTable noobs;
    title "The mapping table";
run;

proc print data=out(obs=10) noobs;
    title "The Binning output";
run;
```

The Performance Information, Binning Information, and Mapping Table outputs are given below.

The performance information output is as follows:

```
The Performance Information
Descr           Value
Host Node                   <yourGridHostName>
Execution Mode              Distributed
Number of Compute Nodes     100
Number of Threads per Node  8
```

The binning information output is as follows:

```
The Binning information
Descr                  Value
Method                 Pseudo-Quantile Binning
Number of Bins         10
Number of Variables    3
Observations           1000000
```

The mapping table output is as follows:

```
The mapping table
NAME  BIN_NAME  LB         UB         EM_BIN_LABEL                  BIN
x1    bin_x1    .          0.101000   x1 < 0.101000                 1
x1    bin_x1    0.101000   0.201000   0.101000 <= x1 < 0.201000     2
x1    bin_x1    0.201000   0.301000   0.201000 <= x1 < 0.301000     3
x1    bin_x1    0.301000   0.401000   0.301000 <= x1 < 0.401000     4
x1    bin_x1    0.401000   0.500000   0.401000 <= x1 < 0.500000     5
x1    bin_x1    0.500000   0.600000   0.500000 <= x1 < 0.600000     6
x1    bin_x1    0.600000   0.700000   0.600000 <= x1 < 0.700000     7
x1    bin_x1    0.700000   0.800000   0.700000 <= x1 < 0.800000     8
x1    bin_x1    0.800000   0.900000   0.800000 <= x1 < 0.900000     9
x1    bin_x1    0.900000   .          0.900000 <= x1                10
x2    bin_x2    .          1.010000   x2 < 1.010000                 1
x2    bin_x2    1.010000   2.000000   1.010000 <= x2 < 2.000000     2
```

x2	bin_x2	2.000000	3.000000	2.000000 <= x2 < 3.000000	3
x2	bin_x2	3.000000	3.999999	3.000000 <= x2 < 3.999999	4
x2	bin_x2	3.999999	4.999999	3.999999 <= x2 < 4.999999	5
x2	bin_x2	4.999999	5.999999	4.999999 <= x2 < 5.999999	6
x2	bin_x2	5.999999	7.009998	5.999999 <= x2 < 7.009998	7
x2	bin_x2	7.009998	8.009998	7.009998 <= x2 < 8.009998	8
x2	bin_x2	8.009998	8.999997	8.009998 <= x2 < 8.999997	9
x2	bin_x2	8.999997	.	8.999997 <= x2	10
x3	bin_x3	.	10.100087	x3 < 10.100087	1
x3	bin_x3	10.100087	20.000066	10.100087 <= x3 < 20.000066	2
x3	bin_x3	20.000066	30.000045	20.000066 <= x3 < 30.000045	3
x3	bin_x3	30.000045	40.000024	30.000045 <= x3 < 40.000024	4
x3	bin_x3	40.000024	50.100003	40.000024 <= x3 < 50.100003	5
x3	bin_x3	50.100003	59.999982	50.100003 <= x3 < 59.999982	6
x3	bin_x3	59.999982	69.999961	59.999982 <= x3 < 69.999961	7
x3	bin_x3	69.999961	80.099940	69.999961 <= x3 < 80.099940	8
x3	bin_x3	80.099940	89.999919	80.099940 <= x3 < 89.999919	9
x3	bin_x3	89.999919	.	89.999919 <= x3	10

Chapter 3

HPDECIDE Procedure

Overview: HPDECIDE Procedure

The HPDECIDE procedure creates optimal decisions that are based on a user-supplied decision matrix, prior probabilities, and 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.

Syntax: HPDECIDE Procedure

```
PROC HPDECIDE DATA=SAS-data-set <options>;
   DECISION DECDATA=SAS-data-set <options>;
   FREQ variable;
   ID variables-list;
   PERFORMANCE <performance-options>;
   POSTERIORS variables-list;
   PREDICTED variable;
   TARGET variable;
RUN;
```

PROC HPDECIDE Statement

The PROC HPDECIDE statement invokes the procedure.

Details

Required Arguments

DATA=*SAS-data-set*

Specifies the input data set, which is the output data set from a modeling procedure.

Note: Strictly speaking, this argument is not required. If you omit this argument, then the HPDECIDE procedure uses the most recently created data set.

Optional Arguments

OUT=*data-set-name*

Specifies the output data set, which contains the following information:

- any variables from the input data set that are specified in the ID statement
- the chosen decision with a prefix of "D_"
- 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 computed from the target value with a prefix of either "CL_" or "CP_"
- the consequence of the best possible decision knowing the target value with a prefix of either "BL_" or "BP_"

Also, if the PRIORVAR= and OLDPRIORVAR= variables are specified, then this data set will contain the recalculated posterior probabilities. The default name for this data set is **`data_n`**, where n is the smallest integer not already used to name a data set.

OUTFIT=*data-set-name*

Specifies an output data set that contains fit statistics. These statistics include the total and average profit or loss. You cannot specify this option with a data set of type SCORE. By default, this data set is not created.

ROLE=*TRAIN* | *VALID* | *VALIDATION* | *TEST* | *SCORE*

Specifies the role of the data set. This option affects the variables that are created in the OUTFIT= data set. The default value is TEST.

DECISION Statement

The DECISION statement specifies the decision matrix, prior probabilities, or both.

Syntax

DECISION DECDATA=*SAS-data-set* <*options*>;

Details

Required Argument

DECDATA=*SAS-data-set*

Specifies the input data set that contains the decision matrix, the prior probabilities, or both. This data set might contain decision variables that are specified with the DECVARS= option. Also, it might contain prior probability variables that are specified with the PRIORVAR= option, the OLDPRIORVAR= option, or both.

This data set must contain the target variable, which is specified in the TARGET statement.

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 twice or more in this data set, an error message is printed and the procedure terminates. Any class value in the input data set that is not found in this data set is treated as a missing class value. Note that the classes in this data set must correspond exactly with 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 interpolating 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 the data set. This enables you to specify discontinuities in the data. The end points, which are the minimum and maximum data points, cannot appear more than once. No target value is allowed to appear more than twice. If the target values are not monotonic, then they are sorted by the procedure and are not allowed to appear more than once.

TIP The DECDATA= data set can be of type LOSS, PROFIT, or REVENUE. PROFIT is assumed by default. TYPE is a data set option that is specified in parentheses after the data set name when the data set is created or used.

Optional Arguments

DECVARS=*list-of-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 one of the following:

- numeric constants that give the cost of a decision
- numeric variables in the input data set that contain case-specific costs
- any combination of constants and variables

There must be the same number of cost constants and variables as there are decision variables in the DECVARS= option. In this option, you cannot use abbreviated variable lists. For any case where a cost variable is missing, the results for that case are considered missing. By default, all costs are assumed to be zero. Furthermore, this option can be used only when the DECDATA= data set is of type REVENUE.

PRIORVAR=*variable*
This option specifies the numeric 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 one. But, if they do not sum to one, then they are scaled by some constant so that they do sum to one. If this option is not specified, 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 PRIORVAR=.

FREQ Statement

The variable in the FREQ statement identifies a numeric variable in the input data set that contains the frequency of occurrence for each observation.

Syntax

FREQ *variable*;

Details

Required Argument

variable
Specify the variable in the input data set that contains the frequency for each observation. The HPDECIDE procedure treats each observation as if it appeared *f* times, where *f* is the value of the frequency variable for that observation. If the frequency value is not an integer, then the fractional part is not truncated. If the frequency value is less than or equal to 0, then the observation does not contribute to the summary statistics. However, all of the variables in the OUT= data set are processed as if the frequency variable is positive.

The frequency variable has no effect on decisions of the adjustment for prior probabilities. It affects only the summary statistics in the OUTFIT= data set.

ID Statement

The ID statement lists one or more variables from the input data set that are transferred to the output data sets. For more information about the ID statement, see “Shared Concepts and Topics” in the *SAS High-Performance Analytics User’s Guide*.

Syntax

ID *variables-list*;

Details

Required Argument

variables-list
Specifies the variables that you want to transfer from the input data set to the output data sets, provided that the output data set produces at least one record per input observation.

PERFORMANCE Statement

The PERFORMANCE statement defines performance parameters for multi-threaded and distributed computing, communicates variable information about the distributed computing environment, and requests detailed results about the performance characteristics of the HPDECIDE procedure. With the PERFORMANCE statement, you can control whether the HPDECIDE procedure executes in symmetric multiprocessing mode or massively parallel mode. For more information about the PERFROMANCE statement, see “Shared Concepts and Topics” in the *SAS High-Performance Analytics User’s Guide*.

Syntax

PERFORMANCE <*performance-options*>;

Details

Optional Arguments

COMMIT=*n*
Specifies the minimum number of observations transferred from the client to the appliance necessary to update the SAS log. For example, if you specify **`COMMIT=5000`**, then every time the number of observations sent exceeds an integer multiple of 5000, a log message is produced. This message indicates the actual number of observations distributed, not the COMMIT= value that triggered the message.

CPUCOUNT= ACTUAL | *number*
Specifies how many processors that PROC HPDECIDE assumes are available on each host in the computing environment. Valid values for *number* are integers between 1 and 256, inclusive. Setting CPUCOUNT= to a value greater than the actual number of available CPUs might result in reduced performance.

Specify **`CPUCOUNT=ACTUAL`** to set CPUCOUNT= to the number of processors physically available. This number can be less than the physical number of CPUs if the SAS process has been restricted by system administration tools.

This option overrides the CPUCOUNT= SAS system option.

If PROC HPDECIDE executes in SMP mode, then this option refers to the client machine of the SAS session. If PROC HPDECIDE executes in MPP mode, then this option applies the nodes on the appliance.

DATASERVER=*“name”*
Specifies the server name on Teradata systems as defined through the *hosts* file and as used in the LIBNAME statement for Teradata. For example, if the *hosts* file defines **`myservercop1 33.44.55.66`** as the server for Teradata, then a LIBNAME statement would be as follows:

```
libname TDLIB terdata server=myserver user= password= database=;
```

To induce PROC HPDECIDE to run alongside the Teradata server, specify the following performance statement:

```
performance dataserver="myserver";
```

DETAILS
Requests a table that shows a timing breakdown of the procedure steps.

TIMEOUT=*s*
Specifies the length of time, in seconds, that PROC HPDECIDE should wait for a connection to the appliance and to establish a connection back to the client. The default value for *s* 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.

HOST=*"name"*
GRIDHOST=*"name"*
Specifies the name of the appliance host. The HOST= option overrides the value of the GRIDHOST environment variable.

INSTALL=*"name"*
INSTALLOC=*"name"*
Specifies the directory where the High-Performance Analytics shared libraries are installed on the appliance. Specifying the INSTALL= option overrides the GRIDINSTALLLOC environment variable.

NODES=*n*
NNODES=*n*
Specifies the number of nodes in the distributed computing environment, provided that the data is not processed alongside the database. Specify **NODES=0** to indicate that you want to process the data in SMP mode on the client machine. If the input data is not alongside the database, this is the default setting. The HPDECIDE procedure then performs multithreaded analysis on the client.

If the data is not read alongside the database, the NODES= option specifies the number of nodes on the appliance that are involved in the analysis.

If the number of nodes can be modified by the application, you can specify a NODES= option where *n* exceeds the number of physical nodes on the appliance. The High-Performance Analytics software then over-subscribes the nodes and associates nodes with multiple units of work. For example, on a system with 16 appliance nodes, the following statement would over-subscribe the system by a factor of 3:

```
performance nodes=48 host="hpa.sas.com";
```

Generally, it is not advisable to over-subscribe the system because the analytic code is optimized for a certain level of multithreading on the nodes that depend on the CPU count.

If the data is 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.

NTHREADS=*n*
Specifies the number of threads used for analytic computations and overrides the SAS system option THREADS | NOTHREADS. If you do not specify the NTHREADS= option, then the number of threads is determined based on the number of CPUs on the host machine where the analytic computations execute.

By default, High-Performance Analytics procedures execute in multiple concurrent threads, unless you disable this behavior with the NOTHREADS system option, or you specify NTHREADS=1 to force single-threaded execution. The value specified here must not exceed 256.

Note: The SAS system option THREADS | NOTHREADS applies to the current machine where the SAS High-Performance Analytics procedures execute. This option does not apply to the compute nodes in a distributed environment.

POSTERIORS Statement

The POSTERIORS statement can be specified only with a categorical target variable. You cannot use both the POSTERIORS statement and the PREDICTED statement.

Syntax

POSTERIORS *variables-list*;

Details

Required Argument

variables-list

Specify 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 a few certain conditions are met, then a case is set to missing and the variable _WARN_ contains the flag **P**.

These conditions are as follows:

- 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.

Note that the order of the variables in this list must correspond exactly to the order of the classes in the DECDATA= data set.

PREDICTED Statement

The PREDICTED statement can be specified only with an interval target variable. You cannot use both the POSTERIORS statement and the PREDICTED statement.

Syntax

PREDICTED *variable*;

Details

Required Argument

variable

Specifies the numeric variable in the input data set that contains the predicted values of an interval target variable.

TARGET Statement

The TARGET statement specifies which variable in the DECDATA= data set is the target variable. The HPDECIDE procedure searches for a target variable with 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= and DECDATA= data sets do not need to be the same type. This is 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 will format the target variable with the format and length from the DATA= data set.

Syntax

TARGET *variable*;

Details

Required Argument

variable

The variable specified here is the target variable and must be in the DECDATA= data set.

Example: The HPDECIDE Procedure

Preprocessing the Data and Basic Usage

This extended example creates a fictitious scenario to illustrate how to adjust prior probabilities and make decisions with a revenue matrix and cost constants. This example considers a population of men who consult urologists for prostate problems. In this population, 70% of the men 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 made on each patient. The training data set also includes the diagnosis made by reliable, conventional methods.

For each patient, three treatments are available. First, the urologist could prescribe antibiotics, which are effective against infection, but might have moderately bad side effects. Antibiotics have no effect on benign enlargement or cancer. Second, the urologist could recommend surgery, which is effective for all diseases, but has potentially severe side effects, such as impotence. Finally, the urologist and patient could decide against both antibiotics and surgery, thereby doing nothing.

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 of your results will be identical to those presented in this example.

```
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;
    stop;
generate:
    do i=1 to n;
    x=rannor(12345)*sx+mx;
    y=rannor(0) *sy+my;
    output;
    end;
run;
```

This code creates the Prostate data set. The first 70 observations represent benign tumors, the next 20 represent infections, and the final 5 are cancer. To visualize the measurements of X and Y, you can plot the data with the GPLOT procedure.

```
title2 'Diagnosis';
proc gplot data=prostate;
    plot y*x=dx;
run;
```

When you plot the data, you should be able to see fairly distinct groups of data points. There can be some overlap between groups, but most of the observations for each diagnosis are tightly grouped. You can also use the DISCRIM procedure to see how well variables X and Y classify each patient.

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

The DISCRIM procedure assumes that all prior probabilities are equal, which is 1/3 for this example. As the Output window indicates, the DISCRIM procedure misidentifies some of the benign tumors as cancer or infections. Also, it misidentifies some of the infections as benign tumors. Therefore, you want to create a data set that contains prior probabilities and revenue information. The revenue information indicates the benefit of each treatment. The costs of each treatment, such as bad side effects, are specified late in a DECISION statement. The revenue matrix is given by the code that follows.

```
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 while prior uses information that is known from the sample data set. The other variables define the revenue of each treatment option. The revenue, or 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 since this is not necessary, it has very little benefit. Surgery completely removes an infection, so it has the same value as antibiotics. Finally, surgery can remove a cancerous tumor, which is an immense benefit to the patient.

You can now use the HPDECIDE procedure to 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.

```
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;
```

The data set decOut indicates that only one benign tumor was misidentified, but a similar number of infections were misidentified as benign, when compared with the DISCRIM procedure. All of 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.

Due to the personal nature of medical decisions, the costs 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 due to the patients' insurance plans. For illustrative purposes, assume a higher cost for surgery and leave the other costs constant.

```
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 50;
run;
```

Notice that the misclassified benign tumor was now correctly classified. However, one of the cancer cases was identified as benign, which is a costly mistake. Notice, in decOut, that the total profit has been reduced from 470 to 285.

Chapter 4

HPIMP Procedure

Overview: HPIMP Procedure

The HPIMP procedure executes high-performance variable imputation. You can specify multiple INPUT and IMPUTE statements, as is shown in the example in this chapter. Any class variables that are referenced by the IMPUTE statement are ignored.

