SAS/STAT® 15.1 User's Guide
The HPSPLIT Procedure

This document is an individual chapter from *SAS/STAT® 15.1 User's Guide*.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2018. *SAS/STAT® 15.1 User's Guide*. Cary, NC: SAS Institute Inc.

SAS/STAT® 15.1 User's Guide

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

November 2018

Chapter 65
The HPSPLIT Procedure

Contents

Overview: HPSPLIT Procedure

The HPSPLIT procedure is a high-performance procedure that builds tree-based statistical models for classification and regression. The procedure produces classification trees, which model a categorical response, and regression trees, which model a continuous response. Both types of trees are referred to as decision trees because the model is expressed as a series of if-then statements.

The predictor variables for tree models can be categorical or continuous. The model is based on a partition of the predictor space into nonoverlapping segments, which correspond to the terminal nodes or leaves of the tree. The partitioning is done recursively, starting with the root node, which contains all the data, and ending with the terminal nodes. At each step of the recursion, the parent node is split into child nodes through selection of a predictor variable and a split value that minimize the variability in the response across the child nodes.

Tree models are built from training data for which the response values are known, and these models are subsequently used to score (classify or predict) response values for new data. For classification trees, the most frequent response level of the training observations in a leaf is used to classify observations in that leaf. For regression trees, the average response of the training observations in a leaf is used to predict the response for observations in that leaf. The splitting rules that define the leaves provide the information that is needed to score new data.

The process of building a decision tree begins with growing a large, full tree. Various measures, such as the Gini index, entropy, and residual sum of squares, are used to assess candidate splits for each node. The full tree can overfit the training data, resulting in a model that does not adequately generalize to new data. To prevent overfitting, the full tree is pruned back to a smaller subtree that balances the goals of fitting training data and predicting new data. Two commonly applied approaches for finding the best subtree are cost-complexity pruning (Breiman et al. 1984) and C4.5 pruning (Quinlan 1993). For more information, see the section "Building a Decision Tree" on page 5029.

SAS/STAT software provides many different methods of regression and classification. Compared with other methods, an advantage of tree models is that they are easy to interpret and visualize, especially when the tree is small. Tree-based methods scale well to large data, and they offer various methods of handling missing values, including surrogate splits.

However, tree models have limitations. Regression tree models fit response surfaces that are constant over rectangular regions of the predictor space, and so they often lack the flexibility needed to capture smooth relationships between the predictor variables and the response. Another limitation of tree models is that small changes in the data can lead to very different splits, and this undermines the interpretability of the model (Hastie, Tibshirani, and Friedman 2009; Kuhn and Johnson 2013).

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

NOTE: Distributed mode requires SAS High-Performance Statistics.

PROC HPSPLIT Features

The main features of the HPSPLIT procedure are as follows:

- provides a variety of methods of splitting nodes, including criteria based on impurity (entropy, Gini index, residual sum of squares) and criteria based on statistical tests (chi-square, *F* test, CHAID, FastCHAID)
- provides a computationally efficient strategy for generating candidate splits
- provides the cost-complexity, C4.5, and reduced-error methods of pruning trees
- supports the use of cross validation and validation data for selecting the best subtree
- provides various methods of handling missing values, including surrogate rules
- creates tree diagrams, plots for cost-complexity analysis, and plots of ROC curves
- computes statistics for assessing model fit, including model-based (resubstitution) statistics and cross validation statistics
- computes measures of variable importance
- produces a file that contains SAS DATA step code for scoring new data
- produces a file that contains node rules
- provides an output data set with leaf assignments and predicted values for observations

The HPSPLIT procedure uses ODS Graphics to create plots as part of its output. For general information about ODS Graphics, see Chapter 21, "Statistical Graphics Using ODS." For specific information about the statistical graphics available with the HPSPLIT procedure, see the PLOTS options in the PROC HPSPLIT statement and the section "ODS Graphics" on page 5048.

Because the HPSPLIT 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 available cores and concurrent threads, regardless of execution mode

For more information, see the section "Processing Modes" (Chapter 2, *SAS/STAT User's Guide: High-Performance Procedures*).

Getting Started: HPSPLIT Procedure

This example explains basic features of the HPSPLIT procedure for building a classification tree. The data are measurements of 13 chemical attributes for 178 samples of wine. Each wine is derived from one of three cultivars that are grown in the same area of Italy, and the goal of the analysis is a model that classifies samples into cultivar groups. The data are available from the UCI Irvine Machine Learning Repository; see Lichman (2013).[1]

The following statements create a data set named Wine that contains the measurements:

```
data Wine;
   %let url = http://archive.ics.uci.edu/ml/machine-learning-databases;
   infile "&url/wine/wine.data" url delimiter=',';
   input Cultivar Alcohol Malic Ash Alkan Mg TotPhen
         Flav NFPhen Cyanins Color Hue ODRatio Proline;
   label Cultivar = "Cultivar"
         Alcohol  = "Alcohol"
         Malic    = "Malic Acid"
         Ash      = "Ash"
         Alkan    = "Alkalinity of Ash"
         Mg       = "Magnesium"
         TotPhen  = "Total Phenols"
         Flav     = "Flavonoids"
         NFPhen   = "Nonflavonoid Phenols"
         Cyanins  = "Proanthocyanins"
         Color    = "Color Intensity"
         Hue      = "Hue"
         ODRatio  = "OD280/OD315 of Diluted Wines"
         Proline  = "Proline";
run;

proc print data=Wine(obs=10); run;
```

Figure 65.1 lists the first 10 observations of Wine.

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

Figure 65.1 Partial Listing of Wine

Obs	Cultivar	Alcohol	Malic	Ash	Alkan	Mg	TotPhen	Flav	NFPhen	Cyanins	Color	Hue	ODRatio	Proline
1	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
2	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
3	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
4	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
5	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
6	1	14.20	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450
7	1	14.39	1.87	2.45	14.6	96	2.50	2.52	0.30	1.98	5.25	1.02	3.58	1290
8	1	14.06	2.15	2.61	17.6	121	2.60	2.51	0.31	1.25	5.05	1.06	3.58	1295
9	1	14.83	1.64	2.17	14.0	97	2.80	2.98	0.29	1.98	5.20	1.08	2.85	1045
10	1	13.86	1.35	2.27	16.0	98	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1045

The variable Cultivar is a nominal categorical variable with levels 1, 2, and 3, and the 13 attribute variables are continuous.

The following statements use the HPSPLIT procedure to create a classification tree:

```
ods graphics on;

proc hpsplit data=Wine seed=15533;
   class Cultivar;
   model Cultivar = Alcohol Malic Ash Alkan Mg TotPhen Flav
                    NFPhen Cyanins Color Hue ODRatio Proline;
   grow entropy;
   prune costcomplexity;
run;
```

The MODEL statement specifies Cultivar as the response variable and the variables to the right of the equal sign as the predictor variables. The inclusion of Cultivar in the CLASS statement designates it as a categorical response variable and requests a classification tree. All the predictor variables are treated as continuous variables because none are included in the CLASS statement.

The GROW and PRUNE statements control two fundamental aspects of building classification and regression trees: growing and pruning. You use the GROW statement to specify the criterion for recursively splitting parent nodes into child nodes as the tree is grown. For classification trees, the default criterion is entropy; see the section "Splitting Criteria" on page 5032.

By default, the growth process continues until the tree reaches a maximum depth of 10 (you can specify a different limit by using the MAXDEPTH= option). The result is often a large tree that overfits the data and is likely to perform poorly in predicting future data. A recommended strategy for avoiding this problem is to prune the tree to a smaller subtree that minimizes prediction error. You use the PRUNE statement to specify the method of pruning. The default method is cost complexity; see the section "Pruning" on page 5035.

The default output includes the four informational tables that are shown in Figures 65.2 through 65.5.

The "Performance Information" table in Figure 65.2 shows that the procedure executes in single-machine mode—that is, all the computations are done on the machine where the SAS session executes. This run of the HPSPLIT procedure was performed on a multicore machine with the same number of CPUs as there are threads; that is, one computational thread was spawned per CPU.

Figure 65.2 Performance Information

The HPSPLIT Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	16

The "Data Access Information" table in Figure 65.3 shows that the input data set is accessed using the V9 (base) engine on the client machine where the MVA SAS session executes. This table includes similar information about output data sets that you can request by using the OUTPUT statement.

Figure 65.3 Data Access Information

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

The "Model Information" table in Figure 65.4 provides information about the model and the methods that are used to grow and prune the tree.

Figure 65.4 Model Information

Model Information	
Split Criterion Used	Entropy
Pruning Method	Cost-Complexity
Subtree Evaluation Criterion	Cost-Complexity
Number of Branches	2
Maximum Tree Depth Requested	10
Maximum Tree Depth Achieved	4
Tree Depth	4
Number of Leaves Before Pruning	8
Number of Leaves After Pruning	8

The "Observation Information" table in Figure 65.5 provides the numbers of observations that are read and used.

Figure 65.5 Observation Information

Number of Observations Read	178
Number of Observations Used	178

These numbers are the same in this example because there are no missing values in the predictor variables. By default, observations that have missing values are not used. However, the HPSPLIT procedure provides methods for incorporating missing values in the analysis, as explained in the sections "Handling Missing Values" on page 5041 and "Primary and Surrogate Splitting Rules" on page 5040.

The plot in Figure 65.6 is a tool for selecting the tuning parameter for cost-complexity pruning. The parameter, indicated on the lower horizontal axis, indexes a sequence of progressively smaller subtrees that are nested within the large tree. The parameter value 0 corresponds to the large tree, and positive values control

the trade-off between complexity (number of leaves) and fit to the training data, as measured by average misclassification rate.

Figure 65.6 Misclassification Rate as a Function of Cost-Complexity Parameter

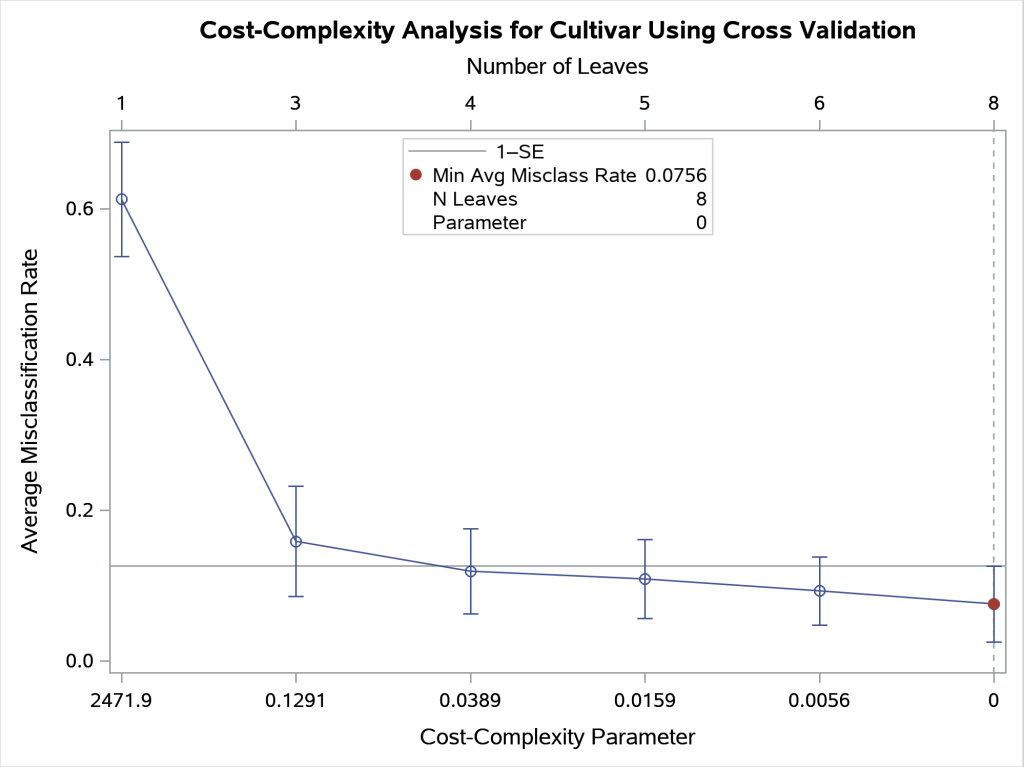


Figure 65.6 shows that PROC HPSPLIT selects the parameter value 0.0159 because it minimizes the average misclassification rate, which is obtained by 10-fold cross validation. However, the average misclassification rates for several other parameter choices are nearly the same.

Breiman's 1-SE rule chooses the parameter that corresponds to the smallest subtree for which the misclassification rate is less than one standard error above the minimum misclassification rate (Breiman et al. 1984). This parameter choice (0.1291) corresponds to a tree that has only three leaves.

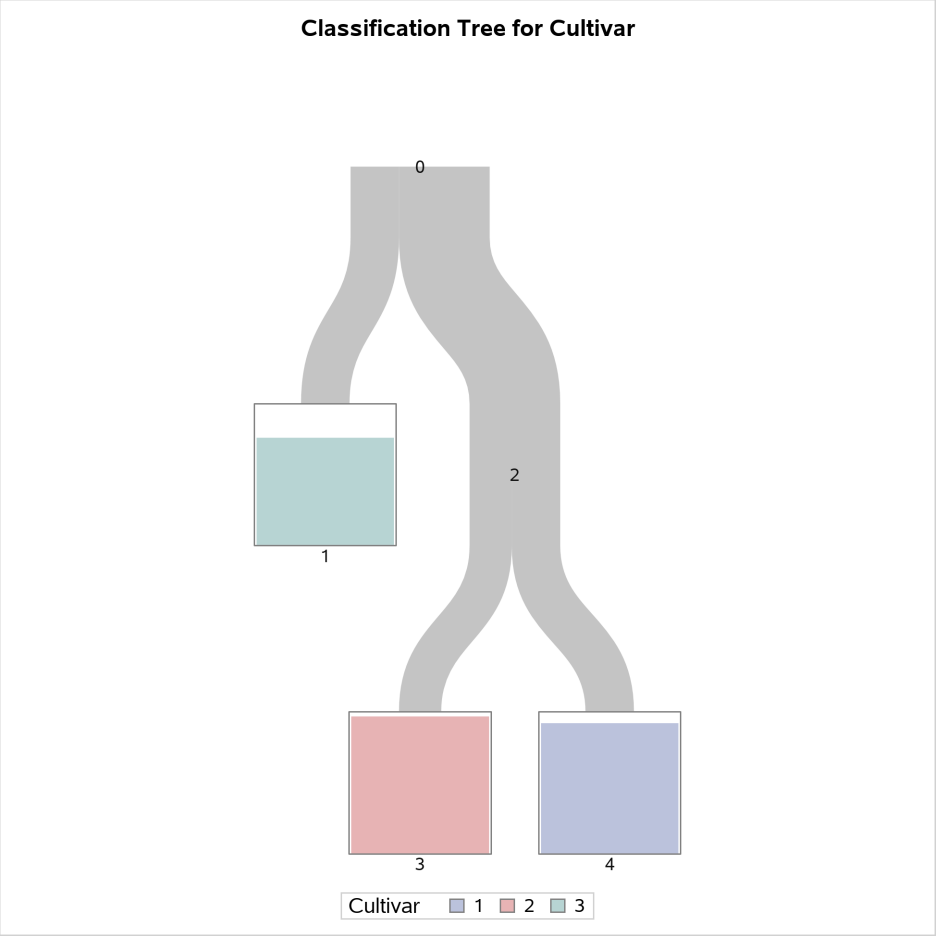
NOTE: The estimates of misclassification rate and their standard errors depend on the random assignment of observations to the folds in the cross validation; the random assignment is determined by the SEED= option.

The following statements build a classification tree by growing a large tree and applying cost-complexity pruning (also known as weakest-link pruning) to obtain a tree that has three leaves:

```
proc hpsplit data=Wine seed=15533;
   class Cultivar;
   model Cultivar = Alcohol Malic Ash Alkan Mg TotPhen Flav
                    NFPhen Cyanins Color Hue ODRatio Proline;
   prune costcomplexity(leaves=3);
run;
```

The tree diagram in Figure 65.7, which is produced by default when ODS Graphics is enabled, provides an overview of the tree as a classifier.

Figure 65.7 Overview Diagram of Final Tree



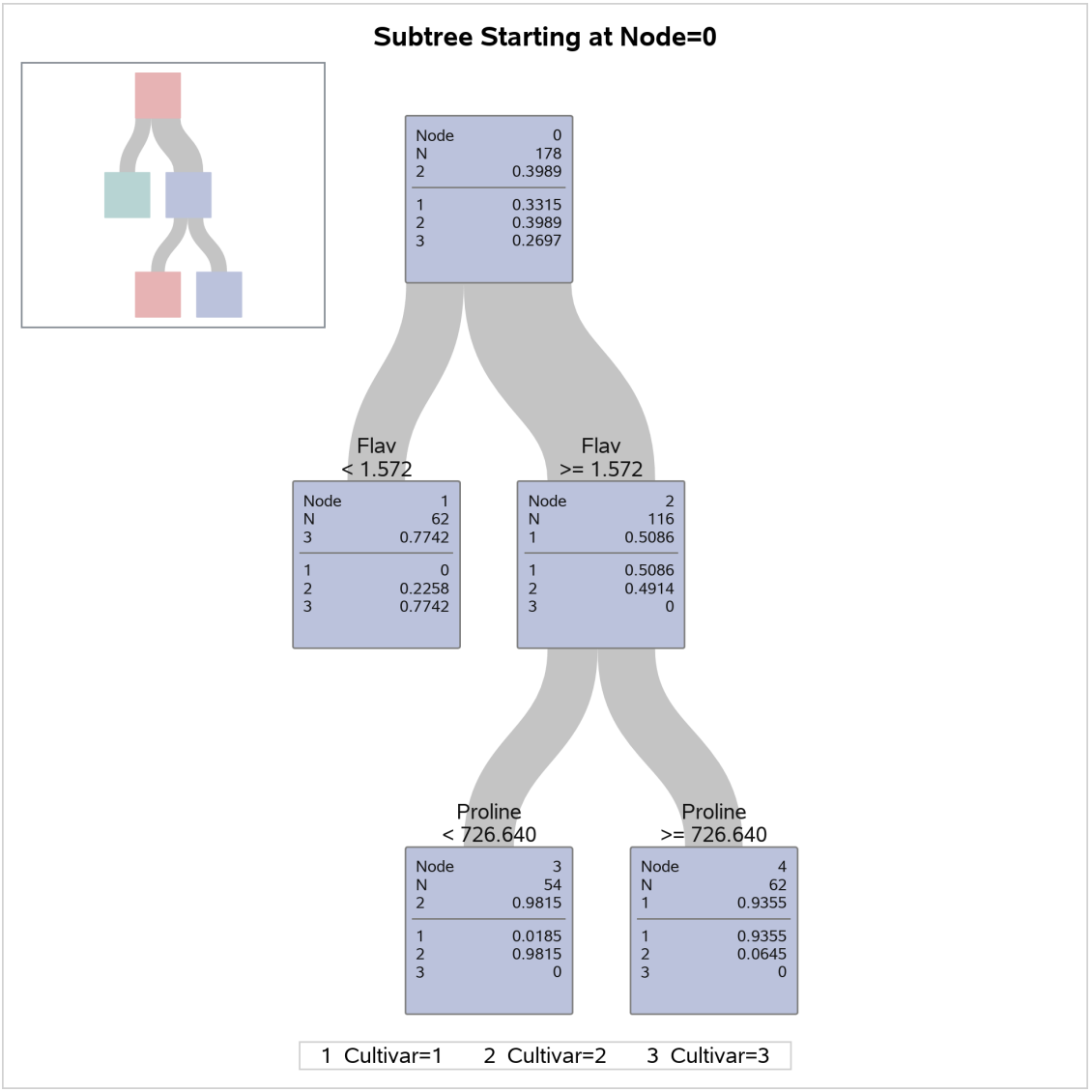
The tree is constructed starting with all the observations in the root node (labeled 0). This node is split into a leaf node (1) and an internal node (2), which is further split into two leaf nodes (3 and 4).

The color of the bar in each leaf node indicates the most frequent level of Cultivar among the observations in that node; this is also the classification level assigned to all observations in that node. The height of the bar indicates the proportion of observations in the node that have the most frequent level. The width of the link between parent and child nodes is proportional to the number of observations in the child node.

The diagram reveals that splitting on just two of the attributes is sufficient to differentiate the three cultivars, and a tree model that has only three leaves provides a high degree of accuracy for classification.

The diagram in Figure 65.8 provides more detail about the nodes and the splits.

Figure 65.8 Detailed Tree Diagram



There are 178 samples in the root node (node 0). The table below the line in the box for node 0 provides the proportion of samples for each level of Cultivar, and the level that has the highest proportion is also given above the line. These samples are divided into 62 samples for which Flav $<$ 1.572 (node 1) and 116 samples for which Flav $\geq$ 1.572 (node 2).

The variable Flav and the split point 1.572 are chosen to maximally decrease the impurity of the root node as measured by the entropy criterion. There are no samples for which Cultivar=1 in node 1, and there are no samples for which Cultivar=3 in node 2.

The samples in node 2 were further divided into 54 samples for which Proline $<$ 726.640 (node 3) and 62 samples for which Proline $\geq$ 726.640 (node 4).

The classification tree yields simple rules for predicting the wine cultivar. For example, a sample for which Flav $\geq$ 1.572 and Proline $<$ 726.640 is predicted to be from the third cultivar (Cultivar=3).

The diagram in Figure 65.8 happens to show the entire tree that was created by the preceding statements, but in general this diagram shows a subtree of the entire tree that begins with the root node and has a depth of four levels. You can use the PLOTS=ZOOMEDTREE option in the PROC HPSPLIT statement to request diagrams that begin with other nodes and have specified depths.

The confusion matrix in Figure 65.9 evaluates the accuracy of the fitted tree for classifying the training data that were used to build the tree model.

Figure 65.9 Confusion Matrix

The HPSPLIT Procedure

Model-Based Confusion Matrix

	Predicted			
Actual	1	2	3	Error Rate
1	58	1	0	0.0169
2	4	53	14	0.2535
3	0	0	48	0.0000

The values in the off-diagonal entries of the matrices show how many times the fitted model misclassified a sample. For example, among the 59 samples for which Cultivar=1, only one sample was misclassified, and it was incorrectly classified as Cultivar=2.

The "Tree Performance" table in Figure 65.10 displays various fit statistics for the tree model.

Figure 65.10 Fit Statistics

Model-Based Fit Statistics for Selected Tree

N Leaves	ASE	Mis-class	Entropy	Gini	RSS
3	0.0583	0.1067	0.4290	0.1749	31.1243

For instance, the misclassification rate is the proportion of the 178 wine samples that were misclassified: (1 + 4 + 14)/178 = 0.1067.

You can use the CVMODELFIT option in the PROC HPSPLIT statement to request a model assessment that is based on cross validation; this is done independently of the cross validation used to assess pruning parameters. For more information about fit statistics, see the section "Measures of Model Fit" on page 5043.

NOTE: Additional fit statistics are available in the case of classification trees for binary responses. See "Example 65.1: Building a Classification Tree for a Binary Outcome" on page 5051.

Syntax: HPSPLIT Procedure

The following statements and options are available in the HPSPLIT procedure:

PROC HPSPLIT < *options* > **;**
CLASS *variable...* < */options* > **;**
CODE FILE=*filename* **;**
GROW *criterion* < */ options* > **;**
ID *variables* **;**
MODEL *response* < **(***response-options***)** > **=** *variable* < *variable...* > **;**
OUTPUT *output-options* **;**
PARTITION < *partition-options* > **;**
PERFORMANCE *performance-options* **;**
PRUNE *prune-method* < **(***prune-options***)** > **;**
RULES FILE=*filename* **;**
WEIGHT *variable* < **/ UNWEIGHTDF** > **;**

The PROC HPSPLIT statement and the MODEL statement are required. If any variables are character or to be treated as categorical, at least one CLASS statement is required. Variables that appear after the equal sign (=) in the MODEL statement are explanatory variables that model the *response* variable. By default, all *variables* that appear in the MODEL statement are treated as continuous variables. CLASS statements cause a *variable* to be treated as categorical. Specifying a *variable* in a CLASS statement but not in a MODEL statement causes the variable to be ignored and a warning to be issued.

