THE POWER TO KNOW.

SAS® Enterprise Miner™ 14.1: High-Performance Procedures

The HPFOREST Procedure

This document is an individual chapter from *SAS® Enterprise Miner™ 14.1: High-Performance Procedures*.

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

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513-2414.

July 2015

Chapter 8
The HPFOREST Procedure

Contents

Overview: HPFOREST Procedure

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

A *decision tree* is a type of predictive model that has been developed independently in the statistics and artificial intelligence communities. The HPFOREST procedure creates a tree recursively. An input variable

is chosen and used to create a rule to split the data into two segments. The process is then repeated in each segment, and then again in each new segment, and so on until some constraint is met. In the terminology of the tree metaphor, the segments are *nodes*, the original data set is the *root* node, and the final unpartitioned segments are *leaves* or *terminal nodes*. A node is an *internal node* if it is not a leaf. The data in a leaf determine the estimates of the value of the target variable. These estimates are subsequently applied to predict the target of a new observation assigned to the leaf.

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

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

PROC HPFOREST Features

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

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

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

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

Beginning with this release, the training algorithm uses concurrent threads whenever available. However, this release of PROC HPFOREST copies all the data to all cores in distributed mode. For more information, see the section "Processing Modes" on page 8 in Chapter 3, "Shared Concepts and Topics."

PROC HPFOREST Contrasted with Other SAS Procedures

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

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

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

Getting Started: HPFOREST Procedure

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

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

```
data diabetes;
   %let url=//archive.ics.uci.edu/ml/machine-learning-databases;
   infile "http:&url/pima-indians-diabetes/pima-indians-diabetes.data"
          device=url delimiter=',';
   input NumPregnancies
         plasmaGlucose
         diastolicBloodPr
         tricepsSkinfold
         hrSerumInsulin
         BodyMassIndex
         DiabetesPedigreeFn
         Age
         diabetes $
   ;
run;
```

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

```
proc hpforest data=diabetes ;
   input NumPregnancies
         plasmaGlucose
         diastolicBloodPr
         tricepsSkinfold
         hrSerumInsulin
         BodyMassIndex
         DiabetesPedigreeFn
         Age ;
   target diabetes;
 ods output FitStatistics=fitstats;
```

```
   save file="model";
run;
```

Figure 8.1 shows that the program ran locally and that four threads were used. The default number of threads is the number of processors in the computer. The listing also shows the values of the training parameters and the number of observations (768). No parameters are specified in the PROC HPFOREST statement; therefore, all the values are default. The maximum number of decision trees to create is 50. The VARS_TO_TRY= option equals 3, indicating that 3 of the 8 input variables are randomly selected to be considered for a splitting rule.

Figure 8.1 PROC HPFOREST Getting Started Example Output

The HPFOREST Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Data Access Information			
Data	**Engine**	**Role**	**Path**
WORK.DIABETES	V9	Input	On Client

Model Information		
Parameter	**Value**	
Variables to Try	3	(Default)
Maximum Trees	100	(Default)
Inbag Fraction	0.6	(Default)
Prune Fraction	0	(Default)
Prune Threshold	0.1	(Default)
Leaf Fraction	0.00001	(Default)
Leaf Size Setting	1	(Default)
Leaf Size Used	1	
Category Bins	30	(Default)
Interval Bins	100	
Minimum Category Size	5	(Default)
Node Size	100000	(Default)
Maximum Depth	20	(Default)
Alpha	1	(Default)
Exhaustive	5000	(Default)
Rows of Sequence to Skip	5	(Default)
Split Criterion	.	Gini
Preselection Method	.	BinnedSearch
Missing Value Handling	.	Valid value

Number of Observations	
Type	**N**
Number of Observations Read	768
Number of Observations Used	768

Figure 8.2 shows the baseline fit statistics. PROC HPFOREST first computes baseline statistics without using a model. The listing shows a baseline misclassification rate of 0.349 because that is the proportion of observations for which diabetes equals 1.

Figure 8.2 PROC HPFOREST Getting Started Example Output

Baseline Fit Statistics	
Statistic	**Value**
Average Square Error	0.227
Misclassification Rate	0.349
Log Loss	0.647

Figure 8.3 shows the first 10 and last 10 observations of the fit statistics. When PROC HPFOREST runs in single-machine mode, it computes fit statistics for a sequence of forests that have an increasing number of trees. As the number of trees increases, the fit statistics usually improve (decrease) at first and then level off and fluctuate in a small range. Forest models provide an alternative estimate of average square error and misclassification rate, called the *out-of-bag* (OOB) estimate. The OOB estimate is a convenient substitute for an estimate that is based on test data and is a less biased estimate of how the model will perform on future data. For more information, see the section "Bagging the Data" on page 150. The listing shows that the out-of-bag error estimate is worse (larger) than the estimate that evaluates all observations on all trees. This is usual. The out-of-bag misclassification rate for the model fluctuates between 0.230 and 0.238, which is in a range much less than the baseline rate of rate of 0.349. Therefore, you can conclude that the model is good.

Figure 8.3 PROC HPFOREST Getting Started Example Output

Number of Trees	Number of Leaves	Average Square Error (Train)	Average Square Error (OOB)	Misclassification Rate (Train)	Misclassification Rate (OOB)	Log Loss (Train)	Log Loss (OOB)
1	**100**	0.1367	0.341	0.1367	0.341	3.148	7.850
2	**212**	0.0801	0.288	0.1172	0.297	0.908	6.267
3	**318**	0.0663	0.280	0.0729	0.318	0.437	5.541
4	**430**	0.0542	0.259	0.0703	0.303	0.205	4.647
5	**537**	0.0463	0.241	0.0417	0.296	0.165	3.975
6	**637**	0.0416	0.222	0.0469	0.277	0.157	3.320
7	**736**	0.0389	0.214	0.0313	0.273	0.154	2.950
8	**849**	0.0362	0.206	0.0339	0.268	0.148	2.645
9	**945**	0.0358	0.202	0.0234	0.265	0.149	2.345
10	**1055**	0.0347	0.198	0.0260	0.260	0.147	2.170
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
91	**9510**	0.0267	0.162	0.0026	0.232	0.142	0.517
92	**9616**	0.0267	0.162	0.0026	0.234	0.142	0.516
93	**9717**	0.0266	0.162	0.0026	0.233	0.142	0.515
94	**9817**	0.0267	0.162	0.0026	0.232	0.142	0.516
95	**9923**	0.0267	0.162	0.0013	0.230	0.142	0.515
96	**10030**	0.0266	0.162	0.0013	0.229	0.142	0.515
97	**10127**	0.0266	0.162	0.0013	0.230	0.142	0.515
98	**10233**	0.0267	0.162	0.0013	0.234	0.142	0.516
99	**10334**	0.0267	0.162	0.0013	0.236	0.142	0.516
100	**10446**	0.0267	0.162	0.0013	0.238	0.142	0.517

Estimates of variable importance appear after the fit statistics. The Number of Rules column in Figure 8.4 shows the number of splitting rules that use a variable. The section "Measuring Variable Importance" on page 176 explains the measures of importance. Each measure is computed twice: once on training data and once on out-of-bag data. As with fit statistics, the out-of-bag estimates are less biased. The rows are sorted by the OOB Gini measure, which is a more stringent measure than the OOB margin measure. The OOB Gini column is negative for seven variables, and the OOB margin column is negative for two variables. The splitting rules that involve these two variables are, on average, spurious. The main conclusion from fitting the forest model to these data is that plasmaGlucose is the most important predictor of future onset of diabetes.

Figure 8.4 PROC HPFOREST Getting Started Example Output

Loss Reduction Variable Importance

Variable	Number of Rules	Gini	OOB Gini	Margin	OOB Margin
plasmaGlucose	1341	0.121668	0.04130	0.243336	0.16453
NumPregnancies	698	0.031116	-0.01226	0.062232	0.01934
BodyMassIndex	1665	0.075439	-0.02387	0.150878	0.04945
hrSerumInsulin	815	0.029283	-0.02493	0.058566	0.00500
tricepsSkinfold	780	0.023756	-0.02584	0.047512	-0.00134
Age	1814	0.068814	-0.02874	0.137628	0.03839
diastolicBloodPr	1228	0.038636	-0.03966	0.077272	-0.00008
DiabetesPedigreeFn	2005	0.064256	-0.05610	0.128512	0.00764

Syntax: HPFOREST Procedure

The following statements are available in the HPFOREST procedure:

PROC HPFOREST < *options* > **;**
FREQ *variable* **;**
INPUT *variables* < *options* > **;**
ID *variables* **;**
PERFORMANCE *performance-options* **;**
SAVE < *options* > **;**
SCORE < *score-options* > **;**
TARGET *variable* < *options* > **;**
TREATMENT *variable* < **ORDER=***order* > **;**

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

PROC HPFOREST Statement

PROC HPFOREST < *options* > **;**

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

DATA=< *libref.* **>***SAS-data-set*
names the SAS data set to be used by PROC HPFOREST for training the model. The default is the most recently created data set.

If the data are already distributed, the procedure reads the data alongside the distributed database. See the section "Processing Modes" on page 8 for the various execution modes and the section "Alongside-the-Database Execution" on page 16 for the alongside-the-database model. Data from all the computer grid nodes are combined into a structure that is optimized for model training and redistributed to the nodes. The different nodes then proceed independently with identical data to create decision trees.

ALPHA=*number*
specifies a threshold *p*-value for the significance level of a test of association of a candidate variable with the target. If no association meets this threshold, the node is not split. The default value is 1.

BALANCE=YES | NO
specifies whether to modify the splitting criterion for a nominal target so that the number of observations in each target class are effectively equal. A weight is applied to the count of observations in a class. The weight is different in different nodes. You can specify the following values:

YES modifies the splitting criterion so that the number of observations in each target class are effectively equal. Setting BALANCE=YES can improve prediction when the class sizes are very different.

NO does not modify the splitting criterion.

By default, BALANCE=NO.

CATBINS=*k*

specifies the maximum number of categories of a nominal candidate variable to use in the association test. *k* refers only to the categories that are present in the training data in the node and that satisfy the MINCATSIZE= option. The categories are counted independently in each node. If more than *k* categories are present, then the least frequent categories are removed from the association test. Many infrequent categories can dilute a strong predictive ability of common categories. The search for a splitting rule uses all categories that satisfy the MINCATSIZE= options. The value of *k* must be a positive integer. The default value is 30.

EXHAUSTIVE=*number*

specifies the maximum number of splits to examine in a complete enumeration of all possible splits when the input variable is nominal and the target has more than two nominal categories. The exhaustive method of searching for a split examines all possible splits. If the number of possible splits is greater than *number*, then a heuristic search is done instead of an exhaustive search. The default value of *number* is 5,000.

GRIDCLASSSIZE=*n*

specifies the minimum number of observations of any value of a binary or nominal target to exist on a grid node when GRIDCOPY=MINIMAL is specified. Observations are copied between grid nodes as necessary to meet this minimum. This option is ignored unless PROC HPFOREST is running in distributed mode. The default value is 10,000.

GRIDCOPY=ALL | MINIMAL | NONE | TRAINING

specifies how many observations to copy between grid nodes when PROC HPFOREST runs in distributed mode. PROC HPFOREST creates a decision tree on a single grid node without using data on other nodes unless observations are copied from other nodes before any tree is created. Generally, the more observations on a node, the larger the tree can grow and the more accurate the forest. The disadvantage is time, because the core processors have more work to do. You can specify the following values:

ALL copies all training and validation data.

MINIMAL copies enough training data to achieve a minimum on each node. This option uses the GRIDCLASSSIZE= and GRIDNODESIZE= option values to determine the minimum number of observations required on each grid node.

NONE does not copy any observations.

TRAINING copies all training data.

By default, GRIDCOPY=MINIMAL.

GRIDNODESIZE=*n*

specifies the minimum number of observations to use on a grid node when PROC HPFOREST runs in distributed mode and GRIDCOPY=MINIMAL is specified. If the number of observations on a node is less than *n*, then observations are copied from other nodes to achieve *n*. The default value is 100,000.

IMPORTANCE=YES | NO
specifies whether to compute variable importance. You can specify the following values:

YES	computes loss reduction variable importance.
NO	does not computes loss reduction variable importance. If you save the model (by specifying the SAVE statement) and subsequently input it to PROC HP4SCORE, then PROC HP4SCORE cannot compute variable importance either.

By default, IMPORTANCE=YES.

INBAGFRACTION=*f*
specifies the fraction of training observations to train a tree with, where *f* can be any number greater than 0 and at most 1. Using less than all the available data often improves the generalization error. A different in-bag sample is taken for each tree. The default value of *f* is 0.6. PROC HPFOREST uses at least four observations in the in-bag data regardless of how small *f* is (assuming four observations exist). If an observation is available for training but is not an in-bag datum, then it is either out-of-bag or a pruning datum. If *f* is too small to accommodate the LEAFSIZE=, LEAFFRACTION=, and SPLITSIZE options then no tree is made. The INBAGN= option accepts an absolute number instead of a fraction to specify the same quantity. Specifying both the INBAGN= and INBAGFRACTION= options is an error.

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

INTERVALBINS=*k*
specifies the number of equally spaced bins into which variables are divided when the PRESELECT=BINNEDSEARCH algorithm is executed. The default value is 100.

LEAFFRACTION=*f*
specifies the smallest number of training observations that a new branch can have, expressed as the fraction of the number *N* of available observations in the DATA= data set. *N* might be less than the total number of observations in the data set because observations with a missing target value or non positive value of the variable specified in the FREQ statement are excluded from *N*. If you specify a number in the LEAFSIZE= option that implies a larger number than that specified in the LEAFFRACTION= option, *f* is ignored. The value *f* must be larger than 0 and less than 1. The default value is 0.00001.

LEAFSIZE=*n*
specifies the smallest number of training observations a new branch can have. If you specify a value for the LEAFFRACTION= option that implies a larger value than *n*, the LEAFSIZE= option is ignored. The default value is 1.

MAXDEPTH=*d*

specifies the maximum depth of a node in any tree that PROC HPFOREST creates. The depth of a node equals the number of splitting rules needed to define the node. The root node has depth 0. The children of the root have depth 1, the children of those children have depth 2, and so on. The smallest acceptable value of *d* is 1. The default value of *d* is 20 (implying a maximum of 1,048,576 leaves).

MAXTREES=*n*

specifies the number of trees in the forest. *n* is a positive integer. The number of trees in the resulting forest can be less than *n* when the HPFOREST procedure fails to split the training data for a tree. Up to two times *n* trees are attempted. If the procedure fails to split the training data for more than *n* trees, then less than *n* trees are created. The ALPHA=, LEAFSIZE=, and MINCATSIZE= options constrain the split search to form trees that are more likely to predict well using new data. Setting all of these options to 1 generally frees the search algorithm to find a split and train a tree, although the tree might not help the forest predict well. The default value of *n* is 100.

MINCATSIZE=*n*

specifies the minimum number of observations that a given nominal input category must have in order to use the category in a split search. Categorical values that appear in fewer than *n* observations are handled as if they were missing. The categories that occur in fewer than *n* observations are merged into the pseudo category for missing values for the purpose of finding a split. The policy for assigning such observations to a branch is the same as the policy for assigning missing values to a branch. The default value of *n* is 5.

MINUSEINSEARCH=*n*

specifies a threshold for utilizing missing values in the split search when MISSING=USEINSEARCH is specified as the missing value policy. If the number of observations in which the splitting variable has missing values in a node is greater than or equal to *n*, then PROC HPFOREST initiates the USEINSEARCH policy for missing values. See the section "Handling Missing Values" on page 172 for a more complete explanation. The default value of *n* is 1.

MISSING=USEINSEARCH | BIGBRANCH

specifies how the training procedure handles an observation with missing values. If MISSING=USEINSEARCH and the number of training observations in the node is more than *n*, where *n* is the value of the MINUSEINSEARCH= option, then the missing value is used as a separate, legitimate value in the test of association and the split search. If MISSING=BIGBRANCH, observations with a missing value of the candidate variable are omitted from the test of association and split search in that node. A splitting rule will assign such an observation to the branch containing the most observations among those used in the split search. See the section "Handling Missing Values" on page 172 for a more complete explanation. By default, MISSING=USEINSEARCH.

NODESIZE=*n* | ALL

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

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

measures and split worth adjust the category counts to the category proportions in the node before sampling.

PRESELECT=BINNEDSEARCH | LOH | HOTHORN
specifies the method for selecting the variable to split with. You can specify the following values:

BINNEDSEARCH selects the variable that produces the rule that has the largest worth. This option searches for a splitting rule for each candidate variable. Values of an interval variable are binned before the search. This option preferentially selects nominal variables that have many categories. But if such variables do not exist or if all the variables have a similar number of categories, then this option potentially selects better variables than the other methods. This option usually takes longer because searching for a splitting rule is more complex. For more information, see the section "Binned Search" on page 156.

HOTHORN selects the variable that produces the rule that has the highest association with the target. For more information, see the section "Hothorn, Hornik, and Zeileis" on page 156.

