



THE
POWER
TO KNOW.

**SAS[®] Enterprise Miner[™] and
SAS[®] Text Miner Procedures
Reference for SAS[®] 9.1.3
Recursive Partitioning
Procedures
(Book Excerpt)**

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., SAS® *Enterprise Miner™* and SAS® *Text Miner Procedures: Reference for SAS® 9.1.3*, Cary, NC: SAS Institute Inc.

SAS® Enterprise Miner™ and SAS® Text Miner Procedures: Reference for SAS 9.1.3

Copyright © 2008 by SAS Institute Inc., Cary, NC, USA.

All rights reserved. Produced in the United States of America.

For a Web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

U.S. Government Restricted Rights Notice. Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19 Commercial Computer Software-Restricted Rights (June 1987).

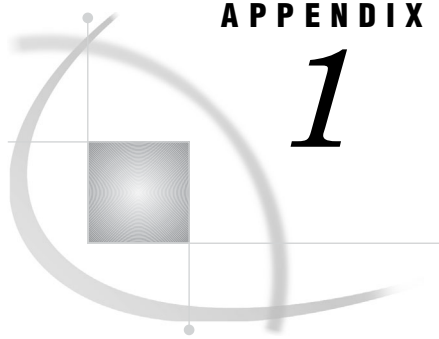
SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, October 2008

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

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.



APPENDIX

1

Recursive Partitioning Procedures

<i>Overview: Recursive Partitioning Procedures</i>	472
<i>Terminology</i>	472
<i>Running a Partitioning Procedure</i>	473
<i>Syntax: Recursive Partitioning Procedures</i>	474
<i>Partitioning Procedure Statement</i>	475
<i>ASSESS Statement</i>	481
<i>CODE Statement</i>	482
<i>DECISION Statement</i>	483
<i>FREQ Statement</i>	485
<i>IMPORTANCE Statement</i>	485
<i>INPUT Statement</i>	487
<i>MAKEMACRO Statement</i>	488
<i>PARTIALDEP Statement</i>	488
<i>PERFORMANCE Statement</i>	489
<i>SAVE Statement</i>	490
<i>SCORE Statement</i>	491
<i>TARGET Statement</i>	491
<i>Details: Recursive Partitioning Procedures</i>	492
<i>Form of a Splitting Rule</i>	492
<i>Posterior and Within Node Probabilities</i>	492
<i>Incorporating Prior Probabilities</i>	493
<i>Incorporating Decisions, Profit, and Loss</i>	493
<i>Splitting Criteria</i>	494
<i>Reduction in Node Impurity</i>	494
<i>Statistical Tests and p-Values</i>	495
<i>Distributional Assumptions</i>	496
<i>Multiple Testing Assumptions</i>	497
<i>Adjusting p-Values for Multiple Tests</i>	498
<i>Adjusting p-Values for the Number of Input Values and Branches</i>	498
<i>Adjusting p-Values for the Depth of the Node</i>	499
<i>Adjusting p-Values for the Number of Input Variables</i>	499
<i>Splitting Criteria for an Ordinal Target</i>	500
<i>Missing Values</i>	500
<i>Unseen Categorical Values</i>	501
<i>Within Node Training Sample</i>	502
<i>Split Search Algorithm</i>	502
<i>Surrogate Splitting Rules</i>	504
<i>Similarity and Dissimilarity of Pairs of Inputs</i>	504
<i>Variable Importance</i>	505
<i>Split-Based Variable Importance</i>	505
<i>SAVE Statement IMPORTANCE= Output Data Set</i>	507

<i>Observation-Based Variable Importance</i>	507
<i>Partial Dependency Functions</i>	509
<i>Formulaic Definition and Additivity</i>	509
<i>Interaction Detection</i>	510
<i>Algorithmic Definition</i>	510
<i>NODESTATS= Output Data Set</i>	511
<i>PATH= Output Data Set</i>	512
<i>RULES= Output Data Set</i>	513
<i>Interval Split</i>	514
<i>Nominal Split</i>	514
<i>Ordinal Split</i>	514
<i>SCORE Statement OUT= Output Data Set</i>	515
<i>Variable Names and Conditions for Their Creation</i>	515
<i>Decision Variables</i>	516
<i>Leaf Assignment Variables</i>	517
<i>SCORE Statement OUTFIT= Output Data Set</i>	517
<i>Performance Considerations</i>	518
<i>Passes Over the Data</i>	518
<i>Memory Considerations</i>	519
<i>References</i>	519

Overview: Recursive Partitioning Procedures

A *partitioning* procedure is one that searches for an optimal partition of the data defined in terms of the values of a single variable. The optimality criterion depends on how another variable, the *target*, is distributed into the partition segments. The more similar the target values are within the segments, the greater the worth of the partition.

Most partitioning procedures further partition each segment in a process called *recursive partitioning*. The partitions are then combined to create a predictive model. The model is evaluated by goodness-of-fit statistics defined in terms of the target variable. These statistics are different from the measure of worth of an individual partition. A good model might result from many mediocre partitions.

The following Enterprise Miner procedures use partitioning:

- ARBOR — Train a decision tree.
- TREEBOOST — Train a tree boosting model.

All the procedures are *predictive* in that they *train* a model or a grouping for the purpose of applying it to data that is unavailable during training and for which the target is absent and the model or grouping cannot be evaluated when it is applied. If the prediction on the new data turns out well, the model is said to *generalize* well. Good generalization is the primary goal of a predictive task. A model might fit the training data well but generalize poorly.

The partitioning procedures have many statements and options in common. These are described in this chapter.

Terminology

Variable Roles and Measurement Levels

A partitioning procedure defines a partition in terms of values of a single variable selected from a set of available variables called *input* variables. A rule assigning the

variable values to branches is called a *splitting rule*. The partitioning procedures can search for a splitting rule that maximizes an association between a target variable and the partition segments. The association is defined by a *splitting criterion* or *grouping criterion* that defines a measure of worth of the rule.

The partitioning procedures accept variables with *nominal*, *ordinal*, and *interval measurement levels*. A *nominal* variable is a numeric or character categorical variable in which the categories are unordered. An *ordinal* variable is a numeric or character categorical variable in which the categories are ordered. An *interval* variable is a numeric variable for which differences of values are informative.

The partitioning procedures use *normalized, formatted* values of categorical variables, and consider two categorical values the same if the normalized values are the same. Normalization removes any leading blank spaces from a value, converts lowercase characters to uppercase, and truncates the value to 32 characters. The `FORMAT` procedure in the SAS Procedures Guide explains how to define a format. A `FORMAT` statement in the current run of a procedure or in the `DATA` step that created the training data associates a format with a variable. By default, numeric variables use the `BEST12` format, and the formatted values of character variables are the same as the unformatted ones.

Data, Prior, and Posterior Probabilities

The data available to a procedure can be divided into sets with different roles. A procedure uses *training data* to find the partitions and construct a family of models. *Validation data* is used to make unbiased estimates and select one model from the family, and *test data* is used to evaluate the result. This division is prudent because the procedure tends to *overfit*, to fit the model to spurious features of the data at hand that will not appear in the data to which the model will be applied later, (the *score data*).

For a categorical target, the proportions of the target values influence the model strongly. If the proportions in the training data differ noticeably from those in the scored data, then the model will generalize poorly. To correct this, the partitioning procedures accept *prior probabilities* that specify what proportions to expect in the score data. The *posterior probability* of a target category for a particular observation is the probability that the observation has that target value, according to the model. A partitioning procedure incorporates prior probabilities when computing posterior probabilities. The procedures also incorporate the prior probabilities in the search for partitions if (and only if) the user requests it.

Recursive Partitioning

Recursive partitioning partitions the data into subsets and then partitions each of the subsets, and so on. In the terminology of the tree metaphor, the subsets are *nodes*, the original data set is the *root node*, and the final, unpartitioned subsets are *terminal nodes* or *leaves*. Nodes that are not terminal nodes are sometimes called *internal nodes*. The subsets of a single partition are commonly called *child nodes*. This mixes the metaphor with genealogy, which also provides the terms *descendant* and *ancestor nodes*. A *branch* of a node consists of a child node and its descendants.

Running a Partitioning Procedure

Each partitioning procedure runs in two or three phases:

- initial declarations
- interactive training (optional)
- model assessment and output

The initial declarations specify the training data, variable roles, and other options that can be set only once. The interactive training statements are optional and only available with the ARBOR procedure. They enable complete control of the creation and modification of splitting rules and of the deletion of nodes in the tree. The model assessment and output statements evaluate and select submodels, apply the predictions to new data, and save model estimates in SAS data sets.

The partitioning procedures execute statements as soon as they are submitted. All interactive training, model assessment, and output statements can be repeated.

The initial declarations must appear first. If interactive training is intended, those statements would typically come next, followed by assessment and output statements. Interactive training statements can follow model assessment and output statements, which in turn can be repeated after interactive training.

Interactive training begins with an INTERACT statement. In the ARBOR procedure, the INTERACT statement specifies which subtree to begin with, and, implicitly, which nodes to permanently delete. Interactive training ends with any model assessment or output statement. If no interactive training statements appear before the first assessment or output statement, and no nontrivial tree is imported using the INMODEL= option in the PROC statement, the procedure will automatically create a model or grouping.

A QUIT statement terminates the procedure. A RUN statement clears error conditions.

Syntax: Recursive Partitioning Procedures

```
PROC name< option(s)>;
  DECISION DECDDATA= SAS-data-set<option(s)> ;
  FREQ variable;
  INPUT variable(s)< /option(s)>;
  TARGET variable< /option(s)>;
  PERFORMANCE <option(s)>;
  ASSESS <option(s)>;
  CODE <option(s)>;
  IMPORTANCE <option(s)>;
  MAKEMACRO keyword= macname;
  PARTIALDEP <option(s)>;
  SAVE <option(s)>;
  SCORE <option(s)>;
```

Statements, statement options, and possible values might depend on the procedure and are listed in the chapter describing the procedure. Most statements are shared by two or more procedures and are described here.

The following table summarizes the function of each statement (other than the PROC statement) in the procedure:

Table A1.1 Statements in a Recursive Partitioning Procedure

Statement	Description
DECISION	Specify profits and prior probabilities
FREQ	Specify a frequency variable

Statement	Description
INPUT	Specify input variables with common options
TARGET	Specify the target variable
PERFORMANCE	Specify memory size and where to locate data
ASSESS	Evaluate models
CODE	Generate SAS DATA step code for scoring new cases
IMPORTANCE	Modify variable values to infer its importance
MAKEMACRO	Define a macro variable
PARTIALDEP	Output partial dependency for plots and interaction detection
SAVE	Output data sets containing model results
SCORE	Use the model to make predictions on new data

The rest of this section gives detailed syntax information for each of these statements, beginning with the PROC statement. The remaining statements are covered in alphabetical order.

Partitioning Procedure Statement

PROC *partitioning_proc_name* <*option(s)*>;

where *partitioning_proc_name*= ARBOR | TREEBOOST

The PROC statement starts the procedure. Either the DATA= option or the INMODEL= option must appear. The DATA= option must appear to begin or resume training a model (or a grouping). The INMODEL= option specifies a previously saved model (or grouping).

ALPHA=*p*

specifies a threshold *p*-value for the significance level of a candidate splitting rule, that is applicable for splitting criteria that depend on *p*-values, namely, CRITERION=PROBF and PROBCHISQ. For splitting criteria not based on *p*-values, a recursive partitioning procedure uses the value associated with the MINWORTH= option instead of *p*.

Default: 0.20

CATEGORICALBINS=*n*

specifies the number of preliminary bins to collect categorical input values when preparing to search for a split. If an input variable has more than *n* categories in the node, then the split search uses the most frequent *n* – 1 categories, and regards the remaining categories as a single pseudo category. The count of categories is done separately in each node.

Default: The default value of *n* is 15 times the maximum value specified in the MAXBRANCHES= option in the PROC statement and INPUT statements. If the MAXBRANCHES= option is unspecified, the default value of the CATEGORICALBINS= option is 30.