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

PROC HPSPLIT Statement

PROC HPSPLIT < *options* > **;**

The PROC HPSPLIT statement invokes the procedure. Table 65.1 summarizes the options in the PROC HPSPLIT statement.

Table 65.1 PROC HPSPLIT Statement Options

Option	Description
Basic Options	
CVCC	Requests a table of the results of cost-complexity pruning based on cross validation
CVMETHOD=	Specifies the cross validation method to use
CVMODELFIT	Requests model assessment and a confusion matrix with cross validation
DATA=	Specifies the predictor data set
INTERVALBINS=	Specifies the number of bins for continuous variables
MINVARIANCE=	Sets the minimum variance for a regression tree leaf to be split
NODES	Requests a table that describes the nodes of the final tree
NOPRINT	Suppresses ODS output
NSURROGATES=	Specifies the number of surrogate rules to create
PLOTS=	Specifies options for plots
SEED=	Specifies the random number seed to use for cross validation
SPLITONCE	Specifies that variables should be split only once per branch
Splitting Options	
ASSIGNMISSING=	Specifies how to handle missing values in a predictor variable
LEVTHRESH1=	Specifies the maximum number of computations to perform in an exhaustive search for a categorical predictor
LEVTHRESH2=	Specifies the number of computations to perform before the splitter uses the fastest greedy search
MAXBRANCH=	Specifies the maximum number of leaves per node
MAXDEPTH=	Specifies the maximum tree depth
MINCATSIZE=	Specifies the number of observations per level in order for the level to be considered for splitting
MINLEAFSIZE=	Specifies the minimum number of observations per leaf

You can specify the following *options*.

ASSIGNMISSING=BRANCH | NONE | POPULAR | SIMILAR

specifies how PROC HPSPLIT creates a default splitting rule to handle missing values, unknown levels, and levels that have fewer observations than you specify in the MINCATSIZE= option. An unknown level is a level of a categorical predictor that does not exist in the training data but is encountered during scoring.

Both the ASSIGNMISSING= and NSURROGATES= options affect training and scoring. See the NSURROGATES= option for the definition of surrogate rules.

During training, the primary splitting rule is created first, along with the default splitting rule (controlled by the ASSIGNMISSING= option). If you request surrogate rules (by using the NSURROGATES= option), they are created after the primary and default splits are made. When the splitting rules have been created, the rules are used as described in the following list for assigning the training data by using the new splitting rules, and splitting rule creation continues on the new children.

Observation assignment during the training phase and during scoring proceeds as follows:

1. The primary splitting rule is applied if the rule's variable is not missing. Otherwise:
2. The first surrogate rule (with the largest agreement, described in the section "Primary and Surrogate Splitting Rules" on page 5040) is applied if the first surrogate rule's variable is not missing. Otherwise:
3. Each subsequent (ordered by agreement) splitting rule is applied as described in the first surrogate item in this list. If all of the surrogate rules' variables are missing, then:
4. The default splitting rule is used.

Because there is always a default splitting rule, all data can be scored, even if the primary rule and all surrogate rules cannot be used on a particular observation.

You can specify one of the following:

BRANCH
specifies that PROC HPSPLIT create a special child (branch) for the default rule and assign to that child missing values, unknown levels, and levels that have fewer observations than you specify in the MINCATSIZE= option. You need to have missing values in the training data set for this to work. If no missing values or levels that have fewer observations than you specify in the MINCATSIZE= option are available at the split in the training data set, then missing values are assigned using ASSIGNMISSING=POPULAR.

NONE
specifies that observations that have any missing variables be excluded from the analysis. In addition, the default rule (to handle unknown levels, which include missing values, because they are excluded from the analysis, and levels that have fewer observations than you specify in the MINCATSIZE= option) is created using ASSIGNMISSING=POPULAR.

POPULAR
specifies that missing values be assigned to the most popular (largest) child.

SIMILAR
specifies that missing values be assigned to the child that they are most similar to (using the chi-square for categorical responses or F-test criterion for continuous responses). For more information about this option and the similarity measurement, see the section "Handling Missing Values" on page 5041. If no missing values or levels that have fewer observations than you specify in the MINCATSIZE= option are available at the split in the training data set, then missing values are assigned using ASSINGMISSING=POPULAR.

By default, ASSIGNMISSING=NONE.

CVCC

CVCOSTCOMPLEXITY

requests a table of the results of cost-complexity pruning based on cross validation. For each fold in the cross validation, the table provides the penalty parameter, the number of leaves, and the average ASE. In addition, the table provides the minimum and maximum ASE, and the minimum, median, and maximum number of leaves. The number of leaves is the floor of the median. You can use the PLOTS=CVCC option to request a plot of the information in this table.

CVMETHOD=NONE | RANDOM < (*k*) >

requests the cross validation method to be performed.

You can specify one of the following:

NONE

suppresses cross validation.

RANDOM < (*k*) >

assigns each training observation randomly to one of the *k* folds (with a probability of 1/*k* for any given fold) for cross validation. The default value of *k* is 10.

Cross validation applies to the CVMODELFIT option and cost-complexity pruning. In *k*-fold cross validation, the training set is divided into *k* folds; *k* trees are then built using all but the one fold. For example, the first tree uses all of the training set except for the observations in the first fold. The second uses all of the training set except for the observations in the second fold, and so on.

The holdout fold for each tree is used to calculate the ASE of that tree. The average ASE across the *k* trees is the cross validation error for that set of trees. This process is repeated for each value of the parameter that is cross validated. The parameter that has the minimum cross validated error is used as the best parameter value. A final tree is then grown using this final parameter value.

NOTE: Attempting to use a PARTITION statement along with cross validation results in an error.

By default, CVMETHOD=RANDOM(10) when you perform cost-complexity pruning with no PARTITION statement.

CVMODELFIT

requests model assessment with cross validation. When you specify this option, the procedure does a cross validation of the final model parameters and produces a table that describes the cross validation error measures of the parameters. The table contains various summary statistics of the ASE, the number of leaves, and, for a categorical response, the misclassification rate across the *k* trees grown. This option also requests a table that contains the cross validation confusion matrix.

If cost complexity is cross validated (the default if you use cost-complexity pruning without a validation set), the assessment is a completely separate cross validation of the final tree penalty parameter, using a different seed for fold assignment.

NOTE: The assessment is not on a holdout sample but instead on *k*-fold cross validation. This is not a second run of the procedure but instead is done automatically and internally.

DATA=*SAS-data-set*
names the predictor SAS data set to be used by PROC HPSPLIT. The default is the most recently created data set.

If the procedure executes in distributed mode, the predictor data are distributed to memory on the appliance nodes and analyzed in parallel, unless the data are already distributed in the appliance database. In that case, the procedure reads the data alongside the distributed database. For more information, see the section "Processing Modes" (Chapter 2, *SAS/STAT User's Guide: High-Performance Procedures*) about the various execution modes and the section "Alongside-the-Database Execution" (Chapter 2, *SAS/STAT User's Guide: High-Performance Procedures*) about the alongside-the-database model.

INTERVALBINS=*number*
specifies the *number* of bins for continuous variables. PROC HPSPLIT bins continuous predictors to a fixed bin size. This option controls the number of bins and thereby also the size of the bins. For more information about interval variable binning, see the section "Details: HPSPLIT Procedure" on page 5029.

By default, INTERVALBINS=100.

LEVTHRESH1=*number*
applies only to categorical predictor variables and specifies the limit for the number of computations in an exhaustive search for the optimal partition of the levels of a particular variable. The splitter first evaluates the number of computations that are needed for an exhaustive search. If this number exceeds the limit, then the splitter falls back to a faster heuristic algorithm. You can use the LEVTHRESH2= option to specify the limit for the number of computations in this faster algorithm.

By default, LEVTHRESH1=500000.

LEVTHRESH2=*number*
applies to categorical predictor variables and continuous predictor variables with multiway splits. This option does not apply to continuous predictor variables with binary splits.

For a categorical predictor variable, the splitter first evaluates the number of computations that are needed for an exhaustive search. If this number exceeds the limit that you specify in the LEVTHRESH1= option, then the splitter falls back to a faster heuristic algorithm. You can use the LEVTHRESH2= option to specify the limit for the number of computations in this faster algorithm. If this number of computations exceeds the limit that you specify in the LEVTHRESH2= option, then the splitter falls back to an even faster algorithm.

For a continuous predictor variable, the splitter first tries to perform an exhaustive search for the optimal split values. The splitter first evaluates the number of computations that are needed for an exhaustive search. If this number exceeds the limit that you specify in the LEVTHRESH2= option, the splitter falls back to a faster heuristic algorithm.

By default, LEVTHRESH2=1000000.

MAXBRANCH=*b*
specifies the maximum number of children per node in the tree. PROC HPSPLIT tries to create this number of children unless it is impossible (for example, if a split variable does not have enough levels). By default, MAXBRANCH=2.

MAXDEPTH=*number*

specifies the maximum depth of the tree to be grown. The default is set using the following equation, where b is the value for the MAXBRANCH= option:

$$\text{MaxDepth} = \left\lceil \frac{10}{\log_2(b)} \right\rceil$$

MINCATSIZE=*number*

specifies the *number* of observations that a categorical variable level must have in order to be considered in the split. Predictor variable levels that have fewer observations than *number* receive the nonsurrogate missing value assignment for that split. See the NSURROGATES= option for the definition of surrogate rules.

By default, MINCATSIZE=1.

MINLEAFSIZE=*number*

LEAFSIZE=

specifies the minimum *number* of observations that each child of a split must contain in the training data set in order for the split to be considered.

By default, MINLEAFSIZE=1.

MINVARIANCE=*value*

specifies the minimum variance for a regression tree leaf to be eligible for splitting. That is, leaves whose variance is less than *value* are not split any further.

By default, MINVARIANCE=1E–8.

The variance at some leaf λ with weight N_λ (number of observations) is calculated using

$$\text{Var} = \frac{\sum_i^\lambda y_i^2}{N_\lambda} - \bar{y}_\lambda^2$$

where y_i is the response value at observation i and $\bar{y}_\lambda$ is the average value of the response within leaf λ.

NODES=DETAIL | SUMMARY

requests a table that contains the description of the paths from each leaf to the root.

You can specify the following values:

DETAIL

prints all nodes and leaves, including the path from each node or leaf to the root of the tree.

SUMMARY

prints all nodes and leaves, without including the path to the root of the tree.

NOPRINT

suppresses the generation of ODS output.

NSURROGATES=*number*

specifies the *number* of surrogate rules to create for each splitting rule. Surrogate rules are backup splitting rules that are used when the variable that corresponds to the primary splitting rule is missing. By default, NSURROGATES=0.

Both the ASSIGNMISSING= option and NSURROGATES= options affect training and scoring.

During training, the primary splitting rule is created first, along with the default splitting rule (controlled by the ASSIGNMISSING= option). If you request surrogate rules (by specifying the NSURROGATES= option), they are created after the primary and default splits are created. When the splitting rules have been created, the rules are used as described in the following list for assigning the training data by using the new splitting rules, and splitting rule creation continues on the new children.

Observation assignment during the training phase and during scoring proceeds as follows:

1. The primary splitting rule is applied if the primary rule's variable is not missing. Otherwise:
2. The first surrogate rule (with the largest agreement, described in the section "Primary and Surrogate Splitting Rules" on page 5040) is applied if the first surrogate rule's variable is not missing. Otherwise:
3. Each subsequent (ordered by agreement) splitting rule is applied as described in the first surrogate item in the list. If all of the surrogate rules' variables are missing, then:
4. The default splitting rule is used.

Because there is always a default splitting rule, all data can be scored, even if the primary rule and all surrogate rules cannot be used on a particular observation.

As auxiliary rules, surrogate rules are used in the following ways:

- Surrogate rules are used in generating the tree.
- Surrogate rules therefore also affect the final tree metrics.
- Surrogate rules are used in the SAS code output from the CODE statement.
- Surrogate rules are used in scoring the predictor data set by using the OUTPUT statement.
- Surrogate rules are *not* written out in the "Node Rules" file, output from the RULES statement.

PLOTS <(*global-plot-option***)> <= *plot-request* <(***options***)>>**

PLOTS <(*global-plot-option***)> <= (***plot-request* <(***options***)> <... *plot-request* <(***options***)>>)>**

controls the plots that are produced through ODS Graphics. When you specify only one *plot-request*, you can omit the parentheses around it. Some examples follow.

You can specify the following *global-plot-option*:

ONLY

suppresses the default plots. Only plots that you specifically request are displayed.

You can specify the following *plot-requests*:

ALL

produces all appropriate plots.

CVCC< (ASE) >
produces a plot that is used to determine the tuning parameter for cost-complexity pruning when cross validation is used. By default, this plot displays the average misclassification rate when the response variable is a classification variable and displays the average square error (ASE) when the response variable is a continuous variable. You can specify the following suboption:

ASE
displays the ASE when the response variable is a classification variable.

NONE
suppresses the default plots. Only plots that you specifically request are displayed.

PRUNEUNTIL
produces a plot of the metric that is used to select the final subtree. This option is enabled by default except in the following cases:

- You request cross validation of cost-complexity pruning.
- You specify the LEAVES= option in the PRUNE statement when you use the cost-complexity pruning method.
- You specify the OFF option in the PRUNE statement. Note that specifying the PRUNEUNTIL option has no effect in this case.

ROC
produces the receiver operating characteristic (ROC) curve. This option is enabled by default.

WHOLETREE < (*whole-tree-options*) >
produces a plot to visualize the entire finished (grown and pruned) tree. This option is enabled by default.

You can specify the following *whole-tree-options*:

LINKSTYLE=*link-style*
specifies the style of links between nodes and leaves in the tree. You can specify the following *link-styles*:

CURVED
requests curved links between the nodes and their children.

ORTHOGONAL
requests that links go straight down partway from a node to its children, create a horizontal line at the base of the vertical line, and then go straight down from that line to each child.

STRAIGHT
requests that links go straight from the nodes to their children.

By default, LINKSTYLE=CURVED.

LINKWIDTH=*link-width*
specifies the width of links between nodes and leaves in the tree. You can specify the following *link-widths*:

CONSTANT
requests that all links have the same thickness.

PROPORTIONAL
requests that links have a thickness proportional to the total number of observations that go between the node and each child.

By default, LINKWIDTH=PROPORTIONAL.

NOLEGEND
turns off the legend.

ZOOMEDTREE < (*zoomed-tree-options*) >
produces a plot to visualize a portion of the finished (grown and pruned) tree. This option is enabled by default.

You can specify the following *zoomed-tree-options*:

DEPTH=*depth*
requests that PROC HPSPLIT create plots that are specified by the ZOOMEDTREE option for each *node-id* down to *depth*.

FRACTIONPRECISION=*integer*
requests that the fraction of total observations that are displayed in the decision tree node have a number of digits after the decimal equal to *integer*.

By default, FRACTIONPRECISION=4.

LINKSTYLE=*link-style*
specifies the style of links between nodes and leaves in the tree. You can specify the following *link-styles*:

CURVED
requests curved links between the nodes and their children.

ORTHOGONAL
requests that links go straight down partway from a node to its children, create a horizontal line at the base of the vertical line, and then go straight down from that line to each child.

STRAIGHT
requests that links go straight from the nodes to their children.

By default, LINKSTYLE=CURVED.

LINKWIDTH=*link-width*
specifies the width of links between nodes and leaves in the tree. You can specify the following *link-widths*:

CONSTANT
requests that all links have the same thickness.

PROPORTIONAL
requests that links have a thickness proportional to the total number of observations that go between the node and each child.

By default, LINKWIDTH=PROPORTIONAL.

NODES=('*node-id***' < '***node-id***' < ... > >)**
requests plots for the subtree that is rooted at nodes at *node-id*. The default node ID is '0', the root of the entire tree. The values of *node-id* are alphanumeric strings that are displayed within the nodes in the plot that is created by the WHOLETREE option. PROC HPSPLIT creates one plot that is specified by the ZOOMEDTREE option for each *node-id* that you specify. The DEPTH=, LINKSTYLE=, and LINKWIDTH= option values apply to each plot that the ZOOMEDTREE option creates; if you specify any of these option values more than once, then only the last value specified is used.

The nodes in the decision tree are named using a base 62 convention, in order to improve the visual quality of the plots. That is, the first 10 nodes are named '0' through '9'. After '9', the next 26 nodes are named 'A' through 'Z'. The following 26 nodes are named 'a' through 'z'. After these first 62 nodes, a second digit is added, with the names starting at '10'. In a similar fashion, the name after '19' is '1A', the name after '1Z' is '1a', and the name after '1z' is '20'.

A quick conversion is the mapping 'A'=10 to 'Z'=35 and 'a'=36 to 'z'=61. Thus, for example, a node named '1F' would equal 1*62 + 15 = 77 (where 15 is the value of 'F').

NOLEGEND
turns off the legend.

PREDICTORPRECISION=*integer*
requests that the split value that is displayed above the decision tree node have a number of digits after the decimal equal to *integer*.

By default, PREDICTORPRECISION=3.

SEED=*number*
specifies the initial seed for random number generation for cross validation. The value of *number* must be an integer. The default seed is based on the date and time.

SPLITONCE
specifies that variables be split only once on a branch. However, a variable can be used more than once across branches. That is, a variable cannot be split more than once on the path from any leaf to the root node.

CLASS Statement

CLASS *variable* < **(***var-options***)** > < *variable* < **(***var-options***)** > < ... > > < / *class-options* > **;**

The CLASS statement causes *variable* to be treated as a categorical variable in the analysis. These variables enter the analysis not through their values but through levels to which the unique values are mapped. For

more information about these mappings, see the section "Levelization of Classification Variables" (Chapter 3, *SAS/STAT User's Guide: High-Performance Procedures*).

Multiple CLASS statements are supported.

NOTE: All class levels are padded or truncated to 32 characters.

By default, PROC HPSPLIT treats *variable*s as categorical variables whose order is specified by the ORDER= option.

You can specify the following *var-options*:

DESC

DESCENDING

reverses the sort order of the classification variable. If you specify both the DESCENDING and ORDER= options, PROC HPSPLIT orders the categories according to the ORDER= option and then reverses that order.

ORDER=*ordering*

specifies the sort order for the levels of classification variables. By default, ORDER=FORMATTED except for numeric CLASS variables that have no specified format, for which ORDER=INTERNAL is the default. You can specify the following values:

FORMATTED orders values in ascending order of the formatted value.

INTERNAL orders values in ascending order of the unformatted value.

You can specify the following *class-options*:

DESC

DESCENDING

reverses the sort order of the classification variable. If you specify both the DESCENDING and ORDER= options, PROC HPSPLIT orders the categories according to the ORDER= option and then reverse that order.

ORDER=*ordering*

specifies the sort order for the levels of classification variables. By default, ORDER=FORMATTED except for numeric CLASS variables that have no specified format, for which ORDER=INTERNAL is the default. You can specify the following values:

FORMATTED orders values in ascending order of the formatted value.

INTERNAL orders values in ascending order of the unformatted value.

UPCASE

uppercases the values of character-valued CLASS variables before levelizing them. For example, if the UPCASE option is in effect and a CLASS variable can take the values 'a', 'A', and 'b', then 'a' and 'A' represent the same level and the CLASS variable is treated as having only two values: 'A' and 'B'.

CODE Statement

CODE FILE=*filename* **;**

The CODE statement converts the final tree into SAS DATA step code that can be used for scoring. The code is written to the file that is specified by *filename*.

If you do not specify a CODE statement, no SAS DATA step code is output.

GROW Statement

GROW *criterion* **<(***options***)>** **;**

The GROW statement specifies the *criterion* by which to grow the tree. For more information, see the section "Splitting Criteria" on page 5032. For categorical responses, the available *criteria* are CHAID, CHISQUARE, ENTROPY, FASTCHAID, and GINI, and the default *criterion* is ENTROPY. For continuous responses, the available *criteria* are CHAID, FTEST, and RSS, and the default *criterion* is RSS.

For either categorical or continuous responses, you can specify the following *criterion*:

CHAID <(*options***)>**

For categorical predictors, CHAID uses values of a chi-square statistic (in the case of a classification tree) or an F statistic (in the case of a regression tree) to merge similar levels until the number of children in the proposed split reaches the number that you specify in the MAXBRANCH= option. The p-values for the final split determine the variable on which to split.

For continuous predictors, CHAID chooses the best single split until the number of children in the proposed split reaches the value that you specify in the MAXBRANCH= option.

You can specify the following *options*:

ALPHA=*value*

specifies the maximum p-value for a split to be considered.

By default, ALPHA=0.3.

BONFERRONI

requests a Bonferroni adjustment to the p-value for a variable after the split has been determined.

By default, no adjustment is made.

For categorical responses only, you can specify the following *criteria*:

CHISQUARE <(*options***)>**

uses a chi-square statistic to split each variable and then uses the p-values that correspond to the resulting splits to determine the splitting variable.

You can specify the following *options*:

ALPHA=*value*
specifies the maximum *p*-value for a split to be considered.

By default, ALPHA=0.3.

BONFERRONI
requests a Bonferroni adjustment to the *p*-value for a variable after the split has been determined.

By default, no adjustment is made.

ENTROPY
uses the gain in information (decrease in entropy) to split each variable and then to determine the split.

FASTCHAID < (*options***) >**
uses a Kolmogorov-Smirnov splitter to determine splits for each variable. The FastCHAID *criterion* follows a recursive method similar to that of Friedman (1977) after ordering the levels according to the response variable. The *criterion* then selects the split variable as the variable that has the smallest *p*-value.

You can specify the following *options*:

ALPHA=*value*
specifies the maximum *p*-value for a split to be considered.

By default, ALPHA=0.3.

BONFERRONI
requests a Bonferroni adjustment to the *p*-value for a variable after the split has been determined.

By default, no adjustment is made.

MINDIST=*number*
specifies the minimum Kolmogorov-Smirnov distance for a candidate split.

By default, MINDIST=0.01.

GINI
uses the decrease in the Gini index to split each variable and then to determine the split.

IGR
uses the entropy metric to split each variable and then uses the information gain ratio to determine the split.

The default *criterion* for categorical responses is ENTROPY.

For continuous responses only, you can specify the following *criteria*:

FTEST < (*options***) >**
uses an *F* statistic to split each variable and then uses the resulting *p*-value to determine the split variable.

You can specify the following *options*:

ALPHA=*value*
specifies the maximum *p*-value for a split to be considered.

By default, ALPHA=0.3.

BONFERRONI
requests a Bonferroni adjustment to the *p*-value for a variable after the split has been determined.

By default, no adjustment is made.

RSS

VARIANCE
uses the change in response variance to split each variable and then to determine the split.

The default *criterion* for continuous responses is RSS.

ID Statement

ID *variables* **;**

The ID statement lists one or more variables from the predictor data set that are to be transferred to the output data set that you specify in the OUTPUT statement.

For more information, see the section "ID Statement" (Chapter 3, *SAS/STAT User's Guide: High-Performance Procedures*).

MODEL Statement

MODEL *response* < **(***response-option***)** > **=** *variable* < *variable...* > **;**

The MODEL statement causes PROC HPSPLIT to create a tree model by using *response* as the response variable and *variable* as a predictor. By default, *variable* is treated as a continuous predictor if it is a numeric variable, or as a categorical variable if the variable also appears in the CLASS statement.

NOTE: Specifying a character variable in a MODEL statement without previously declaring it in a CLASS statement results in an error.

You can specify the following *response-option*:

EVENT='*category***'**
specifies the event level for a binary categorical response variable. PROC HPSPLIT associates this level with the event of interest (sometimes referred to as the positive outcome) for the purpose of computing sensitivity, specificity, and area under the curve (AUC) and creating receiver operating characteristic (ROC) curves. You can specify the value (formatted if a format is applied) of the event category in quotation marks.

The default level is the first level of the response variable as specified by the ORDER= option in the CLASS statement.

NOTE: The EVENT= option has no effect in the case of categorical response variables that have more than two levels, and it has no effect in the case of continuous response variables.