LOH selects the variable that has the smallest p-value of a chi-square test of association in a contingency table. For more information, see the section "Loh and GUIDE" on page 158.

If no nominal variables have more than five categories, then PRESELECT=BINNEDSEARCH by default. Otherwise, if some variables are not nominal or if any nominal variable has five categories more than another nominal variable, then PRESELECT=LOH by default.

PRUNEFRACTION=*g*
specifies the fraction of training observations that are available for pruning a split. The value of *g* can be any number from 0 and to 1, although a number close to 1 would leave little to grow the tree. The default value of *g* is 0; the default action is to not prune.

PRUNETHRESHOLD=*t*
specifies the lower limit of allowable shrinkage when the distance of the child node from the parent is measured by the pruning data instead of the in-bag data. For more information, see the section "Pruning" on page 160. The value of *t* must be between 0 and 1. The default value of *t* is 0.1.

SCOREPROLE=DEFAULT | INBAG | OOB | VALID
specifies which observations are used for the prediction of a new observation in the SCORE statement. Predictions are based on average values of observations that have the same role in leaves. All the observations included in an average have a single role: in-bag, out-of-bag, or validation. The SCOREPROLE= option does not effect the "Fit Statistics" output table. The HP4SCORE procedure outputs the same predictions as the HPFOREST procedure.

You can specify the following values:

DEFAULT selects the default value as if the option was not specified.

INBAG outputs predictions that are based on in-bag observations.

OOB outputs predictions that are based on out-of-bag observations. If out-of-bag observations are not present, then the in-bag observations in the leaf are used.

VALID outputs predictions that are based on validation observations. If validation observations are not present in a leaf node, the out-of-bag observations in that leaf are used. If out-of-bag observations are not present, then the in-bag observations in the leaf are used.

By default, SCOREPROLE=INBAG.

SEED=*n*

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

SKIP_SEQ_ROWS=*n*

specifies the number of rows to skip in the "Fit Statistics" table in distributed mode. After every *n* trees that are trained on a grid node, the fit statistics on the node are updated, consolidated with statistics from other nodes, and eventually output in the "Fit Statistics" table. Each row in the table contains statistics for a specific number of trees in a forest. The table has gaps of up to *n* rows. The gap is smaller if fewer than *n* trees are made on a grid node, and no gap appears if only one tree is made on a node. The SKIP_SEQ_ROWS= option has no effect in single-machine mode. By default, SKIP_SEQ_ROWS=5.

SPLITSIZE=*n*

specifies the requisite number of training observations a node must have for the HPFOREST procedure to consider splitting it. By default, *n* is twice the value of the LEAFSIZE= option (or *n* is the value implied by LEAFFRACTION= option if the procedure ignores the LEAFSIZE= option). The procedure counts the number of observations in a node without adjusting the number with the values of the variable specified in the FREQ statement when it interprets the value specified in the LEAFFRACTION=, LEAFSIZE=, MINCATSIZE=, and SPLITSIZE= options.

VARS_TO_TRY=*m* **| ALL**

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

FREQ Statement

FREQ *variable* **;**

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

INPUT Statement

INPUT *variables* < *options* > **;**

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

LEVEL=*level*

specifies the level of measurement of the variables. Accepted values of *level* are: BINARY, NOMINAL, ORDINAL, and INTERVAL.

ORDER=*order*

specifies the sorting order of the values of an ordinal input variable. Table 8.1 provides recognized values of *order*.

Table 8.1 ORDER= Option Values

Value of ORDER=	Variable Values Sorted By
ASCENDING	Ascending order of unformatted values (default)
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

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

ID Statement

ID *variables* **;**

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

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

PERFORMANCE Statement

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

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

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

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

SAVE Statement

SAVE < *option* > **;**

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

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

Starting in SAS Enterpise Miner 14.1, you can score new data sets against the forest model by specifying the name of this binary file in the FILE= option in the SCORE statement in the HP4SCORE procedure. Earlier releases of SAS Enterprise Miner cannot read the file *filename*.

SCORE Statement

SCORE OUT=*libref.SAS-data-set* < *score-options* > **;**

The SCORE statement applies the forest model to the training data and outputs a data set that contains the ID variables that are specified in the ID statement in addition to predictions, residuals, and decisions.

The prediction variables depend on the measurement type of the target variable in the model. For a target that has an interval measurement level, a single prediction variable is generated. For each level of the target that has a nominal measurement level, a posterior probability variable is generated in addition to the final predicted level. The names of the variables are constructed using the rules that are explained in the SAS Enterprise Miner product documentation.

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

You must specify the output data set with the OUT= option.

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

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

MAXDEPTH=< *n* **>**
produces predictions from trees that are pruned to a depth of *n*. The trees are not truncated by default.

NTREES=< *n* **>**
produces predictions from the first *n* trees only. If the option is omitted then PROC HPFOREST uses all the trees in the model for predictions. Scoring with fewer trees can sometimes increase the speed without significantly reducing the accuracy.

TARGET Statement

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

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

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

TREATMENT Statement (Experimental)

TREATMENT *variable* < **ORDER=***order* > **;**

The TREATMENT statement names a binary treatment *variable* and estimates how much the prediction of an observation changes when the treatment changes. The analysis of differential treatment effects is sometimes called *incremental response modeling* (IRM). The target variable specified in the TARGET statement must have an interval or binary measurement level. For information about how PROC HPFOREST uses the treatment variable, see the section "Modeling Incremental Response from a Treatment" on page 173. Creation of forests with a treatment variable is experimental in PROC HPFOREST.

ORDER=*order*
specifies the sorting order of the treatment values. The smallest and largest sorted values are coded as 0 and 1 respectively in the names of variables in the output data set of the SCORE statement. Table 8.1 provides the recognized values of *order*.

Details: HPFOREST Procedure

Bagging the Data

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

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

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

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

Training a Decision Tree

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

Creating a binary split involves a few subtasks:

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

PROC HPFOREST selects candidate inputs independently in every node. The purpose of preselecting candidate inputs is to increase the differences between the trees, thereby decreasing the correlation and theoretically increasing the quality of the forest predictions. The selection is random, and each input has the same chance. The VARS_TO_TRY= option specifies the number of candidates to select. The quality of the forest often depends on the number of candidates. Unfortunately, a good value for the VARS_TO_TRY= option is generally not known in advance. Data that contain more irrelevant variables generally warrant a larger value.

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

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

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

Controlling for Variable Selection Bias

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

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

A popular solution is to select the splitting variable with a statistical test that does not involve searching for a splitting rule. The test adjusts for the different number of categories. Only the winning variable is used in a search for a splitting rule. Loh and Shih (1997) first proposed this in their QUEST algorithm, and Loh continues to evolve the method in GUIDE. The PRESELECT=LOH and PRESELECT=HOTHORN options in the PROC HPFOREST statement specify a variation of Loh's method and a different method that is attributed to Hothorn, Hornik, and Zeileis (2006), respectively.

Although these methods are fast and mitigate selection bias, they do not focus on selecting a variable that produces the strongest split. When there is no nominal input variable that has many categories, searching for a split with each variable is apt to produce a more accurate model. The PRESELECT=BINNEDSEARCH option specifies this approach. For more information, see the section "Selecting a Splitting Variable" on page 156.

This section illustrates selection bias in traditional methods and the impressive handling by the more recent methods. The data contain a purely random input X_j, which has j nominal categories, and a second input Z_k, which has k categories and is predictive of the target. The plots show the proportions of samples in which a method selects X_j instead of the predictive input, Z_k. The larger the proportion, the greater the bias of the method toward variables that offer more splitting possibilities than predictive power.

The data contain 500 observations. The target Y has values 0 and 1 with equal probabilities. Nominal input Z_k has equally probable integer values from 0 through $k-1$. The probability that $Y = 1$ given that Z_k is greater than or equal to $k/2$ is 0.6 (and therefore $P(Y = 1|Z_k < k/2) = 0.4$). With enough data, a splitting algorithm that uses Z_k assigns values that are less than $k/2$ to one branch, and the rest to the other branch. X_j is equally distributed without reference to Y. With enough data, the best split on X_j has negligible worth.

A variable selection method chooses between two variables. After repeating this for 1,000 samples, the proportion of times X_j is selected is recorded and plotted for several combinations of j and k.

Figure 8.5 shows the proportion of samples for which X_j is selected instead of Z_k when the split-search algorithm uses the reduction in Gini impurity as the splitting criterion. As shown in the figure, Gini reduction selects X_j more often as the number of categories j increases. When j is similar to k, Gini reduction selects X_j between 15% and 30% of the time. For $j - k = 10$, X_j is selected about 75% of the time. Gini reduction selects the wrong variable more often than not for these values of j and k.

Figure 8.5 Proportion of Samples for Which Gini Reduction Selects X_j

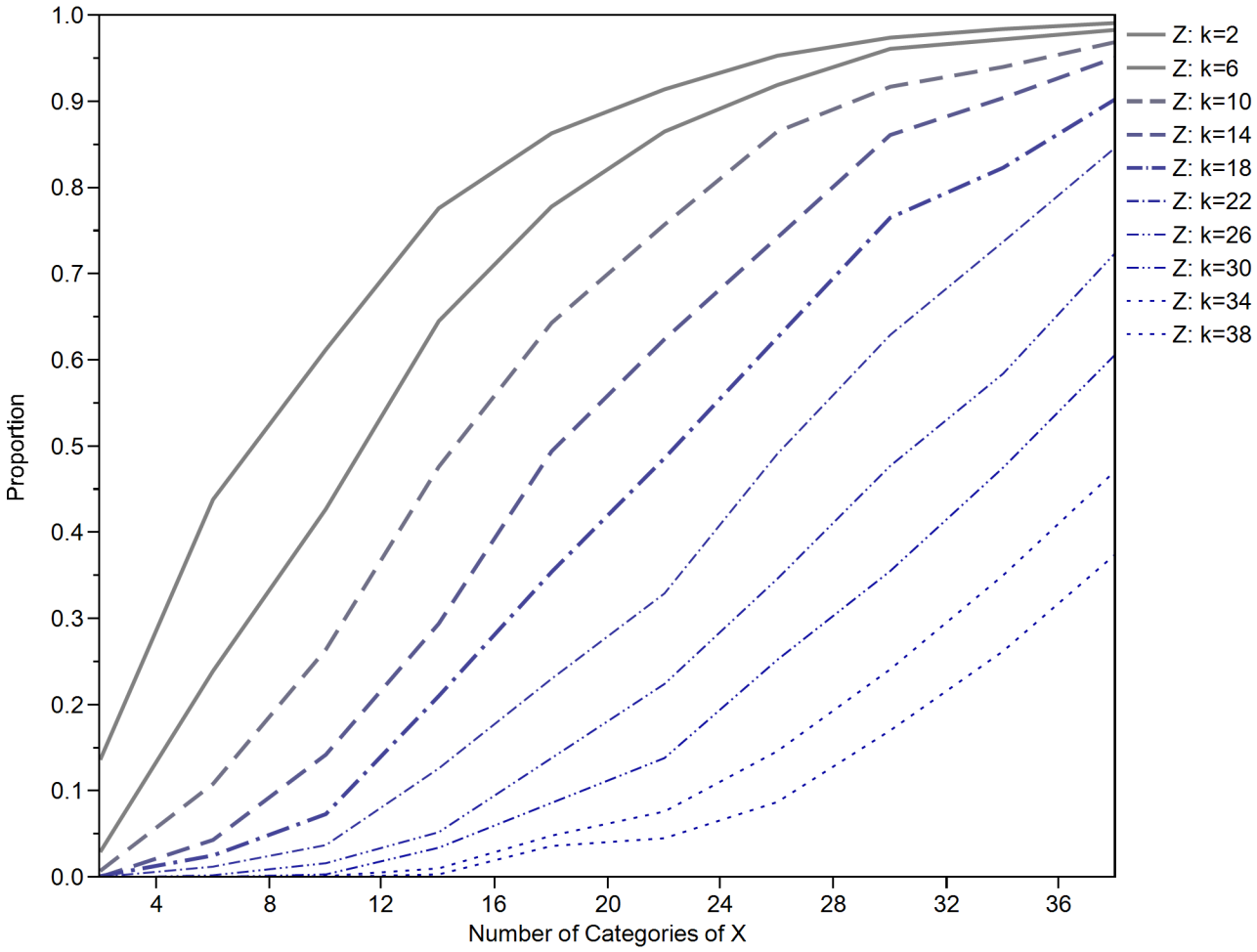


Figure 8.6 shows the proportion of samples for which X_j is selected using the CHAID split-search algorithm. The figure shows that CHAID selects X_j less often as the number of categories j increases. CHAID penalizes variables that have many categories, and the penalty is larger than necessary. For $j - k = 10$, the proportion of times CHAID selects X_j decreases from about 70% ($j = 12$) to less than 10% ($j = 38$). CHAID selects the wrong variable more often than not when j is much less than k.

Figure 8.6 Proportion of Samples for Which CHAID Selects X_j

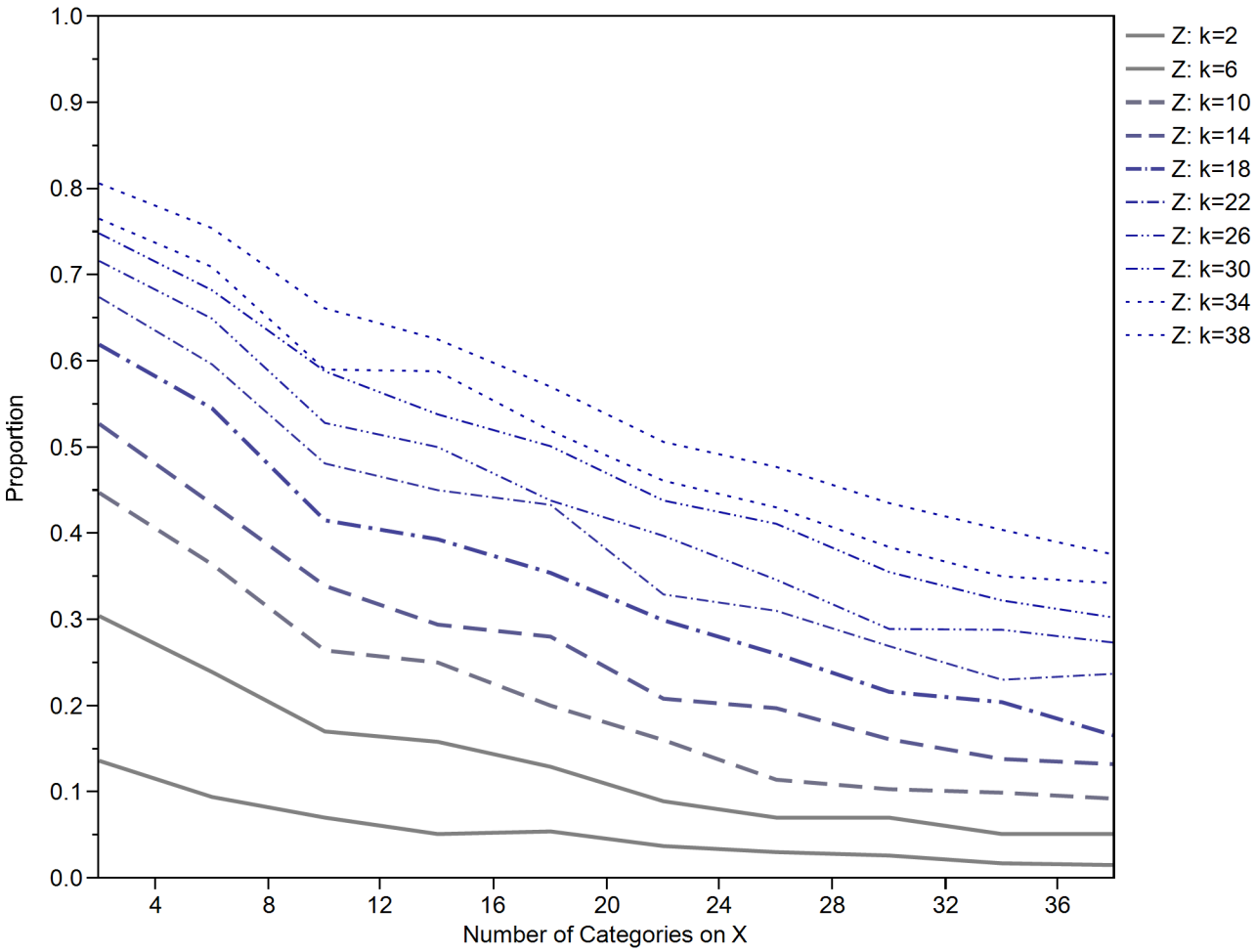
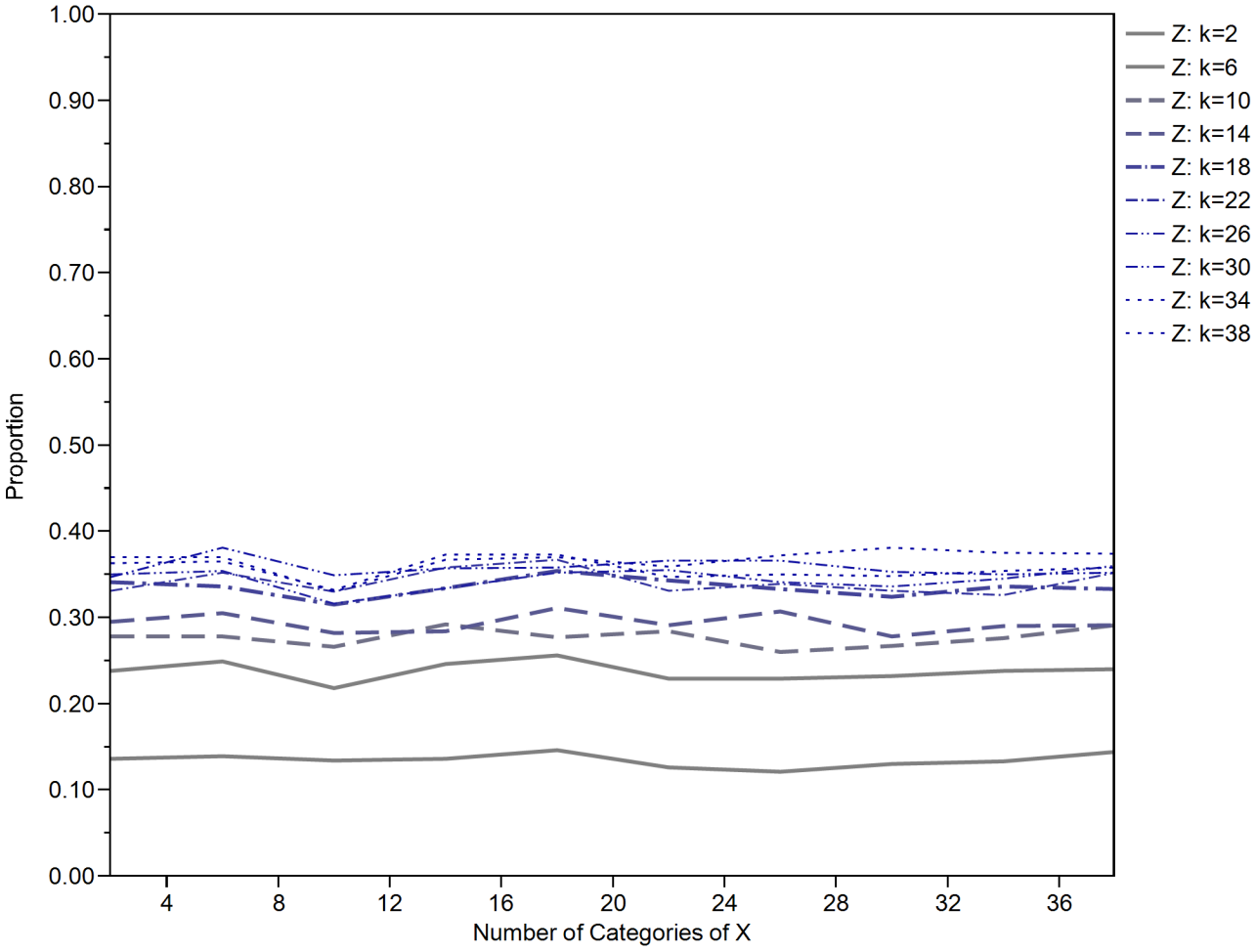