Syntax: HPIMP Procedure

Requirement: At least one INPUT and one IMPUTE statement are required.

PROC HPIMP DATA=<*libref.*>*SAS-data-set* OUT=<*libref.*>*SAS-data-set* <*options*>;
CODE <*options*>
ID *variables-list*;
IMPUTE *variables-list* / <*options*>;
INPUT *variables-list* / <*options*>;
PERFORMANCE <*performance-options*>;

PROC HPIMP Statement

The PROC HPIMP statement invokes the procedure.

Note: WHERE processing is supported in the DATA= and OUT= arguments.

Syntax

PROC HPIMP DATA=<*libref.*>*SAS-data-set* OUT=<*libref.*>*SAS-data-set* <*options*>;

Details

Required Arguments

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

Specifies the input data set that contains the variables to be imputed. The default data set is the most recently created data set. If the data is already distributed, then the procedure reads the data alongside the distributed database.

DMDBCAT=<*libref.*>*SAS-catalog*

Names the SAS catalog that contains the variable metadata. This catalog must exist on the client machine.

Optional Arguments

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

Specifies the output data set that contains the imputed variables. This data set contains the ID variables (if applicable), the imputation indicator variables, and the imputed variables. If the data is already distributed, then the procedure writes the data alongside the distributed database with the ID variables, indicator variables, and imputed variables.

CODE Statement

The CODE statement generates SAS DATA step code that mimics the computations done by the IMPUTE statement.

Syntax

CODE <*options*>;

Details

Optional Arguments

FILE=*file-name*

Specifies the filename that contains the SAS score code.

ID Statement

The ID statement lists one or more variables from the input data set that are transferred to the output data sets. The ID statement accepts numeric and character variables. For example, when an OUTPUT statement is used, the ID variables, followed by the indicator variables and the imputed variables, are added to the output data set. For more information about the ID statement, see “Shared Concepts and Topics” in the *SAS High-Performance Analytics User’s Guide*.

Default: By default, the HPIMP procedure does not include all variables from the input data set in the output data sets.

Requirement: The variables in an ID statement must not appear in any INPUT statement. Otherwise, an error is reported.

Syntax

ID *variables-list*;

Details

Required Arguments

variables-list
Specifies the variables that you want to transfer from the input data set to the output data sets.

IMPUTE Statement

The IMPUTE statement names the variables for PROC HPIMP to impute. You can specify multiple IMPUTE statements

Requirements: The IMPUTE statement accepts only numeric variables that have appeared in an INPUT statement. Class variables are ignored, but specifying a character variable results in an error.

You must specify one of the options METHOD= or VALUE=.

Syntax

IMPUTE *variables-list* / <*options*>;

Details

Required Argument

variables-list
Contains a list of variables to be imputed.

Optional Arguments

METHOD= MEAN | RANDOM
Specifies the method of imputation.

If you specify MEAN, then missing values for each variable are replaced with the algebraic mean of that variable. The mean is obtained from the DMDB catalog. If there is no nonmissing value, the mean is set to 0.

If you specify RANDOM, then missing values for each variable are replaced with a random value between the minimum and maximum value for that variable. The minimum and maximum are obtained from the DMDB catalog.

VALUE=*value*
Replaces missing values with the value specified by the user.

INPUT Statement

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

Note: If you specify any LEVEL= options other than LEVEL=INTERVAL, then the variables are ignored by the IMPUTE statement.

Syntax

INPUT *variables-list* / <*options*>;

Details

Required Argument

variables-list
Contains a list of variables that share common features.

Optional Arguments

LEVEL=*level*
Specifies the level of measurement of the variables. Valid values are BINARY, NOMINAL, ORDINAL, and INTERVAL. The default value is LEVEL=INTERVAL.

ORDER=*order*
Specifies the sorting order for the values of an ordinal input variable. Valid values are given in the table below.

Value of ORDER=	Variable Values Sorted By
ASCENDING	ascending order of unformatted values
ASCFORMATTED	ascending order of formatted values
DESCENDING	descending order of unformatted values
DESFORMATTED	descending order of formatted values
DSORDER	order of appearance in the input data set

PERFORMANCE Statement

The PERFORMANCE statement defines performance parameters for multi-threaded and distributed computing, communicates variable information about the distributed computing environment, and requests detailed results about the performance characteristics of the HPIMP procedure. With the PERFORMANCE statement, you can control whether the HPIMP procedure executes in symmetric multiprocessing or massively parallel mode. For more information about the PERFORMANCE statement, see “Shared Concepts and Topics” in the *SAS High-Performance Analytics User’s Guide*.

Syntax

PERFORMANCE <*performance-options*>;

Details

Optional Arguments

COMMIT=*n*
Specifies the minimum number of observations transferred from the client to the appliance necessary to update the SAS log. For example, if you specify **`COMMIT=5000`**, then every time the number of observations sent exceeds an integer multiple of 5000, a log message is produced. This message indicates the actual number of observations distributed, not the COMMIT= value that triggered the message.

CPUCOUNT= ACTUAL | *number*
Specifies how many processors PROC HPIMP assumes are available on each host in the computing environment. Valid values for *number* are integers between 1 and 256, inclusive. Setting CPUCOUNT= to a value greater than the actual number of available CPUs can result in reduced performance.

Specify **`CPUCOUNT=ACTUAL`** to set CPUCOUNT= to the number of processors physically available. This number can be less than the physical number of CPUs if the SAS process has been restricted by system administration tools.

This option overrides the CPUCOUNT= SAS system option.

If PROC HPIMP executes in SMP mode, then this option refers to the client machine of the SAS session. If PROC HPIMP executes in MPP mode, then this option applies the nodes on the appliance.

DATASERVER=*"name"*
Specifies the server name on Teradata systems as defined through the *hosts* file and as used in the LIBNAME statement for Teradata. For example, if the *hosts* file defines **`myservercop1 33.44.55.66`** as the server for Teradata, then a LIBNAME statement would be as follows:

```
libname TDLIB terdata server=myserver user= password= database= ;
```

To induce PROC HPIMP to run alongside the Teradata server, specify the following performance statement:

```
performance dataserver="myserver";
```

DETAILS
Requests a table that shows a timing breakdown of the procedure steps.

TIMEOUT=*s*
Specifies the length of time, in seconds, that PROC HPIMP should wait for a connection to the appliance and to establish a connection back to the client. The default value for *s* 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.

HOST=*"name"*
GRIDHOST=*"name"*
Specifies the name of the appliance host. The HOST= option overrides the value of the GRIDHOST environment variable.

INSTALL=*"name"*
INSTALLOC=*"name"*
Specifies the directory where the High-Performance Analytics shared libraries are installed on the appliance. Specifying the INSTALL= option overrides the GRIDINSTALLLOC environment variable.

NODES=*n*
NNODES=*n*
Specifies the number of nodes in the distributed computing environment, provided that the data is not processed alongside the database. Specify **`NODES=0`** to indicate that you want to process the data in SMP mode on the client machine. If the input data is not alongside the database, this is the default setting. The HPIMP procedure then performs multithreaded analysis on the client.

If the data is not read alongside the database, the NODES= option specifies the number of nodes on the appliance that are involved in the analysis.

If the number of nodes can be modified by the application, you can specify a NODES= option where *n* exceeds the number of physical nodes on the appliance. The High-Performance Analytics software then over-subscribes the nodes and associates nodes with multiple units of work. For example, on a system with 16 appliance nodes, the following statement would over-subscribe the system by a factor of 3:

```
performance nodes=48 host="hpa.sas.com";
```

Generally, it is not advisable to over-subscribe the system because the analytic code is optimized for a certain level of multithreading on the nodes that depend on the CPU count.

If the data is 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.

NTHREADS=*n*
Specifies the number of threads used for analytic computations and overrides the SAS system option THREADS | NOTHREADS. If you do not specify the NTHREADS= option, then the number of threads is determined based on the number of CPUs on the host machine where the analytic computations execute.

By default, High-Performance Analytics procedures execute in multiple concurrent threads, unless you disable this behavior with the NOTHREADS system option or you specify NTHREADS=1 to force single-threaded execution. The value specified here must not exceed 256.

Note: The SAS system option THREADS | NOTHREADS applies to the current machine where the SAS High-Performance Analytics procedures execute. This option does not apply to the compute nodes in a distributed environment.