OUTPUT Statement

OUTPUT OUT=*SAS-data-set* **;**

The OUTPUT statement creates a data set that contains one observation for each observation in the input data set. The OUT= data set contains the following:

- the response variable
- any variables that you specify by using the ID statement
- the observation's assigned leaf number
- the observation's assigned node number

You can use the leaf number in conjunction with the RULES statement to examine the tree in more detail. You can use the node number in conjunction with the table that is produced using the NODES option to investigate the tree.

In addition, for regression trees the OUT= data set contains the following:

- the prediction for this observation
- the average value within the observation's assigned leaf in the validation partition, if you specify a validation partition in the PARTITION statement

In addition, for classification trees the OUT= data set contains the following for each response variable level:

- the fraction of training partition weight within the assigned leaf for the observation
- the fraction of validation partition weight within the assigned leaf for the observation, if you specify a validation partition in the PARTITION statement

PARTITION Statement

PARTITION < *partition-options* > **;**

The PARTITION statement specifies how observations in the predictor data set are logically partitioned into disjoint subsets for model training and validation. Either you can designate a variable in the predictor data set and a set of formatted values of that variable to determine the role of each observation, or you can specify proportions to use for random assignment of observations to each role.

NOTE: Attempting to use a PARTITION statement along with cross validation results in an error.

You can specify only one of the following *partition-options*:

FRACTION(VALIDATE=*fraction* **< SEED=***number* **>)**

requests that specified proportions of the observations in the predictor data set be randomly assigned to training and validation roles. You specify the proportions for testing and validation by using the VALIDATE= suboption. The SEED= suboption sets the seed. Because *fraction* is a per-observation probability, setting *fraction* too low can result in an empty or nearly empty validation set.

The default seed is based on the date and time.

Using the FRACTION option can cause different numbers of observations to be selected for the validation set because this option specifies a per-observation probability. Different partitions can be observed when the number of nodes or threads changes or when PROC HPSPLIT runs in alongside-the-database mode.

The following PARTITION statement shows how to use a probability of choosing a particular observation for the validation set:

```
partition fraction(validate=0.1 seed=1234);
```

In this example, any particular observation has a probability of 10% of being selected for the validation set. All nonselected records are in the training set. The SEED= suboption specifies the seed that is used for the random number generator.

ROLEVAR=*variable* **(TRAIN='***value***' VALIDATE='***value***')**

names the *variable* in the predictor data set whose values are used to assign roles to each observation. The formatted values of this *variable*, which are used to assign observations roles, are specified in the TRAIN= and VALIDATE= suboptions.

In the following example, the ROLEVAR= option specifies _PARTIND_ as the variable in the predictor data set that is used to select the data set:

```
partition rolevar=_partind_(TRAIN='1' VALIDATE='0');
```

The TRAIN= and VALIDATE= suboptions provide the values that indicate whether an observation is in the training or validation set, respectively. Observations in which the variable is missing or a value corresponds to neither argument are ignored. Formatting and normalization are performed before comparison, so you should specify numeric variable values as formatted values, as in the preceding example.

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 PROC HPSPLIT.

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

The PERFORMANCE statement is documented further in the section "PERFORMANCE Statement" (Chapter 2, *SAS/STAT User's Guide: High-Performance Procedures*).

PRUNE Statement

PRUNE *prune-method* < **(***prune-options***)** > **;**

The PRUNE statement specifies the pruning method and related options. You can specify the following *prune-methods*:

C45 < (*prune-option***) >**
requests C4.5 pruning (Quinlan 1993), which is based on the upper confidence limit for the error rate. For more information, see the section "Pruning" on page 5035. This pruning method is available only for classification trees (categorical responses). PROC HPSPLIT uses the error rate from the validation set, if one is available.

PROC HPSPLIT generates a subtree selection plot during pruning. The subtree selection plot shows the minimum change in prediction error as a function of the number of leaves in the subtree.

You can specify the following *prune-option*:

CONFIDENCE=*confidence-level*
specifies the pruning confidence level, which must be a positive number in the range of [0, 1]. The default confidence level is 0.25.

COSTCOMPLEXITY < (*prune-options***) >**
CC < (*prune-options***) >**
requests cost-complexity pruning (Breiman et al. 1984; Quinlan 1987; Zhang and Singer 2010). You can specify this pruning method for both classification trees and regression trees (continuous response). This is the default pruning method.

If you specify a validation set by using a PARTITION statement, PROC HPSPLIT uses the validation set for subtree selection. If you specify the number of leaves by using the LEAVES= option, the procedure selects the subtree that has the specified *number* of leaves, or if no subtree with exactly that *number* of leaves is available, it selects a subtree with fewer leaves. By default, if you do not specify a validation set by using a PARTITION statement and you do not use the LEAVES= option, the procedure uses k-fold cross validation for subtree selection.

For cost-complexity pruning, PROC HPSPLIT generates either a cost-complexity analysis plot based on cross validation or a cost-complexity pruning plot that shows the error metric for the training and validation sets when a validation set exists. The error metric is misclassification rate for classification trees and average square error (ASE) for regression trees.

You can specify the following *prune-option*:

LEAVES=*number* | **ALL**
specifies the subtree that has the requested *number* of leaves, or if no subtree with exactly that *number* of leaves is available, specifies a subtree with fewer leaves. When LEAVES=ALL, the largest tree is selected.

OFF
turns off pruning completely. No pruning is performed, and no pruning plots are generated.

REDUCEDERROR <(*prune-options*)>

REP <(*prune-options*)>

requests reduced-error pruning (Quinlan 1986). Reduced-error pruning has two stages: subtree sequence generation and subtree selection. For reduced-error pruning, if you specify a validation set by using a PARTITION statement, the validation set is used for subtree sequence generation and for subtree selection. Otherwise, the training set is used. For more information, see the section "Pruning" on page 5035.

For reduced-error pruning, PROC HPSPLIT generates a pruning plot that shows the requested error metric as a function of the number of leaves on the subtree.

You can specify the following *prune-options*:

LEAVES=*number* **| ALL**

specifies the subtree that has the requested *number* of leaves, or if no subtree with exactly that *number* of leaves is available, specifies a subtree with fewer leaves. When LEAVES=ALL, the largest tree is selected.

METRIC=ASE | MISC

specifies the metric for reduced-error pruning. Average square error (ASE) is the default metric. You can specify ASE for both classification trees and regression trees (continuous response). You can specify MISC only for classification trees (categorical response).

RULES Statement

RULES FILE=*filename* **;**

The RULES statement specifies a file for saving the rules that define the leaves for the final tree.

These rules can offer more details than are present in the tree plots and table that are produced using the NODES option in the PROC HPSPLIT statement. For example, if you have a large number of levels, the RULES file can provide extra space for those levels.

WEIGHT Statement

WEIGHT *variable* **< / UNWEIGHTDF > ;**

The *variable* in the WEIGHT statement is used as a weight to perform a weighted analysis of the data. Observations with nonpositive or missing weights are not included in the analysis. If a WEIGHT statement is not included, then all observations used in the analysis are assigned a weight of 1.

You can specify the following:

UNWEIGHTDF
uses the unweighted calculation for the *F* statistic rather than the weighted calculation for the *F* statistic. For more information, see the section "Splitting Criteria" on page 5032.

Details: HPSPLIT Procedure

Building a Decision Tree

Algorithms for building a decision tree use the training data to split the predictor space (the set of all possible combinations of values of the predictor variables) into nonoverlapping regions. These regions correspond to the terminal nodes of the tree, which are also known as leaves.

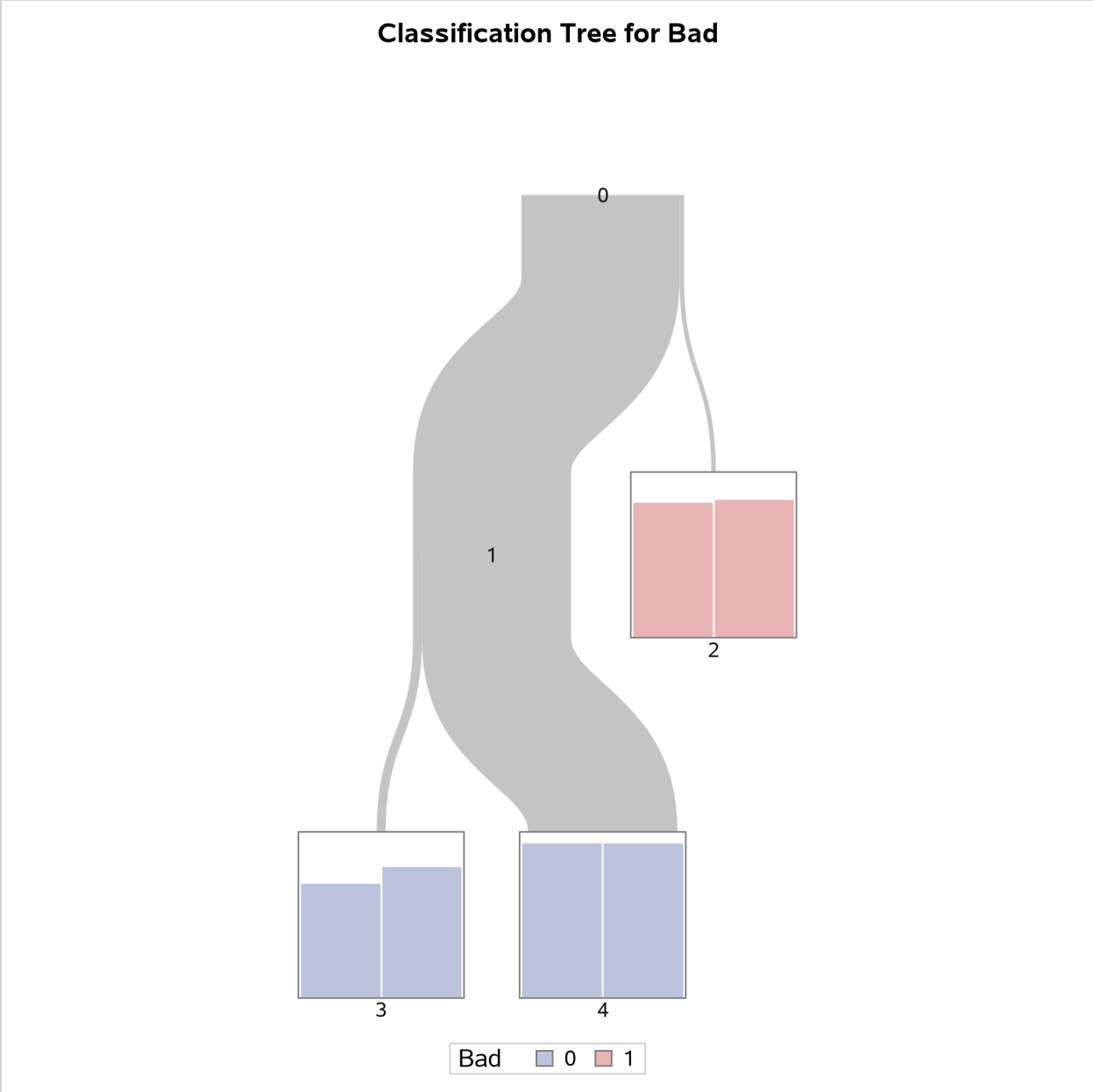
Each region is described by a set of rules, and these rules are used to assign a new observation to a particular region. In the case of a classification tree, the predicted value for this observation is the most commonly occurring level of the response variable in that region; in the case of a regression tree, the predicted value is the mean of the values of the response variables in that region.

The splitting is done by recursive partitioning, starting with all the observations, which are represented by the node at the top of the tree. The algorithm splits this parent node into two or more child nodes in such a way that the responses within each child region are as similar as possible. The splitting process is then repeated for each of the child nodes, and the recursion continues until a stopping criterion is satisfied and the tree is fully built.

At each step, the split is determined by finding a best predictor variable and a best cutpoint (or set of cutpoints) that assign the observations in the parent node to the child nodes. The sections "Splitting Criteria" on page 5032 and "Splitting Strategy" on page 5035 provide details about the splitting methods available in the HPSPLIT procedure.

To illustrate the process, consider the first two splits for the classification tree in "Example 65.4: Creating a Binary Classification Tree with Validation Data" on page 5070, which is shown in Figure 65.11.

Figure 65.11 First Two Splits for Hmeq Data Set



All 2,352 observations in the data are initially assigned to Node 0 at the top of the tree, which represents the entire predictor space. The algorithm then splits this space into two nonoverlapping regions, represented by Node 1 and Node 2. Observations with Debtinc < 48.8434 or with missing values of Debtinc are assigned to Node 1, and observations with Debtinc > 48.8434 are assigned to Node 2. Here the best variable, Debtinc, and the best cutpoint, 48.8434, are chosen to maximize the reduction in entropy as a measure of node impurity.

Next, the algorithm splits the region that is represented by Node 1 into two nonoverlapping regions, represented by Node 3 and Node 4. Observations with values of the categorical predictor variable Delinq equal to 2, 3, 4, 5, 6, or 7 are assigned to Node 3, and observations with Delinq equal to 0, 1, 8, 9, 10, 11, 12, 14, 15, or missing are assigned to Node 4. Note that the levels of Delinq in Node 4 are not adjacent to each other. Here the best variable, Delinq, and the values 2, 3, 4, 5, 6, and 7 are chosen to maximize the reduction in entropy.

Figure 65.12 Scatter Plot of the Predictor Space for the First Split

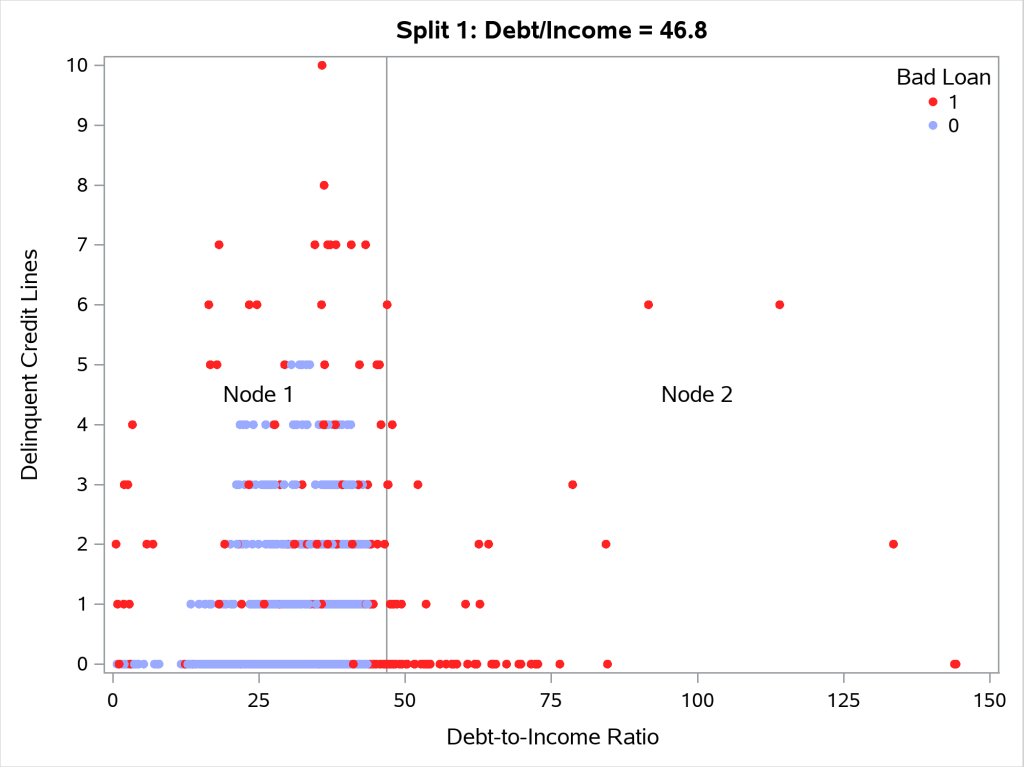
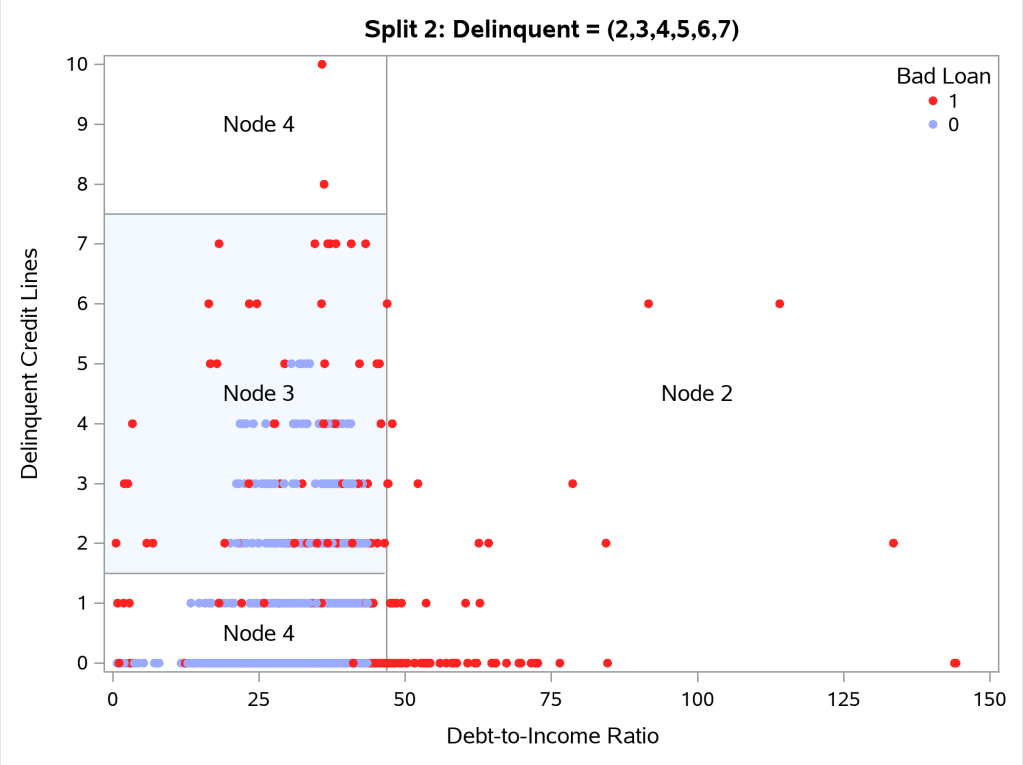


Figure 65.13 Scatter Plot of the Predictor Space for the Second Split



The tree that is defined by these two splits has three leaf (terminal) nodes, which are Nodes 2, 3, and 4 in Figure 65.13. Figure 65.12 and Figure 65.13 present scatter plots of the predictor space for these two splits one at a time. Notice in Figure 65.12 that the first split in Debt-to-Income Ratio divides the entire predictor space into Node 1 and Node 2, represented by two rectangular regions that have different ratios of events to nonevents for the response variable. In a similar way, notice in Figure 65.13 that the second split in Delinquent Credit Lines divides Node 1 into Node 3 and 4, which have different ratios of events to nonevents for the response variable. Also note that several observations have the same values for Debtinc or Delinq, giving the perception of a scatter plot that has fewer observations than there actually are.

For example, Node 4 has a very high proportion of observations for which Bad is equal to 0. In contrast, Bad is equal to 1 for all the observations in Node 2.

This example illustrates recursive binary splitting in which each parent node is split into two child nodes. By default, the HPSPLIT procedure finds two-way splits when building regression trees, and it finds k-way splits when building classification trees, where k is the number of levels of the categorical response variable. You can use the MAXBRANCH= option to specify the k-way splits of your tree.

Splitting Criteria

The goal of recursive partitioning, as described in the section "Building a Decision Tree" on page 5029, is to subdivide the predictor space in such a way that the response values for the observations in the terminal nodes are as similar as possible. The HPSPLIT procedure provides two types of criteria for splitting a parent node τ: criteria that maximize a decrease in node impurity, as defined by an impurity function, and criteria that are defined by a statistical test. You select the criterion by specifying an option in the GROW statement.

Criteria Based on Impurity

The entropy, Gini index, and RSS criteria decrease impurity. The impurity of a parent node τ is defined as $i(\tau)$, a nonnegative number that is equal to zero for a pure node—in other words, a node for which all the observations have the same value of the response variable. Parent nodes for which the observations have very different values of the response variable have a large impurity.

The HPSPLIT procedure selects the best splitting variable and the best cutoff value to produce the highest reduction in impurity,

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

where τ_b denotes the bth child node, $p(\tau_b|\tau)$ is the sum of the weights of observations in τ that are assigned to τ_b divided by the sum of the weights of observations in τ, and B is the number of branches after splitting τ.

If you specify a variable in the WEIGHT statement, then the weight of an observation is the value of the weight variable for that observation. If you omit the WEIGHT statement, then the weight of each observation is equal to 1. In this case, the sum of weights of observations is equal to the number of observations.

The impurity reduction criteria available for classification trees are based on different impurity functions $i(\tau)$ as follows:

- Entropy criterion (default)

  The entropy impurity of node τ is defined as

  $$i(\tau) = -\sum_{j=1}^{J} p_j \log_2 p_j$$

  where p_j is the sum of the weights of observations that have the jth response value divided by the sum of weights of all observations.

- Gini index criterion

  The Gini index criterion defines $i(\tau)$ as the Gini index that corresponds to the ASE of a class response and is given by

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

  For more information, see Hastie, Tibshirani, and Friedman (2009).

The impurity reduction criterion available for regression trees is as follows:

- RSS criterion (default)

  The RSS criterion, also referred to as the ANOVA criterion, defines $i(\tau)$ as the residual sum of squares,

  $$i(\tau) = \frac{1}{N_w(\tau)} \sum_{i=1}^{N(\tau)} (Y_i - \overline{Y}_w)^2$$

  where $N(\tau)$ is the number of observations in τ, $N_w(\tau)$ is the sum of weights of all observations in τ, Y_i is the response value of observation i, $\overline{Y}_w$ is the weighted average response of the observations in τ, and w_i is the weight of observation i.

  The weighted average response is defined as

  $$\overline{Y}_w = \frac{\sum_{i=1}^{N(\tau)} w_i Y_i}{\sum_{i=1}^{N(\tau)} w_i}$$

Criteria Based on Statistical Test

The chi-square, F-test, CHAID, and FastCHAID criteria are defined by statistical tests. These criteria calculate the worth of a split by testing for a significant difference in the response variable across the branches defined by a split. The worth is defined as $-\log(p)$, where p is the p-value of the test. You can adjust the p-values for these criteria by specifying the BONFERRONI option in the GROW statement.

The criteria based on statistical tests compute the worth of a split as follows:

- Chi-square criterion

  For categorical response variables, the worth is based on the p-value for the Pearson chi-square test that compares the frequencies of the levels of the response across the child nodes.

- F-test criterion

  For continuous response variables, the worth is based on the F test for the null hypothesis that the means of the response values are identical across the child nodes. The test statistic is

$$F = \frac{\mathrm{SS}_{\mathrm{between}}/(B-1)}{\mathrm{SS}_{\mathrm{within}}/(N_w(\tau)-B)}$$

  where

$$\mathrm{SS}_{\mathrm{between}} = \sum_{b=1}^{B} N_w(\tau_b)(\overline{Y}_w(\tau_b) - \overline{Y}_w(\tau))^2$$

$$\mathrm{SS}_{\mathrm{within}} = \sum_{b=1}^{B} \sum_{i=1}^{N(\tau_b)} w_i (Y_{bi} - \overline{Y}_w(\tau_b))^2$$

  If you specify the UNWEIGHTDF option in the WEIGHT statement, then the degrees of freedom for the F test are not weighted. The $\mathrm{SS}_{\mathrm{between}}$ and $\mathrm{SS}_{\mathrm{within}}$ formulas are the same when the degrees of freedom are either weighted or unweighted. The F statistic with unweighted degrees of freedom is

$$F = \frac{\mathrm{SS}_{\mathrm{between}}/(B-1)}{\mathrm{SS}_{\mathrm{within}}/(N(\tau)-B)}$$

Available for both categorical and continuous response variables:

- CHAID criterion

  For categorical and continuous response variables, CHAID is an approach first described by Kass (1980) that regards every possible split as representing a test. CHAID tests the hypothesis of no association between the values of the response (target) and the branches of a node. The Bonferroni adjusted probability is defined as $m\alpha$, where α is the significance level of a test and m is the number of independent tests.