Figure 8.7 shows the proportion of samples for which the PRESELECT=HOTHORN option in the PROC HPFOREST statement selects X_j by using a test of association. As shown in the figure, given Z_k, PROC HPFOREST selects X_j the same proportion of times regardless of the number of categories j. When the number of categories k of Z_k increases, PROC HPFOREST selects X_j more often. However, the proportion never reaches 40%. The PRESELECT=HOTHORN option does not select the wrong variable more than 40% of the time for any values of j and k examined. The PRESELECT=LOH option produces similar results and is not shown.

Figure 8.7 Proportion of Samples for Which an Association Test Selects X_j



The material in this section is part of a fuller discussion in De Ville and Neville (2013) of variable selection bias.

Selecting a Splitting Variable

PROC HPFOREST offers three methods for selecting a variable to use in a splitting rule. The PRESELECT=LOH and PRESELECT=HOTHORN options specify two methods that avoid the bias towards nominal variables that is discussed in the section "Controlling for Variable Selection Bias" on page 152. The PRESELECT=HOTHORN is slightly faster on large data sets than PRESELECT=LOH. The third method, PRESELECT=BINNEDSEARCH, is appropriate when bias is less of a concern than finding the strongest variable to use in the splitting rule. The PRESELECT=BINNEDSEARCH method is slower than the other two methods.

Binned Search

Specify the PRESELECT=BINNEDSEARCH option to select the variable that produces the rule that has the largest worth. Unlike the PRESELET=LOH and PRESELECT=HOTHORN options, the PRESELECT=BINNEDSEARCH option searches for a splitting rule for each variable. This method preferentially selects nominal variables that have many categories. However, if such variables do not exist or if all the variables have a similar number of categories, then the PRESELECT=BINNEDSEARCH option potentially selects better variables than the other methods.

The PRESELECT=BINNEDSEARCH option usually takes longer than the other methods because searching for a splitting rule is more complex. To make it run faster, interval variables are divided into equally spaced bins just before the search. If an interval variable is deemed best, then PROC HPFOREST uses the unbinned data to search for a rule that uses the selected variable. Binning is done independently in every node. The INTERVALBINS= option specifies the number of bins.

Hothorn, Hornik, and Zeileis

Specify the PRESELECT=HOTHORN option to select the variable that has the highest association with the target as measured by the C_{quad} statistic that is described in Hothorn, Hornik, and Zeileis (2006).

Specifically, let Y and X denote the target variable and input variable, respectively. Let Y_i and X_i denote their values in observation i. The formulas for the association depend on whether Y and X are categorical. If Y is categorical, let J denote the number of values, and let Y_{ij} equal 1 if Y_i equals j and 0 otherwise. Similarly, if X is categorical, let K denote the number of values, and let X_{ik} equal 1 if X_i equals k and 0 otherwise. Let T denote the statistic defined in Table 8.2.

Table 8.2 Definition of T for Different Types of Variables

Type of Variable		Dimension of	
Y	X	T	Definition
Interval	Interval	1	$T = \sum Y_i X_i$
J classes	Interval	J	$T_j = \sum Y_{ij} X_i$
Interval	K categories	K	$T_k = \sum Y_i X_{ik}$
J classes	K categories	$J \times K$	$T_{jk} = \sum Y_{ij} X_{ik}$

The test statistic is

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

where

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

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

Table 8.3 contains formulas for $\hat{\mu}$.

Table 8.3 Definition of $\hat{\mu}$ for Different Types of Variables

Type of Variable		Dimension of	
Y	***X***	$\hat{\mu}$	**Definition**
Interval	Interval	1	$\hat{\mu} = (\sum Y_i)(\sum X_i)/N$
J classes	Interval	J	$\hat{\mu}_j = (\sum Y_{ij})(\sum X_i)/N$
Interval	K categories	K	$\hat{\mu}_k = (\sum Y_i)(\sum X_{ik})/N$
J classes	K categories	$J \times K$	$\hat{\mu}_{jk} = (\sum Y_{ij})(\sum X_{ik})/N$

To make the formulas more concise, let $J = 1$ and $Y_{ij} = Y_i$ when Y has an interval measurement level. Now J and Y_{ij} are defined for all types of Y. Similarly, let $K = 1$ and $X_{ik} = X_i$ when X has an interval measurement level. The multiple definitions of T in Table 8.2 now reduce to the single formula,

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

and the concise definition of $\hat{\mu}$ becomes

$$\hat{\mu}_{jk} = (\sum_{i=1}^{N} Y_{ij})(\sum_{i=1}^{N} X_{ik})/N$$

The dimension of $\hat{\Sigma}$ equals $(J \times K)^2$, the dimension of T squared. If X has a large number of categories, then K is large, and storing and inverting $\hat{\Sigma}$ can impede performance.

The formula for $\hat{\Sigma}$ is built as follows from the expectation and covariance of Y and a factor that depends on X:

$$\begin{aligned}
\mathrm{E}(Y)_j &= \sum_{i=1}^{N} Y_{ij}/N \\
\mathrm{Cov}(Y)_{hj} &= \sum_{i=1}^{N} (Y_{ih} - \mathrm{E}(Y)_h)(Y_{ij} - \mathrm{E}(Y)_j)/N \\
\Xi_{kl} &= 1(k = l)(\sum_{i=1}^{N} X_{ik} X_{ik}) - (\sum_{i=1}^{N} X_{ik})(\sum_{i=1}^{N} X_{il}) \\
\hat{\Sigma}_{hjkl} &= \mathrm{Cov}(Y)_{hj} \Xi_{kl} N/(N-1)
\end{aligned}$$

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

Loh and GUIDE

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

This method selects the variable that has the smallest p-value of a chi-square test of association in a contingency table. As before, let Y and X denote the target variable and input variable, respectively. Let Y_i and X_i denote their values in observation i. If Y is categorical, let J denote the number of values. Similarly, if X is categorical, let K denote the number of values or the value of the CATBINS= option, whichever is smaller. If the CATBINS= option value is smaller, then only the observations that have the most frequent categories in the node are used to select a variable.

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

If X has interval measurement level, then let

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

where N is the number of observations in the calculations and $J = 2$ if Y has an interval measurement level. If $K = 3$, assign X_i to a table column that is defined by the following boundary points:

$$\begin{aligned} \xi_1 &= \bar{X} - \sqrt{3}\hat{\sigma}/3 \\ \xi_2 &= \bar{X} + \sqrt{3}\hat{\sigma}/3 \end{aligned}$$

Otherwise, use the boundary points

$$\begin{aligned} \xi_1 &= \bar{X} - \sqrt{3}\hat{\sigma}/2 \\ \xi_2 &= \bar{X} \\ \xi_3 &= \bar{X} + \sqrt{3}\hat{\sigma}/2 \end{aligned}$$

where $\bar{X}$ denotes the average value of X and

$$\hat{\sigma}^2 = \frac{\sum_i (X_i - \bar{X})^2}{N}$$

Searching for a Splitting Rule

Rules

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

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

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

Criteria

The HPFOREST procedure searches for rules that maximize the measure of worth that is associated with the splitting criterion. For binary, nominal, and interval targets, the worth of a split s is the reduction in node impurity,

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

where $i(\omega)$ is the impurity of the node ω, and $p(\omega_b|\omega)$ is the proportion of training observations in branch b. Generally, $p(\omega)$ is a nonnegative number that equals 0 if all observations in ω have the same target value, and $p(\omega)$ is large if the target values in ω are very different.

The impurity function for the Gini index is

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

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

$$i(\omega) = \frac{1}{N(\omega)} \sum_{i=1}^{N(\omega)} (Y_i - \bar{Y})^2$$

where $N(\omega)$ is the number of observations in node ω, Y_i is the target value of observation i, and $\bar{Y}$ is the average of Y_i in ω.

For an ordinal target that has J values, the worth of a split is the Kolmogorov-Smirnov statistic,

$$\kappa(s) = \max_{k=1}^{J} |F_k(\text{left branch}) - F_k(\text{right branch})|$$

where

$$F_k(\omega) = \sum_{j=1}^{k} p_j$$

The impurity measures originate with Breiman et al. (1984), and the Kolmogorov-Smirnov splitting criterion originates in Friedman (1977).

Algorithm

PROC HPFOREST searches for a splitting rule as follows:

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

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

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

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

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

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

Pruning

Definitions

An observation can have one of three possible roles when a tree is trained:

INBAG	split construction
PRUNE	split evaluation
OOB	model evaluation

The role of the pruning data is to confirm or reject a split. A center of each node is computed for the in-bag data and again for the pruning data. If the pruning estimate of the center of a child node is much closer to the center of the parent node than the in-bag estimate of the child node, then the split is deemed unreliable and is pruned. The node is never split again.

Specifically, the split is pruned when

$$\frac{d(C_{\text{prune}}, C_{\text{parent}})}{d(C_{\text{inbag}}, C_{\text{parent}})} < \tau$$

where

C_{parent}	is the in-bag center of the parent node,
C_{inbag}	is the in-bag center of a child node,
C_{prune}	is the prune center of a child node,
$d(a,b)$	is a signed measure of distance from a to b, and
τ	is a fixed number between 0 and 1.

The PRUNETHRESHOLD= option specifies τ. For an interval target, C_ω is the average target value in the node, and $d(a,b)$ is $a-b$. For a categorical target, C_ω is the vector of target value proportions, and

$$d(a,b) = \sum_{j=1}^{J} (a_j - b_j)(C_{\text{inbag},j} - C_{\text{parent},j})$$

Pruning a split immediately after the split is created (called *instant pruning*) is a different approach to pruning than the approach in traditional decision trees in which the complete tree is grown before any node is pruned (called *retrospective pruning*). These two terms are used when necessary to distinguish between the two approaches to pruning.

Recommendations

PROC HPFOREST runs faster when it prunes but usually does not fit the data as well. Pruning is more effective with classification than with regression, and with smaller signal-to-noise ratios.

Figure 8.8 shows the proportion of increase in misclassification rate on test data when PROC HPFOREST prunes for various levels of signal-to-noise ratios. The deterioration in misclassification rate is approximately linear with the signal-to-noise ratio. With very small signals, pruning improves the misclassification rate.

More specifically, the vertical axis is

$$\rho = \frac{\text{MISC with pruning} - \text{MISC without pruning}}{\text{MISC without pruning}}$$

The data are generated as follows:

$$\begin{aligned}
Y &= \begin{cases} 1 & \text{if } W \geq 2 \\ 0 & \text{otherwise} \end{cases} \\
W &= f(x) + \sum_{i=1}^{D} U_i + R\epsilon \\
f(x) &= X_1 + 5X_2 + 10X_3 + X_4^2 + 5X_5^2 \\
X_i &\sim \text{Uniform}(-1,1) \\
U_i &\sim \text{Uniform}(-1,1) \\
\epsilon &\sim \text{Normal}(0,1)
\end{aligned}$$

R determines the signal-to-noise ratio. The number of irrelevant variables D varies from 10 to 100 in increments of 10. For each value of R and D, 25 training and test data sets are created and contain 10,000 and 20,000 observations, respectively. For each pair of data sets, PROC HPFOREST creates 100 trees with INBAGFRACTION=100, PRUNEFRACTION= 0 or 0.3, and PRUNETHRESHOLD=0.10. PROC HP4SCORE scores the test data. A DATA step computes the mislcassification rate.