CRITERION=*name*

specifies the criterion for evaluating candidate splitting rules. The criteria available for each level of measurement of the target variable are as follows:

Table A1.2 Split Search Criteria

Criterion	Measure of Split Worth
Criteria for Interval Targets	
VARIANCE	reduction in square error from node means
PROBF	<i>p</i> -value of F test associated with node variances (default)
Criteria for Nominal Targets	
ENTROPY	Reduction in entropy
GINI	Reduction in Gini index
PROBCHISQ	<i>p</i> -value of Pearson <i>chi</i> -square for target versus branches (default)
Criteria for Ordinal Targets	
ENTROPY	Reduction in entropy, adjusted with ordinal distances
GINI	Reduction in Gini index, adjusted with ordinal distances (default)

Default: The default criterion depends on the procedure. For decision trees, the default criterion is PROBF for an interval target, PROBCHISQ for a nominal target, and GINI for an ordinal target. See “Splitting Criteria” on page 494 for more information.

DATA=*SAS-data-set*

specifies the training data set. If the INMODEL= option is specified to input a saved model, the DATA= option causes the partitioning procedure to recompute all the node statistics and predictions in the saved model.

DECSEARCH

specifies that the split search should incorporate the profit or loss function specified in the DECISION statement. See “Incorporating Decisions, Profit, and Loss” on page 493 for more information. The DECSEARCH option only works with a categorical target.

EVENT=*category*

specifies a formatted value of a categorical target to use with some output statistics, such as the LIFT assessment measure. If the EVENT= option is never specified and is needed, the least frequent target value in the training data is used. The EVENT option is ignored with an interval target.

EXHAUSTIVE=*n*

specifies the maximum allowable splits in a complete enumeration of all possible splits. The *exhaustive* method of searching for a split examines all possible splits. If the number of possible splits is greater than *n*, then a *heuristic* search is done instead of an exhaustive search. The exhaustive and heuristic search methods only apply to multiway splits, and to binary splits on nominal targets with more than two values. See “Split Search Algorithm” on page 502 for more information.

Default: 5000

INMODEL=SAS-data-set

names a data set created from the SAVE MODEL= option. When using the INMODEL option, the INPUT, TARGET, FREQ, and DECISION statements are prohibited.

Beginning with SAS 9.1, the MODEL= data set contains the name of the training and validation data. The DATA= option is therefore unnecessary to resume training with the same data as was used to create the saved model (assuming the saved name of the training data is still valid).

INTERVALBINS=*n*

specifies the number of preliminary bins to consolidate interval input values into. The width equals $(\max(x) - \min(x))/n$, where $\max(x)$ and $\min(x)$ are the maximum and minimum of the input variable values in the training data in the node being searched. The width is computed separately for each input and each node. The INTERVALDECIMALS= option, specifying the precision of the split values, can result in fewer bins than n being needed. The INTERVALBINS= option might indirectly modify p -value adjustments. The search algorithm ignores the INTERVALBINS= option if the number of distinct input values in the node is smaller than n .

Default: 100

INTERVALDECIMALS=*n* | MAX

specifies the precision, in decimals, of the split point for an interval input. When searching for a split on an interval input x , the partitioning procedure will combine all observations whose value of x are the same when rounded to n decimal places. If n equals zero, then the values are rounded to the nearest integer. N can be any value from 0 to 8. If INTERVALDECIMALS= MAX then no rounding is done. No rounding is default.

Default: MAX

LEAFFRACTION=*p*

specifies the smallest number of training observations a new branch can have, expressed as the proportion of the number N of available training observations in the DATA= data set specified in the PROC statement. N can be less than the total number of observations in the data set because observations with a missing target value or non-positive value of the variable specified in the FREQ statement are excluded from N . The LEAFSIZE= option specifies the same quantity as an absolute number. The partitioning procedure uses the larger of the two. P can be any number from zero through one.

Default: 0.001

LEAFSIZE=*n*

specifies the smallest number of training observations a new branch can have. The LEAFFRACTION= option specifies the same quantity as a proportion of the original training data. The partitioning procedure uses the larger of the two. The LEAFSIZE= option does not use the values of the variable in the FREQ statement to adjust the count of observations in the leaf.

Default: the number N of available training observations in the DATA= data set specified in the PROC statement, divided by 1,000, or 5, if 5 is larger, or 5,000, if 5,000 is smaller. N can be less than the total number of observations in the data set because observations with a missing target value or non-positive value of the variable specified in the FREQ statement are excluded from N .

MAXBRANCHES=*n*

restricts the number of subsets a splitting rule can produce to n or fewer. Setting n to 2 will create a model or grouping with only binary splits. Any integer from 2 through 50 is permitted.

Default: 2

MAXDEPTH= n | MAX

specifies the maximum depth of a node that the PROC statement will create automatically unless the MAXNEWDEPTH= option equals 1. The *depth* of a node equals the number of splitting rules needed to define the node. The root node has depth zero. The children of the root have depth one, and so on.

The PROC statement will search for a splitting rule in a leaf only when the MAXNEWDEPTH= option equals 1 or the depth of the leaf is $< n$.

The MAXDEPTH=MAX option specifies $n = 50$, the largest possible value of n . The smallest acceptable value of n is 0. Specify MAXDEPTH=0 or MAXNEWDEPTH=0 to avoid searching for any splits while specifying other options. The MAXDEPTH= option remains in effect until explicitly changed. The MAXNEWDEPTH= option reverts to its maximum value after the PROC statement finishes.

Default: The default value of n depends on the procedure. The default value of n is six for PROC ARBOR and two for PROC TREEBOOST.

MAXRULES= n | ALL

specifies how many splitting rules on different input variables are saved in each node, including leaves. The primary splitting rule in an internal node is always saved. Up to $n - 1$ additional competing rules are also saved in an internal node. The MAXRULES=ALL option requests the partitioning procedure to save all the available splitting rules for each node.

Saved rules can be displayed in results and can be output using the RULES option to the SAVE statement.

A valid splitting rule might not exist for some input variables in some nodes. A common explanation is that none of the feasible rules meet the threshold of worth specified in the ALPHA= option in the PROC statement. Other causes occur less often. For example, the MINCATSIZE= option in the PROC statement can prevent creation of a split on a categorical input X if few observations exist for any specific value of X. As another example, the LEAFFRACTION= and LEAFSIZE= options can prevent any split on a specific input, especially one that is nearly constant and consequently permits few candidate splits. No split exists with an input that is constant in a node. As a consequence of these and other possibilities, a node can contain fewer than n rules.

The amount of memory needed to save splitting rules, especially rules using nominal input variables with many values, might be substantial, possibly several megabytes. (Eight bytes for each branch and four more bytes for each categorical value is needed for each rule in each node.)

Default: The default value of n is three for the ARBOR procedure and one for other partitioning procedures.

MAXSURROGATES | MAXSURRS= n

specifies the number of surrogate rules sought for each primary splitting rule. A surrogate rule is a backup to the primary splitting rule. The primary splitting rule might not apply to some observations because the value of the splitting variable might be missing or be a categorical value the rule does not recognize. Surrogate rules are considered for such observations. The search for surrogate rules requires an extra pass over the data, and therefore no surrogates are sought by default. See “Missing Values” on page 500 for more information.

Surrogate rules enhance the split-based importance of the variables they use. See “Split-Based Variable Importance” on page 505 for more information.

MINCATSIZE= n

specifies the minimum number of observations that a given nominal input value must have in order to use the value in a split search. Categorical values that appear in

fewer than n observations are regarded as if they were missing. If USEINSEARCH is specified in the MISSING= option in the input statement for the splitting variable, the categories occurring in fewer than n observations are merged into the pseudo category for missing values for the purpose of finding a split. Otherwise, observations with infrequent categories are excluded from the split search. The policy for assigning such observations to a branch is the same as the policy for assigning missing values to a branch. See “Missing Values” on page 500 for more information.

Default: 5

MINWORTH=*worth*

specifies a threshold value for the worth of a candidate splitting rule, unless the CRITERION= option in the PROC statement is specified as PROBCHISQ or PROBF. A candidate rule whose *worth* is less than *worth* is discarded. When CRITERION=PROBCHISQ or PROBF, the MINWORTH= option is ignored and the ALPHA= option is used instead.

Default: 0

MISSING=*policy*

specifies how a splitting rule handles an observation with missing values. The following table lists the policies available in more than one partitioning procedure.

Table A1.3 Missing Value Policies

Policy	Description
BIGBRANCH	Assign the observation to the largest branch.
DISTRIBUTE	Assign the observation to each branch with a fractional frequency proportional to the number of training observations in the branch.
FIRSTBRANCH	For interval and ordinal inputs, assign to the first branch, otherwise, use missing values during the split search.
LASTBRANCH	For interval and ordinal inputs, assign to the last branch, otherwise, use missing values during the split search.
SMALLRESIDUAL	Assign to the branch minimizing SSE among observations with missing values.
USEINSEARCH	Use missing values during the split search (default).

The MISSING= option in the INPUT statement assigns a policy to the variables listed in the statement, and overrides the MISSING= option to the PROC statement. See INPUT Statement. If a surrogate rule can assign an observation to a branch, then it does, and the missing value policy is ignored for the specific observation. See “Missing Values” on page 500 for a complete description of the missing value options.

Default: USEINSEARCH

PADJUST=*method1* <*method2* <*method3*>>

names one or more methods for adjusting the p -values used with the PROBCHISQ and PROBF criteria. The following methods are available.

- CHAIDAFTER — applies a Bonferroni adjustment after split is chosen.
- CHAIDBEFORE — applies Bonferroni adjustment before split is chosen.
- DEPTH — adjusts for the number of ancestor splits.

- NOGABRIEL — suppresses an adjustment that sometimes overrides CHAID.
- NONE — suppresses all adjustments.

Specifying both CHAID_{AFTER} and CHAID_{BEFORE} is an error. Specifying NONE with any other method is an error. If the PADJUST= option is not specified, the CHAID_{BEFORE} and DEPTH methods are used. The PADJUST= option is ignored unless CRITERION= PROBCHISQ or PROBF. See “Adjusting p -Values for the Number of Input Variables” on page 499 for more information.

Default: CHAID_{BEFORE} and DEPTH if PADJUST= option is not specified.

PRIORSEARCH

requests that the prior probabilities defined in the DECISION statement be incorporated in the split search criterion for a categorical target. See “Incorporating Prior Probabilities” on page 493 for more information.

PVARS= n | ALL

specifies the number of input variables n to regard as independent when adjusting p -values for the number of inputs. PVARS=ALL specifies all the input variables as independent. When searching for a split, the partitioning procedure ignores input variables whose values are constant in the node being split, and ignores categorical variables unless at least two values occur in more observations than specified in the MINCATSIZE= option in the PROC statement. Consequently, the partitioning procedure can search for rules using only $m \leq N$ of the original N input variables. The procedure will regard $\max((n/N)m, 1)$ of the m variables as independent. See “Adjusting p -Values for the Number of Input Variables” on page 499 for more information.

Default: 0, requesting no adjustment for the number of inputs

REUSEVAR= n | NEVER | ALWAYS

influences how many splitting rules can use the same variable. When a splitting rule is considered for a node, the worth of the rule is multiplied by n if the splitting variable has already been used in a primary splitting rule elsewhere in the model. For $n > 1$, once a variable is used in a primary split, it is more likely to appear in another rule because it gets an advantage competing in its worth. The tree ends up using fewer variables, especially correlated variables. Reducing the number of variables in a tree can make the tree easier to interpret or to deploy. Eliminating correlated variables in the tree results in a more accurate measure of importance of those that are in the tree.

An appropriate value for n depends on the context. A value of two or three might be reasonable.

REUSEVAR=NEVER specifies that a variable will appear in two splitting rules in the same path from the root. When a variable is used to split a node, no descendent node will split on that variable. REUSEVAR=NEVER is the same as setting n to zero. Note that n equal to zero prevents using a variable a second time in a path, but allows reusing the variable elsewhere in the model. REUSEVAR=NEVER is the same as the USEVARONCE in previous versions of the ARBORETUM procedure.

A value of n greater than zero and less than one is unusual. It would give a disadvantage for using the same variable in a path, but still allow it.

REUSEVAR=ALWAYS specifies that all splitting rules in a tree will use the same input variable. Setting REUSEVAR=ALWAYS might be appropriate for models using more than one tree, such as in the TREEBOOST procedure, to create a model without any interactions between the variables.

Default: The default value for n is one, giving no advantage or disadvantage for reusing a variable in a path.

SPLITATDATUM

requests that a split on an interval input equal a value of the observation, if the value is an integer, or slightly less than a value if the value is not an integer. The alternative is to split halfway between two data values. The SPLITBETWEEN option requests the alternative.