  For categorical response variables, the HPSPLIT procedure also provides the FastCHAID criterion, which is a special case of CHAID. FastCHAID is faster to compute because it sorts the possible splits according to the response variable.

Splitting Strategy

When you are building a tree, it is computationally intensive to calculate the node purity or the node worth for all possible split points of every predictor variable. For this reason, the HPSPLIT procedure implements a strategy that combines three different methods of generating candidate splits. The exhaustive method computes the split criterion for all the levels of a predictor variable. The greedy method, which is based on the CHAID algorithm, finds candidates for splits by recursively halving the data. The fast-sort method computes splits in order of the proportions of levels of a categorical response or the average of a continuous response.

By default, PROC HPSPLIT first tries to find candidates for splits by using the exhaustive method. If the number of computations exceeds the number that you specify in the LEVTHRESH1= or LEVTHRESH2= option, the procedure switches to the greedy algorithm. If the number of computations for a specified predictor runs out with the greedy algorithm, the procedure switches to the fast-sort method.

For categorical predictor variables, the exhaustive method tries to group levels in every possible combination. If the number of computations exceeds the threshold before this method finishes, the HPSPLIT procedure switches to the greedy algorithm, which groups levels by adding levels one at a time to the number of groups that you specify in the MAXBRANCH= option. If the number of computations exceeds the threshold before the greedy algorithm finishes, the procedure sorts the split points by the proportion of a categorical response in the predictor levels or by the average of a continuous response. Use the LEVTHRESH1= option to specify the number of computations for nominal predictors.

Continuous predictor variables are first binned in equidistant intervals as specified by the INTERVALBINS= option. Binning reduces the number of computations when you are searching for candidate splits. Use the option LEVTHRESH2= to specify the threshold for the number of computations before switching from exhaustive to fast-sort method. The greedy method is not available for interval predictor variables.

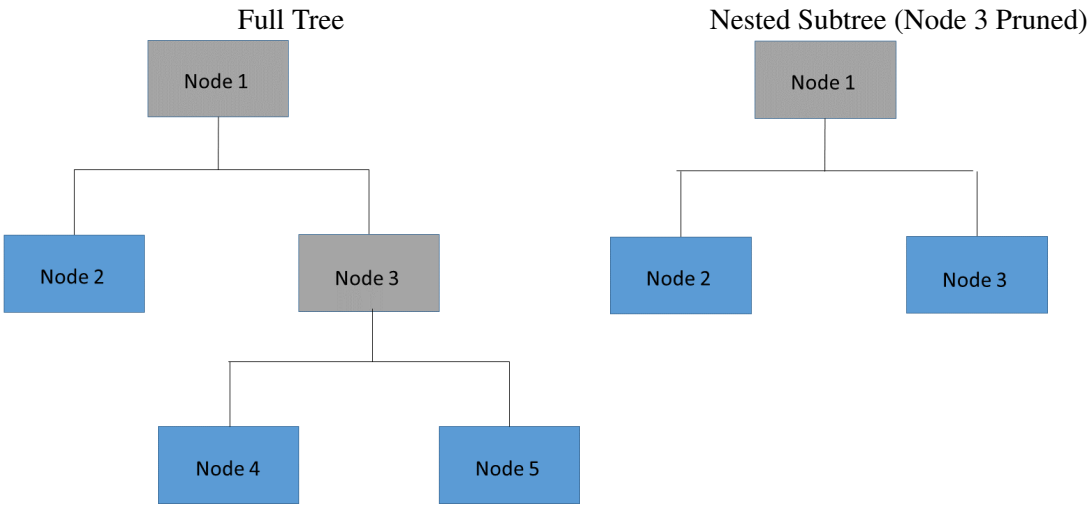
By default, a variable can be used more than once in a branch of a decision tree as long as the split is on a different value of that variable. You can specify the SPLITONCE option in the PROC HPSPLIT statement to request that variables be used only once per branch.

Pruning

The HPSPLIT procedure creates a classification or regression tree by first growing a tree as described in the section "Splitting Criteria" on page 5032. This usually results in a large tree that provides a good fit to the training data. The problem with this tree is its potential for overfitting the data: the tree can be tailored too specifically to the training data and not generalize well to new data. The solution is to find a smaller subtree that results in a low error rate on holdout or validation data.

It is often prohibitively expensive to evaluate the error on all possible subtrees of the full tree. A more practical strategy is to focus on a sequence of nested trees obtained by successively pruning leaves from the tree. Figure 65.14 shows an example of pruning in which the leaves (or terminal nodes) of Node 3 (Nodes 4 and 5) are removed to create a nested subtree of the full tree. In the nested subtree, Node 3 is now a leaf that contains all the observations that were previously in Nodes 4 and 5. This process is repeated until only the root node remains.

Figure 65.14 Tree and Pruned Subtree



Many different methods have been proposed for pruning in this manner. These methods address both how to select which nodes to prune to create the sequence of subtrees and how then to select the optimal subtree from this sequence as the final tree. You can use the PRUNE statement in PROC HPSPLIT to specify which pruning method to apply and related options. Several well-known pruning methods, described in this section, are available, and you can override the final selected tree based on your preferences or domain knowledge.

Cost-Complexity Pruning

Cost-complexity pruning is a widely used pruning method that was originally proposed by Breiman et al. (1984). You can request cost-complexity pruning for either a categorical or continuous response variable by specifying

```
prune costcomplexity;
```

This algorithm is based on making a trade-off between the complexity (size) of a tree and the error rate to help prevent overfitting. Thus large trees with a low error rate are penalized in favor of smaller trees. The cost complexity of a tree T is defined as

$$\mathrm{CC}(T) = \mathrm{R}(T) + \alpha|T|$$

where $\mathrm{R}(T)$ represents its error rate, $|T|$ represents the number of leaves on T, and the complexity parameter α represents the cost of each leaf. For a categorical response variable, the misclassification rate is used for the error rate, $\mathrm{R}(T)$; for a continuous response variable, the residual sum of squares (RSS), also called the sum of square errors (SSE), is used for the error rate. Note that only the training data are used to evaluate cost complexity.

Breiman et al. (1984) show that for each value of α, there is a subtree of T that minimizes cost complexity. When $\alpha = 0$, this is the full tree, T_0. As α increases, the corresponding subtree becomes progressively smaller, and the subtrees are in fact nested. Then, at some value of α, the root node has the minimal cost complexity for any α greater than or equal to that value. Because there are a finite number of possible

subtrees, each subtree corresponds to an interval of values of α; that is,

$$\begin{aligned}
[0, \alpha_1) &= \text{interval where } T_0 \text{ (the full tree) has minimal cost complexity} \\
[\alpha_1, \alpha_2) &= \text{interval where } T_1 \text{ has minimal cost complexity} \\
&\vdots \\
[\alpha_m, \infty) &= \text{interval where } T_m \text{ (the root node) has minimal cost complexity}
\end{aligned}$$

PROC HPSPLIT uses weakest-link pruning, as described by Breiman et al. (1984), to create the sequence of $\alpha_1, \ldots, \alpha_m$ values and the corresponding sequence of nested subtrees, $T_1, \ldots, T_m$.

Finding the optimal subtree from this sequence is then a question of determining the optimal value of the complexity parameter α. This is performed either by using the validation partition, when you use the PARTITION statement to reserve a validation holdout sample, or by using cross validation. In the first case, the subtree in the pruning sequence that has the lowest validation error rate is selected as the final tree. When there is no validation partition, k-fold cross validation can be applied to cost-complexity pruning to select a subtree that generalizes well and does not overfit the training data (Breiman et al. 1984; Zhang and Singer 2010). The algorithm proceeds as follows after creating the sequence of subtrees and α values by using the entire set of training data as described earlier:

1. Randomly divide the training observations into k approximately equal-sized parts, or folds.
2. Define a sequence of β values as the geometric mean of the endpoints of the $[\alpha_i, \alpha_{i+1})$ intervals (that is, $\beta_i = \sqrt{\alpha_i \times \alpha_{i+1}}$) to represent the intervals.
3. For each of the k folds, hold out the current fold for validation and use the remaining $k - 1$ folds for the training data in the following steps:
   a) Grow a tree as done using the full training data set with the same splitting criterion.
   b) Using the $\beta_1, \ldots, \beta_m$ values that are calculated in step 2, create a sequence of subtrees for each β_i as described in the pruning steps given earlier, but now using β_i as a fixed value for α and minimizing the cost complexity, CC(T), to select a subtree at each pruning step.
   c) For each β_i, set T_{ij} to be the subtree that has the minimum cost complexity from the sequence for the jth fold.
   d) Calculate the error $\mathrm{ASE}(T_{ij})$ for each T_{ij} by using the current (jth) fold (the one omitted from the training).
4. Now the error rate can be averaged across folds, $\overline{\mathrm{ASE}}_i = \frac{1}{k} \sum_{j=1}^{k} \mathrm{ASE}(T_{ij})$, and the β_i that has the smallest $\overline{\mathrm{ASE}}_i$ is selected. The tree T_i from pruning the complete training data that corresponds to the selected β_i is the final selected subtree.

The HPSPLIT procedure provides two plots that you can use to tune and evaluate the pruning process: the cost-complexity analysis plot and the cost-complexity pruning plot.

When performing cost-complexity pruning with cross validation (that is, no PARTITION statement is specified), you should examine the cost-complexity analysis plot that is created by default. This plot displays $\overline{\mathrm{ASE}}_i$ as a function of the complexity parameters β_i, and it uses a vertical reference line to indicate the β_i

that minimizes $\overline{\mathrm{ASE}}_i$. You can use this plot to examine alternative choices for β_i. For example, you might prefer to select a smaller tree that has only a slightly higher error rate. The plot also gives you the information that you need to implement the 1-SE rule developed by Breiman et al. (1984).

You can use the LEAVES= option in the PRUNE statement to select a tree that has a specified number of leaves. This is illustrated in "Example 65.2: Cost-Complexity Pruning with Cross Validation" on page 5062.

When you specify validation data by using the PARTITION statement, the cost-complexity pruning plot displays the error rate R(T) as a function of the number of leaves $|T|$ for both the training and validation data. This plot indicates the final selected tree, the tree with the minimum R(T) for the validation data, by using a vertical reference line. Like the cost-complexity analysis plot that is produced when you perform cross validation, this plot can help you identify a smaller tree that has only a slightly higher validation error rate. You could then use the LEAVES= option in a subsequent run of PROC HPSPLIT to obtain the final selected tree that has the number of leaves that you specify. See Output 65.4.7 in "Example 65.4: Creating a Binary Classification Tree with Validation Data" on page 5070 for an example of this plot.

C4.5 Pruning

Quinlan (1987) first introduced pessimistic pruning as a pruning method for classification trees in which the estimate of the true error rate is increased by using a statistical correction in order to prevent overfitting. C4.5 pruning (Quinlan 1993) then evolved from pessimistic pruning to employ an even more pessimistic (that is, higher) estimate of the true error rate. An advantage of methods like pessimistic and C4.5 pruning is that they enable you to use all the data for training instead of requiring a holdout sample. In C4.5 pruning, the upper confidence limit of the true error rate based on the binomial distribution is used to estimate the error rate. The C4.5 algorithm variant that PROC HPSPLIT implements uses the beta distribution in place of the binomial distribution for estimating the upper confidence limit. This pruning method is available only for categorical response variables and is implemented when you specify

```
prune C45;
```

The C4.5 pruning method follows these steps:

1. Grow a tree from the training data set, and call this full, unpruned tree T_0.
2. Solve the following equation for p_l, the adjusted prediction error rate for leaf l, for each leaf in the tree:

$$\alpha = 1 - \frac{\Gamma(N_l + 1)}{\Gamma(F_l + 1)\,\Gamma(N_l - F_l)} \int_0^{p_l} \nu^{F_l} (1-\nu)^{N_l - F_l + 1}\, \mathrm{d}\nu$$

   Here the confidence level α is specified in the CONFIDENCE= option, F_l is the number of failures (misclassified observations) at leaf l, N_l is the number of observations at leaf l, and the function $\Gamma(x)$ is defined as

$$\Gamma(x) = \int_0^{\infty} \nu^{x-1} e^{-\nu} \mathrm{d}\nu$$

3. Given these values of p_l, use the formula for the prediction error E of a tree T to calculate E_0 for the full tree T_0 before pruning:

$$E = \sum_{l \in T} N_l\, p_l$$

4. Consider all nodes that have only leaves as children for pruning in tree T_0.

5. Calculate the prediction error for each possible subtree that is created by replacing a node with a leaf by using the equations from steps 2 and 3, and prune the node that creates the subtree that has the largest decrease (or smallest increase) in prediction error from tree T_0. Let this be the next subtree in the sequence, T_1.

6. Repeat steps 2–5 with subtree T_1 from step 5 as the new "full" tree to use for creating the next subtree in the sequence, T_2. Continue until only the root node remains, represented by T_m.

The change in error is calculated between each pair of consecutive subtrees, $\Delta_i = E_i - E_{i-1}$. For the first change Δ_i that is greater than 0, for $i = 1, \ldots, m$, subtree T_{i-1} is selected as the final subtree. Note that for subtree selection, the change in error Δ_i is calculated on the validation partition when it exists; otherwise the training data are used.

Reduced-Error Pruning

Quinlan's reduced-error pruning (1987) performs pruning and subtree selection based on minimizing the error rate in the validation partition at each pruning step and then in the overall subtree sequence. This is usually based on the misclassification rate for a categorical response variable, but the average square error (ASE) can also be used. For a continuous response, the error is measured by the ASE. To implement reduced-error pruning, you can use the following PRUNE statement:

```
prune reducederror;
```

This pruning algorithm is implemented as follows, starting with the full tree, T_0:

1. Consider all nodes that have only leaves as children for pruning in tree T_0.

2. For each subtree that is created by replacing a node from step 1 with a leaf, calculate the error rate by using the validation data when available (otherwise use the training data).

3. Replace the node that has the smallest error rate with a leaf, and let T_1 be the subtree.

4. Repeat steps 1–3 with subtree T_1 from step 3 as the new "full" tree to use for creating the next subtree in the sequence, T_2. Continue until only the root node remains, represented by T_m.

This algorithm creates a sequence of subtrees from the largest tree, T_0, to the root node, T_m. The subtree that has the smallest validation error is then selected as the final subtree. Note that using this method without validation data results in the largest tree being selected. Also note that pruning could be stopped as soon as the error starts to increase in the validation data as originally described by Quinlan; continuing to prune to create a subtree sequence back to the root node enables you to select a smaller tree that still has an acceptable error rate, as discussed in the next section.

User Specification of Subtree

There might be situations in which you want to select a different tree from the one selected by default but using cost-complexity or reduced-error pruning to create the sequence of subtrees. For example, maybe there is a subtree that has a slightly larger error but is smaller, and thus simpler, than the subtree with the minimum ASE that was selected using reduced-error pruning. In that case, you can override the selected

subtree and instead select the subtree with n leaves that was created using cost-complexity or reduced-error pruning, where n is specified in the LEAVES= option in the PRUNE statement. You can implement the 1-SE rule for cost-complexity pruning described by Breiman et al. (1984) by using this option, as shown in "Example 65.2: Cost-Complexity Pruning with Cross Validation" on page 5062. Alternatively, you might want to select the largest tree that is created. In this case, you have two options: you can specify LEAVES=ALL in the PRUNE statement to still see the statistics for the sequence of subtrees created according to the specified pruning error measure, but the tree with no pruning performed is selected as the final subtree; or you can specify

```
prune off;
```

to select the largest tree with no pruning performed, meaning that statistics are not calculated and plots are not created on a sequence of subtrees.

Memory Considerations

The HPSPLIT procedure is designed for high-performance computing. As a result, it does not create utility files but rather stores all the data in memory. Data sets that have a large number of predictor variables and a large number of response levels can cause PROC HPSPLIT to run out of memory. One way to overcome this problem is to give SAS more memory to use. Another way is to use fewer threads, which reduces the memory that is required. You can use the NTHREADS= option in the PERFORMANCE statement to specify the number of threads. For more information, see the section "PERFORMANCE Statement" (Chapter 2, *SAS/STAT User's Guide: High-Performance Procedures*).

Primary and Surrogate Splitting Rules

The HPSPLIT procedure calculates primary and surrogate splitting rules for assigning the observations in a node to a branch. Both types of splitting rules use the value of a single predictor variable to assign an observation to a branch.

A primary splitting rule is always calculated by default, and it provides for the assignment of observations when the predictor variable is missing, even when there are no missing values in the training data. This allows for the possibility of missing values when you are scoring new data.

In addition, you can request one or more surrogate splitting rules by using the NSURROGATES= option. The purpose of a surrogate rule is to handle the assignment of observations by using an alternative variable that has similar predictive ability and has nonmissing values in observations where the primary predictor is missing. Surrogate rules enable you to make better use of the data. The HPSPLIT procedure uses the method of Breiman et al. (1984) to determine surrogate rules. By default, NSURROGATES=0. If you request one or more surrogate rules, the last column of the "Variable Importance" table shows the number of times that a variable is used as a surrogate. If a variable is used as a surrogate, you can see exactly how it is used in the SAS DATA step code that is generated when you specify the CODE statement.

When you request more than one surrogate rule, the rule that is applied to an observation is the first rule for which the surrogate variable is nonmissing. For example, consider the set of rules in Table 65.2 for a particular node in a decision tree where X, Y, and Z are three continuous predictors.

Table 65.2 Example of Splitting Rules

Rule	Description
Primary	If (NOT MISSING(X)) then (if(X < 1) then (assign to branch 2) else (assign to branch 1))
Surrogate 1	Else if (MISSING(X) and NOT MISSING(Y)) then (if(Y < 0) then (assign to branch 1) else (assign to branch 2))
Surrogate 2	Else if (MISSING(X) and MISSING(Y) and NOT MISSING(Z)) then (if(Z < 100) then (assign to branch 2) else (assign to branch 1))
Default	Else assign to branch 2

Note that the default rule ensures that observations that have missing values are always assigned to a branch regardless of whether you request surrogate rules.

PROC HPSPLIT selects a surrogate rule whose alternative predictor variable has the largest agreement with the predictor variable for the primary split. The measure of agreement between primary and surrogate splitting rules is the proportion of nonmissing observations (that is, observations without a missing value in the predictor variables) in the within-node training sample that the two rules assign to the same branch.

Handling Missing Values

Observations for which the response variable is missing are omitted from the analysis. The HPSPLIT procedure provides various methods of handling missing values of predictor variables.

By default, observations for which predictor variables are missing are omitted from the analysis. This behavior is common to other statistical modeling procedures in SAS/STAT software. Alternatively, you can use the ASSIGNMISSING= option to request different methods of dealing with missing values of predictor variables.

The ASSIGNMISSING=BRANCH option creates an extra branch for missing values. You determine the maximum number of branches by using the MAXBRANCH= option, which specifies the maximum number of branches per node in the tree. By default, MAXBRANCH=2. The ASSIGNMISSING=BRANCH option has no effect if there are no missing values in the training data set for a particular split. However, even if this is not the case, there could be missing values in the data set that is used for scoring, so this option is used to assign missing values that could be encountered in the future. For more information, see the section "Scoring" on page 5042.

The ASSIGNMISSING=POPULAR option assigns missing values to the most popular node of a split. If two or more branches have the same maximum number of observations, then the missing values are assigned to the branch that has the lowest node index.

The ASSIGNMISSING=SIMILAR option assigns missing values to the most similar node. Similarity is calculated using a chi-square test for categorical response variables and an F test for continuous response variables. The ASSIGNMISSING=SIMILAR option has no effect if there are no missing values in the training data set for a certain split. To handle this case, PROC HPSPLIT assigns future missing values to the most popular node of a split.

Unknown Values of Categorical Predictors

Unseen and infrequent levels of categorical predictors are referred to as unknown values, and they present problems for building decision trees in two situations. In the first situation, categorical predictors in validation or scoring data contain levels that do not occur in the training data. Unseen levels of this type are treated as missing values by splitting rules. In the second situation, categorical predictors in training data contain levels that occur very rarely and might be considered as outliers.

You can use the MINCATSIZE= option to specify the minimum number of occurrences that are required for a level to be used in splitting. By default, MINCATSIZE=1. By specifying a number greater than 1, you can filter out rarely occurring levels.

Note that for continuous predictors, the splitting rules always provide for the assignment of values regardless of whether they are unseen or infrequent. For example, a decision tree might have a branch based on the range "(Age > 20) AND (Age < 60)". Here "Age" is a continuous predictor. Any value of "Age" that is not in this range can be assigned to another branch of that decision tree.

Scoring

After you create a tree model, you can apply it to the training data for model diagnosis or to new data in order to make predictions. The process of applying a model to a data set is called *scoring*. You can use scoring to improve or deploy your model. There are two approaches to using PROC HPSPLIT to score a data set.

With the first approach, you can use the OUTPUT statement to score the training data. Usually, the purpose of scoring a training data set is to diagnose the model. The training data set is the data set that you specify by using the DATA= option. When scoring the training data, PROC HPSPLIT creates an output data set that contains one observation for each observation in the training data. You can specify the output data set by using the OUT= option in the OUTPUT statement.

In the following example, the input data set is scored after the tree model has been created:

```
proc hpsplit data=Sampsio.Hmeq;
   class Bad Delinq Derog Job Ninq Reason;
   model Bad = Delinq Derog Job Ninq Reason;
   output out=scored;
run;
```

With the second approach to scoring, the HPSPLIT procedure generates SAS DATA step code that you can use to score new data. Usually, the purpose of scoring new data is to make predictions. PROC HPSPLIT generates SAS DATA step code when you specify the CODE statement. For more information, see the section "Creating Score Code and Scoring New Data" in "Example 65.4: Creating a Binary Classification Tree with Validation Data" on page 5070.

With either approach, scoring a data set creates a data set that contains the new variables shown in Table 65.3.

Table 65.3 Scoring Variables

Variable	Description
Leaf	Leaf number to which the observation is assigned
Node	Node number to which the observation is assigned

In addition, for classification trees, the scored data set also contains new variables with the prefix "P_" and can contain new variables with the prefix "V_". There is one "P_" variable that corresponds to each level of the response variable. For all observations in the same leaf, these variables represent the proportion of the training observations in that leaf that have that particular response level. Similarly, if validation data are used, the variables that have the prefix "V_" represent the proportion of the validation observations in a leaf that have the corresponding response. For example, if the name of the categorical response variable is Color and it has two levels, `'Blue'` and `'Green'`, then the scored data set contains the variables P_ColorBlue and P_ColorGreen, which provide the proportions of training data in this leaf that have the response levels `'Blue'` and `'Green'`. If you use validation data, the HPSPLIT procedure creates two more new variables, V_ColorBlue and V_ColorGreen, which provide the proportions of validation data in this leaf that have the response levels `'Blue'` and `'Green'`.

For regression trees, the scored data set contains a new variable with the prefix "P_" and can contain another new variable with the prefix "V_". For all observations in the same leaf, the new variable that has the prefix "P_" represents the average value of the response variable for those observations. If you use validation data, the new variable with the prefix "V_" represents the average value of the response variable for the validation observations in the same leaf. For example, if the name of the continuous response variable is logSalary, then the scored data set contains a new variable, P_logSalary, to represent the average value of the response variable logSalary in the training data for observations on the same leaf. If you use validation data, the HPSPLIT procedure creates another new variable, V_logSalary, which provides the average value of the response variable for validation observations on the same leaf.

Measures of Model Fit

Various measures of model fit have been proposed in the data mining literature. The HPSPLIT procedure measures model fit based on a number of metrics for classification trees and regression trees.

If you specify a variable in the WEIGHT statement, then the weight of an observation is the value of the weight variable for that observation. If no WEIGHT statement is specified, then the weight of each observation is equal to one. In this case, the sum of weights of observations is equal to the number of observations.

Measures of Model Fit for Classification Trees

The HPSPLIT procedure measures model fit based on the following metrics for classification tree: entropy, Gini index, misclassification rate (Misc), residual sum of squares (RSS), average square error (ASE, also known as the Brier score), sensitivity, specificity, area under the curve (AUC), and confusion matrix.