Figure 8.8 Proportion of Increase in Misclassification Rate with Pruning

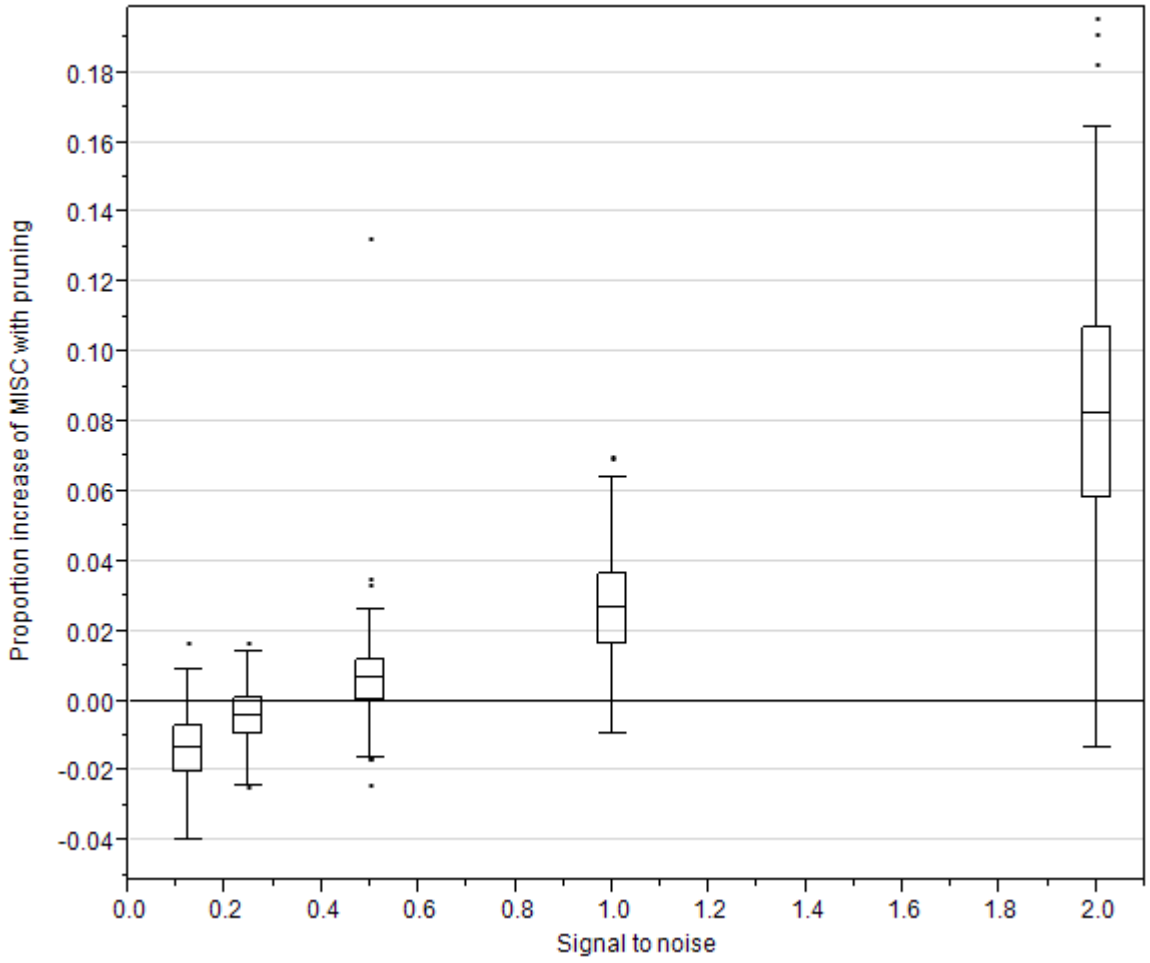
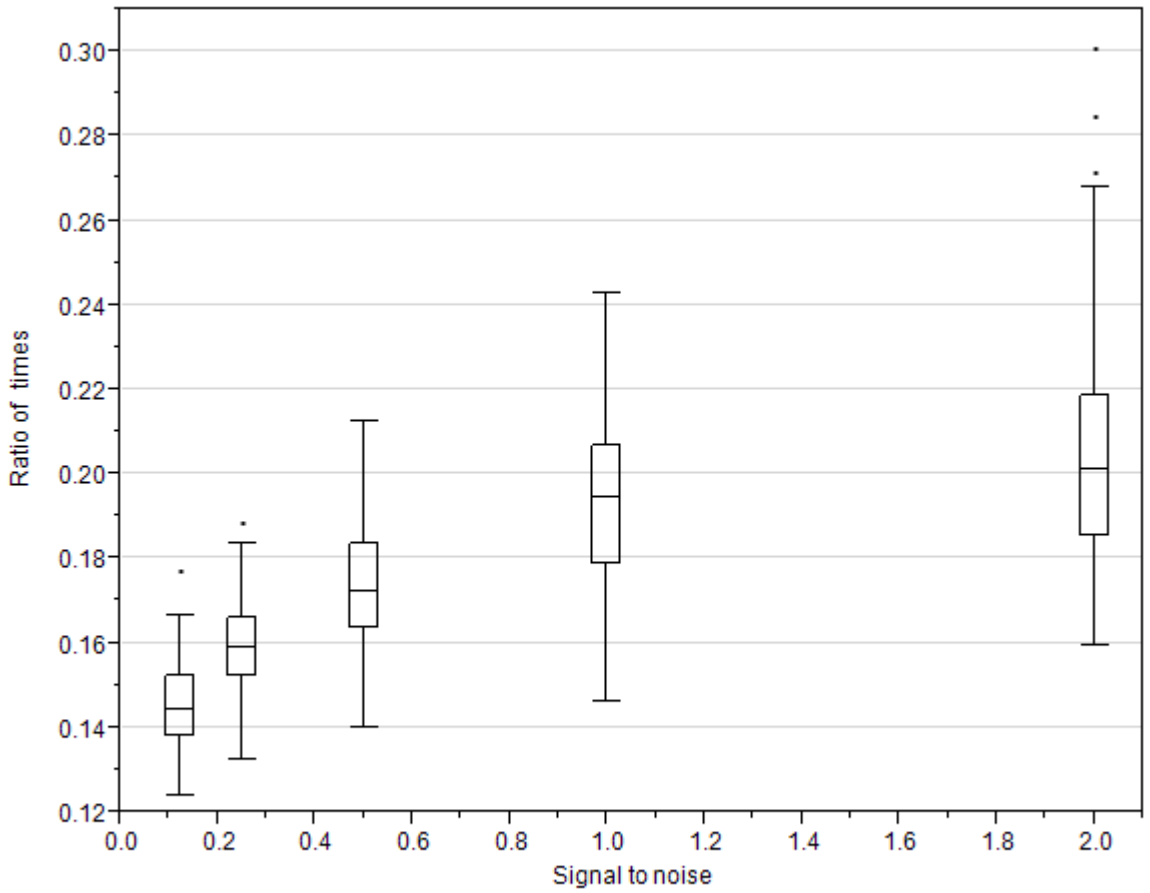


Figure 8.9 shows that the corresponding execution times of runs that use pruning are only 14% to about 22% of the execution times of runs that do not use pruning. The ratio of execution times increases as the signal-to-noise ratio increases. The ratio seems to level off with larger signals.

Figure 8.9 Ratio of Execution Times with Pruning to without Pruning



Predicting an Observation

To predict an observation, the HPFOREST procedure first assigns the observation to a single leaf in each decision tree in the forest, then uses that leaf to make a prediction based on the tree that contains the leaf, and finally simply averages the predictions over the trees. For an interval target, the prediction in a leaf equals the average of the target values among the bagged training observations in that leaf. For a nominal target, the *posterior probability* of a target category equals the proportion of that category among the bagged training observations in that leaf. The predicted nominal target category is the category with the largest posterior probability. In case of a tie, the first category that occurs in the training data is the prediction.

The HPFOREST procedure also computes out-of-bag predictions. The *out-of-bag prediction* of an observation uses only trees for which the observation is out of bag (that is, not selected as part of the training data for that tree).

A model is worthless if its predictions are no better than predictions without a model. For an interval target, the *no-model* prediction of an observation is the average of the target among training observations. For

a nominal target, the no-model posterior probabilities are the class proportions in the training data. The no-model predictions are the same for every observation.

Measuring Prediction Error

The HPFOREST procedure computes the average square error measure of prediction error. For a binary or nominal target, PROC HPFOREST also computes the misclassification rate and the log-loss.

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

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

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

$$\mathrm{MISC} = \sum_{i=1}^{N} \frac{1(y_i \neq \hat{y}_i)}{N}$$

$$\mathrm{LogLoss} = -\sum_{i=1}^{N} \sum_{j=1}^{J} \frac{\delta_{ij} \log(\ddot{p}_{ij})}{N}$$

where $\hat{y}_i$ is the target prediction of observation i, δ_{ij} equals 1 if the nominal target value j occurs in observation i or equals 0 of it does not occur there, $\hat{p}_{ij}$ is the predicted probability of nominal target value j for observation i, N is the number of observations, J is the number of nominal target values (classes), and $\ddot{p}_{ij}$ is $\hat{p}_{ij}$ truncated away from 0 and 1:

$$\ddot{p_{ij}} = \max(\min(\hat{p_{ij}}, 1 - 10^{-10}), 10^{-10})$$

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

Adjusting Statistics When Sampling Target Classes Unevenly

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

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

Formulas for Adjusting the Predictions and Fit Statistics

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

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

$$\widehat{p_{\pi j}} = \frac{p_{\tau j}\gamma_j}{\sum_k p_{\tau k}\gamma_k}$$

where

$\widehat{p_{\pi j}}$ = estimate of $p_{\pi j}$ that is computed with the training data
$p_{\pi j}$ = adjusted probability of class j for the population of interest
$p_{\tau j}$ = unadjusted probability of class j for the training data
$\gamma_j = \pi_j / \tau_j$
π_j = proportion of class j in the population of interest
τ_j = proportion of class j in the training data

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

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

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

$$\begin{aligned}
S_\tau(p_\tau) &= \sum_{i=1}^{N_\tau} \frac{\text{Loss}(y_i, p_{\tau i})}{N_\tau} \\
S_\tau(p_\pi) &= \sum_{i=1}^{N_\tau} \frac{\text{Loss}(y_i, p_{\pi i})}{N_\tau} \\
S_\pi(p_\tau) &= \sum_{i=1}^{N_\pi} \frac{\text{Loss}(y_i, p_{\tau i})}{N_\pi} \\
S_\pi(p_\pi) &= \sum_{i=1}^{N_\pi} \frac{\text{Loss}(y_i, p_{\pi i})}{N_\pi}
\end{aligned}$$

To compute the estimate of a population statistic $S_\pi(p_\pi)$, multiply the terms for target class j by the proportion of class j observations in the population as

$$\widehat{S_\pi} = \sum_j S_{\tau j}(\widehat{p_\pi})\pi_j$$

where

$$S_{\tau j}(\widehat{p_\pi}) = \sum_{i \ni y_i = j} \frac{\text{Loss}(y_i, \widehat{p_{\pi ij}})}{N_{\tau j}}$$
$$N_{\tau j} = \text{number of class } j \text{ observations in the training data}$$

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

$$\text{ASE}_{\tau j} = \sum_{i \ni y_i = j} \sum_{k=1}^{J} \frac{(\delta_{jk} - \hat{p}_{\pi ik})^2}{J N_{\tau j}}$$
$$\text{MISC}_{\tau j} = \sum_{i \ni y_i = j} \frac{1(j \neq \hat{y}_i)}{N_{\tau j}}$$

where

$$\delta_{jk} = 1 \text{ if } j = k \text{ and 0 otherwise}$$
$$\hat{y}_i = \text{the class } k \text{ with the largest value of } \widehat{p_{\pi ik}}$$

Why the Adjustments Matter

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

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

Figure 8.10 shows the misclassification rate of each tree, which is evaluated on the test data. A smooth curve passes near the average misclassification rate at each percentage of class 0 observations. (Technically, the curve is a cubic spline with λ equal to 0.05.) The rate is computed twice: once with the probabilities adjusted for the class proportions in the test data, and once without adjusting. A separate curve is drawn for each. The two rates are the same when the proportion of class 0 observations in the training data is 20 percent, which is the same as in the test data. The misclassification rates increase (get worse) as the class proportions in the training data differ more from the proportions in the test data. If you could choose the proportions of the target classes in the training data, this example suggests that the best choice would be to choose the same proportions as in the population of interest. However, if one target value is rare, oversampling would still be necessary to ensure that the training data have enough rare observations for the algorithm to work with.

In this example, adjusting probabilities is better because the rates that are computed after adjusting probabilities are generally smaller (better) than those computed without adjusting. The difference is larger when the class proportions in the training data differ more from those in the test data.

Figure 8.10 Test Set Misclassification Rates Using Adjusted and Unadjusted Predictions

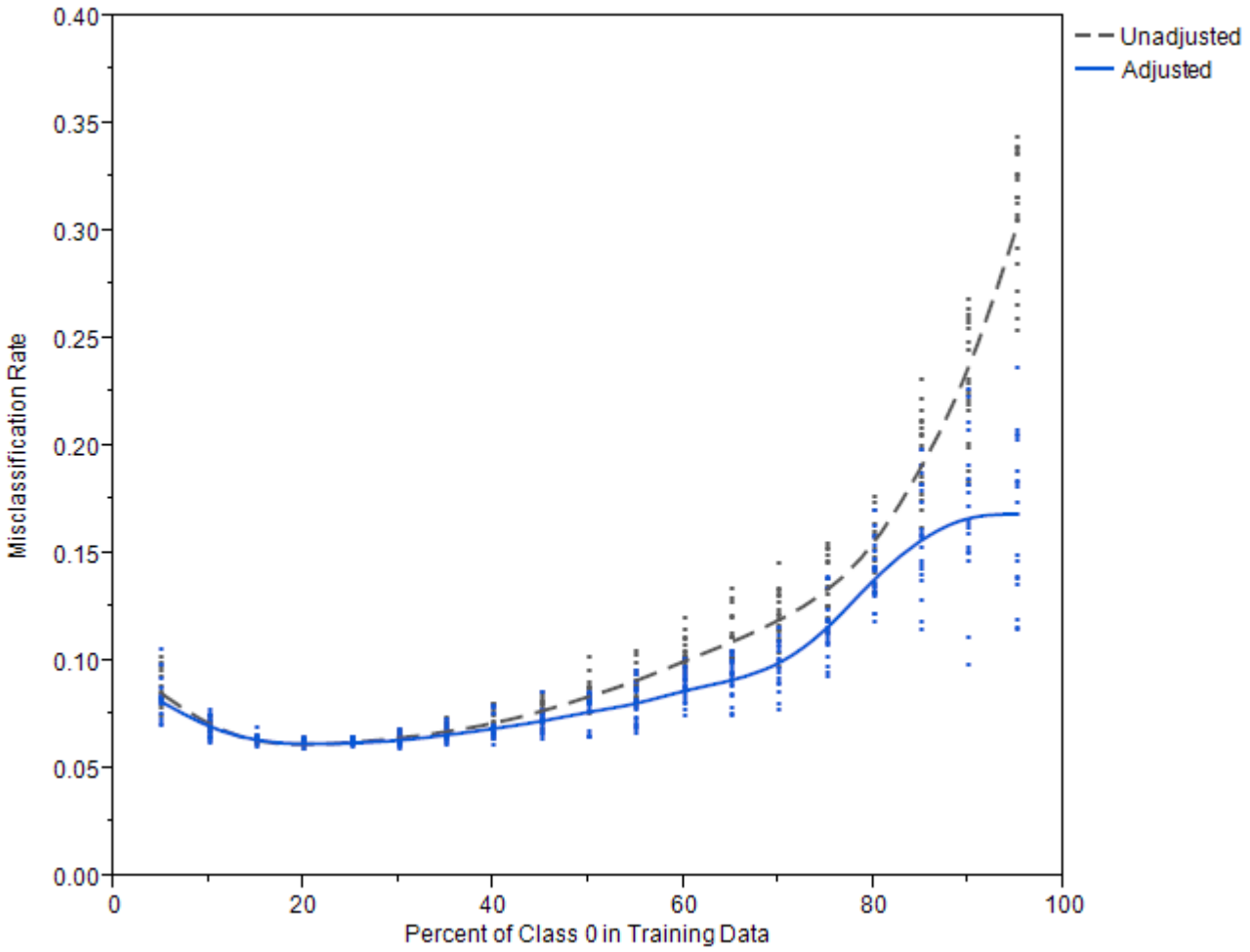


Figure 8.11 shows the average square error (ASE) for each tree, which is evaluated on the test data. A smooth curve passes near the average ASE for each percentage of class 0 observations. When the proportion of class 0 observations in the training data is 0.5 or less, adjusting the probabilities makes no difference to the ASE. For larger proportions of class 0, adjusting the probabilities results in a slightly larger (worse) ASE in this example.

Figure 8.11 Test Set Average Square Error Using Adjusted and Unadjusted Predictions

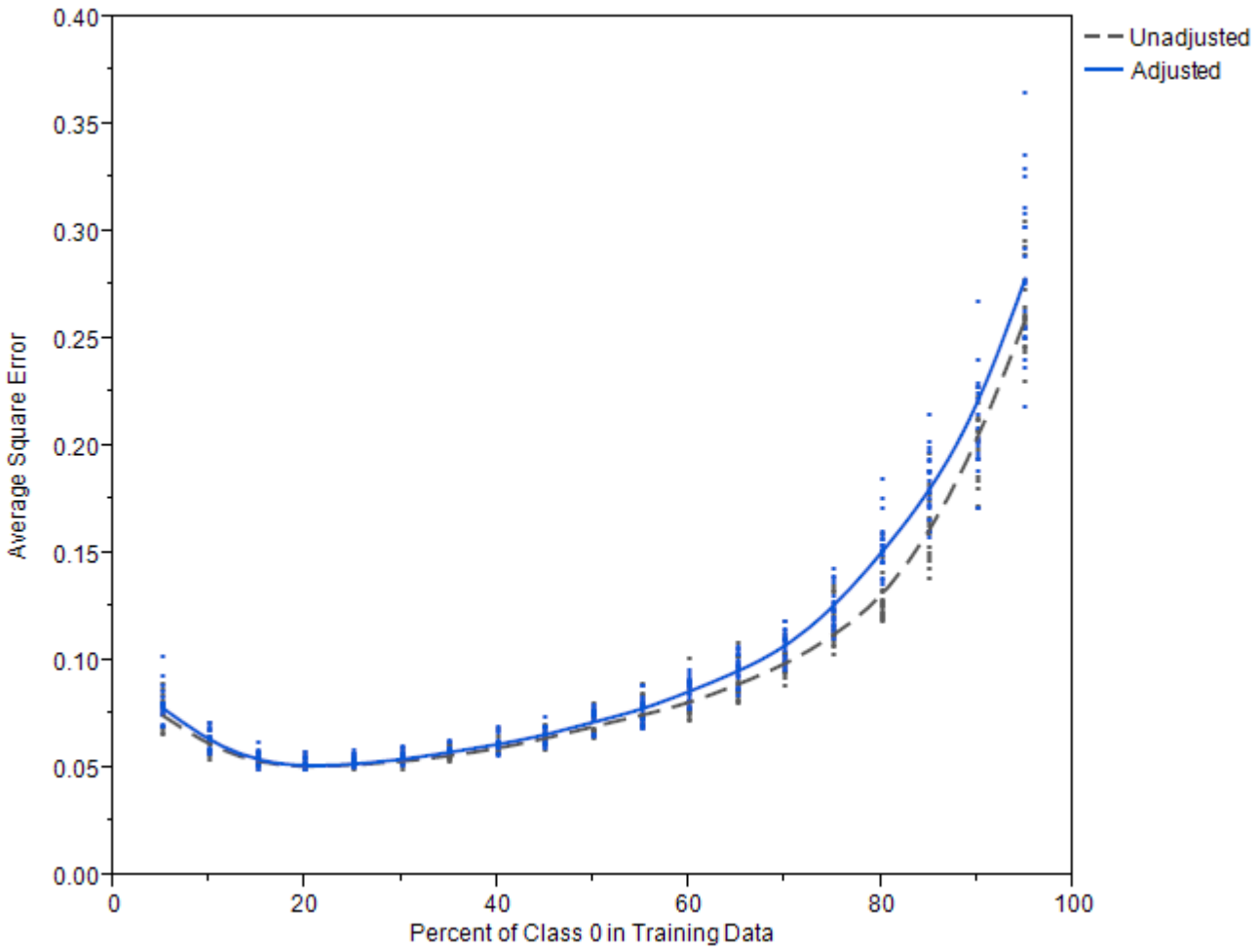


Figure 8.12 shows the misclassification rate that is computed using validation data and adjusted probabilities. The rate is estimated in two ways: one estimate adjusts only the probabilities, and the other adjusts both the statistics and the probabilities. The two are defined respectively as

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

$$\text{MISC}_{\text{adjusted}} = \sum_{j}^{J} \sum_{i \ni y_i = j} \frac{1(j \neq \hat{y}_i)}{N_{\tau j}} \pi_j$$

Only the last factor in each term is different. In Figure 8.12, the curve for $\text{MISC}_{\text{adjusted}}$ is indistinguishable from the curve for the test set misclassification rate. $\text{MISC}_{\text{adjusted}}$ predicts the test set rate perfectly in this example. On the other hand, the curve for $\text{MISC}_{\text{unadjusted}}$ is completely different and unreliable. Adjusting the misclassification rate is necessary in this example to get a good estimate of the rate in the test set.