Example: The HPIMP Procedure

Imputing a Data Set

In this example, you use all three imputation methods available in the IMPUTE statement to manipulate a data set. First, you create the data set with the SAS DATA

step provided below. This data set has four variables, the first being an index variable. The next three variables all have some missing values.

```
data dsopts;
  input ind x y z;
  cards;
1  0.18496   0.97009   0.8496
2  0.39982   .         0.9982
3  0.92160   .         0.2160
4  .         0.53169   .
5  0.04979   0.06657   0.4979
6  0.81932   0.52387   .
7  .         0.06718   .
8  0.95702   0.29719   .
9  0.27261   0.68993   .
10 0.97676   .         .
;
```

Next, you need to run the following code to create the DMDB catalog for your data set:

```
proc dmdb data=dsopts cat=cat;
    var ind x y z;
run;
```

Now, you are ready to run the HPIMP procedure.

```
proc hpimp data=dsopts(where=(y>0.1))
    dmdbcat=cat out=impout(where=(m_y=0));
    input x y z;
    impute x/method=mean;
    impute y/method=random;
    impute z/value=0.888;
    code file="c:\imp.sas";
    id ind;
run;
```

In this call to PROC HPIMP, the variable x is imputed with the method MEAN, y is imputed with the method RANDOM, and z is imputed with a specific value.

The output data set IMPOUT contains seven variables. Note that the two WHERE clauses keep only the rows where y is both nonmissing and greater than 0.1. The variables m_x, m_y, and m_z are the indicator variables and display a 0 if that observation was not imputed and a 1 if it was. The variables imp_x, imp_y, and imp_z contain the imputed variable values.

Part 2

High-Performance Macros

Chapter 5

The %EM_new_assess Macro

Overview

The %EM_new_assess macro computes model assessment measures that are used to evaluate the performance of predictive models and to inform decisions based on the results of the models. For example, the %EM_new_assess macro can help you to determine the following:

- the expected response rate in each of the top 5%, 10%, 15%, and 20% subsets of your campaign
- the probability cutoff that you should use to select customers that areexpected to respond at least 12% of the time.

Model assessment measures are typically referred to as rank order measures. The term rank order measure refers to the fact that these statistics are computed by a descending ranking of the probabilities of a predicted event.

To compute all model assessment measures, %EM_new_assess macro uses a mixture of DATA step code, Base SAS procedures, SAS Enterprise Miner Procedures, and SAS High Performance Analytics procedures. The event and probability for assessment do

not depend on the sort order of the target variable and can be selected independently. You can select any event for assessment.

Syntax

The %EM_new_assess Macro

%macro EM_new_assess (
DATA=*SAS-data-set*
LEVEL=*CLASS* | *INTERVAL*
TARGET=*variable-name*
VAR=*variable-name*
<BINS=*number*>
<BINSTATS=*SAS-data-set*>
<CUTOFF=*number*>
<EVENT=*event-string*>
<EXPAND=*SAS-data-set*>
<FUZZ=*number*>
<HPDS2=*0* | *1*>
<MAX=*number*>
<MIN=*number*>
<OUT=*SAS-data-set*>

Required Arguments

DATA=*SAS-data-set*
Specifies the input data set. The model assessment measures are generated for this data set.

LEVEL=*CLASS* | *INTERVAL*
Specifies the measurement level of the target variable.

TARGET=*variable-name*
Specifies the target variable.

VAR=*variable-name*
Specifies the numeric variable that is used for ranking.

Optional Arguments

BINS=*number*
Specifies the number of bins that are created in the OUT= data set.

BINSTATS=*SAS-data-set*
Specifies an output data set that contains the overall binning statistics.

CUTOFF=*number*
Specifies the cutoff value for new classification. That is, this value specifies the minimum probability value of a bin.

EVENT=*event-string*
Specifies the event string.

EXPAND=*SAS-data-set*
Specifies an output data set that is similar in structure to the scored data. This data set contains columns for target, probability, and classification, and can be used as input to the Model Import node. Note that computations on this data set are approximations and do not equal computations on the full data set. You cannot specify this argument for an interval target variable.

FUZZ=*0 | 1*
Specify *1* to enable FUZZ tie-breaking. FUZZ tie-breaking adds a random number that is generated by an appropriately scaled random normal distribution to the probability in order to create several microbins. This is equivalent to applying a kernel function. In the event of sparse or skewed distributions, this method can still produce bins that contain too many or too few observations.

HPDS2=*0 | 1*
Specify *1* to use PROC HPDS2 to build the microbins.

MAX=*number*

MIN=*number*

OUT=*SAS-data-set*
Specifies the output data set for the binned data.

Details

The OUT= Data Set

The primary output data set is a summary data set that contains one row for each bin. The columns generated depend on the measurement level of the target variable. Cumulative measures are calculated by descending probability. Each bin should have an approximately equal number of observations. The mean probability, event count, and non-event count are stored in each bin.

Note: The tables below do not detail every variable in the output data set. Variables with common or obvious definitions are omitted from these tables.

Table 5.1 *Output Data Set Variables for a Class Target Variable*

Variable	Details
cutoff	minimum probability value of a bin
count	number of nonmissing predicted value observations within a bin
c_count	cumulative number of observations
events	number of events within a bin
c_events	cumulative number of events

Variable	Details
depth	100*(c_count / total_count)
lift	events / ((count*total_events) / total_count)
c_lift	c_events / ((c_count*total_events) / total_count)
correct_rate	100*(c_events + total_events – c_nonevents) / total_count
error_rate	100 – correct_rate
separation	correct_rate – error_rate
sensitivity	c_events / total_events — Also called the true positive rate
specificity	1 – c_nonevents / total_non_events — Also called the true negative rate
one_minus_specificity	1 – specificity

Table 5.2 *Output Data Set Variables for an Interval Target Variable*

Variable	Details
depth	100*(c_count / total_count)
count	number of nonmissing target value observations within a bin
c_count	cumulative number of observations
predicted_count	number of nonmissing predicted observations within a bin
c_predicted_count	cumulative predicted count
target_mean	target variable mean within a bin
target_min	target variable minimum within a bin
target_max	target variable maximum within a bin
predicted_mean	predicted variable mean within a bin
predicted_min	predicted variable minimum within a bin
predicted_max	predicted variable maximum within a bin
residual_mean	residual mean within a bin

Variable	Details
residual_min	residual minimum within a bin
residual_max	residual maximum within a bin
residual_squared_mean	mean of the squared residuals within a bin

The EXPAND= Data Set

This data set is similar in structure to the scored data. This data set contains columns for target, probability, and classification, and can be used as input to the Model Import node. Note that computations on this data set are approximations and do not equal computations on the full data set. This data set is not created for interval target variables.

Variable	Type	Details
target	$	target value, either **`event`** or **`non-event`**
predict	$	predicted value, either **`event`** or **`non-event`**
prob	N	event probability. This is the mean bin probability.
freq	N	frequency. This is the bin count. This variable is necessary for valid results.

The BINSTATS= Data Set

This data set contains a summary of the binning process. It includes point measures of the Kolmogorov-Smirnov statistic, the maximum classification rate, and the depth and probability for each of those measures. These values can be used in subsequent processing. This data set contains one observation.

Table 5.3 *Class Target Variable Summary Information*

Variable	Details
Target	target variable name
Level	target variable measurement level
Var	probability of event variable name. This was used to bin the data.
NBINS	number of bins created

Variable	Details
NOBS	total number of observations. This should equal PredCount + PredMiss.
TargetCount	number of nonmissing target values
TargetMiss	number of missing target values
PredCount	number of nonmissing predicted values
PredMiss	number of missing predicted values
Event	target event value
EventCount	number of events with nonmissing predicted values
NonCount	number of non-events with nonmissing predicted values
EventMiss	number of events with missing predicted values
KSR	the maximum separation between the percentage of captured events and non-events. This value should equal 100*KS.
KS	The Kolmogorov-Smirnov (KS) measure, computed as the maximum difference between sensitivity and one_minus_specificity. Higher numbers indicate better overall confidence in the classification of events and non-events.
KSDEPTH	the sample depth where KS is computed
KSCUT	the predicted event probability where KS is computed
KSREF	the value of one_minus_specificity for the KS reference value
CR	the maximum overall correct classification rate
CRDEPTH	the sample depth where CR is computed
CRCUT	the predicted event probability where CR is computed
MDEPTH	the sample depth that identifies half of the predicted events
MCUT	the cutoff that identifies half of the predicted events

Table 5.4 *Interval Target Variable Summary Information*

Variable	Details
target	target variable name that was used to bin the data
level	target variable measurement level
var	predicted value variable name
NBINS	number of bins created. It is expected to be approximately equal to volume.
NOBS	total number of observation. It is expected to equal TargetCount + TargetMiss
TargetCount	number of nonmissing target values
TargetMiss	number of missing target values
PredCount	number of nonmissing predicted values
PredMiss	number of missing predicted values

MAP-REDUCE

In a distributed grid environment, sets of data rows are located across one or more server units. The high-performance model assessment functions follow the MAP-REDUCE paradigm for distributed grid processing. In this section, MAP-REDUCE is explained briefly.