Entropy for Classification Trees

Entropy for classifications tree is defined as

$$\text{Entropy} = -\sum_{\lambda} \frac{N_{w\lambda}}{N_{w0}} \sum_{\tau} \frac{N^{\lambda}_{w\tau}}{N_{w\lambda}} \log_2 \left(\frac{N^{\lambda}_{w\tau}}{N_{w\lambda}} \right)$$

where

- λ is a leaf
- $N_{w\lambda}$ is the sum of weights of observations on leaf λ
- N_{w0} is the total sum of weights of observations in the entire data set
- τ is a level of the response variable
- $N^{\lambda}_{w\tau}$ is the sum of weights of observations on leaf λ that have response level τ

Gini Index for Classification Trees

The Gini index for classification trees is defined as

$$\text{Gini} = \sum_{\lambda} \frac{N_{w\lambda}}{N_{w0}} \sum_{\tau} \frac{N^{\lambda}_{w\tau}}{N_{w\lambda}} \left(1 - \frac{N^{\lambda}_{w\tau}}{N_{w\lambda}} \right)$$

Misclassification Rate for Classification Trees

Misclassification (Misc) comes from the number of incorrectly predicted observations. It is defined as

$$\text{Misc} = \frac{1}{N_{w0}} \sum \begin{cases} 0 & \text{if prediction is correct} \\ w_i & \text{otherwise} \end{cases}$$

Residual Sum of Squares for Classification Trees

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

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

where

- Φ is the actual response level
- N^{λ}_{Φ} is the number of observations on leaf λ that have response level Φ
- $P^{\lambda}_{w\tau}$ is the weighted posterior probability for response level τ on leaf λ,

$$P_{w\lambda} = \frac{N^{\lambda}_{w\tau}}{N_{w\lambda}}$$

- $P^{\lambda}_{w\Phi}$ is the weighted posterior probability for the actual response level Φ on leaf λ,

$$P_{w\Phi} = \frac{N^{\Phi}_{w\tau}}{N_{w\lambda}}$$

Average Square Error for Classification Trees

The average square error (ASE) is also known as the Brier score for classification trees. It is defined as

$$\text{ASE} = \frac{\text{RSS}}{N_{w0} N_T}$$

where N_T is the number of levels for the response variable.

Sensitivity for Binary Classification Trees

Sensitivity is the probability of predicting an event for the response variable when the actual state is an event. For example, if the event is "an individual is sick," then sensitivity is the probability of predicting that an individual is sick given that the individual is actually sick. For binary classification trees, it is defined as

$$\text{Sensitivity} = \frac{\text{TP}_\text{w}}{P_w}$$

where

- TP is the sum of weights of true positives (predicting that an individual is sick)
- P is the sum of weights of positive observations (sick individuals)

Specificity for Binary Classification Trees

Specificity is the probability of predicting a nonevent for the response variable when the actual state is a nonevent. For example, if the event is "an individual is sick," then specificity is the probability of predicting that an individual is not sick given the fact that the individual is actually not sick. For a binary classification tree, specificity is defined as

$$\text{Specificity} = \frac{\text{TN}_\text{w}}{N_w}$$

where

- TN is the sum of weights of true negatives (predicting that an individual is not sick)
- N is the sum of weights of negative observations (healthy individuals)

Area under the Curve for Binary Classification Trees

Area under the curve (AUC) is defined as the area under the receiver operating characteristic (ROC) curve. PROC HPSPLIT uses sensitivity as the Y axis and 1 – specificity as the X axis to draw the ROC curve. AUC is calculated by trapezoidal rule integration,

$$\text{AUC} = \frac{1}{2} \sum_{\lambda} ((x_\lambda - x_{\lambda-1})(y_\lambda + y_{\lambda-1}))$$

where

- y_λ is the sensitivity value at leaf λ
- x_λ is the 1 – specificity value at leaf λ

NOTE: For a binary response, the event level that is used for calculating sensitivity, specificity, and AUC is specified in the EVENT= option in the MODEL statement.

Confusion Matrix for Classification Trees

A confusion matrix is also known as a contingency table. It contains information about actual values and predicted values from a classification tree. A confusion matrix has N_T rows and N_T columns, where each row corresponds to the actual response level and each column corresponds to the predicted response level. The values in the matrix represent the number of observations that have the actual response represented in the row and the predicted response represented in the column. The error rate per actual response level is also reported,

$$\text{ErrorRate} = \frac{N_{ww}}{N_{w\Phi}}$$

where

- N_{ww} is the sum of weights of wrong predictions
- $N_{w\Phi}$ is the sum of weights of observations that have response level Φ

Measures of Model Fit for Regression Trees

The HPSPLIT procedure measures model fit for regression trees based on RSS and ASE.

Residual Sum of Squares for Regression Trees

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

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

where

- i is an observation on leaf λ
- y_i is the predicted value of the response variable of observation i
- $\hat{y}_{\lambda}^{T}$ is the actual value of the response variable on leaf λ

Average Square Error for Regression Trees

The average square error (ASE) for regression trees is defined as

$$\text{ASE} = \frac{\text{RSS}}{N_{w0}}$$

Variable Importance

A training data set can contain a large number of predictors. Some predictors are useful for predicting the response variable, and others are not. You can use the HPSPLIT procedure to select the most useful predictors based on variable importance. (See "Example 65.5: Assessing Variable Importance" on page 5079.) Variable importance is an indication of which predictors are most useful for predicting the response variable. Various measures of variable importance have been proposed in the data mining literature.

The most important variables might not be the ones near the top of the tree. PROC HPSPLIT measures variable importance based on the following metrics: count, surrogate count, RSS, and relative importance. The count-based variable importance simply counts the number of times in the tree that a particular variable is used in a split. Similarly, the surrogate count tallies the number of times that a variable is used in a surrogate splitting rule.

The RSS-based metric measures variable importance based on the change of RSS when a split is found at a node. The change is

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

where

- d denotes the node
- i denotes the index of a child that this node has
- RSS_d is the RSS if the node is treated as a leaf
- RSS_i^d is the RSS of the node after it has been split

If the change in RSS is negative (which is possible when you use the validation set), then the change is set to 0.

If surrogate rules are in effect, they are also credited with a portion of the change in RSS. The credit is proportional to the agreement between the primary and surrogate splitting rules at the node. The agreement at node d, κ_d, is defined as

$$\kappa_d = \sum_i \frac{N_i}{N_d}$$

where

- N_d is the number of nonmissing observations
- N_i is the number of observations that were assigned to i by both the primary and surrogate rules

The change in RSS from the surrogate rules is defined as

$$\Delta_d = \kappa_d \left(\mathrm{RSS}_d - \sum_i \mathrm{RSS}_i^d \right)$$

The RSS-based importance is then defined as

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

where D is the total number of nodes.

The relative importance metric is a number between 0 and 1. It is calculated in two steps. First, PROC HPSPLIT finds the maximum RSS-based variable importance. Then, for each variable, it calculates the relative variable importance as the RSS-based importance of this variable divided by the maximum RSS-based importance among all the variables.

The RSS and relative importance are calculated from the training set. They are calculated again from the validation set if one exists.

ODS Table Names

Each table that the HPSPLIT procedure creates has a name associated with it, and you must use this name to refer to the table when you use ODS statements. These names are listed in Table 65.4.

Table 65.4 ODS Tables Produced by PROC HPSPLIT

Table Name	Description	Required Statement / Option
ConfusionMatrix	One or more confusion matrices	Default output for categorical response variable
CrossValidatedModel	Model assessment with cross validation	PROC / CVMODELFIT option
CrossValidationValues	Results for cost-complexity pruning based on cross validation	PROC / CVCC option
CVModelConfusionMatrix	Cross validation confusion matrix	PROC / CVMODELFIT option
DataAccessInfo	Information about modes of data access	Default output
ModelInfo	Information about the modeling environment	Default output
NodeTable	Node information	PROC / NODES option
NObs	Observation information	Default output
PerformanceInfo	Information about high-performance computing environment	Default output
Timing	Absolute and relative times for tasks performed by the procedure	PERFORMANCE / DETAILS option
TreePerformance	Fit statistics for the selected tree	Default output
VarImportance	Variable importance	Default output

ODS Graphics

You can refer to every graph that is produced through ODS Graphics by name. The names of the graphs that PROC HPSPLIT generates are listed in Table 65.5, along with the relevant PLOTS= options.

Table 65.5 Graphs Produced by PROC HPSPLIT

ODS Graph Name	Plot Description	PLOTS Option
CrossValidationASEPlot	Cross validation cost-complexity ASE plot	CVCC
PruningPlot	Plot of the error sum of squares, misclassification rate, or cost complexity when it is used for final tree selection	PRUNEUNTIL
ROCPlot	Plot of receiver operating characteristic (ROC) curve	ROC
WholeTreePlot	Overview plot of final tree	WHOLETREE
ZoomedTreePlot	Detailed plot of portion of final tree	ZOOMEDTREE

SAS Enterprise Miner Syntax and Notes

In addition to the syntax that is described in the CLASS and MODEL statement sections, PROC HPSPLIT supports SAS Enterprise Miner INPUT/TARGET syntax that many Enterprise Miner users are familiar with. The INPUT/TARGET syntax cannot be used together with the CLASS/MODEL syntax of SAS/STAT. Doing so is an error.

Enterprise Miner style syntax has one TARGET statement and one or more INPUT statements. If you use the Enterprise Miner syntax, then the PROC HPSPLIT statement, the TARGET statement, and the INPUT statement are required. Depending on the options in those statements, specified variables can be interval or nominal. By default, numeric INPUT variables are treated as interval (or continuous) predictors, and character INPUT variables are treated as nominal (or categorical) predictors.

INPUT *variables* < / *option* > **;**
TARGET *variable* < / *option* > **;**

INPUT Statement

INPUT *variables* < / *option* > **;**

The INPUT statement specifies predictor *variables* for the decision tree or regression tree. The value of *variable* can be a range such as "var_1–var_1000" or the special "_ALL_" value to include all variables in the data set. As with CLASS variables, all nominal INPUT variables are padded or truncated to 32 characters.

It is an error to use an INPUT statement with a MODEL or CLASS statement.

You can specify the following *option*:

LEVEL=INT | NOM
specifies whether the specified predictor *variables* are interval or nominal.

INT
treats all numeric *variables* as interval predictors.

NOM
treats all *variables* as nominal predictors.

By default, numeric *variables* are treated as interval predictors, and character *variables* are treated as nominal predictors. Specifying LEVEL=NOM forces all *variables* in that statement to be treated as nominal. PROC HPSPLIT ignores the LEVEL=INT option for character variables.

TARGET Statement

TARGET *variable* < / *options* > **;**

The TARGET statement names the *variable* whose values PROC HPSPLIT tries to predict. Missing values in the target are ignored except during scoring.

It is an error to use a TARGET statement with a MODEL or CLASS statement.

You can specify the following *options*:

LEVEL=INT | NOM
specifies whether the specified response *variable* is interval or nominal.

INT
treats the response as an interval variable and creates a regression tree.

NOM
treats the response as a nominal variable and creates a decision tree.

By default, LEVEL=NOM, and PROC HPSPLIT creates a decision tree (nominal response).

ORDER=*ordering*
ensures that the response values are levelized in the specified order. You can specify the following values:

ASC | ASCENDING levelizes response values in ascending order.

DESC | DESCENDING levelizes response values in descending order.

FMTASC | ASCFORMATTED levelizes response values in ascending order of the formatted value.

FMTDESC | DESFORMATTED levelizes response values in descending order of the formatted value.

By default, ORDER=DESC.

Example Classification Tree Syntax for SAS/STAT and SAS Enterprise Miner

The following two programs are equivalent. The first is based on the syntax in the section "Syntax: HPSPLIT Procedure" on page 5011, and the second is SAS Enterprise Miner syntax.

```
proc hpsplit data=sashelp.cars;
   class enginesize model;
   model enginesize = mpg_highway model;
run;

proc hpsplit data=sashelp.cars;
   input mpg_highway model;
   target enginesize;
run;
```

Example Regression Tree Syntax for SAS/STAT and SAS Enterprise Miner

The following two programs are equivalent. The first is based on the syntax in the section "Syntax: HPSPLIT Procedure" on page 5011, and the second is SAS Enterprise Miner syntax.

```
proc hpsplit data=sashelp.cars;
   class model;
   model enginesize = mpg_highway model;
run;

proc hpsplit data=sashelp.cars;
   input mpg_highway model;
   target enginesize / level=int;
run;
```

Note for SAS Enterprise Miner Users

NOTE: The RSS splitting criterion is also known as the variance splitting criterion.

Examples: HPSPLIT Procedure

Example 65.1: Building a Classification Tree for a Binary Outcome

This example illustrates how you can use the HPSPLIT procedure to build and assess a classification tree for a binary outcome. Overfitting is avoided by cost-complexity pruning, and the selection of the pruning parameter is based on cross validation.

The training data in this example come from the Lichen Air Quality Surveys (Geiser and Neitlich 2007), which were conducted in western Oregon and Washington between 1994 and 2001. Both the training data and the test data in this example were provided by Richard Cutler, Department of Mathematics and Statistics, Utah State University.

The LAQ data set consists of 30 measurements of environmental conditions, such as temperature, elevation, and moisture, at 840 sites. These variables are treated as predictors for the response variable LobaOreg, which is coded as 1 if the lichen species *Lobaria oregana* was present at the site and 0 otherwise.

```
proc print data=sampsio.LAQ(obs=5);
   var LobaOreg MinMinTemp Aconif PrecipAve Elevation ReserveStatus;
run;
```

Output 65.1.1 lists the first five observations for six of the variables in LAQ.

Output 65.1.1 Partial Listing of LAQ

Obs	LobaOreg	MinMinTemp	Aconif	PrecipAve	Elevation	ReserveStatus
1	0	-5.970	44.897	89.623	1567	Matrix
2	0	-6.430	81.585	91.231	1673	Reserve
3	1	-0.893	229.330	154.610	685	Reserve
4	0	-7.476	45.875	110.330	1971	Reserve
5	0	-5.992	81.679	98.739	1597	Reserve

The following statements invoke the HPSPLIT procedure to create a classification tree for LobaOreg:

```
ods graphics on;

proc hpsplit data=sampsio.LAQ seed=123;
   class LobaOreg ReserveStatus;
   model LobaOreg (event='1') =
      Aconif DegreeDays TransAspect Slope Elevation PctBroadLeafCov
      PctConifCov PctVegCov TreeBiomass EvapoTransAve EvapoTransDiff
      MoistIndexAve MoistIndexDiff PrecipAve PrecipDiff RelHumidAve
      RelHumidDiff PotGlobRadAve PotGlobRadDiff AveTempAve AveTempDiff
      DayTempAve DayTempDiff MinMinTemp MaxMaxTemp AmbVapPressAve
      AmbVapPressDiff SatVapPressAve SatVapPressDiff ReserveStatus;
   grow entropy;
   prune costcomplexity;
run;
```

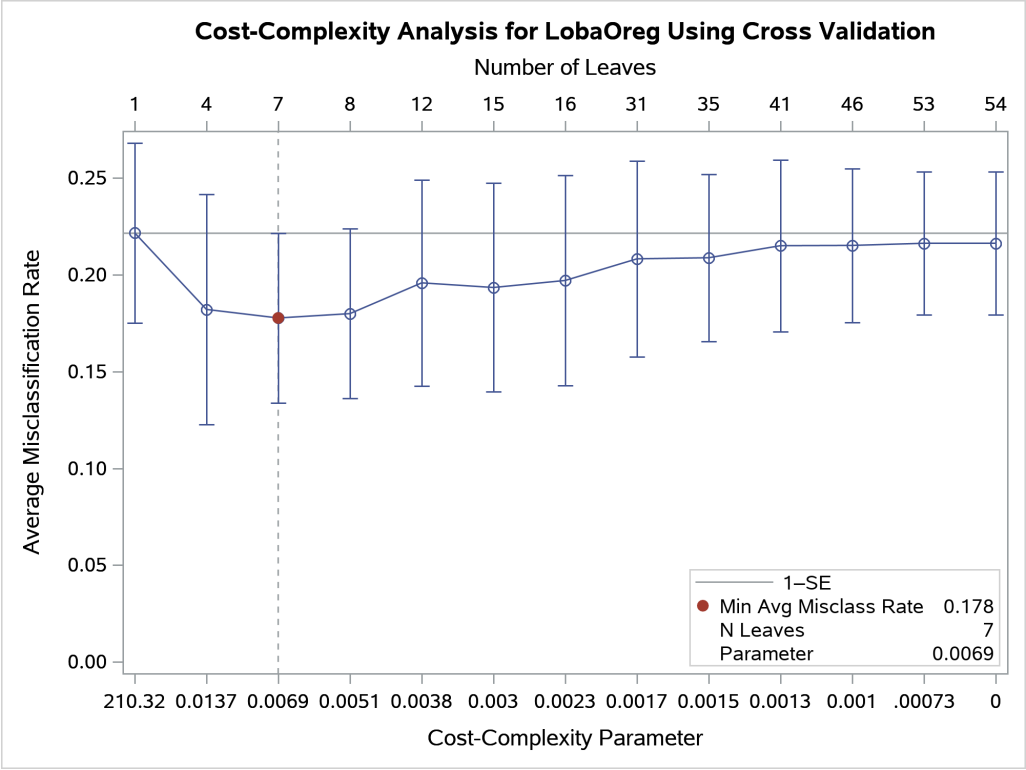
In the case of binary outcomes, the EVENT= option is used to explicitly control the level of the response variable that represents the event of interest for computing the area under the curve (AUC), sensitivity, specificity, and values of the receiver operating characteristic (ROC) curves.

NOTE: These fit statistics do not apply to categorical response variables that have more than two levels, so the EVENT= option does not apply in that situation. Likewise, this option does not apply to continuous response variables.

The GROW statement specifies the entropy criterion for splitting the observations during the process of recursive partitioning that results in a large initial tree. The PRUNE statement requests cost-complexity pruning to select a smaller subtree that avoids overfitting the data.

The plot in Output 65.1.2 displays estimates of the misclassification rate for a series of progressively smaller subtrees of the large tree. The subtrees are indexed by a cost-complexity parameter (also called a pruning parameter or a tuning parameter). The plot provides a tool for selecting the parameter that results in the smallest misclassification rate.

Output 65.1.2 Misclassification Rate as a Function of Cost-Complexity Parameter



The misclassification rate is estimated by 10-fold cross validation; for computational details, see the section "Cost-Complexity Pruning" on page 5036. The subtree size (number of leaves) that corresponds to each parameter is indicated on the upper horizontal axis. The parameter value 0 corresponds to the fully grown tree, which has 58 leaves.

By default, PROC HPSPLIT selects the parameter that minimizes the average misclassification rate, as indicated by the vertical reference line and the dot in Output 65.1.2. Here the minimum rate occurs at a parameter value of 0.0038, which corresponds to a subtree that has seven leaves. However, the rates for two smaller subtrees, one that has four leaves and one that has six leaves, are within one standard error of the minimum rate.

In general, the plot in Output 65.1.2 often shows parameter choices that correspond to smaller subtrees for which the misclassification rates are nearly the same as the minimum misclassification rate. A common approach for choosing the parameter is the 1-SE rule of Breiman et al. (1984), which selects the smallest subtree for which the misclassification rate is less than the minimum rate plus one standard error. In this case, the 1-SE rule selects a subtree that has only four leaves, but this is such a small subtree that the subtree with six leaves and a parameter value of 0.0069 seems preferable.

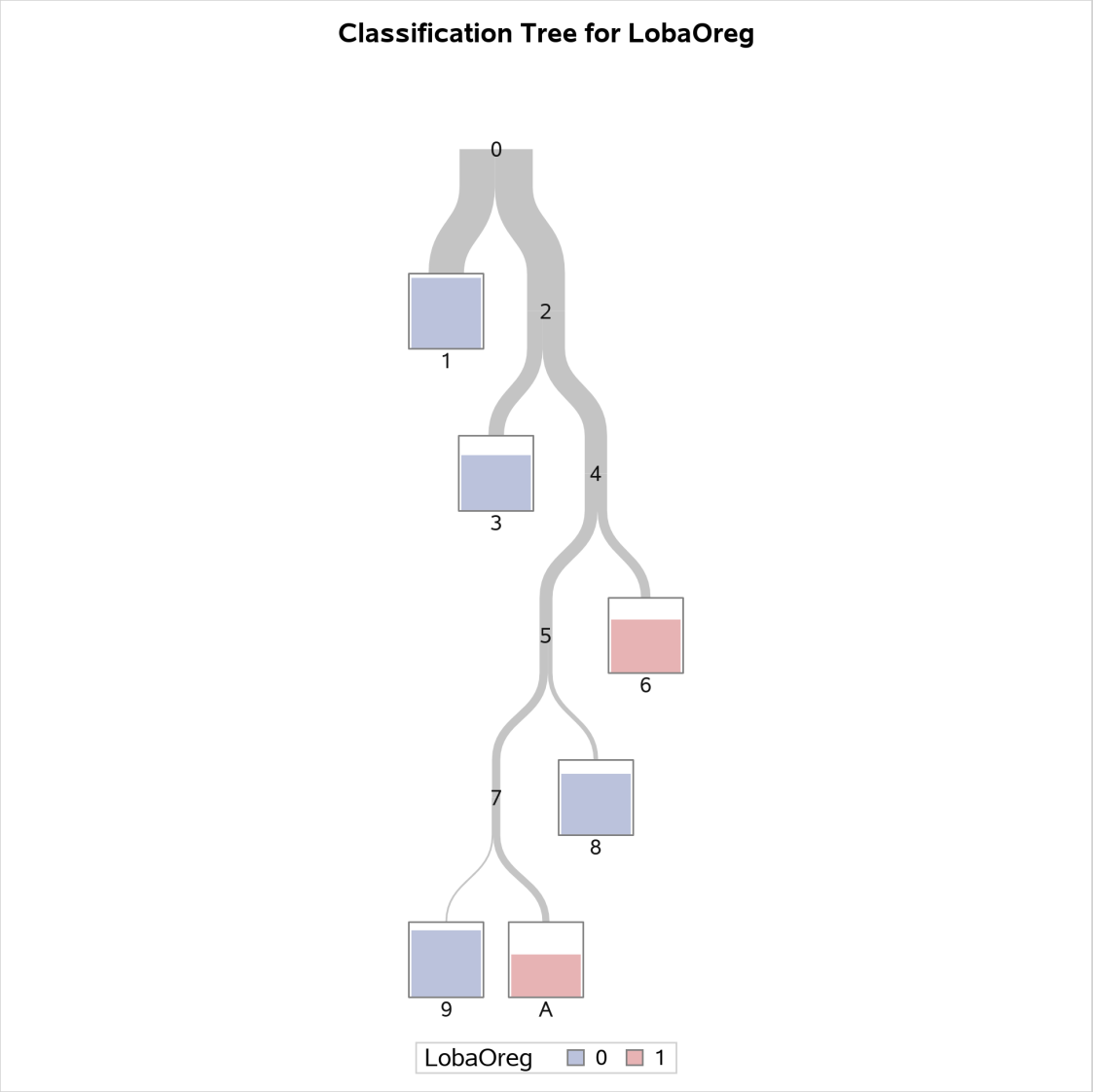
NOTE: The misclassification rates and their standard errors are estimates that depend on the random allocation of the observations to the 10 folds. You obtain different estimates if you specify a different SEED= value. Likewise, the estimates differ if you request a different number of folds by using the CVMETHOD= option.

The following statements rerun the analysis and request a tree with six leaves:

```
proc hpsplit data=sampsio.LAQ cvmodelfit seed=123;
   class LobaOreg ReserveStatus;
   model LobaOreg (event='1') =
      Aconif DegreeDays TransAspect Slope Elevation PctBroadLeafCov
      PctConifCov PctVegCov TreeBiomass EvapoTransAve EvapoTransDiff
      MoistIndexAve MoistIndexDiff PrecipAve PrecipDiff RelHumidAve
      RelHumidDiff PotGlobRadAve PotGlobRadDiff AveTempAve AveTempDiff
      DayTempAve DayTempDiff MinMinTemp MaxMaxTemp AmbVapPressAve
      AmbVapPressDiff SatVapPressAve SatVapPressDiff ReserveStatus;
   grow entropy;
   prune costcomplexity(leaves=6);
   code file='trescore.sas';
run;
```

The tree diagram in Output 65.1.3 provides an overview of the entire tree.