Figure 8.12 Estimates of Test Set Misclassification Using Adjusted and Unadjusted Statistics

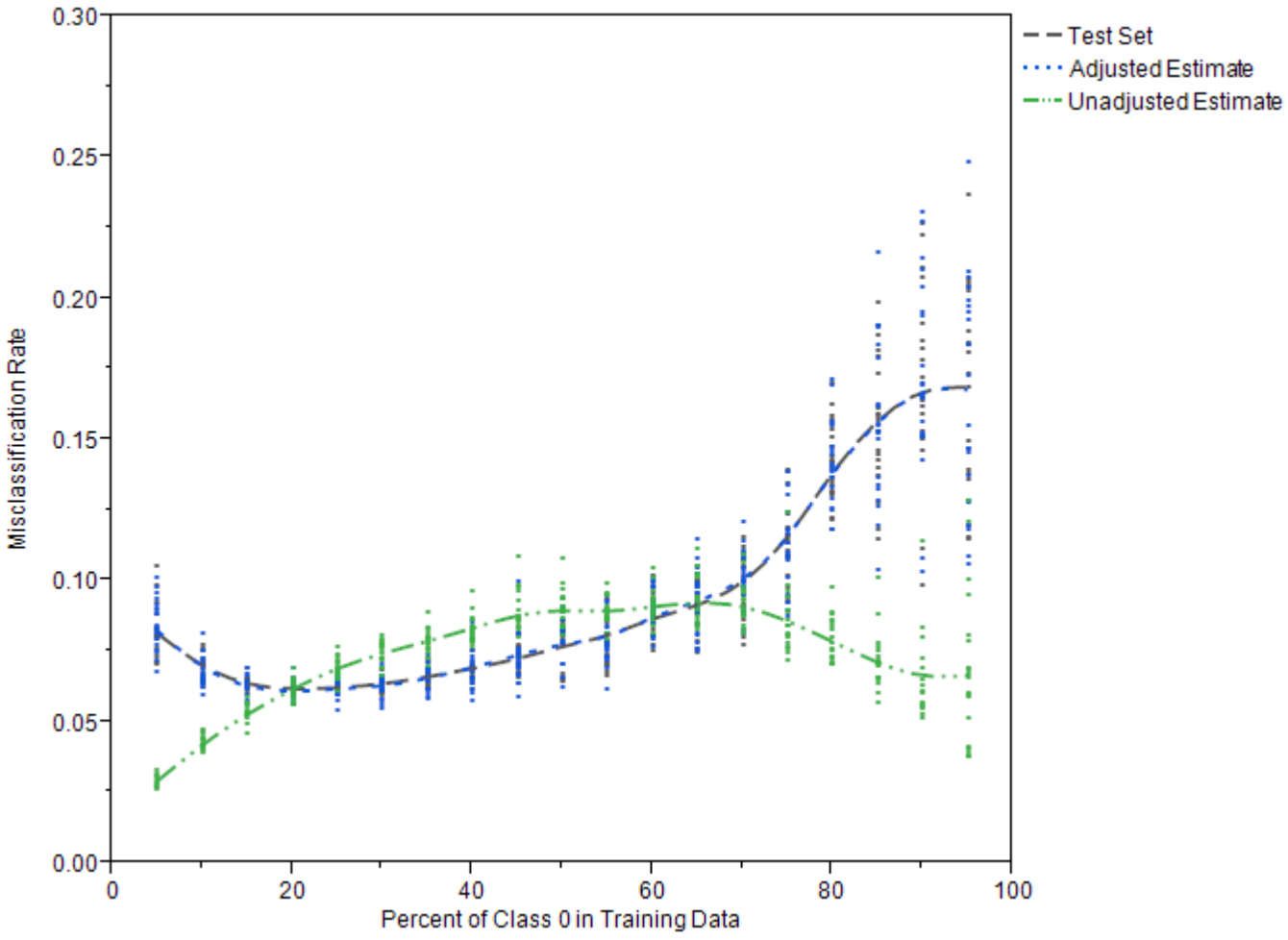
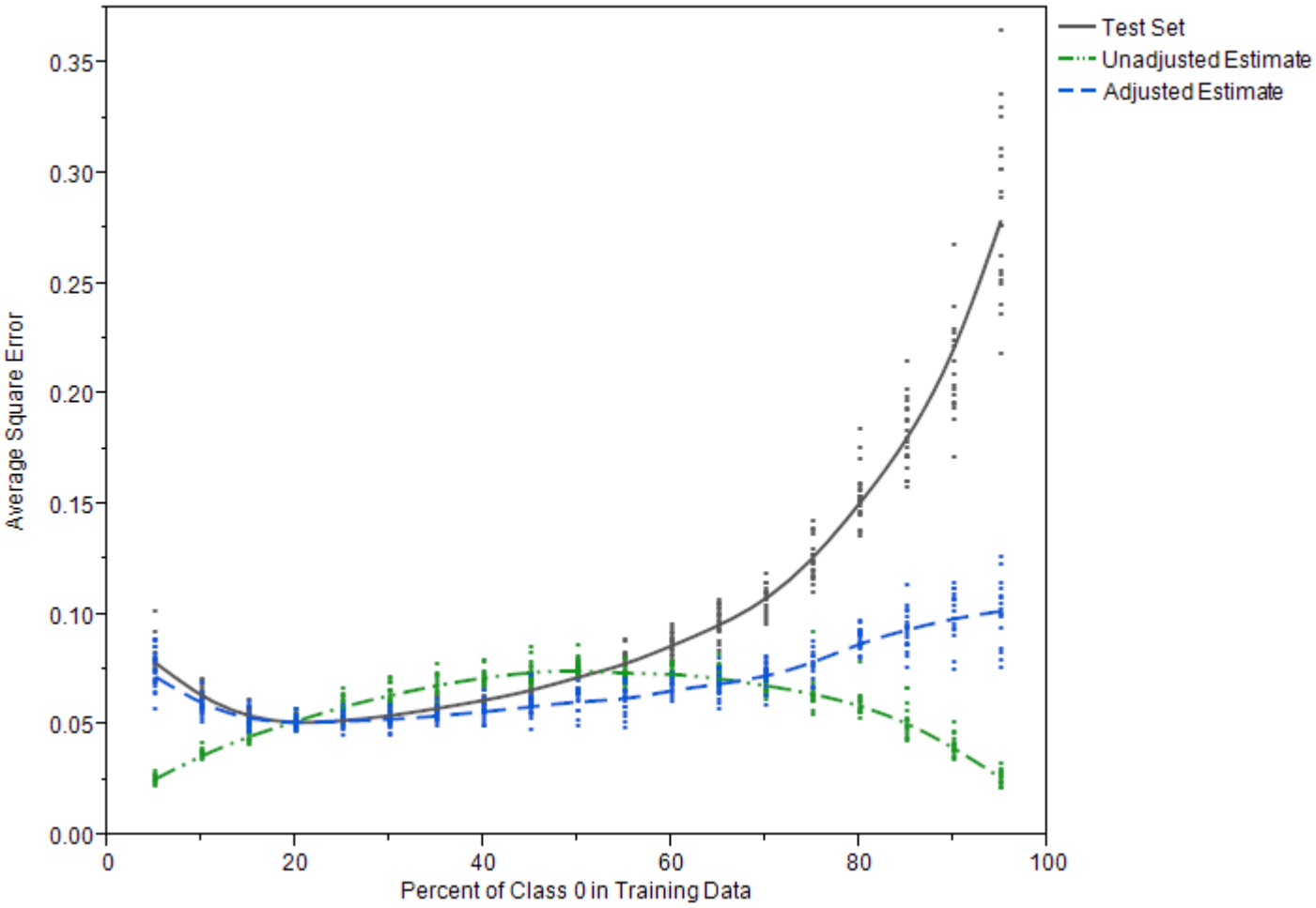


Figure 8.13 is the corresponding plot for ASE. The solid gray curve passes near the average ASE that is computed on the test data. The dashed blue curve that approximately follows the gray curve is the adjusted statistic that is computed on the validation data. The dashed green hill-shaped curve is the unadjusted statistic that is computed on the validation data. The evidence for adjusting the statistics for ASE is less compelling in this example than for the misclassification rate. For class 0 proportions between 45 and 65%, the unadjusted estimates of ASE are closer to the test set ASE than the adjusted estimates. For other class 0 proportions, the adjusted estimates are closer.

Figure 8.13 Estimates of Test Set ASE Using Adjusted and Unadjusted Statistics



Technical Derivations of Adjustment Formulas

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

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

$$\begin{aligned} P(Y_\pi = j|X_\pi) &= P(X_\pi|Y_\pi = j)\frac{P(Y_\pi=j)}{P(X_\pi)} \\ &= P(X_\tau|Y_\tau = j)\frac{P(Y_\pi=j)}{P(X_\pi)} \\ &= P(Y_\tau = j|X_\tau)\frac{P(X_\tau)}{P(Y_\tau=j)}\frac{P(Y_\pi=j)}{P(X_\pi)} \\ &= P(Y_\tau = j|X_\tau)\frac{P(Y_\pi=j)}{P(Y=j)}\frac{P(X_\tau)}{P(X_\pi)} \\ &= P(Y_\tau = j|X_\tau)\frac{P(Y_\pi=j)}{P(Y_\tau=j)}\alpha \end{aligned}$$

The sum of the class probabilities equals 1:

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

Therefore,

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

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

$$\begin{aligned} E_\pi(\text{Loss}(Y_\pi, p_\pi)) &= \iint \text{Loss}(Y_\pi, p_\pi)p(X_\pi, Y_\pi)\,dX_\pi\,dY_\pi \\ &= \sum_j \int \text{Loss}(Y_\pi, p_\pi)p(X_\pi|Y_\pi = j)dX_\pi P(Y_\pi = j) \\ &= \sum_j \int \text{Loss}(Y_\tau, p_\pi)p(X_\tau|Y_\tau = j)dX_\tau P(Y_\pi = j) \\ &= \sum_j E_{\tau j}(\text{Loss}(Y_\tau, p_\pi))P(Y_\pi = j) \end{aligned}$$

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

Handling Missing Values

Strategies

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

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

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

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

When missing values are excluded from the split search, a new policy is needed for assigning an observation with missing values to a branch. Beginning with SAS Enterprise Miner 14.1, PROC HPFOREST assigns such observations to the branch that contains the most observations from the sample that was used during the split search. In previous releases, PROC HPFOREST distributed observations that had a missing value into both branches. Although that approach was better for statistics, it is no longer implemented because it took too long to train and score observations.

Specifics

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

The discussion in this section applies to each candidate variable and each node separately. For example, the test of association that uses input variable X might use all observations, and the test that uses input variable Z might ignore some observations because of missing values. The test that uses X might use all observations in one node but not all observations in another node.

Handling Values That Are Absent from Training Data

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

Modeling Incremental Response from a Treatment

The *incremental response modeling* (IRM) method of analysis predicts how much a target value would change if a treatment were applied to the observation. For example, a conventional model might predict the probability of someone purchasing an item. An incremental response model would predict how much the probability of purchase would increase if the person were to receive a promotional coupon.

PROC HPFOREST provides an experimental implementation that uses a splitting criterion described in Su et al. (2009) and Radcliffe and Surry (2011). The treatment variable must be binary, and the target must have a binary or interval level of measurement.

A treatment variable differs from an input variable because the treatment is assigned at the discretion of some outside agent. The treatments must be assigned in such a way that input values associated with the response are balanced between the two treatment groups. If the data are not balanced, the predictions from many incremental response algorithms, including PROC HPFOREST, can be invalid. For example, if high values of an input X tend to produce high values in a target and a treatment value T is disproportionally assigned to observations that have high values of X, then PROC HPFOREST would incorrectly credit T with high values of the target.

Let T and C denote the two values of the treatment, and let L and R denote two branches of a tree node into which an observation can be assigned. PROC HPFOREST finds a split with the largest value of ζ^2, where

$$\begin{aligned} \zeta^2 &= \frac{(\Delta\Delta p)^2}{\mathrm{SE}^2} \\ \Delta\Delta p &= |(p_{TL} - p_{CL}) - (p_{TR} - p_{CR})| \\ \mathrm{SE}^2 &= \hat{\sigma}^2(1/N_{TL} + 1/N_{CL} + 1/N_{TR} + 1/N_{CR}) \end{aligned}$$

For a binary target,

$$\hat{\sigma}^2 = \frac{\sum_{t\in T,C}\sum_{b\in L,R} N_{tb}\, p_{tb}(1 - p_{tb})}{N - 4}$$

For an interval target,

$$\hat{\sigma}^2 = \frac{\sum_{t\in T,C}\sum_{b\in L,R}(N_{tb} - 1)\sum_{X_i\in tb}(X_i - \bar{X}_{tb})}{N - 4}$$

Increasing Accuracy by Increasing Tree Size

A common mistake is to build trees that are too shallow. Forests usually benefit from some overfitting of the individual trees. Forests with larger trees are usually more accurate than ones with smaller trees, although the benefit of increasing size diminishes eventually. In some cases, as with low signal-to-noise data, pruning the trees can improve accuracy.