In the MAP phase, a single variable, such as probability of event, is chosen for ranking. A frequency table is constructed in an initial pass.The initial pass also returns the maximum and minimum values of the ranking variable. A second pass maps the predictions and event counts to equal-width bins. Bins with zero observations are dropped. For a class target variable, the binned data set has columns for count, event count, and minimum probability. This data is called *microbins*.

Next begins the REDUCE phase. On the SAS side, the microbins are sorted by descending probability. The next step merges bins together until the cumulative count meets or exceeds the next bin's threshold. The result is a set of bins that contain an equal number of observations. These bins are processed to produce measures that are used for reporting for each bin, such as percents, separation, lift, captured response, sensitivity, specificity, and classification rates.

Tie-Breaking

A key point in ranking the probabilities is breaking ties when a small number of probability values have very large frequencies. This macro has two methods to handle ties that can work together.

First, in the MAP phase the FUZZ option adds a random number that is generated by an appropriately scaled random normal distribution to the probability to spread the

distribution into several microbins. This is equivalent to applying a kernel function. In the event of very sparse and skewed probability distributions, this method can still have the effect of producing percentile bins with too many and too few counts.

Second, in the REDUCE phase the code slices microbins to produce final bins with exactly the same numbers of observations. The final bin at the low end of the probability scale might contain a different number of observations to make the final count correct. This is different than in SAS Enterprise Miner 7.1, where the overage is spread among multiple bins at the high end of the probability scale. This difference accounts for some differences in the values of captured response and lift between SAS Enterprise Miner 7.1 and high-performance data mining.

Counting Rows, Events, and Missing Values

Missing values are particularly important when counting targets and predictions. Either the target or the predicted value might be independently missing in the scored data. Counting is handled differently for interval and class target models.

- Interval Targets — In this case, the actual target variable is binned, and missing target values are counted separately. Predicted values are accumulated only for nonmissing target values. The total count of all predictions is equal to the count of nonmissing target values. The values reported are TotalCount, TargetCount, TargetMiss, PredCount, PredMiss.
- Class Targets — The predicted probabilities are binned, and missing probability values are counted separately. Target events and non-events are counted for all real probabilities. In addition, target events are counted for missing probabilities. The values reported are TotalCount, PredCount, PredMiss, TargetCount, TargetMiss, EventCount, and EventMiss. TargetCount includes any nonmissing target value. EventCount includes only the event specified. All other target values are considered non-events with the exception of missing values. Target values are counted as the DMNORM of the formatted value.

Integration with SAS Enterprise Miner

Model assessment within the SAS Enterprise Miner process flow diagram is accomplished in two ways:

- Project Sample — The high-performance model produces score code. The score code is applied to the project sample that is maintained by the input node. These scores are evaluated by the SAS Enterprise Miner model assessment code that runs on the SAS system. The results are displayed in the SAS Enterprise Miner model node results and the SAS Enterprise Miner Model Comparison node. The results are used to select a champion model. In this case, the model has been trained on the high-performance system, and then applied on the SAS system to produce results comparable to other work based in SAS.
- Grid Data — The high-performance procedure outputs a table of training data scores. These scores are evaluated by the %EM_new_assess macro on the grid system and displayed as additional results in the high-performance model node results. These results are similar to the results computed on the project sample, but might differ in the number of observations that are counted and any of the resulting measurements.

Integration with SAS Code

The %EM_new_assess macro can be used directly in a SAS program to evaluate scores on either a SAS system or on a high-performance appliance system. The scores can be generated by a high-performance procedure, by a SAS procedure, or by direct import of a file containing scores. The extended example at the end of this book includes several calls to various high-performance procedures and the %EM_new_assess macro.

Comparison with SAS Enterprise Miner 7.1

In most cases, computational results are similar to the results produced by SAS Enterprise Miner 7.1. However, the new algorithm is fundamentally different from the algorithm used in SAS Enterprise Miner 7.1. The new functions were tested with both simulated and real data and found to adequately preserve the distributions of targets, probabilities, and predictions. Differences can be described by one or more ofthe following conditions:

- The new algorithm creates bins in two stages and handles unlimited quantities of data. The initial set of bins treats all observations within the bin as having the same mean probability. If the distribution of probabilities or predictions is extremely sparse, such that some of these bins both contain at least a centile of data and also multiple modes of data, then resolution in the output might be lost.
- Ties and observations that do not fit into an even number of centiles are handled differently. See Tie-Breaking on page 49 for more details. This effect can produce extremely small differences in the bin counts. This difference should be less than the number of bins for any given bin.
- The distributions of interval target predictions are handled differently. The SAS Enterprise Miner 7.1 functions bin the prediction values and summarize the target values. The new algorithm bins the target values and summarizes the predictions. This is done to encourage diagnostic analysis of residuals by target values. It is also potentially more likely that there will be more missing predictions than missing target values in real data used for modeling.
- The new functions generate additional information about the number of missing values of the target and predicted values. See Counting Rows, Events, and Missing Values on page 50 for more details.

Profiling

The SAS Enterprise Miner 7.1 functions also generate mean and mode values by each centile for variables that have the Report attribute. These values are generated for high-performance models within the model nodes and the Model Compare node based on the project sample data. The new functions do not yet generate full profiles on grid data.

Example: The %EM_new_assess Macro

This example applies the %EM_new_assess macro to a neural network that is created by the HPNEURAL procedure. Before you can run the %EM_new_assess macro, you must run the HPNEURAL procedure, as shown below.

```
filename tools catalog 'sashelp.hpdm.hpdm_tools.source' ;
%include tools ;

proc hpneural data=sampsio.dmagecr ;
input AGE AMOUNT DURATION / level=int ;
input CHECKING COAPP DEPENDS EMPLOYED EXISTCR FOREIGN HISTORY HOUSING
      INSTALLP JOB MARITAL OTHER PROPERTY PURPOSE RESIDENT SAVINGS TELEPHON
      / level=nom ;
target good_bad / level=nom ;
hidden 2 ;
train ;
score out=train_scores ;
run;
```

Now, you can run the %EM_new_assess macro with the following code:

```
%em_new_assess(
    data    =train_scores,     /* input data;                        */
    level   =CLASS,            /* specify the measurement level     */
    target  =good_bad,         /* target variable;                   */
    var     =p_good_badbad,    /* numeric variable for ranking;      */
    event   =BAD,              /* event string (not variable name); */
    bins    =20,               /* final number bins;                 */
    fuzz    =0,                /* handle ties;                       */
    expand  =expand            /* output data                        */
    );
```

Chapter 6

The %HPDM_create_scorecode_bin Macro

Overview

Introduction

The %HPDM_create_scorecode_hpbin macro creates binning score code based on the output of the HPBIN procedure. Refer to the HPBIN procedure on page 9 for more information about its usage and details. Binning transformations must be included in the model score code for scoring data in test and production processes.

Details

The HPBIN procedure only bins numeric interval variables. It cannot be used for character variables. The mapping table, created by the MAPPINGTABLE= argument in PROC HPBIN, contains all the information necessary for binning. Also, a source code line is added to map missing values to bin level zero. Output variables are given the prefix bin_ and are numeric variables. For example, if the HPBIN procedure bins the variable _TEST_, then the output variable is _BIN_TEST_. The actual variable names appear in the output produced by PROC HPBIN and in the mapping table.

Based on the particulars of your data mining project, you must decide whether the original, unbinned variables are kept or dropped. The score code produced by this macro contains no DROP or KEEP statements. The score code created is simple, block DATA step code that can be included within a DATA step that contains other code blocks.

Syntax

The %HPDM_create_scorecode_bin Macro

%macro hpdm_create_scorecode_neural (
BINDATA=*SAS-data-set*
FILEREF=*file-name*

Required Arguments

BINDATA=*SAS-data-set*
Specifies the mapping table that is created by the MAPPINGTABLE= argument in PROC HPBIN.