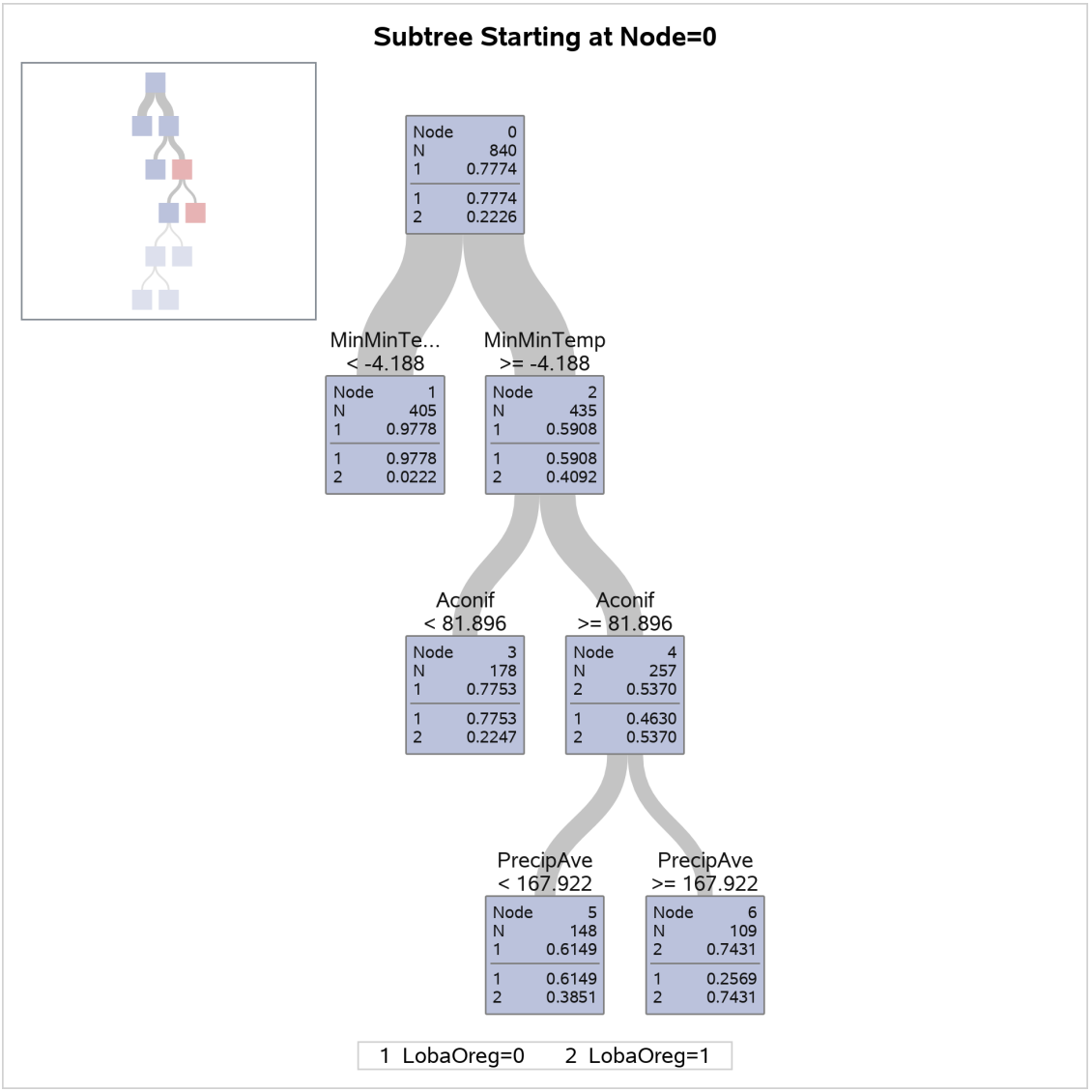
Output 65.1.3 Overview of Fitted Tree



The color of the bar in each leaf node indicates the most frequent level of LobaOreg and represents the classification level assigned to all observations in that node. The height of the bar indicates the proportion of observations (sites) in the node that have the most frequent level.

The diagram in Output 65.1.4 provides more detail about the nodes and splits in the first four levels of the tree. It reveals a model that is highly interpretable.

Output 65.1.4 First Four Levels of Fitted Tree



The first split is based on the variable MinMinTemp, which is the minimum of the 12 monthly minimum temperatures at each site. There are 405 sites whose values of MinMinTemp are less than –4.188 (node 1), and the species *Lobaria oregana* is present at only around 2% of them. Apparently this species does not grow well when the temperature can get very low.

The 435 sites for which MinMinTemp $\geq$ –4.188 (node 2) are further subdivided based on the variable Aconif, which is the average age of the dominant conifer at the site. *Lobaria oregana* is present at 53.7% of the 257 sites for which MinMinTemp $\geq$ –4.188 and Aconif $\geq$ 81.896 years. The cutoff of 81.896 years is notable because coniferous forests in the Pacific Northwest begin to exhibit old-forest characteristics at approximately 80 years of age (Old-Growth Definition Task Group 1986), and *Lobaria oregana* is a lichen species that is associated with old forests.

The 257 sites for which Aconif $\geq$ 81.896 are further subdivided on the basis of PrecipAve (average monthly precipitation) with a cutoff value 167.922 mm. *Lobaria oregana* was present at 74.31% of the 109 sites for which MinMinTemp $\geq$ –4.188, Aconif $\geq$ 81.896 years, and PrecipAve $\geq$ 167.922 mm. Contrast this occupancy percentage with the 2.22% for the sites for which MinMinTemp $<$ –4.188.

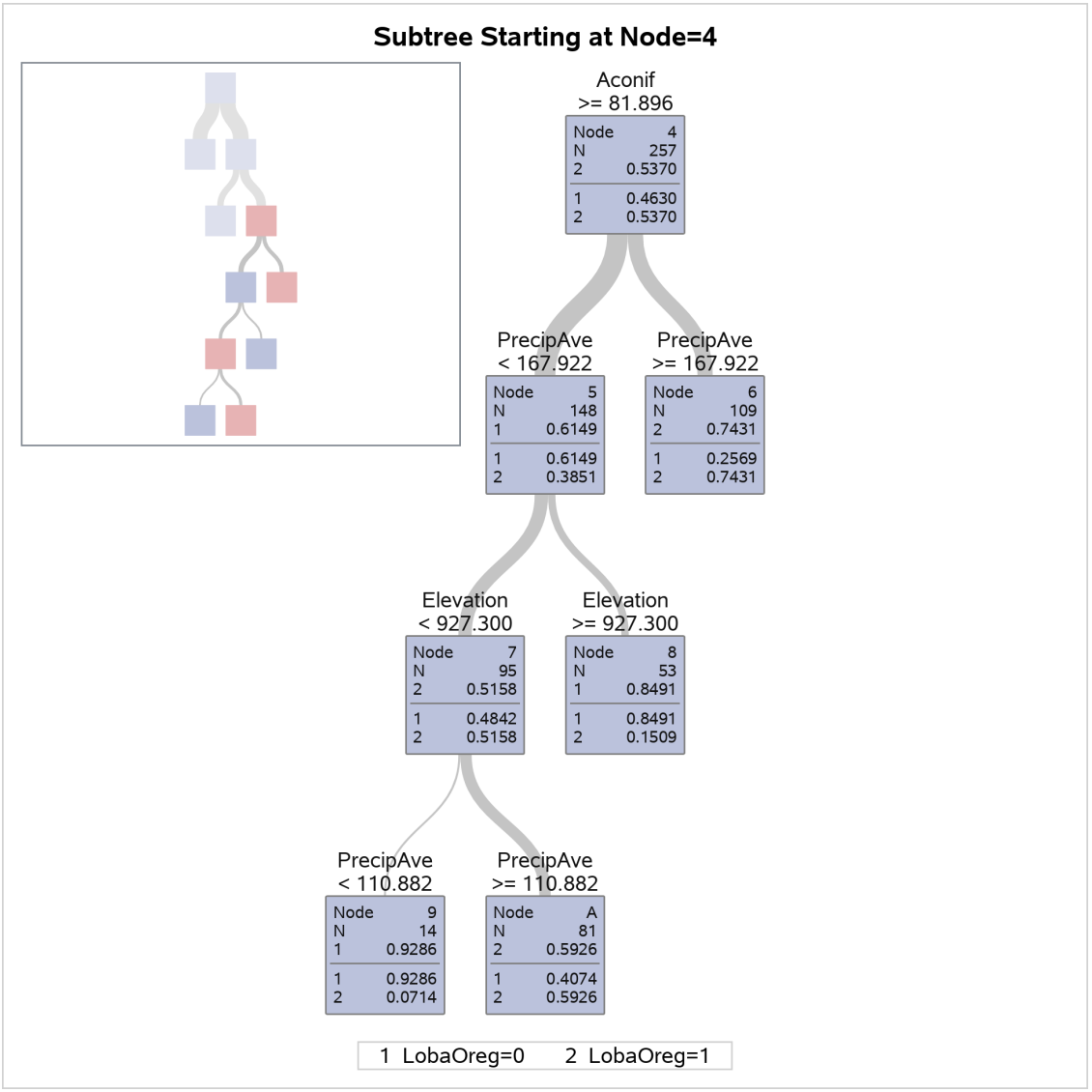
In summary, based on the first three splits, *Lobaria oregana* is most likely to be found at sites for which MinMinTemp $\geq$ –4.188, Aconif $\geq$ 81.896, and PrecipAve $\geq$ 167.922.

The following statements use the PLOTS=ZOOMEDTREE option to request a detailed diagram of the subtree that begins at node 4:

```
proc hpsplit data=sampsio.LAQ cvmodelfit seed=123
   plots=zoomedtree(nodes=('4') depth=4);
   class LobaOreg ReserveStatus;
   model LobaOreg (event='1') =
      Aconif DegreeDays TransAspect Slope Elevation PctBroadLeafCov
      PctConifCov PctVegCov TreeBiomass EvapoTransAve EvapoTransDiff
      MoistIndexAve MoistIndexDiff PrecipAve PrecipDiff RelHumidAve
      RelHumidDiff PotGlobRadAve PotGlobRadDiff AveTempAve AveTempDiff
      DayTempAve DayTempDiff MinMinTemp MaxMaxTemp AmbVapPressAve
      AmbVapPressDiff SatVapPressAve SatVapPressDiff ReserveStatus;
   grow entropy;
   prune costcomplexity(leaves=6);
   code file='trescore.sas';
run;
```

This subtree, shown in Output 65.1.5, provides details for the portion of the entire tree that is not shown in Output 65.1.4.

Output 65.1.5 Bottom Four Levels of Fitted Tree



NOTE: You can use the NODES= option to request a table that describes the path from each leaf in the fitted tree back to the root node.

The next three displays evaluate the accuracy of the selected classification tree. Output 65.1.6 provides two confusion matrices.

Output 65.1.6 Confusion Matrices for Classification of LAQ

The HPSPLIT Procedure

Confusion Matrices

		Predicted		
	Actual	0	1	Error Rate
Model Based	**0**	592	61	0.0934
	1	58	129	0.3102
Cross Validation	**0**	571	82	0.1256
	1	81	106	0.4332

The model-based confusion matrix is sometimes referred to as a resubstitution confusion matrix, because it results from applying the fitted model to the training data. The cross validation confusion matrix is produced when you specify the CVMODELFIT option. It is based on a 10-fold cross validation that is done independently of the 10-fold cross validation that is used to estimate ASEs for pruning parameters.

The table in Output 65.1.7 provides fit statistics for the selected classification tree.

Output 65.1.7 Fit Statistics for Classification of LAQ

Fit Statistics for Selected Tree

	N Leaves	ASE	Mis-class	Sensitivity	Specificity	Entropy	Gini	RSS	AUC
Model Based	6	0.1046	0.1417	0.6898	0.9066	0.4825	0.2093	175.8	0.8805
Cross Validation	6	0.1236	0.1914	0.5668	0.8744				

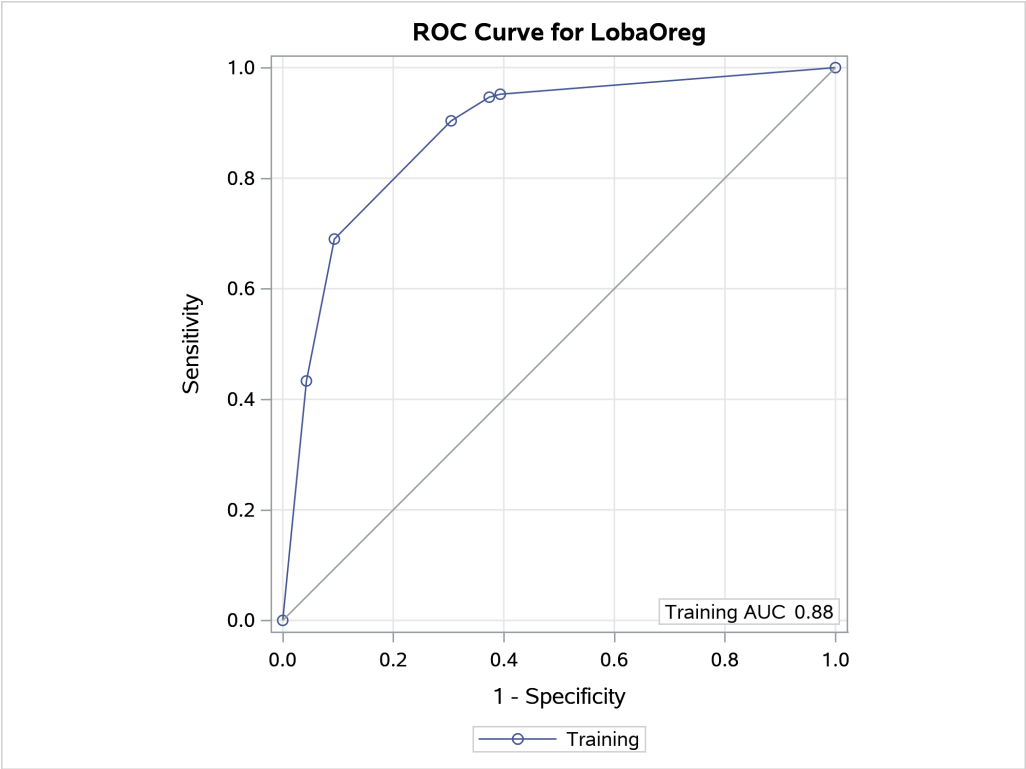
Two sets of fit statistics are provided. The first is based on the fitted model, and the second (requested by the CVMODELFIT option) is based on 10-fold cross validation.

The model-based misclassification rate is low (14.2%), but the corresponding sensitivity, which measures the prediction accuracy at sites where the species is present, is only 69%. Good overall prediction accuracy but poor prediction of a particular level can occur when the data are not well balanced (in this case there are three times as many sites where the species is absent as there are sites where the species is present).

The cross validation misclassification rate is higher (18.1%) than the model-based rate, and the cross validation sensitivity is 64%, which is lower than the model-based sensitivity. Model-based error rates tend to be optimistic.

Output 65.1.8 displays an ROC curve, which is produced only for binary outcomes.

Output 65.1.8 ROC Curve for Classification of LAQ



The AUC statistic and the values of the ROC curve are computed from the training data. When you specify a validation data set by using the PARTITION statement, the plot displays an additional ROC curve and AUC statistic, whose values are computed from the validation data.

NOTE: In this example, the computations of the sensitivity, specificity, AUC, and values of the ROC curve depend on defining LobaOreg=1 as the event of interest by using the EVENT= option in the MODEL statement.

Another way to assess the predictive accuracy of the fitted tree is to apply it to an independent test data set. In 2003, data on the presence of *Lobaria oregana* were collected at 300 sites as part of the Survey and Manage Program within the Northwest Forest Plan, the conservation plan for the northern spotted owl (Molina et al. 2003). These sites were in approximately the same area where the earlier Lichen Air Quality Surveys were conducted, with similar (but not identical) sampling protocols. The 2003 surveys are sometimes called the Pilot Random Grid surveys (Edwards et al. 2005). *Lobaria oregana* was detected at 26.7% (80) of the sites, an occupancy rate comparable to that of the Lichen Air Quality Surveys (187 / 840 = 22.3%).

The CODE statement in the preceding PROC HPSPLIT step requests a file named *trescore.sas* that contains SAS DATA step code for the rules that define the fitted tree model. You can use this code to classify the observations in the PRG data set as follows:

```
data lichenpred(keep=Actual Predicted);
   set sampsio.PRG end=eof;
   %include "trescore.sas";
   Actual    = LobaOreg;
   Predicted = (P_LobaOreg1 >= 0.5);
run;
```

The variables P_LobaOreg1 and P_LobaOreg0 contain the predicted probabilities that *Lobaria oregana* is present or absent, respectively. The value of the expression `(P_LobaOreg1 >= 0.5)` is 1 when P_LobaOreg1 $\geq$ P_LobaOreg0.

The following statements use these probabilities to produce a confusion matrix:

```
title "Confusion Matrix Based on Cutoff Value of 0.5";
proc freq data=lichenpred;
   tables Actual*Predicted / norow nocol nopct;
run;
```

The matrix is shown in Output 65.1.9.

Output 65.1.9 Confusion Matrix Based on Cutoff Value of 0.5

Confusion Matrix Based on Cutoff Value of 0.5

The FREQ Procedure

Frequency	Table of Actual by Predicted		
	Predicted		
Actual	0	1	Total
0	191	29	220
1	40	40	80
Total	231	69	300

The misclassification rate is (40 + 29) / 300 = 0.23, which is higher than the rates in Output 65.1.7. The sensitivity is 50%, which is less than the cross validation sensitivity in Output 65.1.7. The specificity is 86.8%.

When the sensitivity is much smaller than the specificity (which is common for highly unbalanced data), you can change the probability cutoff to a smaller value, increasing the sensitivity at the expense of the specificity and the overall accuracy. The following statements produce the confusion matrix for a cutoff value of 0.1:

```
data lichenpred(keep=Actual Predicted);
   set sampsio.PRG end=eof;
   %include "trescore.sas";
   Actual    = LobaOreg;
   Predicted = (P_LobaOreg1 >= 0.1);
run;

title "Confusion Matrix Based on Cutoff Value of 0.1";
proc freq data=lichenpred;
```

```
   tables Actual*Predicted / norow nocol nopct;
run;
```

The matrix is shown in Output 65.1.10.

Output 65.1.10 Confusion Matrix Based on Cutoff Value of 0.1

Confusion Matrix Based on Cutoff Value of 0.1

The FREQ Procedure

Frequency	Table of Actual by Predicted		
	Predicted		
Actual	0	1	Total
0	160	60	220
1	35	45	80
Total	195	105	300

Based on the cutoff value of 0.1, the sensitivity is 56.3% and the specificity is 72.7%.

Example 65.2: Cost-Complexity Pruning with Cross Validation

In this example, data were collected to study the damage to pine forests from mountain pine beetle attacks in the Sawtooth National Recreation Area (SNRA) in Idaho (Cutler et al. 2003). (The data in this example were provided by Richard Cutler, Department of Mathematics and Statistics, Utah State University.) A classification tree is applied to classify various types of vegetation in the area based on data from satellite images. This classification can then be used to track how the pine beetle infestation is progressing through the forest. Data from 699 points in the SNRA are included in the sample.

This example creates a classification tree to predict the response variable Type, which contains the 10 vegetation classes represented in the data: `'Agriculture'`, `'Dirt'`, `'DougFir'`, `'Grass'`, `'GreenLP'`, `'RedTop'`, `'Road'`, `'Sagebrush'`, `'Shadow'`, and `'Water'`. The predictor variables include the following:

- the spectral intensities on four bands of the satellite imagery: Blue, Green, Red, and NearInfrared
- Elevation
- NDVI, a function of Red and NearInfrared
- "Tasseled cap transformations" of the intensities on the four bands of imagery: SoilBrightness, Greenness, Yellowness, and NoneSuch

The first step in the analysis is to run PROC HPSPLIT to identify the best subtree model:

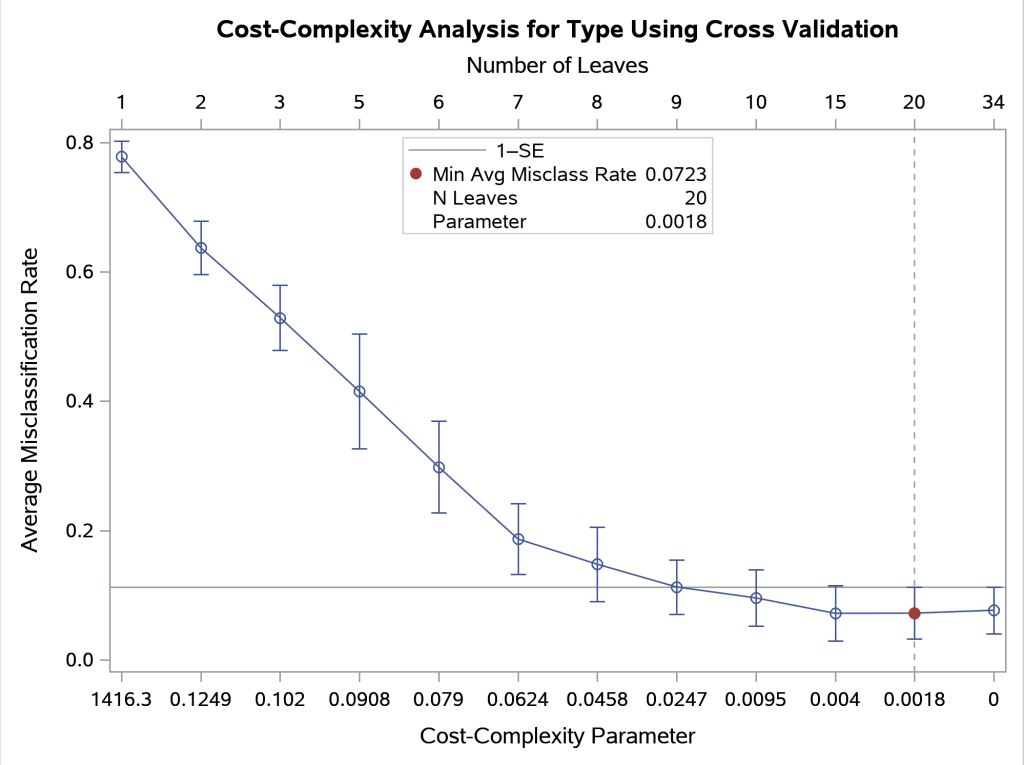
```
ods graphics on;

proc hpsplit data=sampsio.snra cvmethod=random(10) seed=123 intervalbins=500;
   class Type;
   grow gini;
```

```
   model Type = Blue Green Red NearInfrared NDVI Elevation
                SoilBrightness Greenness Yellowness NoneSuch;
   prune costcomplexity;
run;
```

You grow the tree by using the Gini index criterion, specified in the GROW statement, to create splits. This is a relatively small data set, so in order to use all the data to train the model, you apply cross validation with 10 folds, as specified in the CVMETHOD= option, to the cost-complexity pruning for subtree selection. An alternative would be to partition the data into training and validation sets. The SEED= option ensures that results remain the same in each run of the procedure. Different seeds can produce different trees because the cross validation fold assignments vary. When you do not specify the SEED= option, the seed is assigned based on the time.

By default, PROC HPSPLIT creates a plot of the estimated misclassification rate at each complexity parameter value in the sequence, as displayed in Output 65.2.1.

Output 65.2.1 Misclassification Rate as a Function of Cost-Complexity Parameter



The ends of the error bars correspond to the misclassification rate plus or minus one standard error (SE) at each of the complexity pruning parameter values. A vertical reference line is drawn at the complexity parameter that has the lowest misclassification rate, and the subtree of the corresponding size for that complexity parameter is selected as the final tree. In this case, the 15-leaf tree is selected as the final tree. The horizontal reference line represents the minimum misclassification rate plus one standard error.