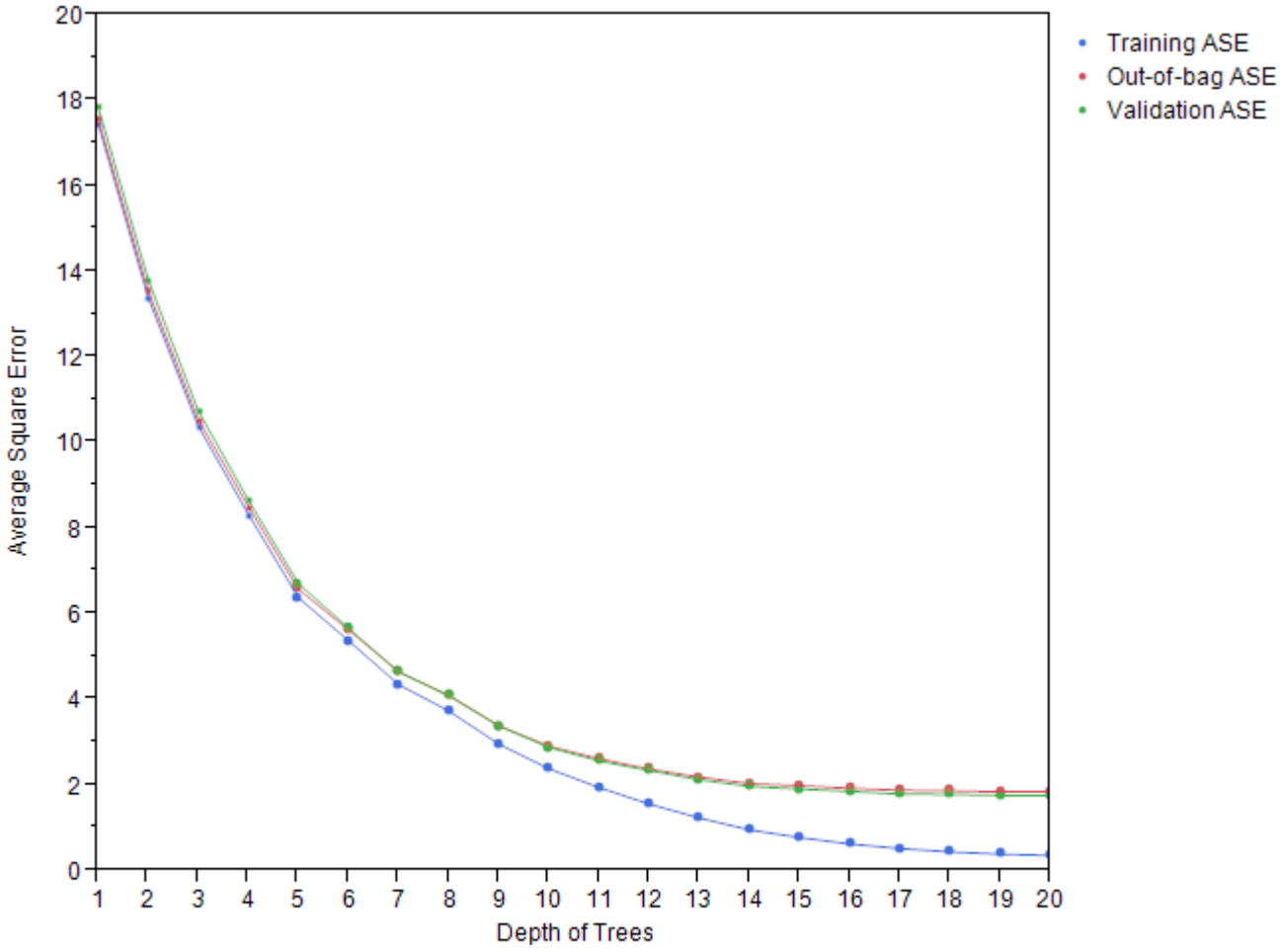
The principle is illustrated by increasing the depth of the tree. The data consist of 50,000 and 25,000 observations for training and validation, respectively. These observations are generated by the relationship,

$$Y = 10\sin(\pi X_1 X - 2) + 20(X_3 - 1/2)^2 + 10X_4 + 5X_5 + \epsilon$$

where X_i is uniform between 0 and 1 and ϵ is normally distributed with mean 0 and variance 1. The relationship originally appeared as equation 61 in Friedman (1991).

Figure 8.14 shows the average square error of the prediction versus the values of the MAXDEPTH= option. The bottom, blue line represents the errors from the training data. The green and red lines represent errors from the out-of-bag and validation data, respectively. The average square error in the out-of-bag and validation data decreases with depth and then begins to level in this example.

Figure 8.14 ASE versus Depth of Trees



Using Distributed Data Wisely

Data are said to be distributed when the observations do not all exist on a single machine. Jobs run faster with distributed data because more computer processors are used and the burden on an individual processor is limited to the observations on its single machine. PROC HPFOREST creates a single decision tree with data on a single machine. When the data are distributed, only the observations on a specific machine are available to create the trees on that machine. Accuracy for this approach might be worse than when trees are created by using all the data because using more data usually results in larger trees and greater accuracy. However, after some minimal number of observations, the gain in accuracy might not warrant the extra computational effort. PROC HPFOREST can copy some or all observations from one machine to all other machines in order to increase the number of observations on each machine.

You can use the GRIDCOPY= option to specify whether to copy any observations. You can copy all observations (GRIDCOPY=ALL), all training observations (GRIDCOPY=TRAINING), or no observations(GRIDCOPY=NONE). You can also specify GRIDCOPY=MINIMAL, which requests that a minimum number of observations be copied to establish a minimum number of observations on each machine, and, for a nominal target, a minimum number of observations for each target value. The GRIDNODESIZE= and GRIDCLASSSIZE= options specify these minimums, respectively.

Increasing the value of the MAXTREES= option to a multiple of the number of machines that contain data is good practice because accuracy can improve without taking any more time: if one machine is creating T trees, each machine should create T trees or some machine will become idle and wait for the others to finish.

When a single machine does not contain all observations, PROC HPFOREST outputs baseline statistics for the training data on each machine. (These statistics are set to 0 on machines that have no trees.) The statistics are useful for detecting whether the data differ between machines. If large differences exist, then the training data on a single machine might not be representative of the data on which the forest will be applied. The generalization accuracy of the forest would be degraded.

The NODES= option in the PERFORMANCE statement specifies the number of machines to use.

Measuring Variable Importance

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

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

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

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

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

reason Breiman introduced loss reduction. A more recent method called random branch assignments (RBA) (Neville and Tan 2014) uses the same theory as Strobl's method but runs as fast as Breiman's method. PROC HP4SCORE uses the RBA method.

Loss Reduction

Loss reduction is also called Gini increase, Gini importance, or impurity reduction. It was introduced in Breiman et al. (1984) for decision trees, later modified for gradient boosting machines (Friedman 2001), and later used in forests (Breiman and Cutler 2003).

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

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

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

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

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

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

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

where

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

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

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

where

$$\text{SAE}(\omega) = \sum_{i=1}^{N(\omega)} |Y_i - \hat{Y}(\omega)|$$

For a categorical target, the formula for $\text{SSE}(\omega)$ reduces to

$$\text{SSE}(\omega) = \begin{cases} N(1 - \sum_{j=1}^{J} \hat{p}_j^2) & \text{for training data} \\ N(1 - \sum_{j=1}^{J} (2p_j - \hat{p}_j)\hat{p}_j) & \text{for validation data} \end{cases}$$

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

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

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

where

$$\text{SNM}(\omega) = -\sum_{j=1}^{J} N_j(\hat{p}_j - \max_{k \neq j} \hat{p}_k)$$

and N_j is the number of class j observations in ω in the data set being used to evaluate the variable importance.

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

Breiman's Method

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

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

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

Bias and Correlation

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

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

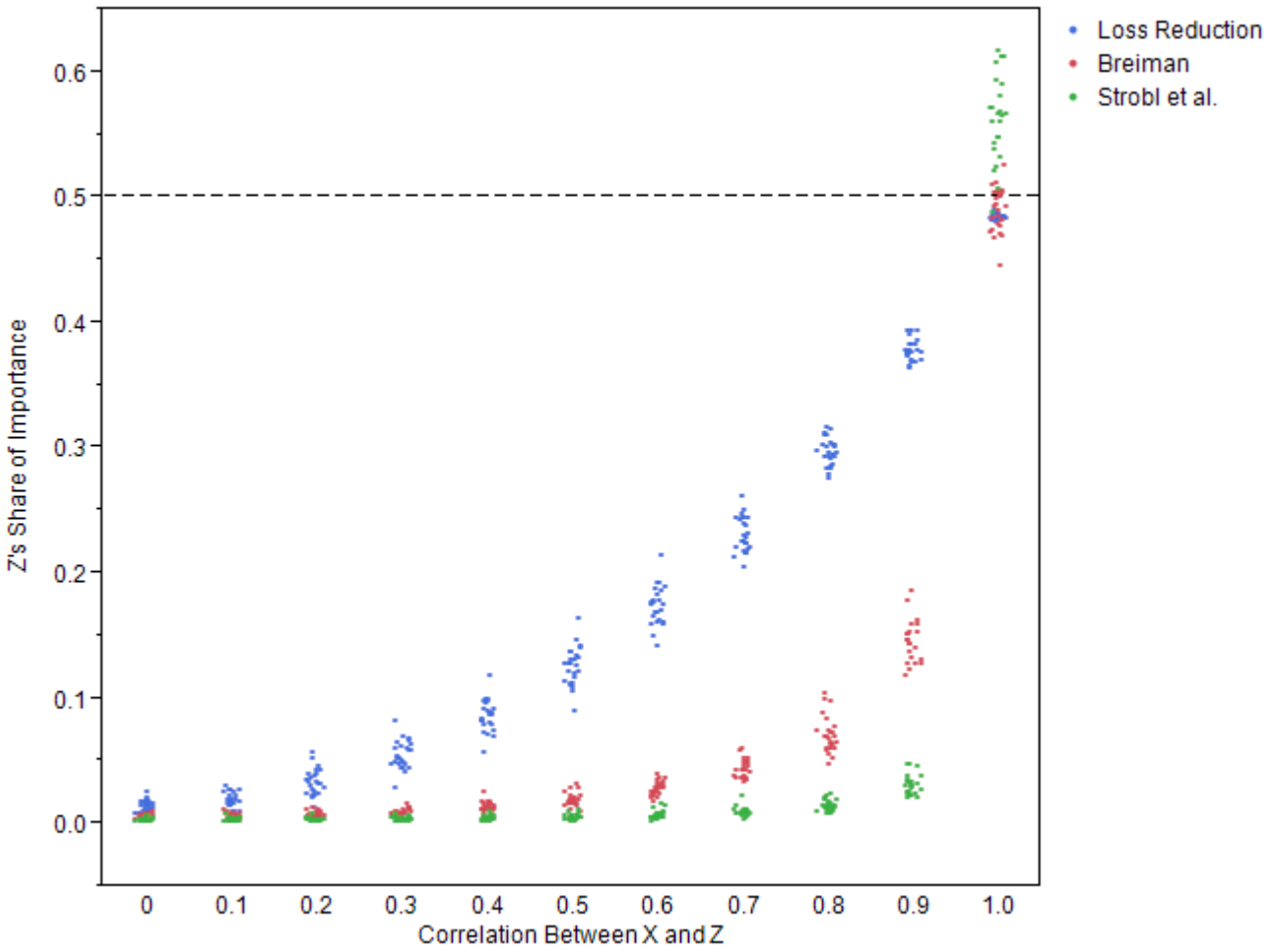
Conditional and Marginal Importance

The difference between loss reduction and Breiman's method can be illustrated with a simple example. Suppose X and Z are correlated and $Y = X$. Should Z be assigned any importance? One answer is no: Z is not even in the formula that generates Y. The other answer is yes: observing Z provides information about Y, and therefore Z is helpful in explaining the variation of Y. An input has *conditional* importance if it is needed for prediction even after values of the other variables are given; Z has no conditional importance. An input has *marginal* importance if it is predictive of Y by itself; Z has marginal importance.

The degree to which correlations determine the final importance values depends both on the algorithm for importance and on the algorithmic parameters for the model. In the current example, if VARS_TO_TRY=2, then X and Z compete in every node, PROC HPFOREST selects X to split almost all the nodes, and all importance measures assign Z negligible importance. If VARS_TO_TRY=1, then PROC HPFOREST must use Z to split some nodes, and it assigns some importance to Z. In general, loss reduction assigns more importance to correlated variables than Breiman's method, and Breiman's method assigns more importance than a conditional permutation method that is introduced in Strobl et al. (2008) and summarized in the section "Strobl's Method" on page 180.

Figure 8.15 shows the result from generating 25 samples at each of several correlation values that range from 0 to 1. Each sample has 500 bivariate correlated normal observations. PROC HPFOREST is run with VARS_TO_TRY=1 and outputs the proportion of total importance that is assigned to Z. Figure 8.15 has a point for each of three measures of importance that are evaluated on each sample. Unless the correlation equals 1, Breiman's method and Strobl's method assign little importance to Z. All methods choose X as the most important variable unless they are perfectly correlated.

Figure 8.15 Z's Proportion of Total Importance



Archer and Kimes (2008) present a simulation that compares the variable importance methods. To emulate genomic data, correlated variables appear in groups, and at most one variable in a correlated group is in the regression equation that generates the target. Both forest methods are about as good as using a method of variable importance from regression. This is consistent with Figure 8.15, which deems X the most important variable by every method and every correlation value except a correlation of 1.

Strobl's Method

Strobl et al. (2008) obtain a more conditional measure of importance by analyzing the logic of Breiman's method and revealing an omission. The analysis centers around statistical independence. If Y and Z are independent, then for sets B and C,

$$P((Y \in B) \cap (Z \in C)) = P(Y \in B)P(Z \in C)$$

A bivariate histogram of the data would be approximately equal to the product of univariate histograms. If the values of Z are permuted, the univariate histogram of those values would remain unchanged and so would the bivariate histogram. On the other hand, if permuting the values of Z does significantly change the bivariate histogram, then Y has some dependence on Z.

Strobl et al. say that this is the essence of Breiman's logic and that this logic ignores the influence of other variables. To incorporate the other variables, the reasoning should proceed from conditional independence:

$$P((Y \in B) \cap (Z \in C) \mid X \in A) = P(Y \in B \mid X \in A)P(Z \in C \mid X \in A)$$

Probabilities that involve Y and Z are computed separately for separate values of X. When the values of Z are permuted, only values from observations that have the same value of X are permuted. Strobl et al. develop an algorithm around conditional independence. The resulting variable importance measure is closer to the conditional end of the spectrum than Breiman's.

The method of Strobl et al. works well for data that contain a few hundred observations, but it becomes prohibitively slow for hundreds of thousands of observations. Let V denote the number of variables, and let T denote the number trees. Then Breiman's method requires approximately $VT/3$ permutations. Strobl's method multiplies that number by some proportion of the number of observations.

Random Branch Assignments

Neville and Tan (2014) present a simple algorithm, called random branch assignments (RBA), that satisfies the logic of Strobl et al. (2008) and avoids all the permutations. When the trees are created, the number of observations in each node is saved. The statistical distribution of training data must be similar to that of the data with which variable importance is being evaluated because the algorithm assumes that the observation sizes that are saved in each node are proportional to the number of observations in the evaluation data that visit the node.

To compute the importance of a variable Z, randomize the branch assignment rules that involve Z and then apply the randomized model to the data and compute a goodness-of-fit measure. The randomized rule is one that randomly assigns an observation to a branch with a probability that is proportional to the number of observations in the branch. For example, suppose a node that contains 100 training observations is split by values of Z into two nodes: one contains 25 training observations and the other contains 75. When the importance of Z is evaluated, an observation that reaches the node is randomly assigned to the smaller branch with probability 0.25. As in the Breiman method, the importance of Z is proportional to the randomized fit minus the fit without randomization.

Neville and Tan claim that RBA satisfies the objectives of the methods of Breiman and of Strobl et al. The purpose of permuting the values is to break any relationship between the response variable and Z without changing the univariate distribution of Z. The same thing can be accomplished by replacing an observation value with a value that is randomly chosen from the univariate distribution. The branch assignment rule lumps together the values that are assigned to the same branch. To assign a branch to an observation, all that is needed is the probability that Z is in a lump, not the probabilities of the individual values. The RBA method uses the lump probabilities that are conditioned on arriving at the node that is being split, which is exactly the conditional requirement of Strobl et al.

Preferences

Breiman's method and variants are generally not used in practice. One reason is the long running time needed to score every observation for every variable that is evaluated. A more important reason is the lack of a convincing example where Breiman's method succeeds and others fail. The comparison in the previous section applies only to forests because only in forests can the VARS_TO_TRY= option be set. No comparison of importance measures is published with the VARS_TO_TRY= option equal to all the variables.

Some authors insist on using Breiman's method. Berk (2008) says that importance must be measured outside of the procedure that is used to measure it. Otherwise, it is not a practical measure. It simply restates a part

of the model itself without reference to the practical reason for creating the model. Nicodemus and Malley (2009) present plots that clearly show loss reduction as hopelessly biased, while Breiman's method gives the correct results on the same data. Actually, they are not the same data. All these authors use training data with loss reduction and holdout data with Breiman's method. Berk even says loss reduction is a fit measure, and as such should be used with the training data. This is misguided. Using holdout data to evaluate a predictive model is generally recommended. Computing loss reduction with both training and validation data can reveal which inputs are fooling the training algorithm, and corrective action can be taken.

Displaying the Output

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

Performance Information

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

Model Information

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

Number of Observations

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

Baseline Fit Statistics

The "Baseline Fit Statistics" table contains fit statistics that are calculated without a model. Compare the baseline statistics with the model fit statistics to determine how beneficial the model is. Fit statistics are described in the section "Measuring Prediction Error" on page 164. The table is produced by default.

Fit Statistics

The "Fit Statistics" table contains statistics that measure the model's goodness of fit. The fit of the model to the data improves with the number of trees in the forest. Successive rows in the table contain fit statistics for a forest that has more trees. The SKIP_SEQ_ROWS= option controls how many more trees successive rows represent. Compare these fit statistics with the baseline fit statistics to determine how beneficial the model is. Fit statistics are described in the section "Measuring Prediction Error" on page 164. The table is produced by default.

Baseline Grid Statistics

The "Baseline Grid Statistics" table contains baseline fit statistics of the training data on each grid node when the data are distributed over multiple machines. If the statistics are markedly different on different machines, predictions of new data might be worse than if the data on different machines were similar. The statistics are not computed in single-machine mode nor when all the data are copied to all the machines.

Loss Reduction Variable Importance

The "Variable Importance" table displays variable importance based on loss reduction, which is explained in the section "Loss Reduction" on page 177. The table is produced by default. If you do not want this table, then specify IMPORTANCE=NO to turn off the calculation and conserve some memory resources.