SPLITBETWEEN

requests that a split on an interval input be halfway between two data values. The SPLITBETWEEN option is default. The SPLITATDATUM option is an alternative.

SPLITSIZE=*n*

specifies the requisite number of training observations a node must have for the recursive partitioning procedure to consider splitting it. By default, *n* is twice the value of the LEAFSIZE= option. For the LEAFFRACTION=, LEAFSIZE=, MINCATSIZE=, and SPLITSIZE= options in the PROC statement, and the NODESIZE= option in the PERFORMANCE statement, the procedure counts the number of observations in a node without adjusting the number with the values of the variable specified in the FREQ statement.

Default: twice the value of the LEAFSIZE= option

ASSESS Statement

ASSESS <*option(s)*>;

The ASSESS statement specifies the validation data and the statistical measure for evaluating submodels: subtrees of a decision tree in the ARBOR procedure and subseries of a boosted series of trees in the TREEBOOST procedure. Other recursive partitioning procedures do not have submodels and so do not recognize the ASSESS statement.

The ASSESS statement acts without waiting for further statements. If no partition exists, the ASSESS statement trains the model. All statements necessary for training, such as DECISION, FREQ, INPUT, and TARGET, must precede the ASSESS statement. Statements that output results such as CODE, SAVE, and SCORE, logically follow the ASSESS statement. If such a statement occurs without a preceding ASSESS statement, and no partition exists, then an ASSESS statement is executed automatically with default settings. The ASSESS statement can be repeated.

The table below summarizes the options that are available. The ARBOR procedure has a few more options pertaining to lift.

An option remains in effect in subsequent occurrences of the ASSESS statement unless explicitly specified differently.

Table A1.4 Assess Statement Options

Option	Description
MEASURE=	specifies the assessment measure for selecting a model
NOPRIORS	ignores prior probabilities in model selection
PRIORS	incorporates prior probabilities in model selection

Option	Description
VALIDATA=	specifies validation data set
NOVALIDATA	terminates a previous VALIDATA= option

MEASURE=PROFIT | ASE | MISC

specifies the assessment measure for selecting a model from a family of models. The available assessment measures are as follows:

- ASE — Average square error
- MISC — Proportion misclassified
- PROFIT — Average profit or loss from the decision function

MISC is applicable to nominal and ordinal targets. ASE is applicable to any kind of target.

Default: The default measure is PROFIT if the DECISION statement specifies a profit or loss function or if the target variable is ordinal. Otherwise the default measure for a nominal target is MISC, and the default for an interval target is ASE.

NOPRIORS | PRIORS

specifies whether to use prior probabilities when selecting a model.

Default: NOPRIORS, ignoring prior probabilities.

NOVALIDATA | VALIDATA= <SAS-data-set>

specifies the validation data set. The NOVALIDATA option nullifies any VALIDATA= option appearing in a previous ASSESS statement.

CODE Statement

CODE <option(s)>;

The CODE statement generates SAS DATA step code that mimics the computations done by the SCORE statement. The DATA step code creates the same variables described in SCORE Statement OUT= Output Data Set. Using the CODE statement for a model containing a rule with MISSING=DISTRIBUTE is an error.

CATALOG= catname | FILE= filename

specifies where to output the code. Catname specifies a catalog entry by providing a compound name with one to four of the levels in the form, library.catalog.entry.type. The default library is determined by the SAS system option USER=, usually WORK. The default entry is SASCODE, and the default type is SOURCE. Filename specifies the name of the file to contain the code. Filename can be either:

- A quoted string, the value of which is the name (including the extension, if any) of the file to be opened.
- An unquoted SAS name of no more than eight characters. If this name has been assigned as a fileref in a FILENAME statement, the file specified in the FILENAME statement is opened. The special filerefs LOG and PRINT are always assigned. If the specified name is not an assigned fileref, the specified value is concatenated with the extension .txt before opening. For example, if FOO is not an assigned fileref, FILE=FOO would cause FOO.txt to be opened. If the name has more than eight characters, an error message is printed.

If no catalog or file is specified, then the code is output to the SAS log.

FORMAT= *format*

specifies the format to use in the DATA step code for numeric values that do not have a format from the input data set.

Default: BEST20.

LINESIZE | LS= *n*

specifies the line size for generated code. The permissible range is 64 to 254.

Default: 72

NOLEAFID

suppresses the creation of variables `_NODE_` and `_LEAF_` containing the node and leaf identification numbers of the leaf to which the observation is assigned.

Default: `_NODE_` and `_LEAF_` variables are created

NOPREDICTION

suppresses the code for computing predicted variables, such as `P_:`.

Default: PREDICTION, requesting such code

RESIDUAL

requests the DATA step code to create variables, such as residuals, that require the target variable. These variables are the ones with a "yes" in the "Target" column of the table in section "Variable Names and Conditions for Their Creation" on page 515. Using the DATA step code generated by the RESIDUAL option with a data set that does not contain the target variable produces confusing notes and warnings.

Default: NORESIDUAL, suppressing the generation of the DATA step code for these variables.

DECISION Statement

DECISION DECDATA= *SAS-data-set* <option(s)> ;

The DECISION statement specifies decision functions and prior probabilities for categorical targets. A recursive partitioning procedure uses the term *decision* in the sense of decision theory: a *decision* is one of a set of alternatives, each associated with a function of posterior probabilities. For an observation i , a model determines the decision d_i whose associated function evaluates to the best value, $E_i(d)$. The interpretation of *best* as well as the form of the function depends on whether the type of the DECDATA= data set is *profit*, *revenue*, or *loss*. The SAS DATA step TYPE= option specifies the data set type. If the DECDATA= data set has no type, a partitioning procedure assumes a type of profit.

The following formulas define $E_i(d)$ and d_i . The sum is over the J categorical target values, and p_{ij} denotes the posterior probability of target value j for observation i . The coefficient, A_{jd} , for target value j , decision d , is specified in the DECDATA= data set.

PROFIT:

$$E_i(d) = \sum_{j=1}^J A_{jd} p_{ij}$$

where

$$d_i = \arg \max_d E_i(d)$$

REVENUE:

$$E_i(d) = \sum_{j=1}^J A_{jd} p_{ij} - C_{id}$$

where

$$d_i = \arg \max_d E_i(d)$$

and C_{id} is the cost of decision d for observation i , specified in the COST= option.

LOSS:

$$E_i(d) = \sum_{j=1}^J A_{jd} p_{ij}$$

where

$$d_i = \arg \min_d E_i(d)$$

The decision functions do not affect the creation of the model unless the DECSEARCH option is specified in the PROC statement. However, the decision functions determine a profit or loss measure for assessing submodels in the ARBOR and TREEBOOST procedures, and consequently might greatly affect what nodes or trees are pruned and omitted from the final submodel.

FREQ, INPUT, and TARGET statements must appear before the DECISION statement. The DECISION statement is optional. When the DECISION statement is omitted, neither decision alternatives nor prior probabilities are defined. Specifying the DECISION statement and the INMODEL= option in the PROC statement is an error.

COST=costs

specifies a list of cost constants and cost variables associated with the decision alternatives specified in the DECVAR= option. The first cost in the list corresponds to the first alternative in the DECVAR= list, the second cost with the second alternative, and so on. The number of costs must equal the number of alternatives specified in the DECVAR= list.

The costs specify the terms C_{id} in the REVENUE formula for $E_i(d)$, and consequently the COST= option requires a DECDATA= data set of type REVENUE.

A cost constant is a number specifying the same value to C_{id} for all observations i . A cost variable is the name of a numeric variable in the training data set specified in the DATA= option in the PROC statement. The value of this variable for observation i is assigned to C_{id} . A partitioning procedure does not recognize abbreviations of lists

of variables in the COST= option. For example, D1-D3, ABC-XYZ, and PQR: are invalid representations of lists of variables.

DECADATA=SAS-*data-set*

specifies the input data set containing the decision coefficients A_{jd} and prior probabilities. The DECADATA= data set must contain the target variable. One observation must appear for each target value in the training data set specified in the DATA= option of the PROC statement.

DECVARS=*decision-alternatives*

specifies the variables in the DECADATA= data set defining the coefficients, A_{jd} . The labels of the variables define the names of the decision alternatives. For a variable without a label, the name of the decision alternative is the name of the variable.

If the DECVARS= option is omitted, no decision functions are defined.

PRIORVAR=*pvar*

specifies the variable *pvar* in the DECADATA= data set that contains the prior probabilities of categorical target values. See “Data, Prior, and Posterior Probabilities” on page 473 for more information. *Pvar* must have nonnegative numeric values. A partitioning procedure rescales the values to sum to one, and ignores training observations with a target value for which *pvar* equals zero.

Prior probabilities do not affect the creation of the model unless the PRIORSSEARCH option to the PROC statement is specified. Prior probabilities affect the posterior probabilities, and consequently affect the model predictions and assessment.

FREQ Statement

FREQ *variable*;

The FREQ statement identifies a variable that contains the frequency of occurrence of each observation. A partitioning procedure treats each observation as if it appears n times, where n is the value of the FREQ variable for the observation. The value of n can be fractional to indicate partial observations. If the value of n is close to zero, negative, or missing, the observation is ignored. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

The LEAFFRACTION=, LEAFSIZE=, MINCATSIZE=, and SPLITSIZE= options in the PROC statement, and the NODESIZE= option in the PERFORMANCE statement ignore the FREQ statement in that the options do not use the variable values to adjust the specified number of observations.

IMPORTANCE Statement

IMPORTANCE *<option(s)>*;

The IMPORTANCE statement implements an observation-based approach to evaluate the importance of a variable or of a pair of variables to the predictions of the model.

For each observation, the value of the variable or pair of variables being evaluated is rendered uninformative in a process described in detail later. The IMPORTANCE statement outputs the prediction once using the actual value and a second time using

the uninformative value. The difference between the two predictions shows the dependence of the prediction on the variable or pair of variables being evaluated. The differences for all the observations can be plotted against the actual variable value or observation number to explore where the dependence is stronger or weaker.

A partitioning procedure also computes goodness-of-fit statistics once using the actual value and a second time using the uninformative value. Comparing the two shows the dependence of the statistics on the variable. Evaluating several variables produces statistics that can then be ranked to reveal the relative observation-based importance of the variables.

The observation-based importance differs from the split-based importance computed in the IMPORTANCE= option of the SAVE statement. The latter importance is based on the contribution a variable makes in reducing the residual sum of squares.

The DATA=, OUT=, and OUTFIT= options are the same as those in the SCORE statement. The NVAR=, N2WAY=, and VAR= options specify the variables to evaluate. If the NVAR=, N2WAY=, and VAR= options are absent, then a partitioning procedure assumes NVAR= 5. The IMPORTANCE statement can be repeated.

DATA=SAS-data-set

names the input data set. If the DATA= option is absent, the procedure uses the training data.

N2WAY=m n

requests to evaluate the best m variables paired with the best n variables, where the term “best” here refers to the split-based variable importance rankings computed in the IMPORTANCE= option of the SAVE statement. If n is missing, then n is set to m . When a procedure evaluates a pair of variables, it also evaluates the two variables individually and outputs results as if the variables were specified in the VAR= option.

Default: 0 0

NVAR=n

requests to evaluate the best n variables as ranked by the split-based variable importance that was computed in the IMPORTANCE= option of the SAVE statement. If the N2WAY=, NVAR=, and VAR= options are absent, then a partitioning procedure assumes NVAR= 5.

OUT=SAS-data-set

names the output data set to contain the scored data. If the OUT= option is absent, a partitioning procedure creates a data set name using the DATA n convention.

The OUT= data set in the IMPORTANCE statement has the same variables as the OUT= data set specified in the SCORE statement, plus one or two more, `_INPUT1_` and `_INPUT2_`, that contain the name of a variable whose values were treated as uninformative when making the predictions. If `_INPUT1_` is blank, then `_INPUT2_` is blank and the predictions are the same as in the OUT= data set of the SCORE statement. The OUT= data set becomes very large if many variables are being evaluated. The number of observations in the OUT= data set equals the number of variables and pairs of variables being evaluated plus one times the number of observations in the data set. Specify OUT= `_NULL_` to avoid creating a scored data set. “Variable Names and Conditions for Their Creation” on page 515 describes the variables in the OUT= data set.

OUTFIT=SAS-data-set

names the output data set to contain the fit statistics. The number of observations in the OUTFIT= data set equals the number of variables and pairs of variables being evaluated plus one.