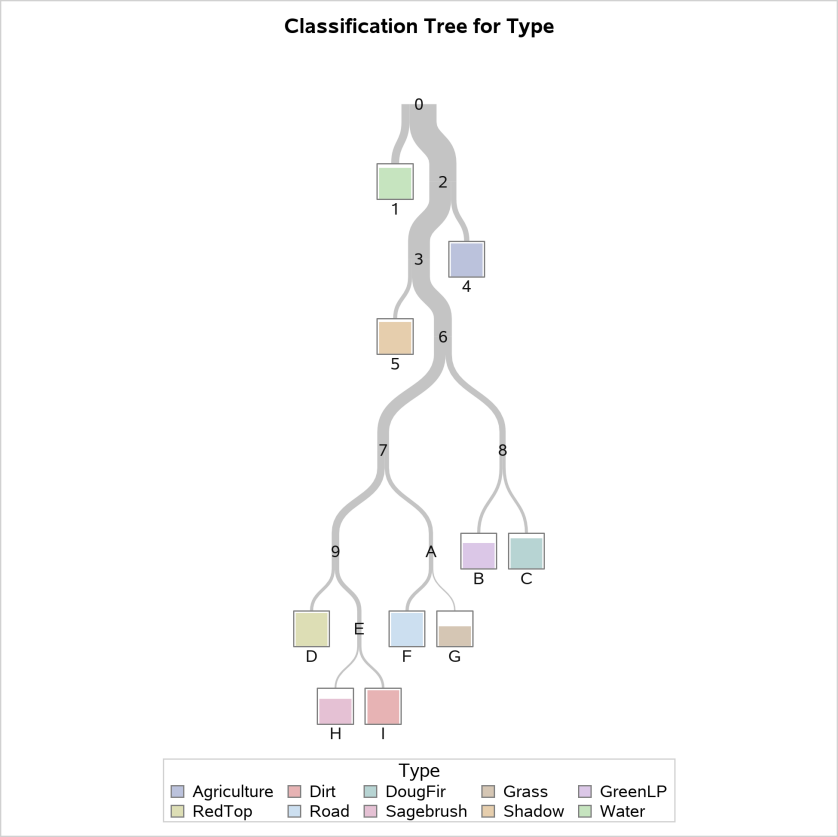
Often, you would apply the 1-SE rule (Breiman et al. 1984) when you are pruning via the cost-complexity method to potentially select a smaller tree that has only a slightly higher error rate than the minimum error. Selecting the smallest tree that has a misclassification rate below the horizontal reference line is in effect implementing the 1-SE rule. The subtree that has 10 leaves would be selected according to this rule, so you can run PROC HPSPLIT again as follows to override the subtree that was automatically selected in the first run:

```
proc hpsplit data=sampsio.snra plots=zoomedtree(node=7) seed=123 cvmodelfit
   intervalbins=500;
   class Type;
   grow gini;
   model Type = Blue Green Red NearInfrared NDVI Elevation
                SoilBrightness Greenness Yellowness NoneSuch;
   prune costcomplexity (leaves=10);
run;
```

This code includes specification of the LEAVES=10 option in the PRUNE statement to select this smaller subtree that performs almost equivalently to the subtree with 15 leaves from the earlier run. Specifying ZOOMEDTREE(NODE=7) in the PLOTS= option requests that the ODS graph ZoomedTreePlot displays the tree rooted at node 7 instead of at the root node. The CVMODELFIT option requests fit statistics for the final model by using cross validation as well as the cross validation confusion matrix.

Output 65.2.2 provides an overview of the final tree that has 10 leaves as requested.

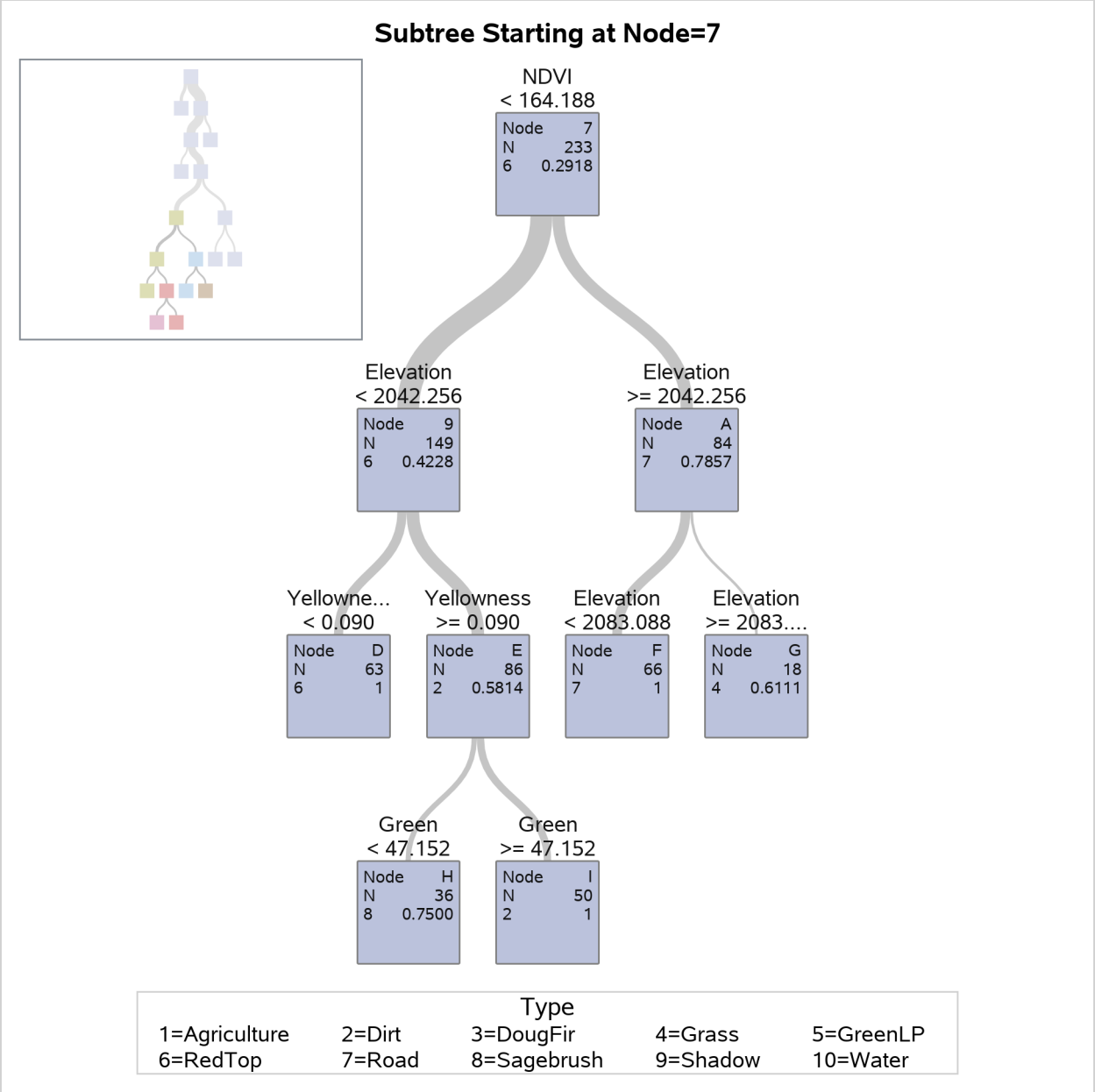
Output 65.2.2 Diagram of 10-Leaf Tree Selected Using 1-SE Rule



It turns out that there is exactly one leaf in the classification tree that corresponds to each of the 10 vegetation classes; this does not usually occur. The leaf color indicates the most frequently observed response among observations in that leaf, which is then the predicted response for all observations in that leaf. The height of the bars in the nodes represents the proportion of observations that have that particular response. For example, in node D, all observations have the value of `'RedTop'` for the response variable Type, whereas in node G, it appears that slightly over half of the observations have the value `'Grass'`.

Output 65.2.3 shows more details about a portion of the final tree, including splitting variables and values.

Output 65.2.3 Detailed Diagram of 10-Leaf Tree



As requested, the detailed tree diagram is displayed for the portion of the tree rooted at node 7 so that you can view the splits made at the bottom of the tree. You can see that several splits are made on the variable Elevation. The vegetation type most common in node G, whose observations have an elevation of at least 2083.0880, is `'Grass'`.

Confusion matrices are displayed in Output 65.2.4.

Output 65.2.4 Confusion Matrices for SNRA Data

The HPSPLIT Procedure

Confusion Matrices

		Predicted										
	Actual	Agriculture	Dirt	DougFir	Grass	GreenLP	RedTop	Road	Sagebrush	Shadow	Water	Error Rate
Model Based	Agriculture	105	0	0	0	1	0	0	0	0	0	0.0094
	Dirt	0	50	0	0	1	0	0	0	0	2	0.0566
	DougFir	0	0	55	0	11	0	0	0	0	0	0.1667
	Grass	1	0	0	11	1	0	0	2	0	0	0.2667
	GreenLP	0	0	5	0	49	0	0	1	0	0	0.1091
	RedTop	0	0	0	5	0	63	0	0	0	0	0.0735
	Road	0	0	0	0	0	0	66	0	0	2	0.0294
	Sagebrush	0	0	0	2	0	0	0	27	0	0	0.0690
	Shadow	0	0	0	0	0	0	0	2	76	6	0.0952
	Water	0	0	0	0	0	0	0	4	3	148	0.0452
Cross Validation	Agriculture	104	0	0	0	2	0	0	0	0	0	0.0189
	Dirt	0	49	0	0	1	0	0	1	0	2	0.0755
	DougFir	1	0	55	0	10	0	0	0	0	0	0.1667
	Grass	2	0	0	11	0	0	0	2	0	0	0.2667
	GreenLP	0	0	16	0	37	0	0	1	1	0	0.3273
	RedTop	0	1	0	4	1	62	0	0	0	0	0.0882
	Road	0	0	0	0	0	2	64	0	0	2	0.0588
	Sagebrush	0	1	0	2	0	0	0	26	0	0	0.1034
	Shadow	0	0	0	0	0	2	0	2	74	6	0.1190
	Water	0	0	0	0	0	3	0	1	3	148	0.0452

This table contains two matrices—one for the training data that uses the final tree and one that uses the cross validation folds—requested by the CVMODELFIT option in the PROC HPSPLIT statement. The values on the diagonal of each confusion matrix are the number of observations that are correctly classified for each of the 10 vegetation types. For the model-based matrix, you can see that the only nonzero value in the `'RedTop'` column is in the `'RedTop'` row. This is consistent with what is shown in Output 65.2.2, where the bar in node D with a predicted response of `'RedTop'` is the full height of the box representing the leaf, indicating that all observations on that leaf are correctly classified. You can also see from the matrix that the `'DougFir'` and `'GreenLP'` vegetation types are hard to distinguish; 11 of the 66 observations that have an actual response of `'DougFir'` are incorrectly assigned the response of `'GreenLP'`, corresponding to the 0.1667 error rate reported for `'DougFir'`.

Fit statistics are shown in Output 65.2.5.

Output 65.2.5 Fit Statistics for SNRA Data

Fit Statistics for Selected Tree

	N Leaves	ASE	Mis-class	Entropy	Gini	RSS
Model Based	10	0.0120	0.0701	0.3597	0.1197	83.6514
Cross Validation	10	0.0166	0.0974			

You can see from this table that the subtree with 10 leaves fits the training data very accurately, with 93% of the observations classified correctly. Because no validation data are present in this analysis, you get a better indication of how well the model fits and will generalize to new data by looking at the cross validation statistics, also requested by the CVMODELFIT option, that are included in the table. The misclassification rate that is averaged across the 10 folds is higher than the training misclassification rate for the final tree, suggesting that this model is slightly overfitting the training data.

Example 65.3: Creating a Regression Tree

This example performs an analysis similar to the one in the "Getting Started" section of Chapter 64, "The HPREG Procedure," where a linear regression model is fit. You can alternatively fit a regression tree to predict the salaries of Major League Baseball players based on their performance measures from the previous season by using almost identical code. Regression trees are piecewise constant models that, for relatively small data sets such as this, provide succinct summaries of how the predictors determine the predictions. These models are usually easier to interpret than linear regression models. The Sashelp.Baseball data set contains salary and performance information for Major League Baseball players (excluding pitchers) who played at least one game in both the 1986 and 1987 seasons (Time Inc. 1987). The following statements create a regression tree model:

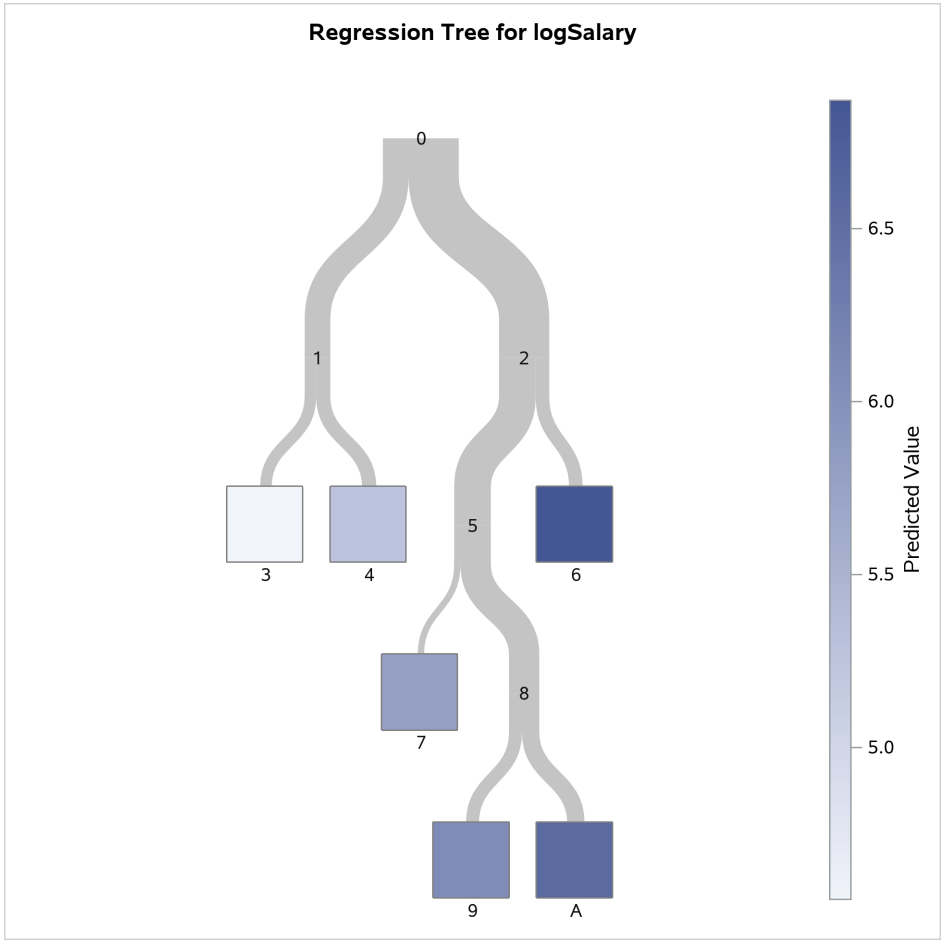
```
ods graphics on;

proc hpsplit data=sashelp.baseball seed=123;
   class league division;
   model logSalary = nAtBat nHits nHome nRuns nRBI nBB
                     yrMajor crAtBat crHits crHome crRuns crRbi
                     crBB league division nOuts nAssts nError;
   output out=hpsplout;
run;
```

By default, the tree is grown using the RSS criterion, and cost-complexity pruning with 10-fold cross validation is performed. The OUTPUT statement requests generation of the data set hpsplout, which contains the predicted salary from the tree model for each observation.

Much of the output for a regression tree is identical to the output for a classification tree. Tables and plots where there are differences are displayed and discussed on the following pages.

Output 65.3.1 displays the full regression tree.

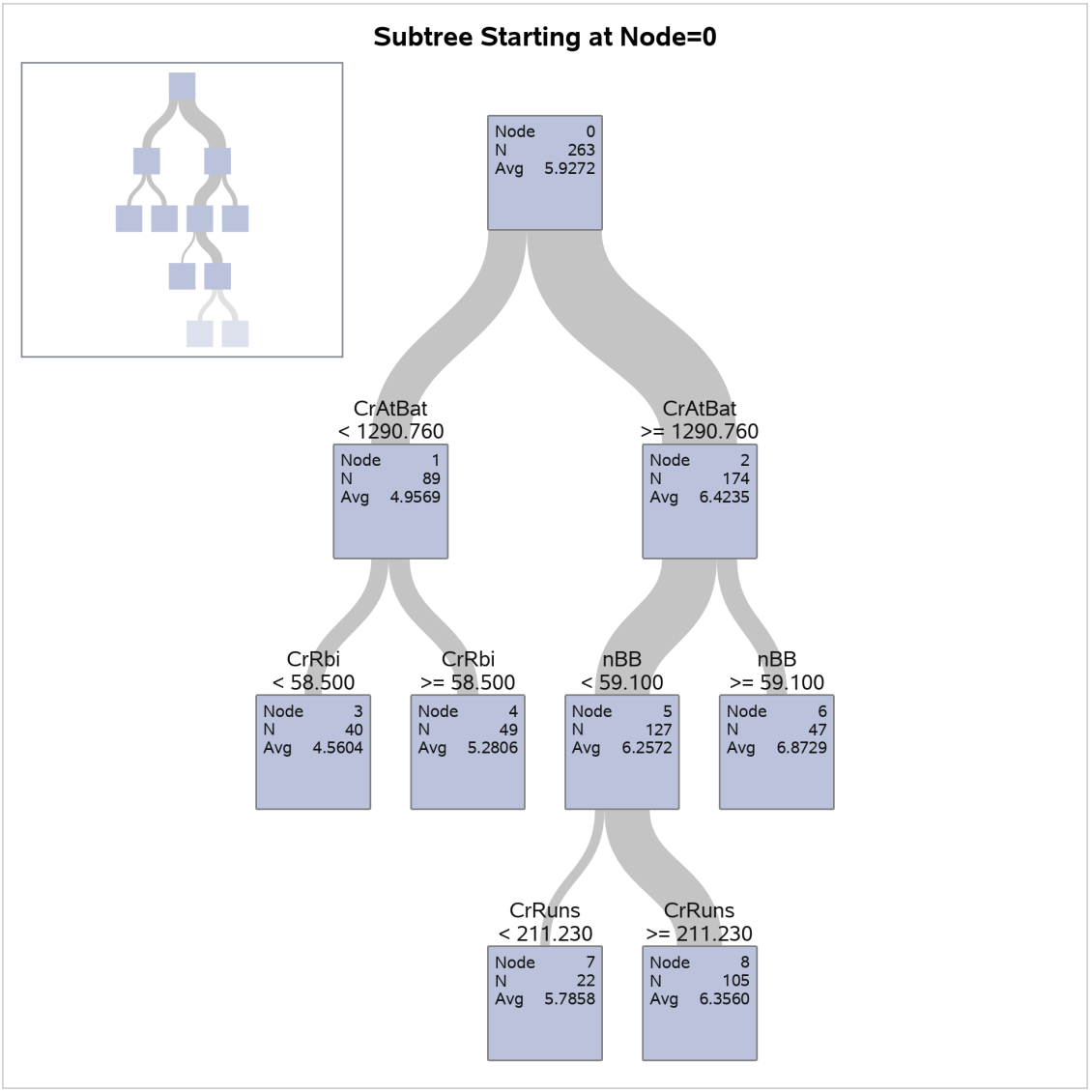
Output 65.3.1 Overview Diagram of Regression Tree



You can see from this diagram that the final selected tree has eight leaves. For a regression tree, the shade of the leaves represents the predicted response value, which is the average observed logSalary for the observations in that leaf. Node 3 has the lowest predicted response value, indicated by the lightest shade of blue, and Node A has the highest, indicated by the dark shade.

Output 65.3.2 shows details of the tree.

Output 65.3.2 Detailed Diagram of Regression Tree



As with a classification tree, you can see split variables and values for a portion of the tree in this view. You can also see the exact predicted response value, which is the average observed response, in each node.

The confusion matrix is omitted from the output when you are modeling a regression tree because it is relevant only for a categorical response.

Output 65.3.3 displays fit statistics for the final regression tree.

Output 65.3.3 Regression Tree Performance

The HPSPLIT Procedure

Model-Based Fit Statistics for Selected Tree		
N Leaves	ASE	RSS
6	0.1681	44.2142

Note that this table contains different statistics from those included for a classification tree. The ASE and RSS are reported here to help you assess the model fit. You could also use the CVMODELFIT option in the PROC HPSPLIT statement to obtain the cross validated fit statistics, as with a classification tree.

Output 65.3.4 shows the hpsplout data set that is created by using the OUTPUT statement and contains the first 10 observations of the predicted log-transformed salaries for each player in Sashelp.Baseball based on the regression tree model.

Output 65.3.4 Scored Predictor Data Set

Obs	logSalary	_Node_	_Leaf_	P_logSalary
1	.	3	0	4.56042
2	6.16331	9	4	6.08107
3	6.17379	6	2	6.87289
4	6.21461	10	5	6.56218
5	4.51634	3	0	4.56042
6	6.62007	10	5	6.56218
7	4.24850	3	0	4.56042
8	4.60517	3	0	4.56042
9	4.31749	3	0	4.56042
10	7.00307	6	2	6.87289

The variable P_logSalary contains the predicted salaries on the log scale. Note that all observations in the same leaf have the same predicted response. The OUT= data set can contain additional variables from the DATA= data set if you specify them in the ID statement.

Example 65.4: Creating a Binary Classification Tree with Validation Data

A common use of classification trees is to predict the likelihood that a mortgage applicant will default on a loan. The data set Hmeq, which is in the Sampsio library that SAS provides, contains observations for 5,960 mortgage applicants. A variable named Bad indicates whether the applicant paid or defaulted on the loan that was given.

This example uses Hmeq to build a tree model that is used to score the data and can be used to score data about new applicants. Table 65.6 describes the variables in Hmeq.

Table 65.6 Variables in the Home Equity (Hmeq) Data Set

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

The response variable for the tree model is Bad, a CLASS variable that has two values (0 indicates payment of loan, and 1 indicates default). The other variables are predictor variables for the model. The following statements display the first 10 observations of the data set:

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

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

Output 65.4.1 shows the partial listing of Hmeq.

Output 65.4.1 Partial Listing of the Hmeq Data

Obs	Bad	Loan	MortDue	Value	Reason	Job	YoJ	Derog	Delinq	CLAge	nInq	CLNo	DebtInc
1	1	1100	25860	39025	HomeImp	Other	10.5	0	0	94.367	1	9	.
2	1	1300	70053	68400	HomeImp	Other	7.0	0	2	121.833	0	14	.
3	1	1500	13500	16700	HomeImp	Other	4.0	0	0	149.467	1	10	.
4	1	1500	.	.			.	.	.	.	.	.	.
5	0	1700	97800	112000	HomeImp	Office	3.0	0	0	93.333	0	14	.
6	1	1700	30548	40320	HomeImp	Other	9.0	0	0	101.466	1	8	37.1136
7	1	1800	48649	57037	HomeImp	Other	5.0	3	2	77.100	1	17	.
8	1	1800	28502	43034	HomeImp	Other	11.0	0	0	88.766	0	8	36.8849
9	1	2000	32700	46740	HomeImp	Other	3.0	0	2	216.933	1	12	.
10	1	2000	.	62250	HomeImp	Sales	16.0	0	0	115.800	0	13	.

The following statements use the HPSPLIT procedure to create a classification tree:

```
ods graphics on;

proc hpsplit data=hmeq maxdepth=5;
   class Bad Delinq Derog Job nInq Reason;
   model Bad(event='1') = Delinq Derog Job nInq Reason CLAge CLNo
               DebtInc Loan MortDue Value YoJ;
   prune costcomplexity;
   partition fraction(validate=0.3 seed=123);
   code file='hpsplexc.sas';
   rules file='rules.txt';
run;
```

The MAXDEPTH= option specifies the maximum depth of the tree to be grown.

Specifying Bad to the left of the equal sign in the MODEL statement indicates that it is the response variable. Because Bad is a binary response, the EVENT= option specifies that the calculations for sensitivity, specificity, AUC, and ROC be based on the level Bad=1.