ODS Table Names

Table 8.4 lists the names of the data tables created by the HPFOREST procedure. Use these names in ODS statements.

Table 8.4 ODS Tables Produced by PROC HPFOREST

Table Name	Description	Required Statement and Option
PerformanceInfo	Performance information	Default output
NObs	Number of observations	Default output
Baseline	Fit statistics without a model	Default output
FitStatistics	Fit statistics from the model	Default output
GridStatistics	Baseline statistics by grid node	Default for distributed data
VariableImportance	Loss reduction variable importance	Default output
ModelInfo	Model information	Default output
Timing	Absolute and relative times for tasks performed by the procedure	PERFORMANCE DETAILS

Examples: HPFOREST Procedure

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

Example 8.1: Out-Of-Bag Estimate of Misclassification Rate

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

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

The following SAS statements create a SAS data set from an URL:

```
data spambase;
   %let url=//archive.ics.uci.edu/ml/machine-learning-databases;
   infile "http:&url/spambase/spambase.data"
          device=url delimiter=',';
   input wf_make wf_adress wf_all wf_3d wf_our
         wf_over wf_remove wf_internet wf_order wf_mail
         wf_receive wf_will wf_people wf_report wf_addresses
         wf_free wf_business wf_email wf_you wf_credit
         wf_your wf_font wf_000 wf_money wf_hp
         wf_hpl wf_george wf_650 wf_lab wf_labs
         wf_telnet wf_857 wf_data wf_415 wf_85
         wf_technology wf_1999 wf_parts wf_pm wf_direct
         wf_cs wf_meeting wf_original wf_project wf_re
         wf_edu wf_table wf_conference
         cf_semicolon cf_parenthese cf_bracket cf_exclamation
         cf_dollar cf_pound
         average longest total
         spam;
run;
```

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

data fitstats;
   set fitstats;
   label Trees = 'Number of Trees';
   label MiscAll = 'Full Data';
   label Miscoob = 'OOB';
run;
```

```
proc sgplot data=fitstats;
   title "OOB vs Training";
   series x=Trees y=MiscAll;
   series x=Trees y=MiscOob/lineattrs=(pattern=shortdash thickness=2);
   yaxis label='Misclassification Rate';


run;
title;
```

Output 8.1.1 Plot of OOB versus Training Misclassification Rate

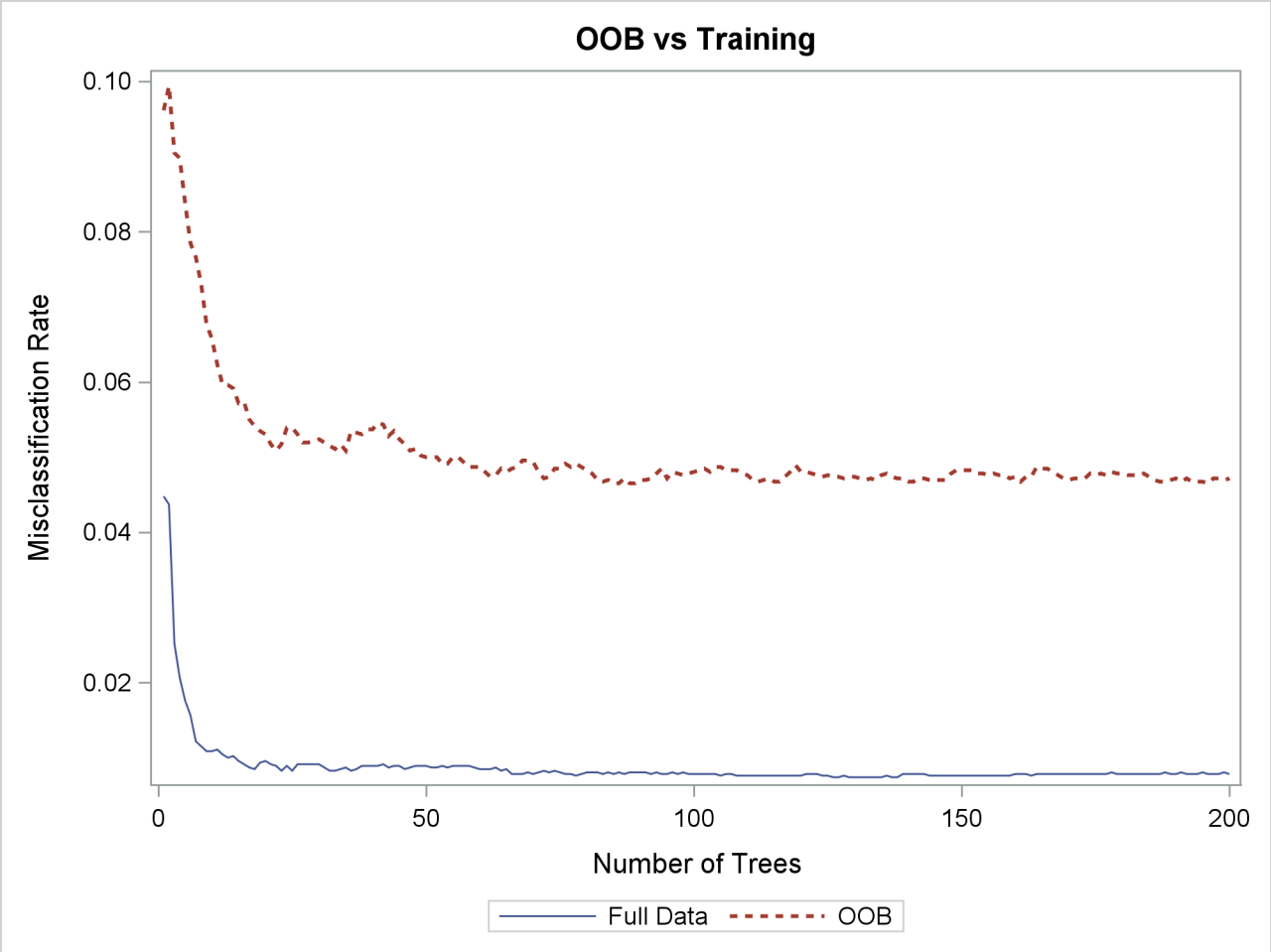


Figure 8.1.1 shows the misclassification rate is worse (larger) based on the out-of-bag (OOB) data, and more trees are needed for the out-of-bag rates to level off. Both characteristics are typical of a forest.

Example 8.2: Number of Variables to Try When Splitting a Node

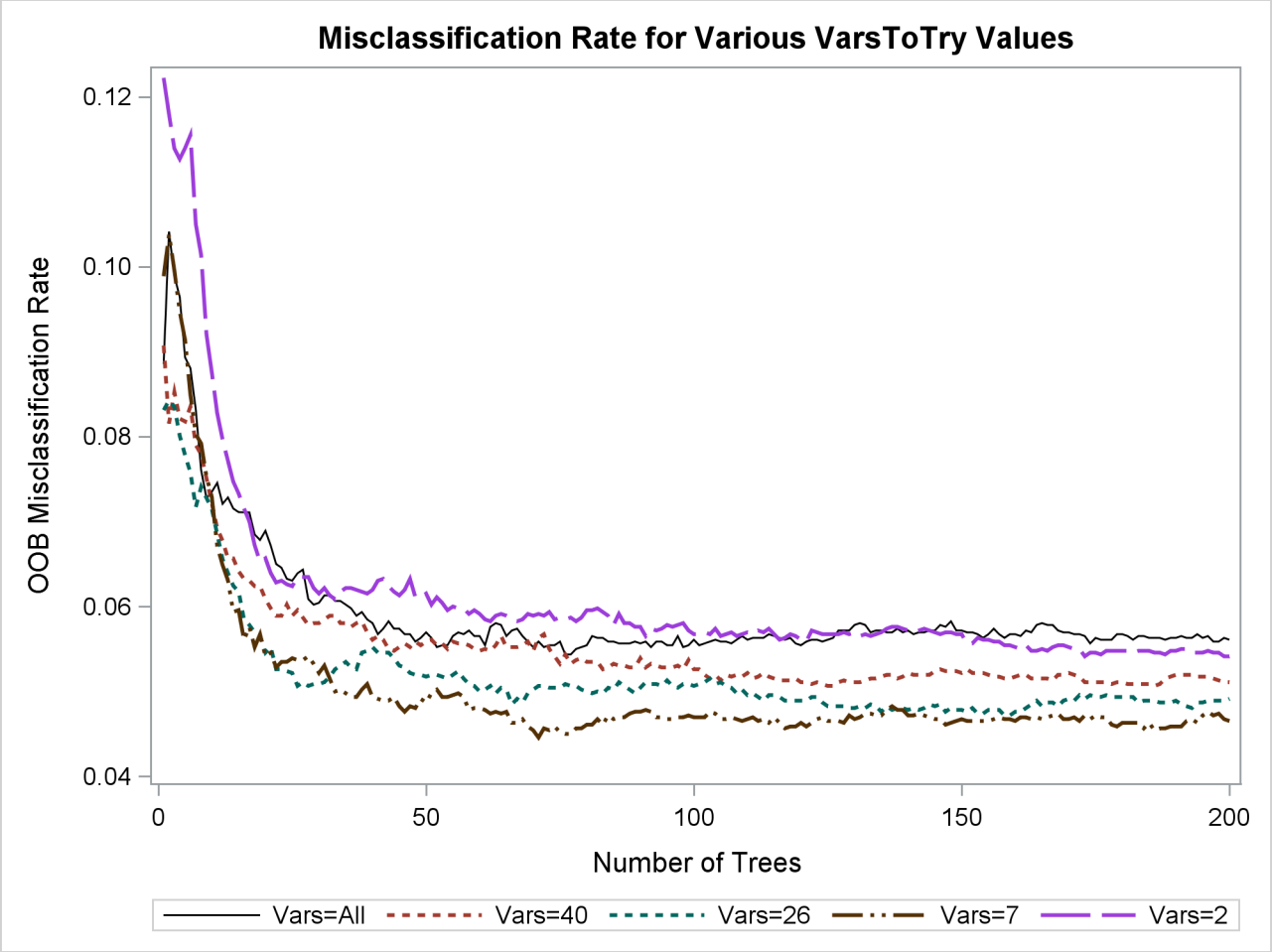
This example illustrates the effect of changing the number of variables to randomly select as candidate splitting variables in a node. In each node in each tree, *m* variables are randomly selected to be candidates to split on. Use the VARS_TO_TRY= option to specify *m*. Specifying *m* less than the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

The following SAS statements create a SAS data set from an URL:

```
data spambase;
     %let url=//archive.ics.uci.edu/ml/machine-learning-databases;
     infile "http:&url/spambase/spambase.data"
            device=url delimiter=',';
     input wf_make wf_adress wf_all wf_3d wf_our
           wf_over wf_remove wf_internet wf_order wf_mail
           wf_receive wf_will wf_people wf_report wf_addresses
           wf_free wf_business wf_email wf_you wf_credit
           wf_your wf_font wf_000 wf_money wf_hp
           wf_hpl wf_george wf_650 wf_lab wf_labs
           wf_telnet wf_857 wf_data wf_415 wf_85
           wf_technology wf_1999 wf_parts wf_pm wf_direct
           wf_cs wf_meeting wf_original wf_project wf_re
           wf_edu wf_table wf_conference
           cf_semicolon cf_parenthese cf_bracket cf_exclamation
           cf_dollar cf_pound
           average longest total
           spam;
  run;
```

```
%macro hpforest(Vars=);
proc hpforest data=spambase maxtrees=200
   vars_to_try=&Vars.;
   input w: c: average longest total/level=interval;
   target spam/level=binary;
   ods output
   FitStatistics = fitstats_vars&Vars.(rename=(Miscoob=VarsToTry&Vars.));
run;
%mend;

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

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

proc sgplot data=fitstats;
   title "Misclassification Rate for Various VarsToTry Values";
   series x=Trees y = VarsToTryAll/lineattrs=(Color=black);
   series x=Trees y=VarsToTry40/lineattrs=(Pattern=ShortDash Thickness=2);
   series x=Trees y=VarsToTry26/lineattrs=(Pattern=ShortDash Thickness=2);
   series x=Trees y=VarsToTry7/lineattrs=(Pattern=MediumDashDotDot Thickness=2);
   series x=Trees y=VarsToTry2/lineattrs=(Pattern=LongDash Thickness=2);
   yaxis label='OOB Misclassification Rate';
run;
title;
```

Output 8.2.1 Effect of the VARS_TO_TRY= Option on the Misclassification Rate



Specifying a value of 7 or 26 for the VARS_TO_TRY= option results in a more accurate forest than would occur without random selection of variables (VARS_TO_TRY=ALL). Specifying VARS_TO_TRY=2 is no better than specifying VARS_TO_TRY=ALL. A good value for the VARS_TO_TRY= option depends on the data. In this example, the HPFOREST procedure uses a default value of $\sqrt{58} = 7$, which worked well.

Example 8.3: Fraction of Training Data to Train a Tree

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

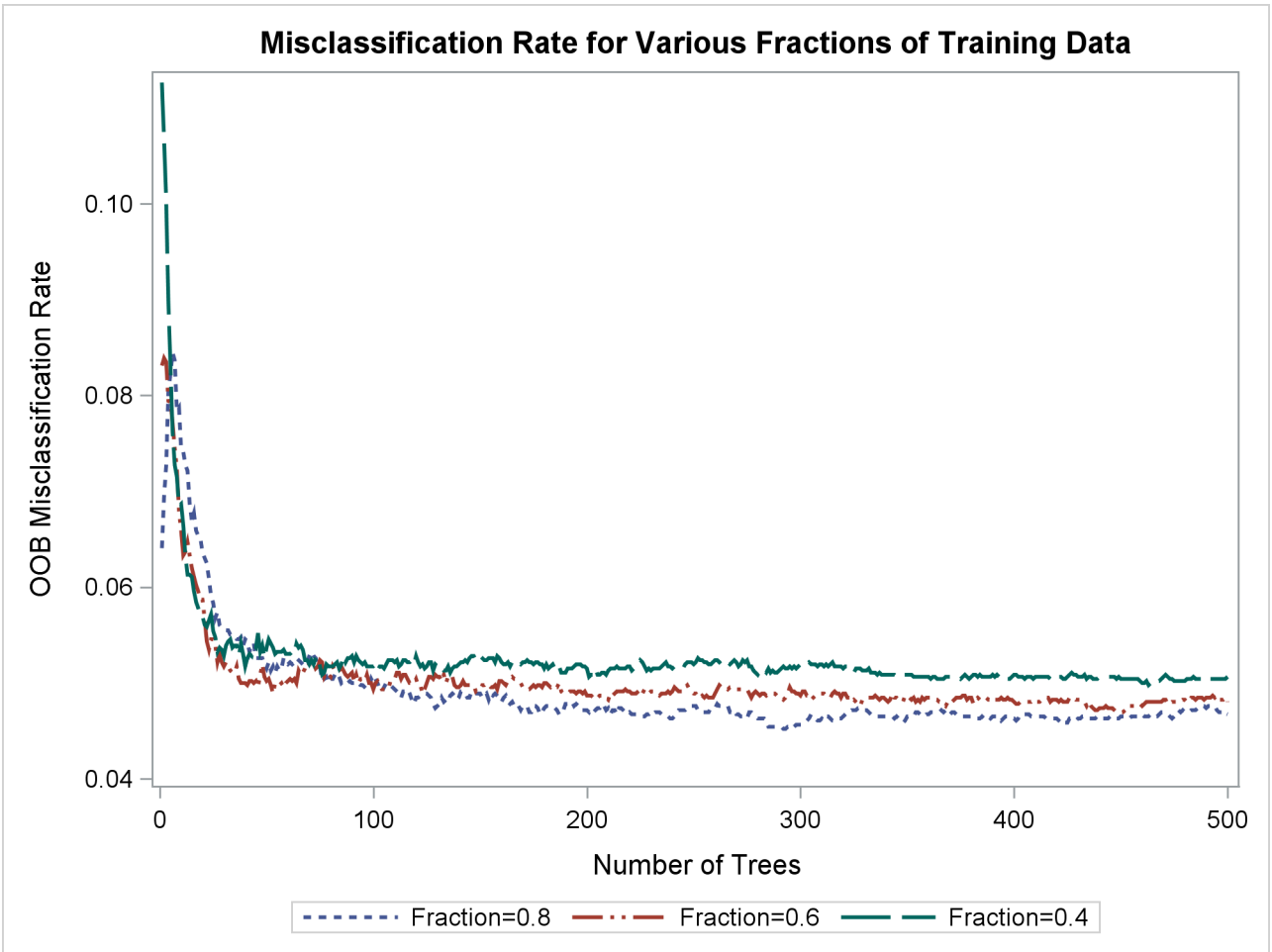
The following SAS statements create a SAS data set from an URL:

```
 data spambase;
      %let url=//archive.ics.uci.edu/ml/machine-learning-databases;
      infile "http:&url/spambase/spambase.data"
             device=url delimiter=',';
      input wf_make wf_adress wf_all wf_3d wf_our
            wf_over wf_remove wf_internet wf_order wf_mail
            wf_receive wf_will wf_people wf_report wf_addresses
            wf_free wf_business wf_email wf_you wf_credit
            wf_your wf_font wf_000 wf_money wf_hp
            wf_hpl wf_george wf_650 wf_lab wf_labs
            wf_telnet wf_857 wf_data wf_415 wf_85
            wf_technology wf_1999 wf_parts wf_pm wf_direct
            wf_cs wf_meeting wf_original wf_project wf_re
            wf_edu wf_table wf_conference
            cf_semicolon cf_parenthese cf_bracket cf_exclamation
            cf_dollar cf_pound
            average longest total
            spam;
   run;
```

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

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

data fitstats;
   merge
   fitstats_f08
   fitstats_f06
   fitstats_f04;
   rename Ntrees=Trees;
```

```
   label fraction08 = "Fraction=0.8";
   label fraction06 = "Fraction=0.6";
   label fraction04 = "Fraction=0.4";
run;

proc sgplot data=fitstats;
   title "Misclassification Rate for Various Fractions of Training Data";
   series x=Trees y=fraction08/lineattrs=(Pattern=ShortDash Thickness=2);
   series x=Trees y=fraction06/lineattrs=(Pattern=MediumDashDotDot Thickness=2);
   series x=Trees y=fraction04/lineattrs=(Pattern=LongDash Thickness=2);
   yaxis label='OOB Misclassification Rate';
run;
title;
```

Output 8.3.1 Effect of the INBAGFRACTION Option on the Misclassification Rate



In this example, INBAGFRACTION=0.4 and INBAGFRACTION=0.6 produce the best OOB misclassification rate initially, with few trees. When more trees are used, INBAGFRACTION=0.8 is best.