FILEREF=*file-name*
Specifies a SAS file reference for an output code file.

Example: The %HPDM_create_scorecode_bin Macro

This example applies the %HPDM_create_scorecode_bin macro to a binned data set. The data set that is binned is the HMEQ data set from the SAMPSIO library. Before you can run the %HPDM_create_scorecode_bin macro, you need to run PROC HPBIN, as shown below.

```
/*--- load the score code creation macro ---*/
filename h catalog 'sashelp.hpdm.hpdm_create_scorecode_hpbin.source';
%include h;

/*--- run proc hpbin to create bins table --*/
proc hpbin data=sampsio.hmeq pseudo_quantile;
    performance details;
    var LOAN MORTDUE;
    ods table mappingTable=binmap;
run;
```

Next, run the %HPDM_create_scorecode_bin to create the score code for the binned data set. Also, the code below applies the score code and monitors the distribution of the bins.

```
/*--- create scorecode ---------------------*/
filename bincode catalog 'work.sample.bincode.source';
%hpdm_create_scorecode_hpbin(bindata=binmap,fileref=bincode);

/*--- apply scorecode  ---------------------*/
data hmeq_bins; set sampsio.hmeq;
    %include bincode;
run;
```

```
/*--- check distribution of bins -----------*/
proc freq data=hmeq_bins;
    table bin_loan bin_mortdue;
run;
```

Chapter 7

The %HPDM_create_scorecode_logistic Macro

Overview

Introduction

The %HPDM_create_scorecode_logistic macro creates SAS code to score the logistic model created by the HPLOGISTIC procedure.

Syntax

The %HPDM_create_scorecode_logistic Macro

%macro hpdm_create_scorecode_logistic (
DATA=*SAS-data-set*
EVENTLEVEL=*number*
MODEL=*SAS-data-set*
MODELINFO=*SAS-data-set*
NONEVENTLEVEL=*number*
FILEREF=*file-name*
<CLASSIFY=*Y* | *N*>
<IMPUTE=*Y* | *N*>
<RESIDUAL=*Y* | *N*>

Required Arguments

DATA=*SAS-data-set*
Specifies the data set that is used to create the regression model in PROC HPLOGISTIC.

EVENTLEVEL=*number*
Specifies the event level for a binary target variable.

FILEREF=*file-name*
Specifies a SAS file reference for an output code file. Failure to specify this argument results in an error.

MODEL=*SAS-data-set*
Specifies the parameter estimate data set that is generated by the HPLOGISTIC procedure.

MODELINFO=*SAS-data-set*
Specifies the model information data set output that is generated by the HPLOGISTIC procedure.

NONEVENTLEVEL=*number*
Specifies the nonevent level for a binary target variable.

Optional Arguments

CLASSIFY=*Y* | *N*
Specify *Y* to create the F_, I_, and U_ variables. Specify *N* to suppress the residual parameters. The default value is *N*.

IMPUTE*Y* | *N*
Specify *Y* to impute the predicted values. The default value is *N*.

RESIDUAL=*Y* | *N*
Specify *Y* to create residual variables. The default value is *N*.

Example: The %HPDM_create_scorecode_logistic Macro

This example applies the %HPDM_create_scorecode_logistic macro to a regression model that is created by the HPLOGISTIC procedure. The data set that you model is the HMEQ data set from the SAMPSIO library. Before you can run the %HPDM_create_scorecode_logistic macro, you need to run PROC HPLOGISTIC and output the parameter estimate table and model information table, as shown below.

```
/*--- load the score code creation macro ---*/
filename source1 catalog 'sashelp.hpdm.hpreg_macros.source';
%include source1;
filename source1;

/*--- run proc HPLOGISTIC to create the model --*/
proc hplogistic data=sampsio.hmeq;
  class JOB REASON DELINQ DEROG NINQ;
  model BAD(order=internal descending) =
```

```
    JOB REASON CLAGE CLNO DEBTINC DELINQ DEROG
    LOAN MORTDUE NINQ VALUE YOJ / link=LOGIT ;
    performance details;
  ods output ParameterEstimates = ParamEsts1 ModelInfo = MInfo1;
run; quit;
```

Next, run the %HPDM_create_scorecode_logistic macro to create the score code for the logistic model. Also, the code below applies the score code.

```
filename code1 catalog 'work.model.scorecode_HMEQ_BAD.source';

%HPDM_create_scorecode_HPLOGISTIC(
 data = sampsio.hmeq,
 model = ParamEsts1,
 modelinfo = Minfo1,
 classify = Y,
 residual = Y,
 fileref = code1,
 eventLevel = 1,
 nonEventLevel= 0
);

data scored_hmeq;
  set sampsio.hmeq;
  %include code1;
run;
```

Chapter 8

The %HPDM_create_scorecode_neural Macro

Overview

Introduction

The %HPDM_create_scorecode_neural macro creates neural network score code based on the output of the HPNEURAL procedure. This macro reads the output data set created by the OUTMODEL= option in the TRAIN statement of PROC HPNEURAL. See "The HPNEURAL Procedure" in the *SAS High-Performance Analytics User's Guide* for information about its usage.

The generated score code is used inside a DATA step to calculate the predicted value for an interval target variable and the predicted probabilities for a nominal target variable in a neural network model. The residuals are calculated on request.

Usage

Before you can invoke the %HPDM_create_scorecode_neural macro, you must include the macro source file. The macro source file is located in the HPDM catalog. The following is an example of how to include the macro source file:

```
filename NNSC catalog 'sashelp.hpdm.hpdmneural_score_macros.source';
%include NNSC;
filename NNSC;
```

You must create the file reference that stores the score codes before you invoke the %HPDM_create_scorecode_neural macro. Otherwise, the score code output prints to the SAS log.

Syntax

The %HPDM_create_scorecode_neural Macro

%macro hpdm_create_scorecode_neural (
DATA=*SAS-data-set*
MODEL=*SAS-data-set*
<FILEREF=*file-name*>
<RESIDUAL=*Y* | *N*>

Required Arguments

DATA=*SAS-data-set*
Specifies the training data that is used for the HPNEURAL procedure.

MODEL=*SAS-data-set*
Specifies the parameter estimates produced by PROC HPNEURAL. The data set specified here is the data set created by the OUTMODEL= argument in the TRAIN statement of the HPNEURAL procedure.

Optional Arguments

FILEREF=*file-name*
Specifies a SAS file reference for an output code file. If this option is omitted, then the output prints to the SAS log.

RESIDUAL=*Y* | *N*
Specify *Y* to create residual variables. The default value is *N*.

Example: The %HPDM_create_scorecode_neural Macro

This example applies the %HPDM_create_scorecode_neural macro to neural network model created for the SAMPSIO.HMEQ data set. Before you can run the %HPDM_create_scorecode_neural macro, you need to prepare the data set and run PROC HPNEURAL. The code below accomplishes both of these steps.

```
%let hpdm = <directoryPath>;

libname hpdm "&hpdm.";
filename CodeFile "&hpdm.\hpneural_scorecode.sas";
filename CodeDS2 "&hpdm.\hpneural_scorecode_DS2.sas";

/* Create a case ID */
data hpdm.hmeq;
   set sampsio.hmeq;
   casnum = _N_;
```

```
run;

/* Partition data into training and hold-out samples */
data hpdm.hmeq_train
     hpdm.hmeq_holdout;
   set hpdm.hmeq;
   call streaminit(27513);
   if (rand('uniform') le 0.7) then output hpdm.hmeq_train;
   else output hpdm.hmeq_holdout;
run;

/* Assign variables into interval predictors and nominal predictors */
%let INTPRED = CLAGE CLNO DEBTINC LOAN MORTDUE VALUE YOJ;
%let NOMPRED = DELINQ DEROG JOB NINQ REASON ;

/* Predict bad loan using all available predictors by a neural*/
/* network model (2 layers with 10 hidden nodes)              */
proc hpneural data = hpdm.hmeq_train;
   id casnum;
   input &INTPRED. / level = int;
   input &NOMPRED. / level = nom;
   target BAD / level = nom;
   hidden 10 / act = sig;
   architecture layer2;
   train outmodel = hpdm.hmeq_model numtries=4;
   performance details;
run;
```

Note: You must replace **<directoryPath>** with the directory path to a valid location on your network.