The OUTFIT= data set in the IMPORTANCE statement has the same variables as the OUTFIT= data set specified in the SCORE statement, plus one or two more, `_INPUT1_` and `_INPUT2_`, that contain the name of a variable whose values were

treated as uninformative when computing the statistics. If `_INPUT1_` is blank, then `_INPUT2_` is blank and the statistics are the same as in the `OUTFIT=` data set of the `SCORE` statement.

VAR=(varlist)

specifies variables and pairs of variables to evaluate. *Varlist* is a list of variable names optionally containing asterisks to indicate a pair of variables. Variables on the left or right of an asterisk can be grouped within square brackets. Brackets cannot be nested. Parentheses must enclose the list, *varlist*.

When a procedure evaluates a pair of variables, it also evaluates the two variables individually and outputs the results. For example, the following *varlist* would specify variables A, B, C, D, and E, and pairs of variables, B-C, D-E, D-C, and E-C:

```
A B*C [D E] * [E C]
```

INPUT Statement

INPUT *variables* < /*option(s)* >;

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

LEVEL= INTERVAL | NOMINAL | ORDINAL

specifies the level of measurement. See “Terminology” on page 472 for more information.

Default: The default level is `INTERVAL` for a numeric variable, `NOMINAL` for a character variable.

MISSING= policy

specifies the missing value policy for the inputs. The option is the same as the `MISSING=` option in the `PROC` statement, except that it only applies to the variables in the `INPUT` statement. If the option is omitted, the policy specified in the `MISSING=` option in the `PROC` statement applies to these variables.

ORDER=sortorder

specifies the sorting order of the values of an ordinal input variable. The `ORDER=` option is available only when `LEVEL=ORDINAL` is specified. The following table shows how a partitioning procedure interprets values of the `ORDER=` option.

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

The “Terminology” section discusses formatted values. When `ORDER=ASCFORMATTED` or `DESFORMATTED` for numeric input variables for

which no explicit format is declared, the ordering often deviates from the numeric one. It is consequently unexpected and undesired. ORDER=ASCENDING and DESCENDING orders the values of numeric variables by their numeric values.

When ORDER=ASCENDING or DESCENDING, and more than one unformatted value has the same formatted value, a partitioning procedure uses the smallest unformatted value (with the same formatted value) to determine the ordering of the formatted values. A splitting rule on an ordinal input assigns a range of formatted values to a branch. The range will correspond to a range of unformatted values if all unformatted values with the same formatted value define an interval that contains no other values.

The sorting order of character values, including formatted values, might be machine dependent. For more information on sorting order, see the chapter on the SORT procedure in the *SAS Procedures Guide*.

Default: ASCENDING

SPLITATDATUM

requests that a split on an interval input equal the value of the observation, if the value is an integer, or slightly less than the value if the value is not an integer. The alternative is to split an interval variable halfway between two values.

SPLITBETWEEN

requests that a split on an interval input be halfway between two data values. The SPLITBETWEEN option is default, unless the SPLITATDATUM option is specified in the PROC statement. The SPLITATDATUM option is the alternative.

MAKEMACRO Statement

MAKEMACRO keyword= *macname*;

The MAKEMACRO statement specifies the name of a macro variable to contain a statistic of the model. The macro variable is available immediately, before the execution of the next statement. See the documentation of an individual procedure for a list of available keywords.

If the MAKEMACRO statement appears before a model is trained, the procedure trains a model before executing the statement. Subsequent initialization statements are prohibited.

PARTIALDEP Statement

PARTIALDEP <DATA=><OUT=>VAR=(varlist);

The PARTIALDEP statement outputs a partial dependency function for plots and interaction detection. See “Partial Dependency Functions” on page 509 for a more complete description.

DATA=SAS-*data-set*

names the input data set. If the DATA= option is absent, the procedure uses the training data. Only the variables listed in the VAR= option are used.

OUT=SAS-*data-set*

names the output data set to contain the partial dependency function. If the OUT= option is absent, a partitioning procedure creates a data set name using the DATAn convention. The output variables are the same as output with the OUT= option to the SCORE statement. See “SCORE Statement OUT= Output Data Set” on page 515

VAR=(varlist)

specifies the variable or pair of variables to compute the partial dependency for. Parentheses must enclose the list, varlist. The PARTIALDEP statement requires at least one variable and at most two variables. The procedure ignores the value of all other variables.

PERFORMANCE Statement

PERFORMANCE <option(s)>;

The PERFORMANCE statement specifies options affecting the speed of computations with little or no impact on the results. See “Performance Considerations” on page 518 for more information.

WORKDATALOCATION = RAM | DISK | SOURCE

specifies the location to put the working copy of the training data. The RAM location requests that the working copy be stored in memory if enough memory is available for it and still allow for a single split search in one pass of the data. The DISK location requests that the working copy be stored in a disk utility file. Storing the copy on disk can free a considerable amount of memory for calculations, possibly speeding up the program. The SOURCE location requests that the training data be read multiple times instead of copying it to memory or a disk utility file. SOURCE is slower than DISK because the DISK copy is converted to encodings directly usable in the calculations. The SOURCE location is preferable only when the training data will not fit in RAM or in a disk utility file.

MEMSIZE=m<B | K | M | G>

specifies the maximum amount of memory to allocate for the computations and the working copy of the training data if the data is stored in memory. The optional suffix, B, K, M, G, specifies bytes, kilobytes, megabytes, and gigabytes respectively. Without a suffix, m specifies the number of bytes. M can be fractional.

The SAS MEMSIZE system option sets an upper limit to the number of bytes.

Default: The default value depends on the computer and might considerably prolong the execution time if SAS cannot distinguish physical memory from virtual memory.

NODESIZE=n | ALL

specifies the number of training observations to use when searching for a splitting rule. NODESIZE=ALL requests to use all the observations. For larger data sets, using a large within-node sample might require more passes of the data, resulting in a longer running time. See “Performance Considerations” on page 518 for more information.

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

See “Within Node Training Sample” on page 502 for more information.

Default: 10,000

SAVE Statement

SAVE< *option(s)*>;

The SAVE statement outputs model information into SAS data sets. For partitioning procedures that generate a family of models, the information describes the model selected in the ASSESS or SUBTREE or SUBSERIES statement.

DISSIMILARITY = SAS-data-set

names the output data set to contain a dissimilarity statistic for pairs of input variables. The data set has type DISTANCE and is suitable for input to the DATA= option of the CLUSTER procedure. The data set includes an ID variable, _VAR_. The dissimilarity matrix equals one minus the similarity matrix output in the SIMILARITY= option. Similarity relies on surrogate rules. Use the MAXSURROGATES= option to the PROC statement to create surrogate rules when the model is trained. See “Similarity and Dissimilarity of Pairs of Inputs” on page 504 for more information.

IMPORTANCE= SAS-data-set

names the output data set to contain the split-based variable importance. See “Split-Based Variable Importance” on page 505 for more information.

MODEL= SAS-data-set

names the output data set to encode the information necessary for use with the INMODEL= option in a subsequent invocation of a partitioning procedure.

NODES=nodes

specifies what nodes to output in the NODESTAT=, PATH=, and RULES= data sets. By default, the NODESTAT= and RULES= data sets contain information for all nodes, and the PATH= data set contains information for all leaves in the current submodel.

NODESTAT= SAS-data-set

names the output data set to contain node information. See “NODESTATS= Output Data Set” on page 511 for more information.

PATH= SAS-data-set

names the output data set describing the path to nodes. See “PATH= Output Data Set” on page 512 for more information.

RULES= SAS-data-set

names the output data set describing the splitting rules. See “RULES= Output Data Set” on page 513 for more information.

SIMILARITY = SAS-data-set

names the output data set to contain a similarity statistic for pairs of input variables. The data set contains a variable for every input variable used in a primary splitting rule, and an additional identification variable, _VAR_, whose value is the name of an input variable. Similarity relies on surrogate rules. Use the MAXSURROGATES= option to the PROC statement to create surrogate rules when the model is trained. The similarity matrix equals one minus the dissimilarity

matrix output using the DISSIMILARITY= option. The DISSIMILARITY= option creates a DISTANCE matrix suitable for input into the CLUSTER procedure. See “Similarity and Dissimilarity of Pairs of Inputs” on page 504 for more information.

SUMMARY= SAS-data-set

names the output data set to contain summary statistics. For categorical targets, the summary statistics consists of the counts and proportions of observations correctly classified. For interval targets, the summary statistics include the average square error and R-squared ($= 1 - \text{average squared error} / \text{sum of square errors from the prediction}$).

SCORE Statement

SCORE< *option(s)*>;

The SCORE statement reads a data set containing the input variables used by the tree and outputs a data set containing the original variables plus new variables to contain predictions, residuals, decisions, and leaf assignments. The SCORE statement can be repeated.

DATA=SAS-data-set

names the input data set. If the DATA= option is absent, the procedure uses the training data.

PREDICTION | NOPREDICTION

requests generation of prediction variables, such as P_*

Default: PREDICTION, requesting prediction variables

OUT=SAS-data-set

names the output data set to contain the scored data. If the OUT= option is absent, a partitioning procedure creates a data set name using the DATAn convention. Specify OUT= _NULL_ to avoid creating a scored data set. The "SCORE Statement OUT= Output Data Set" section on page 49 describes the variables in the OUT= data set.

OUTFIT=SAS-data-set

names the output data set to contain the fit statistics.

ROLE=TRAIN | VALID | TEST | SCORE

specifies the role of the input data set, and determines the fit statistics to compute.

For ROLE=TRAIN, VALID, or TEST, observations without a target value are ignored.

TARGET Statement

TARGET *variable*< /*option(s)*>;

The TARGET statement names the variable that the model tries to predict. See “INPUT Statement” on page 487 for more information.

LEVEL= INTERVAL | ORDINAL | NOMINAL | BINARY

specifies the level of measurement of the target variable.

Default: The default is INTERVAL for a numeric variable, NOMINAL for a character variable.

**ORDER= ASCENDING | ASCFORMATTED |
DESCENDING | DESFORMATTED | DSORDER**

specifies the ordering of the values of an ordinal target variable. The ORDER= option is available only when LEVEL=ORDINAL is specified, and would have no impact with a target variable with only two values. The option is the same as the ORDINAL= option in the INPUT statement.

Details: Recursive Partitioning Procedures

Form of a Splitting Rule

A splitting rule uses the value of a single input variable to assign an observation to a branch. The branches are ordered and numbered consecutively starting with 1. Every splitting rule (other than a surrogate rule) includes an assignment of missing values to one or all branches, even if no missing values appear in the data. Rules defining a branch exclusively for missing values assigns the missing values to the last branch.

For interval and ordinal inputs, observations with smaller input values are assigned to branches with smaller numbers. Consequently, a list of increasing input values suffices to specify a splitting rule. A surrogate rule can disregard the ordering and assign smaller values of the input to any branch.

Rules need not assign any training observations to a particular branch. A partitioning procedure does not automatically generate such rules, but a user can specify them in the SETRULE or SPLIT interactive training statements, available in the ARBOR procedure.

Posterior and Within Node Probabilities

The predicted proportions of categorical target values for an observation are called the *posterior probabilities* of the target values.

For an observation assigned to a node in a tree, the posterior probabilities equal the *predicted within node probabilities*, which are the proportions of the target values of all the training observations assigned to the node, adjusted for prior probabilities (if any), and not adjusted for any profit or loss coefficients. For an observation assigned to more than one leaf with fractional weights that sum to one, the posterior probabilities are the weighted averages over the leaves of the predicted within node probabilities.

The *within node probabilities for a split search* are the proportions of the target values in the *within node training sample*, adjusted for the bias from stratified sampling, and adjusted for prior probabilities if requested by the PRIORSSEARCH option in the PROC statement, and adjusted for profit or loss coefficients if requested by the DECSEARCH option in the PROC statement.

When neither priors, profits, nor losses are specified, and observations are assigned to a single leaf, and within node sampling is not used, the posterior and within node probabilities are simply the proportions of the target values in a node τ :

$$p_j = \text{proportion}_j(\tau) = N_j(\tau) / N(\tau)$$

where $N_j(\tau)$ is the number of training observations in τ with target value j .