Example 8.4: Loss Reduction Variable Importance

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

$$Y = X1 + X2 + 2X3 + X5 + X6 + 2X7$$

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

```
data output;
   call streaminit(54321);
   do i=1 to 10000;
      x1 = rand('normal', 0, 1);
      x2 = rand('normal', 0, 1);
      x3 = rand('normal', 0, 1);
      x4 = rand('normal', 0, 1);
      output;
   end;
run;

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

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

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

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

Output 8.4.1 shows the PROC HPFOREST variable importance table. The NRules column contains the number of splitting rules that use each variable. The next four columns are loss reduction measures of variable importance. The mean square error and the absolute error are computed with the training data. The OOB

columns contain the same measures computed with out-of-bag data. In this example, the relative importance of any pair of variables is similar in every measure.

Output 8.4.1 Loss Reduction Variable Importance

The HPFOREST Procedure

Loss Reduction Variable Importance

Variable	Number of Rules	MSE	OOB MSE	Absolute Error	OOB Absolute Error
x7	72098	14.28476	13.87263	1.714890	1.551729
x3	27499	3.83790	3.61276	0.733673	0.614372
x2	55172	0.89924	0.69273	0.317925	0.175285
x1	43608	0.87310	0.68100	0.305458	0.174619
x5	25653	0.65375	0.49699	0.193121	0.103127
x6	29150	0.53833	0.37682	0.182109	0.089835
x4	14346	0.03388	-0.03754	0.029768	-0.010899
x8	330263	0.14825	-0.07709	0.194676	-0.021982

PROC HPFOREST reports X7 as the most important variable. Although X7 and X3 have the same coefficient, X7 steals importance from correlated variables X5 and X6. PROC HPFOREST assigns less importance to X5 and X6 than to the uncorrelated variables X1 and X2 as a result, even though all four variables have the same coefficient in the formula for Y.

Variables X4 and X8 are not in the formula for Y. The OOB measures of importance are negative for both variables because, on balance, the spurious splits assign out-of-bag observations to the branch that has the worse prediction. The in-bag measures of importance are much larger for X8 than for X4 because X8 is correlated with variables that are predictive. Some splits that use X8 have validity because of this correlation.

Specifying VARS_TO_TRY=ALL in this example requests that PROC HPFOREST compare all inputs when it selects a variable to split a node on. The larger the number, the more dominant the importance of X7 is in this example. If VARS_TO_TRY=3 or less, variables X5, X6, and X7 would each get approximately the same importance, which would be slightly higher than the importance given to X3. Changing the VARS_TO_TRY= option has little effect on the importance of X1, X2, and X3.

Example 8.5: Missing Values and Imputed Values

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

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

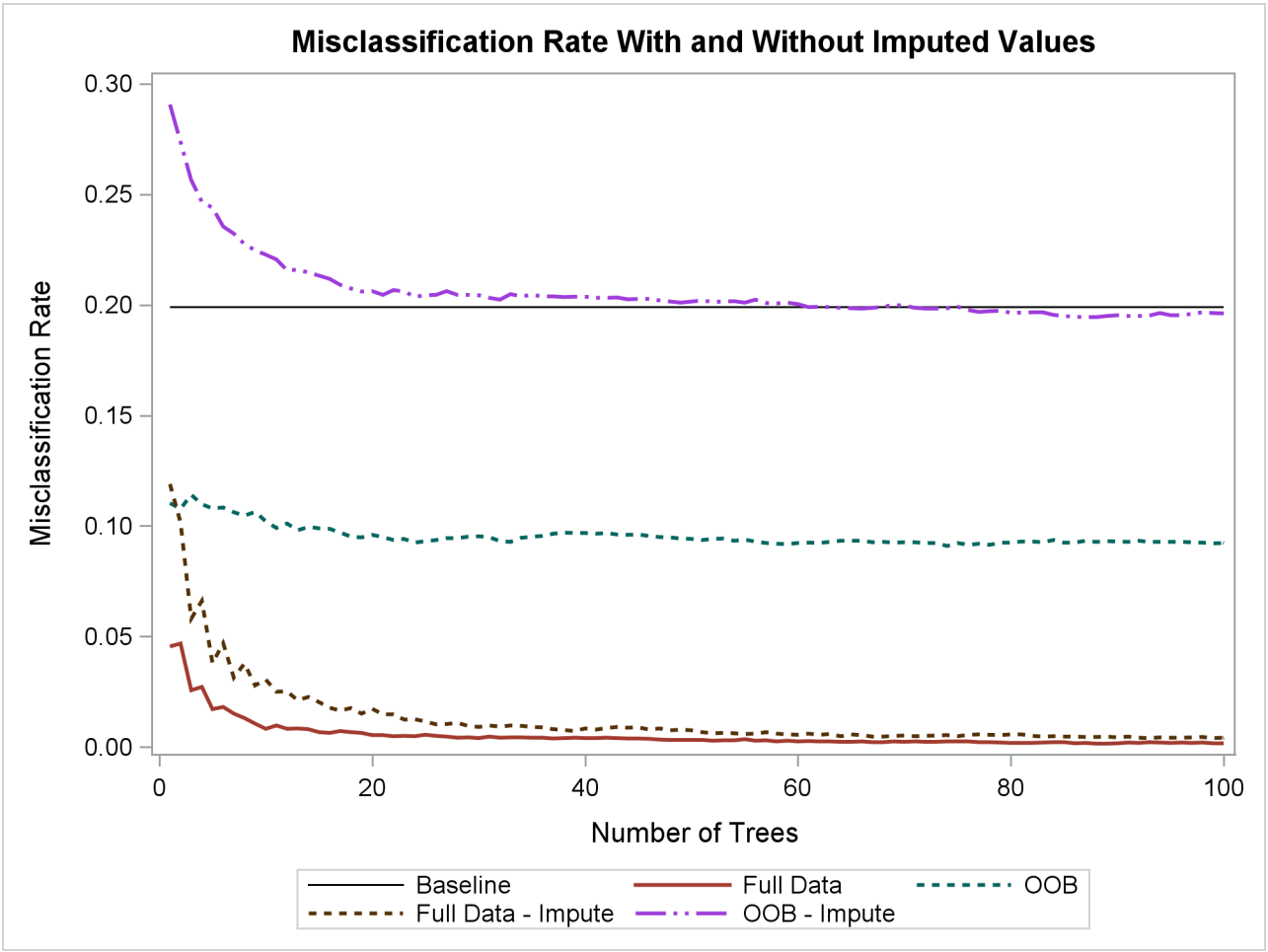
```
proc hpimpute data=sampsio.hmeq out=imout;
   input mortdue value yoj clage ninq clno debtinc derog delinq;
   impute mortdue value yoj clage ninq clno debtinc derog delinq/method=mean;
run;

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

data imout;
   merge imout job_reason;
run;

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

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

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

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

proc sgplot data=fitstats;
   title "Misclassification Rate With and Without Imputed Values";
```

```
   series x=trees y=Miscbaseline/lineattrs=(Pattern=Solid Color=black);
   series x=Trees y=MiscAll/lineattrs=(Pattern=Solid Thickness=2);
   series x=Trees y=Miscoob/lineattrs=(Pattern=ShortDash Thickness=2);
   series x=Trees y=ImMiscAll/lineattrs=(Pattern=ShortDash Thickness=2);
   series x=Trees y=ImMiscoob/lineattrs=(Pattern=MediumDashDotDot Thickness=2);
   yaxis label='Misclassification Rate';
run;

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

proc sort data=vi;
  by Variable;
run;

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

proc sort data=imvi;
  by Variable;
run;

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

proc sort data=vi;
  by rank;
run;

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

proc print data=t1;
run;
```

```
proc print data=t2;
run;

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

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

Figure 8.5.1 shows the misclassification rate and the out-of-bag misclassification rate with and without imputed missing values. Without any model, the misclassification rate equals 0.1995. The out-of-bag rate from the model trained with imputed values is not much better. The out-of-bag rate with the original data is much better, equal to half the baseline rate.

Output 8.5.1 The Effect of Imputing Missing Values



Output 8.5.2 shows the number of times each model uses each variable. The Rules column shows the number

of times by using the original data; the RankImputed column shows the number of times by using the imputed data. The numbers in the Rules column vary much more than those in the RuleImputed column, which suggests that variables with missing values have more distinctive information than variables with imputed values.

The order of the variables is the order of importance from the model that uses the original data. The RankImpute column shows the order of importance from the model that uses imputed values. DEBTINC is the most important variable when the original data are used, and among the last in importance when imputed values are used. Imputing changes the characteristics of these data dramatically.

Output 8.5.2 Variable Importance Ranking

Misclassification Rate With and Without Imputed Values

Obs	Variable	Rules	RulesImputed	RankImputed
1	DEBTINC	2165	8675	7
2	DELINQ	1688	5587	4
3	DEROG	1850	3694	3
4	REASON	1045	1334	1
5	CLAGE	3943	7179	9
6	JOB	1667	2208	2
7	NINQ	3252	6048	5
8	YOJ	3476	6971	6
9	CLNO	4373	7827	8
10	MORTDUE	4201	8676	10
11	VALUE	8538	17806	11

Output 8.5.3 shows the in-bag and out-of-bag Gini measures of importance for each variable. PROC HPFOREST uses only the training data in a tree to compute the in-bag measure, and it uses only the out-of-bag data in a tree for the out-of-bag measure. The out-of-bag measure is a better estimate of the contribution the variable makes to predicting new observations. A negative value indicates that the variable makes the prediction worse on average. The GiniOOB column shows that DEBTINC is twice as important than the next variable. However, when missing values are imputed, GinioOOB is slightly negative for DEBTINC, indicating that DEBTINC makes prediction slightly worse.

Output 8.5.3 Variable Importance Ranking

Misclassification Rate With and Without Imputed Values

Obs	Variable	Gini	GiniOOB	GiniImputed	GiniOOBImputed
1	DEBTINC	0.128626	0.07500	0.042435	-0.02535
2	DELINQ	0.020325	0.03773	0.019351	-0.01336
3	DEROG	0.011766	0.00453	0.012612	-0.00977
4	REASON	0.002426	0.00037	0.005074	0.00007
5	CLAGE	0.029483	-0.00020	0.033833	-0.02829
6	JOB	0.005599	-0.00085	0.009318	-0.00150
7	NINQ	0.014454	-0.00587	0.019901	-0.01792
8	YOJ	0.018382	-0.00628	0.028516	-0.02495
9	CLNO	0.019739	-0.00808	0.031076	-0.02675
10	MORTDUE	0.020341	-0.00866	0.036683	-0.02939
11	VALUE	0.041805	-0.00871	0.065968	-0.05408

Output 8.5.4 shows the count of observations for which DEBTINC is or is not missing for each value of the target BAD. DEBTINC is missing in 21% of the observations. If DEBTINC is missing, then the proportion of observations with BAD equal to 1 (which indicates an applicant who becomes delinquent) is 62%. If DEBTINC is not missing, then the proportion is only 8.6%. Missing values in DEBTINC are highly predictive of BAD in this example. Imputing the missing values destroys the predictive power.

Output 8.5.4 Contingency Table of BAD by DEBTINC_IS_MISSING

Misclassification Rate With and Without Imputed Values

The FREQ Procedure

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

References

Archer, K. J., and Kimes, R. V. (2008). "Empirical Characterization of Random Forest Variable Importance Measures." *Computational Statistics and Data Analysis* 52:2249–2260.

Asuncion, A., and Newman, D. J. (2007). "UCI Machine Learning Repository." http://archive.ics.uci.edu/ml/.

Berk, R. A. (2008). *Statistical Learning from a Regression Perspective*. New York: Springer.

Breiman, L. (1996). "Bagging Predictors." *Machine Learning* 24:123–140.

Breiman, L. (2001). "Random Forests." *Machine Learning* 45:5–32.

Breiman, L., and Cutler, A. (2003). "Manual—Setting Up, Using, and Understanding Random Forests V4.0." http://oz.berkeley.edu/users/breiman/Using_random_forests_v4.0.pdf.

Breiman, L., Friedman, J., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth.

De Ville, B., and Neville, P. G. (2013). *Decision Trees for Analytics Using SAS Enterprise Miner*. Cary, NC: SAS Institute Inc.

Fisher, W. D. (1958). "On Grouping for Maximum Homogeneity." *Journal of the American Statistical Association* 53:789–798.

Freedman, J. H., and Popescu, B. E. (2003). *Importance Sampled Learning Ensembles*. Technical report, Department of Statistics, Stanford University.

Friedman, J. H. (1977). "A Recursive Partitioning Decision Rule for Nonparametric Classification." *IEEE Transactions on Computers* 26:404–408.

Friedman, J. H. (1991). "Multivariate Adaptive Regression Splines." *Annals of Statistics* 19:1–67.

Friedman, J. H. (2001). "Greedy Function Approximation: A Gradient Boosting Machine." *Annals of Statistics* 29:1189–1232.

Grömping, U. (2009). "Variable Importance Assessment in Regression: Linear Regression versus Random Forest." *American Statistician* 63:308–319.

Hothorn, T., Hornik, K., and Zeileis, A. (2006). "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics* 15:651–674.

Kass, G. V. (1980). "An Exploratory Technique for Investigating Large Quantities of Categorical Data." *Journal of the Royal Statistical Society, Series C* 29:119–127.

King, G., and Zeng, L. (2001). "Logistic Regression in Rare Events Data." *Political Analysis* 9:137–163.

Loh, W.-Y. (2002). "Regression Trees with Unbiased Variable Selection and Interaction Detection." *Statistica Sinica* 12:361–386.

Loh, W.-Y. (2009). "Improving the Precision of Classification Trees." *Annals of Applied Statistics* 3:1710–1737.

Loh, W.-Y., and Shih, Y.-S. (1997). "Split Selection Methods for Classification Trees." *Statistica Sinica* 7:815–840.

Neville, P. G., and Tan, P.-Y. (2014). "A Forest Measure of Variable Importance Resistant to Correlations." In *Proceedings of the 2014 Joint Statistical Meetings*. Alexandria, VA: American Statistical Association.

Nicodemus, K. K., and Malley, J. D. (2009). "Predictor Correlation Impacts Machine Learning Algorithms: Implications for Genomic Studies." *Bioinformatics* 25:1884–1890.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Francisco: Morgan Kaufmann.

Radcliffe, N., and Surry, P. (2011). *Real-World Uplift Modelling with Significance-Based Uplift Trees*. Portrait technical report tr-2011-1, Stochastic Solutions. http://www.stochasticsolutions.com/pdf/sig-based-up-trees.pdf.

Smith, J. W., Everhart, J. E., Dickson, W. C., Knowler, W. C., and Johannes, R. S. (1988). "Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus." In *Proceedings of the Symposium on Computer Applications and Medical Care*, 261–265. Los Alamitos, CA: IEEE Computer Society Press.

Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., and Zeileis, A. (2008). "Conditional Variable Importance for Random Forests." *Bioinformatics* 9:307.

Su, X., Tsai, C.-L., Wang, H., Nickerson, D. M., and Li, B. (2009). "Subgroup Analysis via Recursive Partitioning." *Journal of Machine Learning Research* 10:141–158.

Van der Laan, M. J. (2006). "Statistical Inference for Variable Importance." *International Journal of Biostatistics* 2, article 2.

Subject Index

Syntax Index