The above code partitions the SAMPSIO.HMEQ data set into a training and a holdout data set, and then models the training data set with the HPNEURAL procedure. Now, you can run the %HPDM_create_scorecode_neural macro with the following code:

```
/* Invoke the macro to generate score codes */
filename NNSC catalog 'sashelp.hpdm.hpdmneural_score_macros.source';
%include NNSC;
filename NNSC;

%hpdm_create_scorecode_neural
(
   data = hpdm.hmeq_train,
   model = hpdm.hmeq_model,
   fileref = CodeFile,
   residual = N
);

/* Translate scoring code into DS2 codes using DSTRANS */
proc dstrans ds_to_ds2 nocomp aster
             in = CodeFile
             out = CodeDS2;
run;

/* Calculate scores (i.e. predicted probabilities) for the hold-out sample */
%let ASTER_INPUT = sasep.in;
%let ASTER_OUTPUT = sasep.out;
```

```
proc hpds2 in = hpdm.hmeq_holdout
           out = hpdm.hmeq_holdout_score;
   %include CodeDS2;
run;

/* Print the first 20 records of scores in the hold-out samples */
proc print data = hpdm.hmeq_holdout_score (obs = 20);
   var casnum BAD _WARN_ P_:;
run;
```

Chapter 9

The %HPDM_create_scorecode_reg Macro

Overview

Introduction

The %HPDM_create_scorecode_reg macro creates SAS code to score the regression model that is created by the HPREG procedure.

Syntax

The %HPDM_create_scorecode_reg Macro

%macro hpdm_create_scorecode_reg (
DATA=*SAS-data-set*
FILEREF=*file-name*
MODEL=*SAS-data-set*
MODELINFO=*SAS-data-set*
<IMPUTE=*Y* | *N*>
<RESIDUAL=*Y* | *N*>

Required Arguments

DATA=*SAS-data-set*
Specifies the data set that is used to create the regression model in PROC HPREG.

FILEREF=*file-name*
Specifies a SAS file reference for an output code file. Failure to specify this argument results in an error.

MODEL=*SAS-data-set*
Specifies the parameter estimate data set that is generated by the HPREG procedure.

MODELINFO=*SAS-data-set*
Specifies the model information data set output that is generated by the HPREG procedure.

Optional Arguments

IMPUTE=*Y* | *N*
Specify *Y* to impute the predicted values. The default value is *N*.

RESIDUAL=*Y* | *N*
Specifies *Y* to create residual variables. The default value is *N*.

Example: The %HPDM_create_scorecode_reg Macro

This example applies the %HPDM_create_scorecode_reg macro to a regression model that is created by the HPREG procedure. The data set that you model is the HMEQ data set from the SAMPSIO library. Before you can run the %HPDM_create_scorecode_reg macro, you need to run PROC HPREG, as shown below.

```
/*--- load the score code creation macro ---*/
filename source1 catalog 'sashelp.hpdm.hpreg_macros.source';
%include source1;
filename source1;

/*--- run proc hpreg to create the model --*/
proc hpreg data=sampsio.hmeq;
  class JOB REASON BAD DELINQ DEROG NINQ;
  model LOAN =
    JOB REASON CLAGE CLNO DEBTINC DELINQ DEROG MORTDUE NINQ VALUE YOJ BAD;
  performance details;
  ods output ParameterEstimates = ParamEsts2 ModelInfo = MInfo2;
run; quit;
```

Next, run the %HPDM_create_scorecode_reg macro to create the score code for the regression model. Also, the code below applies the score code.

```
filename code2 catalog 'work.model.scorecode_HMEQ_LOAN.source' ;

%HPDM_create_scorecode_HPREG(
  data = sampsio.hmeq,
  model = ParamEsts2,
  modelinfo = Minfo2,
  residual = Y,
  fileref = code2
);
```

```
data scored_hmeq;
  set sampsio.hmeq;
  %include code2;
run;
```

Part 3

Example High-Performance Procedure and Macro Code

Chapter 10

Home Equity Loan Default Model

Overview

This example program uses SAS Enterprise Miner High Performance (HP) procedures and macros together to build a data mining model with model scoring. The example code uses the HMEQ (Home Equity) data set from the SAMPSIO example data library that ships with SAS Enterprise Miner.

For a variable map of the SAMPSIO.HMEQ home equity data set, see "SAMPSIO.HMEQ Data Set Map" on page 78.

For ease of use and understanding, this program is written for deployment on a single SAS Enterprise Miner client, and not on a grid.

Example Program Flow

The following outline indicates the sequence of functional operations that are performed by the example program. Actions that are new high-performance components are in bold.

Action	Function
setup	specify SAMPSIO.HMEQ data set, select BAD as target variable, create macro variables, create titles
DATA step	partition SAMPSIO.HMEQ into train and test data sets and create a partition variable for the train data set

Action	Function
proc hpdmdb	identify variables with missing values, determine the range for LOAN and MORTDUE variables.
proc hpimp	impute missing values in interval variables, create score code.
DATA step	merge training data with new imputed variables
proc hpbin	create bin class variables for LOAN and MORTDUE
DATA step	merge training data with new bin variables
macro %hpdm_create_scorecode_hpbin	create score code for bin variables
proc hpreduce	select a subset of the variable set. The list used for variable selection includes impute variables, impute indicator variables, and bin variables. Variable transformations affect the model.
proc hpneural	train the neural network
macro %em_new_assessmacro %em_new_report	build and report on model performance
macro %hpdm_create_scorecode_neural	create score code for the neural model
DATA step	apply the generated score code to the test data
macro %em_new_assessmacro %em_new_report	report on the test data set
proc hplogistic	use backwards model selection algorithm to train the logistic regression model
macro %em_new_assessmacro %em_new_report	build and report on model assessment
macro %hpdm_create_scorecode_reg	create score code for the regression model
DATA step	apply the generated score code to the test data
macro %em_new_assessmacro %em_new_report	build and report on model performance
proc hpdecide	create profit matrix and bias decisions.