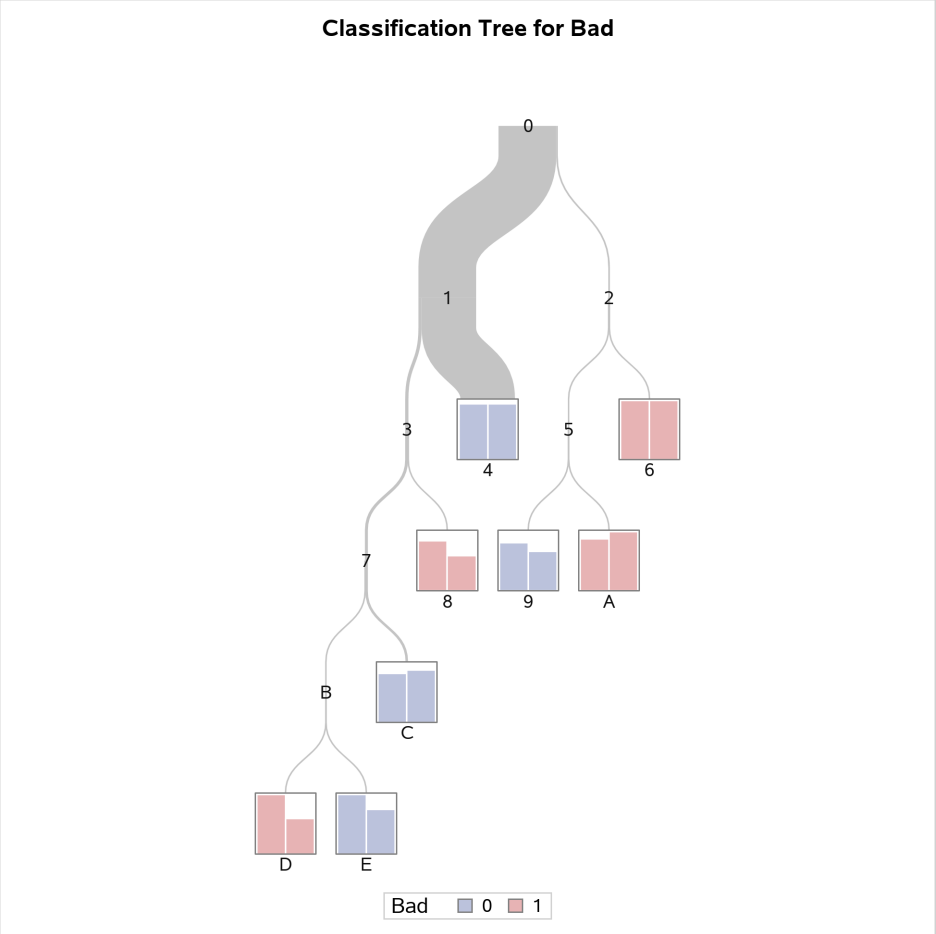
By default, the entropy metric is used to grow the tree. The PRUNE statement requests cost-complexity pruning.

The PARTITION statement specifies that the observations in Hmeq be logically partitioned into disjoint subsets for model training and validation. Observations are randomly selected for the validation subset with the probability 0.3; the remaining observations are selected for the training subset.

The FILE= option in the CODE statement requests that SAS DATA step score code be saved to a file named *hpsplexc.sas*, and the FILE= option in the RULES statement requests that the node rules be saved to a file named *rules.txt*.

The tree diagram in Output 65.4.2 provides an overview of the full tree.

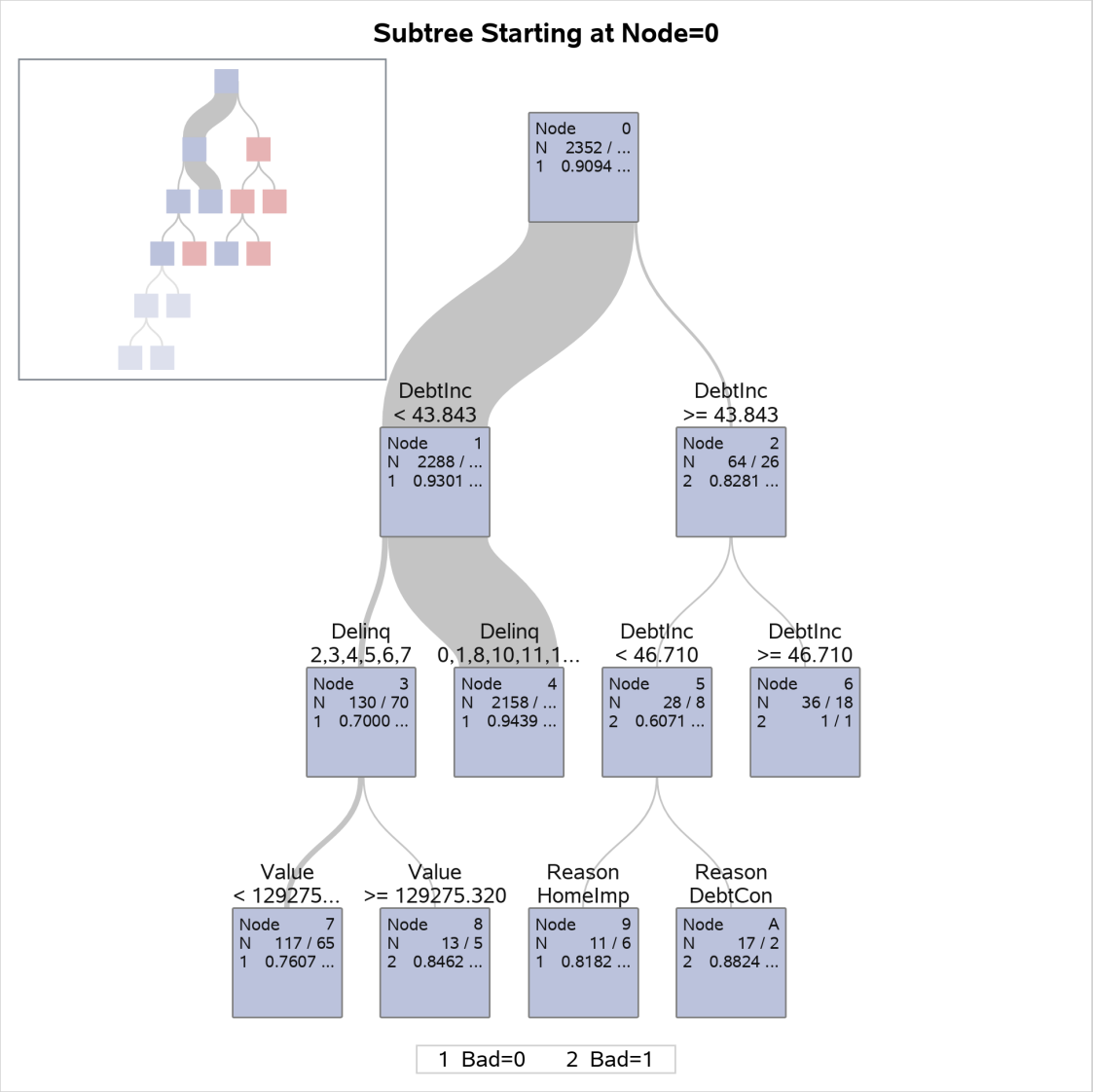
Output 65.4.2 Overview Diagram of Final Tree



You can see from this diagram that the observations in terminal nodes 4, 9, C, and E are assigned a prediction of Bad=0 and those in terminal nodes 6, 8, A, and D are assigned a prediction of Bad=1. You can easily see that node 4 contains the most observations, as indicated by thickness of the link from its parent node.

The tree diagram in Output 65.4.3 is a detailed view of the top portion of the tree. You can use the PLOTS= option in the PROC HPSPLIT statement to control which nodes are displayed.

Output 65.4.3 Detailed Tree Diagram



By default, this view provides detailed splitting information about the first three levels of the tree, including the splitting variable and splitting values. The splitting rule above each node determines which observations from the parent node are included in the node. The first row of the table inside the node provides the node identifier. The second row of the table provides the number of training and validation observations, separated by a slash. The third row shows the predicted response for observations in that node if classification was occurring at that point, along with the proportion of training and validation observations with that observed response. Note that the legend shows what actual value of the response variable is represented by the value shown in the node. For example, in node 6, all 36 observations in the training data and all 18 observations in the validation data have an observed response value of Bad=1, as indicated by the value 2 shown on the third line.

Because the response is categorical, the confusion matrices for the training data and for the validation data are displayed in the table shown in Output 65.4.4.

Output 65.4.4 Training and Validation Confusion Matrices

The HPSPLIT Procedure

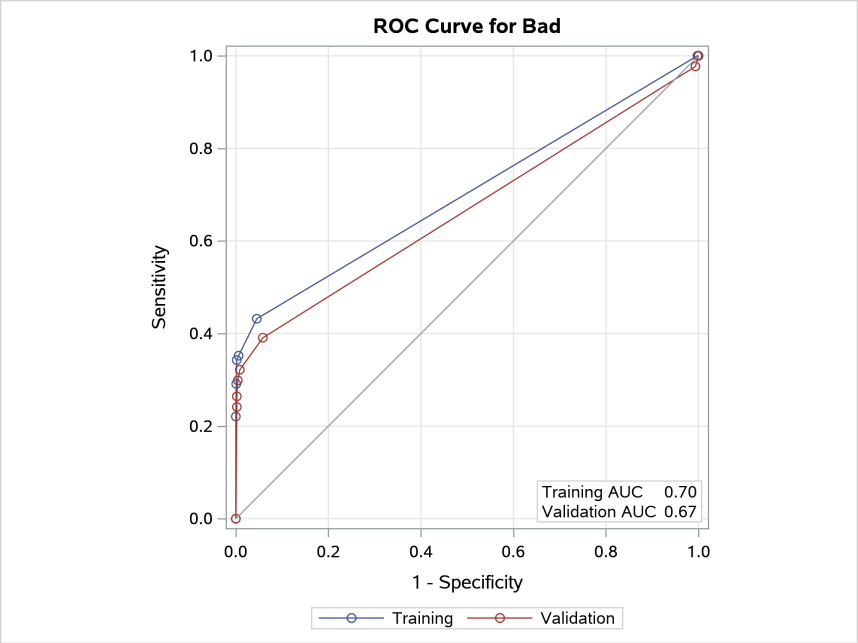
Confusion Matrices

		Predicted		
	Actual	**0**	**1**	**Error Rate**
Training	**0**	2135	4	0.0019
	1	140	73	0.6573
Validation	**0**	921	4	0.0043
	1	61	26	0.7011

For the training data, there are 73 observations where the model correctly predicted Bad=1 and 2,135 observations where the model correctly predicted Bad=0. The values in the off-diagonal entries of the matrices show how many times the model misclassified observations.

When you are modeling a binary response variable, a plot of the ROC curve is displayed as shown in Output 65.4.5.

Output 65.4.5 ROC Plot



This plot summarizes the performance of the tree model in terms of sensitivity and specificity.

Output 65.4.6 displays the "Tree Performance" table, which provides several impurity measures and fit statistics for the final tree.

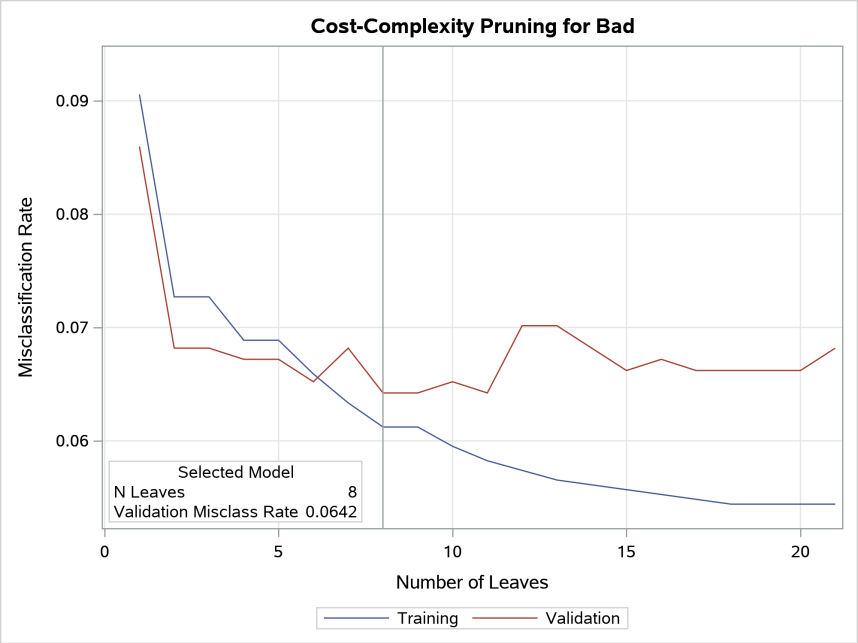
Output 65.4.6 Tree Performance

Fit Statistics for Selected Tree									
	N Leaves	ASE	Mis-class	Sensitivity	Specificity	Entropy	Gini	RSS	AUC
Training	8	0.0567	0.0612	0.3427	0.9981	0.3244	0.1135	266.9	0.7016
Validation	8	0.0599	0.0642	0.2989	0.9957	0.3286	0.1160	121.2	0.6659

The last three columns of this table contain statistics related to the ROC curve plotted in Output 65.4.5. This model does not classify the observations with Bad=1 (the event of interest) very well, as you can see in the confusion matrix and from the low sensitivity reported in this table. This is often the case with a relatively rare event. The AUC measures the area under the ROC curve. A model that fits the data perfectly would have an AUC of 1. These three columns are included only for a binary response variable.

Output 65.4.7 displays the pruning plot.

Output 65.4.7 Pruning Plot



This plot displays misclassification rates for the training and validation data as the tree is pruned. The tree with eight leaves is selected as the final tree because it has the lowest misclassification rate for the validation data.

Creating Score Code and Scoring New Data

In addition to seeing information about the tree model, you might be interested in applying a model to predict the response variable in other data sets where the response is unknown. The following statements show how you can use the score code file *hpsplexc.sas*, created by the FILE= option in the CODE statement, to score the data in Hmeq and save the results in a SAS data set named Scored.

```
data scored;
   set hmeq;
   %include 'hpsplexc.sas';
run;
```

Output 65.4.8 shows a partial listing of Scored.

Output 65.4.8 Partial Listing of the Scored Hmeq Data

Obs	Bad	Loan	MortDue	Value	Reason	Job	YoJ	Derog	Delinq	CLAge	nInq	CLNo	DebtInc
1	1	1100	25860	39025	HomeImp	Other	10.5	0	0	94.367	1	9	.
2	1	1300	70053	68400	HomeImp	Other	7.0	0	2	121.833	0	14	.
3	1	1500	13500	16700	HomeImp	Other	4.0	0	0	149.467	1	10	.
4	1	1500	.	.			.	.	.	.	.	.	.
5	0	1700	97800	112000	HomeImp	Office	3.0	0	0	93.333	0	14	.
6	1	1700	30548	40320	HomeImp	Other	9.0	0	0	101.466	1	8	37.1136
7	1	1800	48649	57037	HomeImp	Other	5.0	3	2	77.100	1	17	.
8	1	1800	28502	43034	HomeImp	Other	11.0	0	0	88.766	0	8	36.8849
9	1	2000	32700	46740	HomeImp	Other	3.0	0	2	216.933	1	12	.
10	1	2000	.	62250	HomeImp	Sales	16.0	0	0	115.800	0	13	.

Obs	_Node_	_Leaf_	_WARN_	P_Bad0	P_Bad1	V_Bad0	V_Bad1
1	4	0		0.94393	0.05607	0.94432	0.05568
2	12	5		0.83168	0.16832	0.88462	0.11538
3	4	0		0.94393	0.05607	0.94432	0.05568
4	4	0		0.94393	0.05607	0.94432	0.05568
5	4	0		0.94393	0.05607	0.94432	0.05568
6	4	0		0.94393	0.05607	0.94432	0.05568
7	12	5		0.83168	0.16832	0.88462	0.11538
8	4	0		0.94393	0.05607	0.94432	0.05568
9	12	5		0.83168	0.16832	0.88462	0.11538
10	4	0		0.94393	0.05607	0.94432	0.05568

The data set contains the 13 original variables and the 7 new variables that are created by the score code. The variable P_Bad1 is the proportion of training observations on this leaf for which Bad=1, and this variable can be interpreted as the probability of default. The variable V_Bad1 is the proportion of validation observations on this leaf for which Bad=1. Similar variables are created for the other response value, Bad=0. Also included in the scored data set are the node and leaf assignments, which are shown in the variables _Node_ and _Leaf_, respectively. Note that these same variables are included in the OUT= data set when you specify that option in the OUTPUT statement. For more information about these variables, see the section "Scoring" on page 5042.

You can use the preceding statements to score new data by including the new data set in place of Hmeq in the SET statement. The new data set must contain the same variables as the data that are used to build the tree model, but not the unknown response variable that you are now trying to predict.

Creating a Node Rules Description of a Tree

When you specify the FILE= option in the RULES statement, a file that contains the node rules is created. In this example, PROC HPSPLIT saves the node rules of the tree model to a file named *rules.txt*; a partial listing of this file is shown in Output 65.4.9.

Output 65.4.9 Rules File

```
*------------------------------------------------------------*
NODE = 4
*------------------------------------------------------------*
MISSING(Delinq) OR (Delinq IS ONE OF 0, 1, 8, 10, 11, 12, 13, 15)
AND MISSING(DebtInc) OR (DebtInc < 43.843383)
    PREDICTED VALUE IS 0
    PREDICTED 0 = 0.9439( 2037/2158)
    PREDICTED 1 = 0.05607( 121/2158)
*------------------------------------------------------------*
NODE = 13
*------------------------------------------------------------*
MISSING(CLNo) OR (CLNo < 30.08)
AND (Delinq IS ONE OF 4, 5, 6, 7)
AND MISSING(Value) OR (Value < 129275.32)
AND (Delinq IS ONE OF 2, 3, 4, 5, 6, 7)
AND MISSING(DebtInc) OR (DebtInc < 43.843383)
    PREDICTED VALUE IS 1
    PREDICTED 0 = 0( 0/11)
    PREDICTED 1 = 1( 11/11)
*------------------------------------------------------------*
NODE = 14
*------------------------------------------------------------*
(CLNo >= 30.08)
AND (Delinq IS ONE OF 4, 5, 6, 7)
AND MISSING(Value) OR (Value < 129275.32)
AND (Delinq IS ONE OF 2, 3, 4, 5, 6, 7)
AND MISSING(DebtInc) OR (DebtInc < 43.843383)
    PREDICTED VALUE IS 0
    PREDICTED 0 = 1( 5/5)
    PREDICTED 1 = 0( 0/5)
*------------------------------------------------------------*
NODE = 9
*------------------------------------------------------------*
(Reason IS HomeImp)
AND (DebtInc < 46.7104)
AND (DebtInc >= 43.843383)
    PREDICTED VALUE IS 0
    PREDICTED 0 = 0.8182( 9/11)
    PREDICTED 1 = 0.1818( 2/11)
```

In this listing, the predicted value and the fraction of observations for each level of the response variable are displayed for each terminal node. The nodes are not numbered consecutively because only terminal nodes (leaves) are included. The splits that lead to each leaf are shown above the predicted value and fractions.

Example 65.5: Assessing Variable Importance

This example creates a classification tree model to determine important variables (parameters) during the manufacture of a semiconductor device. Some of the variables that are involved in the manufacturing process are as follows: gTemp is the growth temperature of substrate, aTemp is the anneal temperature, Rot is rotation speed, Dopant is the atom that is used during device growth, and Usable indicates whether the device is usable.

The following statements create a data set named MBE_Data, which contains measurements for 20 devices:

```
data MBE_Data;
   label gTemp  = 'Growth Temperature of Substrate';
   label aTemp  = 'Anneal Temperature';
   label Rot    = 'Rotation Speed';
   label Dopant = 'Dopant Atom';
   label Usable = 'Experiment Could Be Performed';

   input gTemp aTemp Rot Dopant $ 39-40 Usable $ 47-54;
   datalines;
   384.614    633.172    1.01933      C       Unusable
   363.874    512.942    0.72057      C       Unusable
   397.395    671.179    0.90419      C       Unusable
   389.962    653.940    1.01417      C       Unusable
   387.763    612.545    1.00417      C       Unusable
   394.206    617.021    1.07188      Si      Usable
   387.135    616.035    0.94740      Si      Usable
   428.783    745.345    0.99087      Si      Unusable
   399.365    600.932    1.23307      Si      Unusable
   455.502    648.821    1.01703      Si      Unusable
   387.362    697.589    1.01623      Ge      Usable
   408.872    640.406    0.94543      Ge      Usable
   407.734    628.196    1.05137      Ge      Usable
   417.343    612.328    1.03960      Ge      Usable
   482.539    669.392    0.84249      Ge      Unusable
   367.116    564.246    0.99642      Sn      Unusable
   398.594    733.839    1.08744      Sn      Unusable
   378.032    619.561    1.06137      Sn      Usable
   357.544    606.871    0.85205      Sn      Unusable
   384.578    635.858    1.12215      Sn      Unusable
   ;
```

The following statements create the tree model:

```
proc hpsplit data=MBE_Data maxdepth=6;
   class Usable Dopant;
   model Usable = gTemp aTemp Rot Dopant;
   prune none;
run;
```

Output 65.5.1 shows the "Variable Importance" table.

Output 65.5.1 Variable Importance

The HPSPLIT Procedure

Variable Importance

		Training		
Variable	**Variable Label**	**Relative**	**Importance**	**Count**
gTemp	Growth Temperature of Substrate	1.0000	2.1022	2
Dopant	Dopant Atom	0.7522	1.5811	1
aTemp	Anneal Temperature	0.6228	1.3093	1
Rot	Rotation Speed	0.3250	0.6831	1

This table shows that the predictor gTemp has the largest value. This means that the growth temperature of substrate is the most important consideration in determining the usability of the sample.

References

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

Cutler, R., Brown, L., Powell, J., Bentz, B., and Cutler, A. (2003). "Identifying 'Redtops': Classification of Satellite Imagery for Tracking Mountain Pine Beetle Progression through a Pine Forest." In *Security and Infrastructure Protection: Proceedings of the 35th Symposium on the Interface*, edited by A. Braverman, T. Hesterberg, M. Minnotte, J. Symanzik, and Y. Said. Fairfax Station, VA: Interface Foundation of North America Inc.

Edwards, T. C., Jr., Cutler, D. R., Zimmermann, N. E., Geiser, L., and Alegria, J. (2005). "Model-Based Stratifications for Enhancing the Detection of Rare Ecological Events." *Ecology* 86:1081–1090.

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

Geiser, L. H., and Neitlich, P. N. (2007). "Air Pollution and Climate Gradients in Western Oregon and Washington Indicated by Epiphytic Macrolichens." *Environmental Pollution* 145:203–218.

Hastie, T. J., Tibshirani, R. J., and Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. New York: Springer-Verlag.

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.

Kuhn, M., and Johnson, K. (2013). *Applied Predictive Modeling*. New York: Springer.

Lichman, M. (2013). "UCI Machine Learning Repository." School of Information and Computer Sciences, University of California, Irvine. http://archive.ics.uci.edu/ml.

Molina, R., McKenzie, D., Lesher, R., Ford, J., Alegria, J., and Cutler, R. (2003). *Strategic Survey Framework for the Northwest Forest Plan Survey and Manage Program*. General Technical Report PNW-GTR-573, US Department of Agriculture, Forest Service, Pacific Northwest Research Station.

Old-Growth Definition Task Group (1986). *Interim Definitions for Old-Growth Douglas-Fir and Mixed-Conifer Forests in the Pacific Northwest and California*. Research Note PNW-447, US Department of Agriculture, Forest Service, Pacific Northwest Research Station.

Quinlan, J. R. (1986). "Induction of Decision Trees." *Machine Learning* 1:81–106.

Quinlan, J. R. (1987). "Simplifying Decision Trees." *International Journal of Man-Machine Studies* 27:221–234.

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

Time Inc. (1987). "What They Make." *Sports Illustrated* (April 20): 54–81.

Zhang, H., and Singer, B. H. (2010). *Recursive Partitioning and Applications*. 2nd ed. New York: Springer.

Subject Index

Syntax Index