When incorporating priors, profits, or losses, this becomes

$$p_j = \frac{\rho_j \text{proportion}_j(\tau) / \text{proportion}_j(\text{root})}{\sum_i \rho_i \text{proportion}_i(\tau) / \text{proportion}_i(\text{root})}$$

or equivalently,

$$p_j = \frac{\rho_j N_j(\tau) / N_j(\text{root})}{\sum_i \rho_i N_i(\tau) / N_i(\text{root})}$$

where $N_j(\text{root})$ is the number of training observations in the root node with target value j , and ρ_j depends on whether p_j incorporates priors, profits, or losses. The following table defines ρ_j by type of quantity being incorporated.

Table A1.5 ρ_j by Type of Incorporated Quantity

Incorporated Quantity	ρ_j	Description
Nothing	$N_j(\text{root})$	number of j observations in root
Prior Probabilities	π_j	prior probability
Profit or Loss	π_j^a	altered prior probability

Incorporating Prior Probabilities

The PRIORVAR option in the DECISION statement declares the existence of prior probabilities. If prior probabilities exist, they are always incorporated in the posterior probabilities. If the PRIORSEARCH option is specified in the PROC statement, the priors will also be incorporated in the search for a splitting rule. If the PRIORS option to the ASSESS statement is specified, the priors will also be incorporated in the evaluation of subtrees and consequently influence which nodes are automatically pruned.

In all cases, the priors are incorporated by defining the within node probabilities above with $\rho_j = \pi_j$, the prior probability of target value j , or, when incorporating a profit or a loss, π_j^a , the altered prior probability defined below.

Incorporating Decisions, Profit, and Loss

The DECSEARCH option in the PROC statement requests that the split search for a nominal target incorporate the profit or loss functions specified in the DECISION statement. Unequal misclassification costs of Breiman et al. (1984) are a special case in which the decision alternatives equal the target values and the DECDATA= data set is type LOSS. A partitioning procedure generalizes the method of *altered priors* introduced in Breiman et al.

The search incorporates the decisions, profit, or loss functions by using $\rho_j = \pi_j^a$ in the definition of within node probability, p_j . Let A_{jd} denote the coefficient for decision d , target value j , in the decision matrix. Define

$$a_j = \sum_d |A_{jd}|$$

If the PRIORSEARCH option is specified in the PROC statement requesting the search to incorporate prior probabilities, then

$$\pi_j^a = \frac{a_j \pi_j}{\sum_i a_i \pi_i}$$

defines the *altered prior probability* for target value j , where π_j denotes the prior probability of j . Intuitively, the alteration inflates the prior probability for those target values having large profit or loss coefficients, thereby giving observations with those target values more weight in the split search. The search incorporates the altered priors instead of incorporating the original priors.

If the PRIORSEARCH option is not specified, then the definition of π_j^a changes by replacing π_j with $N_j(\text{root})/N(\text{root})$, the simple proportion of observations having target value j :

$$\pi_j^a = \frac{a_j N_j(\text{root})/N(\text{root})}{\sum_i a_i N_i(\text{root})/N(\text{root})}$$

Splitting Criteria

A partitioning procedure searches for rules that maximize the measure of worth associated with the splitting criterion specified in the CRITERION= option in the PROC statement. Some measures are based on a node impurity measure, others on p -values of a statistical test. A p -value can be adjusted for the number of branches and input values, the depth of the node in the tree, and the number of independent input variables for which candidate splits exist in the node. A measure for a categorical target can incorporate prior probabilities. A measure for a nominal target can incorporate profit or loss functions, including unequal misclassification costs. A measure for ordinal targets must incorporate distances between target values. A recursive partitioning procedure creates a distance function from a loss function specified with the DECISION statement.

This section defines the formulas for computing the worth of a rule s that splits node τ into B branches, creating nodes, $\{\tau_b : b = 1, 2, \dots, B\}$. $N(\tau)$ denotes the number of observations in node τ used in the search for the rule s .

Reduction in Node Impurity

The impurity $i(\tau)$ of node τ is a nonnegative number that equals zero if all observations in τ have the same target value, and is large if the target values in τ are very different. The option CRITERION=VARIANCE specifies average square error as the impurity measure for an interval target:

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

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

The option CRITERION=ENTROPY specifies entropy as the impurity measure for a categorical target:

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

where p_j is the proportion of observations with target value j in τ , possibly adjusted by prior probabilities, a profit function, or a loss function.

The option CRITERION=GINI specifies the Gini index as the impurity measure, which is also the average square error for a categorical target:

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

For a binary target, CRITERION=GINI creates the same binary splits as CRITERION=ENTROPY.

The worth of a split s is defined as the reduction in node impurity:

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

where the sum is over the B branches that the split s defines, and $p(\tau_b | \tau)$ is the proportion of observations in τ assigned to branch b . For CRITERION=VARIANCE, the reduction in node impurity is $N(\tau)$ times the reduction in within-node sum of squares:

$$\Delta i(s, \tau) = \left(w(\tau) - \sum_{b=1}^B w(\tau_b) \right) / N(\tau)$$

where

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

Statistical Tests and p -Values

An alternative to using the reduction in node impurity is to test for a significant difference of the target values between the different branches defined by a candidate split. The worth of the split is equal to $-\log_{10}(p)$, where p is the p -value (possibly adjusted) of the test. The minus sign ensures that the worth is nonnegative, with larger values being more significant. A partitioning procedure never computes the raw p -value because it is often smaller than the precision of the computer. Instead, the procedure computes $\log_{10}(p)$ directly.

For an interval target, the CRITERION=PROBF option requests to use the F -statistic:

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

where

$$SS_{between} = \sum_{b=1}^B N(\tau_b) (\bar{Y}(\tau_b) - \bar{Y}(\tau))^2$$

$$SS_{within} = \sum_{b=1}^B \sum_{i=1}^{N(\tau_b)} (Y_{bi} - \bar{Y}(\tau_b))^2$$

The p -value equals the probability $z \geq F$ where z is a random variable from an F distribution with $N(\tau) - B$, $B - 1$ degrees of freedom.

For a nominal target, the CRITERION=PROBCHISQ option requests using the chi-square statistic

$$\chi^2 = N(\tau) \sum_{b=1}^B \sum_{j=1}^J \frac{(p_j(\tau_b) - p(\tau_b|\tau) p_j(\tau))^2}{p(\tau_b|\tau) p_j(\tau)}$$

The p -value equals the probability $\chi_v^2 \geq \chi^2$, where χ_v^2 is a random variable from a chi -square distribution with $v = (B-1)(J-1)$ degrees of freedom.

A recursive partitioning procedure provides no statistical test for an ordinal target.

Distributional Assumptions

The F-test assumes that the interval target values $Y_i(\tau_b)$ are normally distributed around a mean that might depend on the branch, τ_b . The chi -square test assumes that the difference between the actual and predicted number of observations for a given target value j in a given branch τ_b , $N_j(\tau_b) - p(\tau_b|\tau) N_j(\tau)$, is normally distributed.

Normality is never checked in practice. Even if the distribution overall training observations were normal, the distribution in a branch need not be. The central limit theorem guarantees approximate normality for large $N(\tau_b)$. However, every split decreases $N(\tau_b)$ and thereby degrades the approximation provided by the theorem.

The search for a split on a variable does not depend on normality, but the evaluation of the selected split on the variable in terms of a p -value does. The procedure uses the p -value to compare the best split on one variable to that of another, and to compare against the threshold significance level specified in the ALPHA= option in the TRAIN statement. Consequently, one potential risk of nonnormality is that the best splitting variable is rejected in favor of another variable because the p -values are incorrect.

The more important risk is that of mistaking an insignificant split as significant, one whose p -value is smaller than ALPHA, thereby creating a split that disappoints when applied to a new sample drawn from the same distribution. The risk is that significance testing would not prevent the tree from overfitting.

No assumption is made about the distribution of the input variable defining the splitting rule. The split search only depends on the ranks of the values of an interval or ordinal input. Consequently, any monotonic transformation of an interval input variable results in the same splitting rule.

Multiple Testing Assumptions

The application of significance tests to splitting rules began in the early 1970s with the CHAID methodology of Kass (1980). CHAID and its derivatives assume that the number of independent significance tests equals the number of candidate splits. Even though a p -value is computed only for a single candidate split on a variable, and the split search might not examine every possible split, CHAID regards every possible split as representing a test.

Let α denote the significance level of a test; α equals the probability of mistakenly rejecting the hypothesis of no association between the target values and the branches when there is indeed no association. For m independent tests, the probability of mistakenly rejecting at least one of them equals one minus the probability of rejecting none of them, which equals

$$P(\text{one or more spurious splits}) = 1 - (1 - \alpha)^m$$

For example, this equals 0.401 for $\alpha = 0.05$ and $m = 10$.
Expanding the polynomial of degree m ,

$$(1 - \alpha)^m = \sum_{k=0}^m (-1)^k \frac{m!}{k!(m-k)!} \alpha^k$$

Multiplying by minus one and then adding one yields

$$P(\text{one or more spurious splits}) =$$

$$\sum_{k=0}^m (-1)^{k+1} \frac{m!}{k!(m-k)!} \alpha^k$$

The *Bonferroni approximation* assumes that terms in α^2 and higher are ignorable:

$$P_{\text{Bonferroni}}(\text{one or more spurious splits}) = m\alpha$$

The CHAID methodology uses this expression to evaluate the best split on a variable, using m equal to the number of possible splits on the variable in the node, and α equal to the p -value of the split. Let $\kappa(\tau, v, b)$ denote the number of candidate splits of node τ into b branches using input variable v . The CHAID methodology sets m equal to $\kappa(\tau, v, b)$.

Setting m equal to the number of possible splits on all variables would produce a much larger value of m than using the number of splits on a single variable. If no input variable were predictive of the target in node τ , a split of node τ would occur by chance using

$$m = \sum_v \sum_{b=2}^{\text{MAXBRANCH}} \kappa(\tau, v, b)$$

in the above expression for the probability, where MAXBRANCH denotes the value of the MAXBRANCH= option in the TRAIN statement.

This value of m and the CHAID value of m are often unrealistically large for computing the probability of a spurious split in a node. The main difficulty is that candidate splits are not independent, but formulating an estimate of the significance

probability without assuming that independence seems impossible. Incorporating the correlation between tests would decrease the estimated probability of a spurious split. Consider an extreme example for illustration: suppose two variables are identical. The candidate splits using one of the variables would be identical to those of the other, and the tests using one would simply repeat those of the other. Incorporating the (perfect) correlation of the two variables would reduce the estimate of the probability of a spurious split by half.

A common situation exposing the awkwardness of the assumption of independent tests is that of a search for a binary split on an interval variable with no tied values. A split at one point assigns most observations to the same branch that a split on a nearby point does, and consequently all splits on nearby points are highly correlated. Regarding all candidate splits as independent creates an m so unrealistically large that an estimate of the probability of a spurious split is near certainty. To avoid this, some analysts first group the values of an interval input variable into 10 or so ordinal values. The INTERVALBINS= option in the TRAIN statement sets the number of groups for this purpose. The groups are created separately in each node. Even after this grouping, a partitioning procedure might consolidate the remaining values, thereby reducing the number of candidate splits. See the “Split Search Algorithm” on page 502 for more information.

Adjusting p -Values for Multiple Tests

When specifying CRITERION=PROBF or CRITERION=PROBCHISQ, a partitioning procedure might adjust the p -value of the significance test when comparing candidate splits with each other, or when comparing a p -value with the significance threshold specified in the ALPHA= option in the PROC statement.

For a particular node, variable, and number of branches, the procedure can find the best candidate without computing a p -value by finding the candidate with the largest F statistic or χ -square statistic.

If the PADJUST=CHAIDBEFORE option is specified, the p -value is multiplied by $\kappa(\tau, v, b)$. Otherwise no adjustment is made yet. This procedure repeats for each possible number of branches, producing a single candidate split for each number of branches, and chooses the one with the best adjusted or unadjusted p -value according to whether PADJUST=CHAIDBEFORE is or is not specified.

If the PADJUST=CHAIDAFTER is specified, the final candidate split in the node for the variable is multiplied by $\kappa(\tau, v, b)$. If either the PVARs= n or PADJUST=DEPTH option is specified in the PROC statement, the p -value is further multiplied by a factor to adjust for the number of variables or the depth of the node τ in the tree, to arrive at a final adjusted p -value of the candidate split.

If the adjusted p -value is greater than the value of the ALPHA= option in the PROC statement, the candidate is discarded, and the procedure proposes no split of τ using the variable.