Example Code

```
/*-------------------------------------------------------------*/
/* HPDM 1.1 HMEQ SAMPLE PROGRAM                                */
/*-------------------------------------------------------------*/
/* load use useful macros for hp data mining                   */
/*-------------------------------------------------------------*/
filename h catalog 'sashelp.hpdm.hpdm_tools.source' ;
  %include h ;
filename h catalog 'sashelp.hpdm.hpreg_macros.source' ;
  %include h ;
filename h catalog 'sashelp.hpdm.hpdmneural_score_macros.source' ;
  %include h ;
filename h catalog 'sashelp.hpdm.hpdm_score_create_hpbin.source' ;
  %include h ;
%global em_keytargetlevel ;


/*-------------------------------------------------------------*/
/* Create train and test partitions of the data                */
/* Grid data should have an ID column for matching results     */
/*-------------------------------------------------------------*/
data train test; set sampsio.hmeq ;
    length id 8 ;
    ID= 10101010 +  _N_ ;
    if ranuni(1) < 0.1 then output test ; else output train ;
run ;


/*-------------------------------------------------------------*/
/* Create a partition variable in the train data               */
/*-------------------------------------------------------------*/
data train ; set train ;
    length partition $2 ;
    if ranuni(1) < 0.5 then partition='T';
else    partition='V';
run ;


/*-------------------------------------------------------------*/
/* use these variables for the analysis.  the target is BAD    */
/*-------------------------------------------------------------*/
%let class= JOB REASON ;
%let vars= CLAGE CLNO DEBTINC DELINQ DEROG LOAN MORTDUE NINQ VALUE YOJ ;


/*-------------------------------------------------------------*/
/* create summary of the data.  locate missing var values      */
/*-------------------------------------------------------------*/
title1 'SAS High Performance Data Mining 1.1 DEMO' ;
proc hpdmdb data=train varout=v classout=c dmdbcat=d ;
```

```
    performance details ;
    var &vars ;
    class &class ;
run ;

title2 'Class Variables' ; proc print data=c noobs ; run ;

title2 'Interval Variables' ; proc print data=v noobs; run ;


/*-------------------------------------------------------------*/
/* impute the missing interval variables                       */
/* procedure saves scorecode for test and production data      */
/*-------------------------------------------------------------*/
title2 'Impute missing interval values' ;
filename impute 'impute.sas' ;
proc hpimp data=train out=hpimp dmdbcat=d;
    performance details ;
    id id ;
    input &vars ;
    impute &vars / method=mean ;
    ods table ImputeResults=ir ;
    code file= impute ;
run ;


/*-------------------------------------------------------------*/
/* Update train table by merging output                        */
/*-------------------------------------------------------------*/
data train ; merge train hpimp ; by  id ; drop &vars ; run;


/*-------------------------------------------------------------*/
/* fetch names of new variables                                */
/*-------------------------------------------------------------*/
%global n_imp_vars imp_vars m_vars ;
data _null_ ; set ir end=eof ;
    length imp_vars $2000 m_vars $2000 ;
    retain imp_vars ' ' m_vars ' ' ;
    imp_vars= strip(imp_vars) !! ' ' !! impvarname ;
    m_vars= strip(m_vars) !! ' ' !! indicatorvarname ;
    if eof then do ;
        call symput('imp_vars',strip(imp_vars)) ;
        call symput('m_vars',strip(m_vars)) ;
        call symput('n_imp_vars', strip(put(_N_,6.))) ;
    end ;
run ;

%put NOTE: IMPUTE VARS: &n_imp_vars : &imp_vars ;
%put NOTE: INDICATOR VARS: &n_imp_vars : &m_vars ;


/*-------------------------------------------------------------*/
/* bin variables with wide distributions                       */
/*-------------------------------------------------------------*/
proc hpbin data=train output=hpbin pseudo_quantile ;
```

```
    performance details ;
    id id ;
    var imp_LOAN imp_MORTDUE ;
    ods table mappingTable=binmap ;
run ;


/*-------------------------------------------------------------*/
/* Update train table by merging output                        */
/*-------------------------------------------------------------*/
data train ; merge train hpbin ; by  id ; run;

%let class= &class bin_imp_LOAN bin_imp_MORTDUE ;


/*-------------------------------------------------------------*/
/* build score code for test and production data               */
/*-------------------------------------------------------------*/
filename bincode 'bincode.sas' ;
%hpdm_create_scorecode_bin(bindata=binmap,fileref=bincode) ;


/*-------------------------------------------------------------*/
/* select variables based on unsupervised variance reduction   */
/*-------------------------------------------------------------*/
title2 'Reduce dimensionality' ;
proc hpreduce data=train outcp= cp;
    performance details ;
    id id partition ;
    class &class / missing ;
    reduce unsupervised &class &imp_vars &m_vars / varexp=0.95 ;
    ods table selectionsummary=s ;
run ;


/*-------------------------------------------------------------*/
/* fetch names of selected variables                           */
/*-------------------------------------------------------------*/
proc freq data=s noprint ; table variable / missing out=sf ; run ;
%global n_reduce_vars reduce_vars n_reduce_class reduce_class ;
data _null_ ; set sf end=eof ;
    length cvars $2000 ivars $2000 ;
    retain cvars ' ' ivars ' ' cnv 0 inv 0 ;
    if count gt 1 then do ;
   cvars= strip(cvars) !! ' ' !! variable ;
   cnv+1 ;
end ;
    else do ; ivars= strip(ivars) !! ' '  !! variable ;
   inv+1 ;
end ;
    if eof then do ;
        call symput('reduce_vars',strip(ivars)) ;
        call symput('n_reduce_vars', strip(put(inv,6.))) ;
        call symput('reduce_class',strip(cvars)) ;
        call symput('n_reduce_class', strip(put(cnv,6.))) ;
    end ;
```

```
run ;

%put NOTE: REDUCE VARS: &n_reduce_vars : &reduce_vars ;
%put NOTE: REDUCE CLASS: &n_reduce_class : &reduce_class ;


/*-------------------------------------------------------------*/
/* Train a Neural Network model                                */
/*-------------------------------------------------------------*/
title2 'Neural Network' ;
proc hpneural data=train ;
    performance details ;
    id id  ;
    input &reduce_vars / level=int ;
    input &reduce_class / level=nom  ;
    target bad / level=nom ;
    hidden 4 ;
    architecture layer1 ;
    train outmodel=nn maxiter=40 ;
    score  out=scores ;
run ;


/*-------------------------------------------------------------*/
/* create model performance measures                           */
/*-------------------------------------------------------------*/
%em_new_assess(data=scores,out=bins,target=bad,event=1,var=p_bad1,
  from=from,into=into) ;
%em_new_report(bins=bins,from=from,into=into) ;


/*-------------------------------------------------------------*/
/* build score code for test and production data               */
/*-------------------------------------------------------------*/
filename neural 'neural.sas' ;
%hpdm_create_scorecode_neural(data=train,model=nn,fileref=neural) ;


/*-------------------------------------------------------------*/
/* score test data and measure performance                     */
/*-------------------------------------------------------------*/
data testscores ; set test ;
    %include impute ;
    %include bincode ;
    %include neural ;
    keep id bad p_: ;
run ;

title2 'Neural Model Applied to Test Data' ;
%em_new_assess(data=testscores,out=testbins,target=bad,event=1,var=p_bad1,
  from=from,into=into) ;
%em_new_report(bins=testbins,from=from,into=into) ;


/*-------------------------------------------------------------*/
/* train a logistic regression model.                          */
```

```
/* use all imputed vars and backward model selection           */
/*-------------------------------------------------------------*/
title2 'Logistic Regression' ;
proc hplogistic data=train ;
    performance details ;
    id id partition bad ;
    class &class ;
    model bad(ref=first) = &imp_vars &m_vars &class ;
    selection method=backward ;
    output out=scores(rename=(pred=p_bad1)) pred ;
    ods output ParameterEstimates=lr_est ModelInfo=lr_info ;
run ;


/*-------------------------------------------------------------*/
/* create model performance measures                           */
/*-------------------------------------------------------------*/
%em_new_assess(data=scores,out=bins,target=bad,event=1,var=p_bad1,
  from=from,into=into) ;
%em_new_report(bins=bins,from=from,into=into) ;


/*-------------------------------------------------------------*/
/* build score code for test data and production systems       */
/*-------------------------------------------------------------*/
filename logistic 'logistic.sas' ;
%HPDM_create_scorecode_HPLOGISTIC(data=train,model=lr_est,modelinfo=lr_info,
  fileref=logistic,eventLevel=1,nonEventLevel=0) ;
/*-------------------------------------------------------------*/
/* score test data and measure performance                     */
/*-------------------------------------------------------------*/
title2 'Logistic Model Applied to Test Data' ;
data testscores ; set test ;
    %include impute ;
    %include bincode ;
    %include logistic ;
    keep id bad p_: ;
run ;

%em_new_assess(data=testscores,out=testbins,target=bad,event=1,
  var=p_bad1,from=from,into=into) ;
%em_new_report(bins=testbins,from=from,into=into) ;


/*-------------------------------------------------------------*/
* Create a profit matrix ;
* introduce bias to keep more risky borrowers ;
* use basis points as units ;
/*-------------------------------------------------------------*/
data profit(type=profit) ;
    length BAD $32 keep reject 8;
    label keep='keep' reject='reject';
    bad='1' ; keep=150 ; reject=170 ; output ;
    bad='0' ; keep=200 ; reject=100 ; output ;
run;
```

```
title2 'Decision Processing' ;
proc hpdecide data=testscores out=decisions outFit=hpdecide_outFit role=train ;
    decision decdata=profit decvars=keep reject;
    target BAD;
    posteriors P_BAD1 P_BAD0;
run;

proc hpdmdb data=decisions classout=c;
class  i_bad d_profit;
run ;

proc print data=c ; run ;
/*-------------------------------------------------------------*/
* create hpreg example ;
/*-------------------------------------------------------------*/
```

SAMPSIO.HMEQ Data Set Map

The following table provides summary information about the HMEQ Home Equity data set that is included in the SAS Enterprise Miner SAMPSIO example data library:

Variable	Model Role	Measurement	Description
BAD	target	binary	default or seriously delinquent
CLAGE	input	interval	age of oldest trade (credit) line in months
CLNO	input	interval	number of trade (credit) lines
DEBTINC	input	interval	debt-to-income ratio
DELINQ	input	interval	number of delinquent trade lines
DEROG	input	interval	number of major derogatory reports
JOB	input	nominal	job category
LOAN	input	interval	amount of current loan request
MORTDUE	input	interval	amount due on existing mortgage
NINQ	input	interval	number of recent credit inquiries

Variable	Model Role	Measurement	Description
REASON	input	binary	home improvement or debt consolidation
VALUE	input	interval	value of current property
YOJ	input	interval	years on current job