Adjusting p -Values for the Number of Input Values and Branches

The PADJUST=CHAIDAFTER or CHAIDBEFORE option in the PROC statement requests the partitioning procedure to multiply the p -value of the χ^2 statistic computed for the PROBCHISQ criterion for a nominal target by a Bonferroni factor κ to adjust for using multiple significance tests. If κp is larger than the p -value of an alternative conservative significance test called *Gabriel's*, then Gabriel's p -value is used instead of κp unless the PADJUST=NOGABRIEL option is specified.

Let B denote the number of branches, and c the number of input variable values available to the split search. If the MISSING=USEINSEARCH option is specified in the

INPUT statement, c includes the missing value. For an interval input, c represents consolidated values described in the “Split Search Algorithm” on page 502.

The Bonferroni factor κ depends on whether the input variable is nominal, and whether the MISSING=USEINSEARCH option is specified.

$$\kappa = \begin{cases} \sum_{i=0}^{B-1} (-1)^i \frac{(B-i)^c}{i!(B-i)!} & \text{for a nominal input} \\ \binom{c-1}{B-1} & \text{for non-nominal, without USEINSEARCH} \\ \frac{B-1+B(c-B)}{c-1} \binom{c-1}{B-1} & \text{for non-nominal, with USEINSEARCH} \end{cases}$$

The Bonferroni adjustment is described further in Kass (1980). Hawkins and Kass (1982) suggested bounding κp with a p -value for a more conservative test. Unless the PADJUST=NOGABRIEL is specified,

$$p = \min \left(\kappa Pr \left(\chi_{(B-1, J-1)}^2 > \chi^2 \right), Pr \left(\chi_{(c-1, J-1)}^2 > \chi^2 \right) \right)$$

where J is the number of target values.

Adjusting p -Values for the Depth of the Node

The PADJUST=DEPTH option in the PROC statement requests the recursive partitioning procedure to multiply the p -value by a depth factor to account for the probability of error in creating the current node. The unadjusted p -value estimates the probability that the observed association between the target values and the split of the data into subsets could happen by chance, given the existence of the current node. The depth adjustment attempts to incorporate the probability that the current node being split is a chance occurrence to begin with.

The depth factor for node τ is the product of the number of branches in each ancestor node:

$$Depth(\tau) = \prod_{\tau' \prec \tau} B(\tau')$$

Adjusting p -Values for the Number of Input Variables

The PVARs= m option in the PROC statement requests the recursive partitioning procedure to adjust the p -value to account for multiple significance tests with independent input variables. Let $M(root)$ denote the number of input variables, and $M(\tau)$ denote the number of input variables for which the recursive partitioning procedure searches for a splitting rule in a specific node. ($M(\tau)$ might be less than $M(root)$ because the partitioning procedure does not search on variables that are constant in τ , or on categorical variables that do not satisfy the MINCATSIZE= option in the TRAIN statement, or on variables that have been excluded in an ancestor node.) A recursive partitioning procedure multiplies the p -value by

$$\max \left((m/M(root)) M(\tau), 1 \right)$$

to adjust for the multiple tests on different input variables in the node. Specifying $m = 0$ requests the procedure to make no adjustment for the number of independent input variables.

Splitting Criteria for an Ordinal Target

To evaluate splitting rules for an ordinal target, a recursive partitioning procedure uses loss coefficients A_{jk} defining the penalty of misclassifying target value j as k . The coefficients are the same as the ones in the decision matrix, if one is specified in DECADATA= option in the DECISION statement. For an ordinal target, the decision matrix must have type LOSS, the decision alternatives must equal the target values, and A_{jk} must be ≥ 0 . By default, $A_{jk} = |k - j|$.

A recursive partitioning procedure always incorporates A_{jk} into the node impurity measure in the splitting criteria for an ordinal target. Let $\hat{k}(\tau)$ denote a target value in node τ minimizing the loss,

$$\sum_j A_{jk} p_j$$

For CRITERION=ENTROPY, define the impurity measure,

$$i(\tau) = - \sum_{j=1}^J \left(A_{j\hat{k}(\tau)} + 1 \right) p_j \log_2 p_j$$

For CRITERION=GINI, define the impurity measure,

$$i(\tau) = \sum_{j=1}^J \left(A_{j\hat{k}(\tau)} + 1 \right) p_j (1 - p_j)$$

Missing Values

If the value of the target variable is missing, the observation is excluded from training and evaluating the tree.

If the value of an input variable X is missing, the MISSING= option in the INPUT statement that declares X determines how a partitioning procedure treats the observation. If the option is omitted from the INPUT statement, then the MISSING= option in the PROC statement determines the policy for X . If the option is omitted from the PROC statement also, then MISSING=USEINSEARCH is assumed for X .

Specify MISSING=USEINSEARCH to incorporate missing values in the calculation of the worth of a splitting rule, and consequently to produce a splitting rule that associates missing values with a branch that maximizes the worth of the split. For a nominal input variable, a new nominal category representing missing values is created for the duration of the split search. For an ordinal or interval input variable, a rule preserves the ordering of the nonmissing values when assigning them to branches, but might assign missing values to any single branch. Specifying MISSING=USEINSEARCH can produce a branch exclusively for missing values. This is desirable when the existence of a missing value is predictive of a target value.

If the MISSING=BIGBRANCH, DISTRIBUTE, or SMALLRESIDUAL option is specified for X and X is missing, the observation is excluded from the search for a split on X.

If MISSING= SMALLRESIDUAL, the rule uses the branch with the smallest residual sum of squares among observations in the within-node training sample with missing values of X. For a categorical target, the residual sum of squares is

$$\sum_{i=1}^{N_{\text{missing}}} \sum_{j=1}^J (\delta_{ij} - p_j(\text{nonmissing}))^2$$

where the outer sum is over observations with missing values of X, δ_{ij} equals 1 if observation i has target value j , and equals 0 otherwise, and $p_j(\text{nonmissing})$ is the within node probability of target value j based on observations with nonmissing X in the within-node training sample and assigned to the branch. When prior probabilities are not specified, $p_j(\text{nonmissing})$ is the proportion of such observations with target value j . Otherwise, $p_j(\text{nonmissing})$ incorporates the prior probabilities (and never incorporates profit or loss coefficients) using the formula described in “Posterior and Within Node Probabilities” on page 492.

If MISSING= SMALLRESIDUAL or USEINSEARCH and no missing values occur in the within-node training sample for X, then the splitting rule assigns missing values to the branch with the most observations in the within-node sample, as if MISSING= BIGBRANCH were specified. If more than one branch has this same maximum number of observations, then the missing values are assigned to the first such branch. Assigning observations to the largest branch does not help create homogeneous branches. But some branch must be assigned in order for the rule to handle missing values in the future (when applied to observations not in the training data). The MISSING=BIGBRANCH policy is the least harmful one possible without any information about the association of missing values with the target.

When a rule is applied to an observation, and the rule requires an input variable whose value is missing or an unrecognized category, surrogate rules are considered before the MISSING= option is. A surrogate rule is a backup to the main splitting rule. For example, the main rule might use variable CITY and the surrogate might use variable REGION. If CITY is missing and REGION is not missing, the surrogate is used. If REGION is also missing, then the next surrogate is considered.

If none of the surrogates can be applied to the observation, then the MISSING= option for the splitting variable governs what happens to the observation. If MISSING=USEINSEARCH and no surrogates are applicable, the observation is assigned to the branch for missing values specified in the splitting rule. If MISSING=DISTRIBUTE, the observation is in effect copied, one copy for each branch. The copy assigned to a branch is given a fractional frequency proportional to the number of training observations assigned to the branch. The CODE statement cannot handle rules with MISSING=DISTRIBUTE.

Unseen Categorical Values

A splitting rule using a categorical variable might not recognize all possible values of the variable. Some categories might not have been in the training data. Others might have been so infrequent in the within-node training sample that the partitioning procedure excluded them. The MINCATSIZE= option in the TRAIN statement specifies the minimum number of occurrences required for a categorical value to participate in the search for a splitting rule. Splitting rules treat unseen categorical values as they would missing values.

Within Node Training Sample

The search for a splitting rule is based on a sample of the training data assigned to the node. The `NODESIZE= n` option in the `PERFORMANCE` statement specifies the number of observations to use in the sample. The procedure counts and samples the observations in a node without adjusting for values of the variable specified in the `FREQ` statement, if any. If the count is larger than n , then the split search for that node is based on a random sample of size n .

For a categorical target variable, the sample uses as many observations as possible in each category. Some categories might occur infrequently enough so that all the observations are in the sample. Let J_{rare} denote the number of these categories, and let n_{rare} denote the total number of observations in the node with these infrequent categories. $J - J_{rare}$ is the number of remaining categories. The sampling algorithm selects

$$s = \frac{n - n_{rare}}{J - J_{rare}}$$

observations from each of the $J - J_{rare}$ remaining categories. If s is not an integer, then the sample will contain one more observation from some of $J - J_{rare}$ categories than others so that the total equals $n - n_{rare}$. The sampling algorithm depends only on the order of the observations in the training data and not on other random factors.

When the node is split into branches, all the observations are passed to the branches, and new samples are created in each branch as needed.

Split Search Algorithm

A partitioning procedure always selects a splitting rule with the largest worth among all the splits evaluated. The algorithm is simple. The details, however, seem complicated because of the many special situations.

The search for splitting and surrogate rules is done on the within-node training sample. The search involving a specific input variable eliminates observations with a missing input value unless the `MISSING=` option for that input equals `USEINSEARCH`. The search involving a specific categorical input variable eliminates observations whose input value occurs less frequently in the within-node sample than the threshold specified in the `MINCATSIZE=` option in the `TRAIN` statement.

The decision to eliminate an observation from the within-node sample is made independently for each input and for each node. An observation in which input `W` is missing and input `Z` is not missing is eliminated from the search using `W` but not from `Z`, unless `MISSING=USEINSEARCH`. An observation in which categorical input `X` has a value that is common in the root node but occurs infrequently in some branch is eliminated from searches in that branch but not the root node.

In most situations the algorithm sorts the observations. For interval and ordinal input variables, the algorithm sorts the input values, and arbitrarily puts observations with missing input values at the end. For nominal inputs with interval targets, the algorithm sorts the input categories by the average target value among the observations in the category. For nominal inputs and observations containing two categorical target values in the search sample, the algorithm sorts the input categories by the proportion of one of the target values. In the remaining situation, the algorithm does not sort.

The remaining situation consists of a nominal input and a categorical target with at least three different values in the sample presented to the algorithm.

After the algorithm sorts, if it sorts at all, the algorithm passes over the data evaluating every permissible binary split preserving the sort. A split between tied input

values or categories with the same average target value is not permitted, nor is a split that leaves fewer observations in a branch than specified in the LEAFSIZE= option in the TRAIN statement.

If the MAXBRANCH= option in the TRAIN statement specifies binary splits, then the search is finished; the best split evaluated is the best binary split. Otherwise the algorithm consolidates the observations into N bins, where N is less than or equal to the limit specified in the SEARCHBINS= option in the TRAIN statement. Observations collected into the same bin remain together during the search and are assigned to the same branch.

The consolidation algorithm uses the best of the binary split points already found to create the bins, subject to some constraints. If the input variable is interval or ordinal and missing values appear in the sample (and are therefore acceptable values), then one bin is always reserved exclusively for missing values. For an interval input X , if the number of distinct values of X is greater than the number specified in the INTERVALBINS= n option in the TRAIN statement, then the split points will be at least

$$\frac{\max(X) - \min(X)}{n + 1}$$

apart, where $\max(X)$ and $\min(X)$ are the maximum and minimum values of the input in the search sample. The algorithm makes as many bins as possible that satisfy these constraints.

Next the algorithm computes the number of candidate splits, including m -ary splits, where $2 \leq m \leq n$. If the number does not exceed the threshold specified in the EXHAUSTIVE= option in the TRAIN statement, then all candidate splits are evaluated. The one with the largest measure of worth is chosen. Otherwise, the algorithm uses a merge-and-shuffle heuristic approach to select which splits to evaluate.

The merge-and-shuffle algorithm first creates a branch for each bin. The algorithm then merges a pair of branches, and then merges another pair, and so on, until only two branches remain. To choose a pair, the algorithm evaluates the worth of splitting the merged candidate branch back into the original pair of branches, and selects a pair that defines the splitting rule with the smallest worth. After each merge, the algorithm reassigns a bin of observations to a different branch if the worth of the splitting rule increases. The algorithm continues the reassignment of single bins as long as the worth increases. The merge-and-shuffle algorithm evaluates many, but not all, permissible splitting rules, and chooses the one with the largest worth. The algorithm is heuristic because it does not guarantee that the best possible split is found.

The previous paragraphs describe the search when the algorithm sorts the observations. The algorithm does not sort when the input variable is nominal and the target variable is categorical with at least three categories in the sample being searched. For this situation, if the number of input categories occurring in the search sample is greater than the threshold specified in the SEARCHBINS= option, then categories with the largest entropy of target values are consolidated into one bin. The remaining $N - 1$ categories are assigned to the remaining $N - 1$ bins, one bin for each category. The rest of the search is similar to the one for n -ary splits of sorted observations. If the number of candidate splits does not exceed the threshold specified in the EXHAUSTIVE= option, then all candidate splits are evaluated. Otherwise the algorithm uses the merge-and-shuffle heuristic approach.

Every rule includes an assignment of missing values to one or all branches, as described in “Missing Values” on page 500.

Surrogate Splitting Rules

A surrogate splitting rule is a backup to the main splitting rule. For example, the main rule might use the variable CITY and the surrogate might use the variable REGION. If CITY is missing and REGION is not missing, the surrogate rule is applied to the observation.

The measure of agreement between a main splitting rule and a surrogate is the proportion of observations in the within-node training sample that the two rules assign to the same branch. The definition excludes observations with missing values or unseen categorical values of the variable used in the main splitting rule. However, remaining observations with missing or unseen values of the surrogate variable count as observations not assigned to the same branch. Therefore, an observation whose value is missing for the variable used in the surrogate rule but not the variable in the main rule diminishes the measure of agreement between the two rules.

The search for a surrogate rule treats infrequent categorical values as missing values. A categorical value is considered infrequent when it appears in fewer observations than the number specified in the MINCATSIZE= option. This policy does not diminish the agreement measure because the search for the main splitting rule also treats infrequent values as missing.

A recursive partitioning procedure discards a surrogate unless its agreement is larger than the proportion of nonmissing observations that the main rule assigns to any single branch. This ensures that a surrogate has a better agreement than the trivial rule that assigns all observations to the largest branch. The MAXSURROGATES= option in the PROC statement specifies the maximum number of surrogate rules to use with a splitting rule.

A partitioning procedure always finds a surrogate rule achieving the maximum possible agreement with the main splitting rule, except when the surrogate variable is an interval and the main rule creates more than two branches. A best surrogate is usually found even in this situation. The search begins by finding the best surrogate among binary splits, creating two intervals, and proceeds recursively by finding the best binary split of one of the new intervals.

Similarity and Dissimilarity of Pairs of Inputs

The SIMILARITY= option in the SAVE statement outputs a data set containing a *similarity* statistic S_{vw} for each pair of input variables, V, W, used in a primary or surrogate splitting rule in the model, T . The similarity depends on the *agreement* of two splitting rules: the proportion of observations that the surrogate rule s_v or s_w sends to the same branches as the primary rule does. If the agreement is one for every node in which either V or W is used in a primary split of τ , then S_{vw} equals one. Generally, S_{vw} ranges from zero to one, with larger values indicating greater similarity. If S_{vw} equals zero, then either V and W have random agreement or no surrogate rule between V and W was saved. In the PROC statement, specify a value for the MAXSURROGATES= option large enough to ensure that a surrogate rule is saved for all variables Z similar to a given variable V. (Specifying too large a value for the MAXSURROGATES= option can strain memory and performance.)

A formal definition of similarity is as follows. Let T_{vw} denote the set of nodes τ in T in which V and W are used in a primary or surrogate split of τ . If V and W are both surrogates, then their agreement with the primary split of τ must be at least 0.80 for τ to be in the set T_{vw} . Define the similarity of V and W in model T as,

$$S_{VW} = \frac{\sum_{\tau \in T_{vw}} p(\tau) a(s_V, \tau) a(s_W, \tau)}{\sum_{\tau \in T_{vw}} p(\tau)}$$

where s_v denotes the primary or surrogate splitting rule using v , and $a(s_v, \tau)$ is the measure of agreement for the rule using v in node τ :

$$a(s_v, \tau) = \begin{cases} 1 & \text{if } sv \text{ is the primary splitting rule} \\ \text{agreement} & \text{if } sv \text{ is the surrogate rule} \\ 0 & \text{otherwise} \end{cases}$$

The DISSIMILARITY= option to the SAVE statement outputs a data set containing a *dissimilarity* statistic D_{vw} for each pair of input variables, V, W, equal to one minus the similarity, $1 - S_{vw}$. The output is a data set with type DISTANCE with ID variable _VAR_, suitable for specifying in the DATA= option in the CLUSTER procedure.

Variable Importance

Each recursive partitioning procedure provides two approaches to evaluating the importance of a variable: split-based and observation-based. The split-based approach uses the reduction in the sum of squares from splitting a node, summing over all nodes. The observation-based approach uses the increase in a fit statistic due to making the observation values of a variable uninformative. The split-based approach measures the contribution to a model. The observation-based approach measures the contribution to prediction. A detailed explanation of each approach is given in separate sections below.

Measures of variable importance generally underestimate the importance of correlated variables. Two correlated variables can make a similar contribution to a model. The total contribution is usually divided between them, and neither variable acquires the rank it deserves. Eliminating either variable generally increases the contribution attributed to the other.

The split-based approach to variable importance fully credits both correlated variables when using surrogate rules. When the primary splitting rule uses one of two highly correlated variables, a surrogate splitting rule will use the other variable. Both variables get about the same credit for the reduction in residual sums of squares in the split of that node. The overall importance of correlated variables is about the same and in the correct relation to other variables.

The observation-based variable importance is misleading when some variables are correlated. Consider using the split-based importance with surrogates first to discover superfluous correlated variables and to eliminate them before relying on observation-based importance.

Split-Based Variable Importance

The split-based approach to evaluating the importance of a variable uses the idea that the reduction in the sum of squares due to the model can be expressed as a sum over all nodes of the reduction in the sum of squares due to splitting the data in the node. The credit for the reduction in a particular node goes to the variable used to split the node. (More than one variable gets credit when surrogate rules exist.) The formula for variable importance sums the credits over all splitting rules, scales the sums so that the largest sum is one, and takes a square-root to revert to linear units.

The split-based approach uses statistics already saved in a node and does not need to read the data again. The `IMPORTANCE=` option of the `SAVE` statement outputs the relative importance computed with the training data and again computed with the validation data. If the validation data indicates a much lower importance than the training data, then the variable is overfitting. The overfitting usually occurs in an individual node that uses the variable to split with. The validation statistics in the branches will differ substantially from the training data statistics in such a node.

A recursive partitioning procedure computes the split-based relative importance of input variable v in model T as

$$I(v; T) \propto \sqrt{\sum_{\tau \in T} a(s_v, \tau) \Delta SSE(\tau)}$$

where the sum is over nodes τ in T , and s_v denotes the primary or surrogate splitting rule using v .

$$a(s_v, \tau)$$

is the measure of agreement for the rule using v in node τ :

$$a(s_v, \tau) = \begin{cases} 1 & \text{if } sv \text{ is the primary splitting rule} \\ \text{agreement} & \text{if } sv \text{ is the surrogate rule} \\ 0 & \text{otherwise} \end{cases}$$

$\Delta SSE(\tau)$ is the reduction in sum of square errors from the predicted values:

$$\Delta SSE(\tau) = \max \left(SSE(\tau) - \sum_{b \in B(\tau)} SSE(\tau_b), 0 \right)$$

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

where

$B(\tau)$ = set of branches from τ

τ_b = child node of τ in branch b

$N(\tau)$ = number of observations in τ

$\hat{Y}(\tau)$ = average Y in training data in τ

$\delta_{ij} = 1$ if $Y_i = j$, 0 otherwise

$\hat{p}_j(\tau)$ = average δ_{ij} in training data in τ

For a categorical target, the formula for $SSE(\tau)$ reduces to

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

where p_j is the proportion of the validation data with target value j , and N , p_j , and \hat{p}_j are evaluated in node τ .

Note that the expression

$$SSE(\tau) - \sum_{b \in B(\tau)} SSE(\tau_b)$$

is always nonnegative with training data, but can be negative with validation data.

SAVE Statement IMPORTANCE= Output Data Set

The IMPORTANCE= option in the SAVE statement specifies the output data set to contain the split-based measure of relative importance for each input variable in the selected subtree. The ASSESS and SUBTREE statements determine the subtree. Each observation describes an input variable. The observations are in order of decreasing importance as computed with the training data.

The IMPORTANCE= data set contains the following variables:

- NAME — the input variable name
- LABEL — the input variable label
- NRULES — the number of splitting rules using this variable
- NSURROGATES — the number of surrogate rules using this variable
- IMPORTANCE — the relative importance computed with the training data
- V_IMPORTANCE — the relative importance computed with the validation data
- RATIO — the ratio of V_IMPORTANCE to IMPORTANCE, or missing if IMPORTANCE is less than 0.0001

The NSURROGATES variable is omitted unless surrogates are requested in the MAXSURROGATES= option in the TRAIN statement.

The V_IMPORTANCE and RATIO variables are omitted unless the VALIDATA= option appears in the ASSESS statement.

Observation-Based Variable Importance

The observation-based approach to evaluating the importance of a variable entails reading a data set and applying the model several times to each observation. It is applied first to compute the standard prediction of the observation, and then once for each variable or pairs of variables being evaluated to compute a prediction in which variables being evaluated are made uninformative. The difference between the standard prediction and the prediction using an uninformative rendering of a variable (or pair of variables) is a measure of the influence of the variable (or pair of variables) on prediction. A plot of all observations of the differences versus the value of the variable can show which values influence the prediction the most. The difference in a fit statistic computed first the standard way and then computed using the uninformative rendering of the variable is a measure of the overall importance of the variable for prediction.

To make a variable uninformative, the procedure replaces its value in a given observation with the empirical distribution of the variable, and replaces the standard prediction with the expected prediction integrated over the distribution. The process is similar to making several copies of a given observation, altering the values of the variable being evaluated, and then taking the average of the standard predictions of these copies. The choice of altered values follows the empirical distribution: Each value that appears in the DATA= data set also appears among the imaginary copies of the observation, and appears with the same frequency. Notice that the uninformative prediction of an observation depends on the other observations in the DATA= data set because of the empirical distribution.

When a splitting rule encounters an observation with a distributional value instead of a single value, the rule assigns the observation to all the branches, and assigns a fractional frequency to the observation in a specific branch equal to the amount of the distribution of the values that the rule would assign to the branch. The prediction in the child nodes is thus equal to the average prediction when many copies of the observation are made in which the values of the splitting variable are taken randomly from the given distribution.

The simple description of the method as taking an average of the prediction of duplicate observations works for a single split when the prediction is computed as an average of individual predictions, but not necessarily otherwise. Classification with gradient boosting machines, for example, transforms the node average, F , into $\exp(F)$ to compute a posterior probability. These probabilities are not averages of probabilities of many individual observations. Even in models such as a decision tree in which the node prediction is an average of individual predictions, successive splitting of distributional values along a single path to a leaf results in a fractional frequency which is the product of the fractional frequencies from the individual splits. This fraction generally differs from the proportion of duplicate observations assigned to the leaf, the duplicate observations having different single values for variables being split on. The same phenomenon occurs in a two-way contingency table: the product of a row proportion and a column proportion generally differs from the proportion of observations in the cell intersecting the row and column. The latter corresponds to the proportion of duplicate observations assigned to a particular leaf.

Some software applications use a similar approach to variable importance, but take a significant short-cut: Instead of replacing a variable value with the empirical distribution, they replace it with a single other value drawn at random from the empirical distribution.

The IMPORTANCE statement options DATA=, OUT=, and OUTFIT= specify the input data set, the output data set of predictions, and the output data set of fit statistics, respectively. For each observation in the DATA= data set, the OUT= data set contains an observation for each variable or pair of variables being evaluated, plus one more observation containing the standard prediction. For example, if two variables are evaluated, then the number of observations in the OUT= data set is three times the number of observations in the DATA= data set. The OUT= data set becomes very large when many variables are evaluated. Unlike the split-based approach to variable importance, which evaluates all variables, the observation-based approach evaluates only the variables requested with the VAR=, NVAR=, and N2WAY= options of the IMPORTANCE statement.

The variables in the OUT= data set are the same as in the data set specified in the OUT= option to the SCORE statement, plus one or two more: _INPUT1_ and _INPUT2_ contain the names of the variables being evaluated. The same is true of the OUTFIT= data set. _INPUT2_ only appears if a pair of variables is evaluated.

The first observation in the OUTFIT= data set is the same as that in the OUTFIT= data set in the SCORE statement. The OUTFIT= data set in the IMPORTANCE statement contains additional observations, one for each variable or pair of variables

being evaluated, in which the values are increases in goodness-of-fit statistics when the variable (or variable pair) becomes uninformative. For a specific fit statistic f , let f_X denote its value when variable X is made uninformative, f_{XY} its value when variables X and Y are both uninformative, and f_0 the value when no variables are uninformative. f_0 appears in the first observation of the OUTFIT= data set. The additional observations have values Δ_X or Δ_{XY} :

$$\Delta_X = f_X - f_0$$

$$\Delta_{XY} = f_{XY} - f_0 - \Delta_X - \Delta_Y$$

A positive Δ_X means the uninformative version of X produces a larger f than the original X does. A larger f generally indicates a poorer fit of the model to the data. Consequently, the larger Δ_X is, the larger the contribution of X is to the fit of the model to the data. Similarly, the larger Δ_{XY} is, the larger the contribution the pair of values (X, Y) makes to the fit of the model beyond the individual contributions of X and Y. Δ_{XY} therefore measures interactions between variables.

Partial Dependency Functions

The partial dependency of a model $F(x, y, z, \dots)$ on a variable X is an approximation of the model prediction expressed as a function $F_X(x)$. Plotting $F_X(x)$ versus X will show the dependence of F on X, provided F has little interaction between X and the other variables, Y, Z, Such interactions might be detected comparing different partial dependency functions.

Friedman (1999) introduced partial dependency and gives both a formulaic and an algorithmic definition. The formulaic definition allows a discussion motivating the use of partial dependency functions. Unfortunately, the formula is computationally impractical on many data sets. The algorithmic definition is sought to remedy this. While the algorithm is feasible, it does not always implement the formula.

Formulaic Definition and Additivity

The partial dependence of a function $F(x, y, z, \dots)$ on X is the expected value of $F(x, y, z, \dots)$ taken over all variables except X:

$$F_X(x) = E_{\setminus X}(F(x, y, z, \dots))$$

For a specific value $X = x$, the estimate of $F_X(x)$ is $\hat{F}_X(x)$, the average of the model predictions over the training data with X kept constant:

$$\hat{F}_X(x) = \sum_{i=1}^N F(X_i = x, Y_i, Z_i, \dots) / N$$

where Y_i, Z_i, \dots represent the input values of observation i , and $X_i = x$ indicates that the value of X is set to x in every observation. The partial dependency of a model on two variables X and Y is similar:

$$F_{XY}(x, y) = E_{\setminus XY}(F(x, y, z, \dots))$$

and is estimated by,

$$\hat{F}_{XY}(x, y) = \sum_{i=1}^N F(X_i = x, Y_i = y, Z_i, \dots) / N$$

When the model F is additive in X ,

$$F(x, y, z, \dots) = G(x) + H(y, z, \dots),$$

or multiplicative

$$F(x, y, z, \dots) = G(x)H(y, z, \dots)$$

for some functions G and H , then the partial dependency function F_X equals $G(x)$.

Interaction Detection

A model additive in variables X and Y has the form,

$$F(x, y, z, \dots) = G_X(X) + G_Y(y) + H(z, \dots)$$

for some functions G_X, G_Y , and H . The partial dependency function obeys the relation,

$$F_{XY}(x, y) = G_X(x) + G_Y(y) = F_X(x) + F_Y(y).$$

The relation fails when F depends on the interaction of X and Y . Friedman and Popescu (2005) define a statistic H_{XY} to measure the interaction between X and Y :

$$H_{XY}^2 = \sum_{i=1}^N \left[\hat{F}_{XY}(x, y) - \hat{F}_X(x) - \hat{F}_Y(y) \right]^2 / \sum_{i=1}^N \hat{F}_{XY}^2(x, y)$$

Algorithmic Definition

The algorithmic implementation of the partial dependency of X regards all variables except X as missing and uses the MISSING=DISTRIBUTE missing value policy. More explicitly, for a specific value $X = x$, $\hat{F}_X(x)$ equals a weighted average of the model

predictions in each leaf. The algorithm assigns weights to nodes recursively. The weight of the root node is one. For a node τ with an assigned weight, if τ is split on X , the branch into which x is assigned gets the same weight as τ , and the other branches get zero weight. If τ is split on a different variable, then the weight of a branch is the node weight $w(\tau)$ times the fraction of training observations assigned to that branch.

The algorithmic and formulaic definitions differ when X occurs in more than one splitting rule in a path with at least one intervening split on another variable. Suppose the root node τ splits on X and has branches τ_1 and τ_2 , with x assigned to τ_1 . Suppose τ_1 is split on some other variable Z , and has branches τ_{11} and τ_{12} . Observations with $X = x$ will appear in both branches. Finally suppose τ_{11} is split on X and has branches τ_{111} and τ_{112} , with x assigned to τ_{111} . Observations with $X = x$ will appear in leaves, τ_{12} and τ_{111} . The partial dependency function evaluated at $X = x$, $\hat{F}(x)$, depends on the weights assigned to these leaves. The weights, in turn, depend on the proportions of observations that the rule using Z splitting τ_1 assigns to the branches. The formulaic definition applies the rule to all observations in the training data. The algorithm only applies the rule to those observations that fall into τ_1 , ignoring the observations in τ_2 . The formulaic definition thus uses more observations, the observations in τ_2 , than the algorithm does to determine the weights in τ_{12} and τ_{111} . The weights will differ to the extent that the observations in τ_1 and τ_2 differ with respect to the splitting rule using Z .

A notable difference in the dependency functions is that the algorithmic definition produces one that generally does not equal an additive component $G(x)$ even when the model is additive:

$$F(x, y, z, \dots) = G(x) + H(y, z, \dots)$$

Using the formulaic definition, the dependency function $F_X(x)$ equals $G(x)$ for such an additive model.

The USEVARONCE option to the PROC statement ensures the variable will appear only in one splitting rule in a path. Consequently the two definitions will produce the same partial dependency functions.

NODESTATS= Output Data Set

The NODESTATS= option in the SAVE statement specifies the output data set to contain statistics for each node in the selected subtree. The ASSESS and SUBTREE statements determine the subtree. Each observation describes one node. The NODESTATS= data set contains the following variables:

- NODE — the ID of the node
- PARENT — the ID of the parent node, or missing if the node is the root
- BRANCH — an integer, beginning with 1, indicating which branch of this node is from the parent, or missing if the node is the root
- LEAF — an integer, beginning with 1, indicating the left-to-right position of the leaf in the tree, or missing if the node is not a leaf
- NBRANCHES — the number of branches emanating from this node, or 0 for leaf nodes
- DEPTH — the number of splits from the root node to this node
- TRAVERSAL — an integer indicating when this node appears in a depth-first, left-to-right traversal
- LINKWIDTH — a suggested width for displaying the line from the parent to this node

- LINKCOLOR — a suggested RGB color value for displaying the line from the parent to this node
- NODETEXT — a character value of a node statistic
- ABOVETEXT — a character value pertaining to the definition of the branch to this node
- BELOWTEXT — the name or label of the input variable used to split this node, or blank
- N — the number of training observations
- NPRIORS — the number N adjusted for prior probabilities
- VN — the number of validation observations
- VNPRIORS — the number VN adjusted for prior probabilities
- _RASE_ — the root average square error
- _VRASE_ — the root average square error based on validation data
- I_: D_: EL_: EP_: P_: U_: V_: — variables output in the OUT= option in the SCORE statement

The variables VN, VNPRIORS, and _VRASE_ occur only if validation data is specified. The variables NPRIORS and VNPRIORS occur only for categorical targets. The variables _RASE_ and _VRASE_ occur only for interval targets. The colon in a name expression such as, I_: refers to all variables whose name begins with, I_. “Variable Names and Conditions for Their Creation” on page 515 describes the variables output by the SCORE statement.

If no prior probabilities are specified in the DECISION statement, then N and NPRIORS are equal. NPRIORS times P_namej equals the number of training observations with categorical target value j , adjusted for prior probabilities. VNPRIORS times V_namej equals the number of validation observations with category j , adjusted for prior probabilities.

The number of training observations with target value j , not adjusted for prior probabilities, is

$$N_j = N \frac{P_{namej} N_j(\text{root}) / \pi_j}{\sum_i P_{namei} N_i(\text{root}) / \pi_i}$$

where $N_j(\text{root})$ is the number of observations in the root node with target value j , and π_j denotes the prior probability for j .

PATH= Output Data Set

The PATH= option in the SAVE statement specifies the output data set that describes the observations that the tree assigns to a node. The description consists of a set of relationships between variables and values. Observations that satisfy all the relations are assigned to the node.

The PATH= output data set describes the path to each leaf in the current subtree unless the NODES= option specifies which nodes to describe.

The PATH= data set contains the following variables:

- NODE — the ID of the node
- LEAF — the leaf number if the node is a leaf
- VARNAME — the name of the variable

- VARIABLE — the variable label, or name if no label
- RELATION — a character variable containing the relation that an observation value must have to be in the node
- CHARACTER_VALUE — the formatted value of the variable
- NUMERIC_VALUE — the numeric value of a numeric variable

Each observation contains a single variable value, unless the relation is ISMISSING or ISNOTMISSING. The relation ISMISSING indicates that missing values of the variable are accepted in the node. The relation ISNOTMISSING indicates that missing values of the variable are excluded from the node, all nonmissing values are accepted. If the relation is not ISMISSING or ISNOTMISSING, then the contents of the observation depend on the level of measurement of the variable.

For a nominal variable, CHARACTER_VALUE contains one formatted value of the variable, RELATION is “=”, and NUMERIC_VALUE is missing.

For an interval or ordinal variable, the path determines a range of values in the node. The upper end of the range can be infinite, or the lower end can be infinitely negative, but at least one end will be finite (otherwise RELATION would equal ISNOTMISSING). The first observation contains the lower end of the range, and the second contains the upper end. If an end is unbounded, CHARACTER_VALUE is blank and NUMERIC_VALUE is missing for that observation. Otherwise, for an interval variable, both CHARACTER_VALUE and NUMERIC_VALUE contain the end value, and RELATION contains “>=” or “<”.

For an ordinal variable, CHARACTER_VALUE contains the formatted value of an end, and NUMERIC_VALUE is missing. RELATION is “>=” or “<=”.

RULES= Output Data Set

The RULES= option in the SAVE statement specifies the output data set describing the splitting rules in each node, including surrogate rules, unused competing rules, and candidate rules in leaf nodes. The data set contains only nodes in the subtree determined by the ASSESS or SUBTREE statement.

The RULES= data set contains the following variables:

- NODE — the ID of the node
- ROLE — a character variable with four possible values:
 - PRIMARY for the primary splitting rule
 - COMPETITOR for a competing rule
 - SURROGATE for a surrogate rule
 - CANDIDATE for a candidate splitting rule in a leaf
- RANK — the rank among other rules with the same role
- STAT — a character variable containing the name of the statistic in the NUMERIC_VALUE or CHARACTER_VALUE variable.
- NUMERIC_VALUE, the numeric value of the statistic, if any
- CHARACTER_VALUE, the character value of the statistic, if any

A single rule is described using several observations. The STAT variable determines what an observation describes. The following table summarizes the possible values of STAT.

Statistics

STAT	NUMERIC_VALUE	CHARACTER_VALUE
VARIABLE		Variable name
LABEL		Variable label
MISSING	Branch	"MISSING VALUES ONLY" or blank
WORTH	worth, or $-\log_{10}(p)$	
AGREEMENT	agreement	
BRANCHES	number of branches	
DECIMALS	decimals of precision	
INTERVAL	interval split value	
NOMINAL	branch	formatted category value
ORDINAL	branch	formatted category value

Interval Split

For an interval input, STAT equals INTERVAL once for each but the last branch containing nonmissing values. The value of NUMERIC_VALUE equals the split value. Successive occurrences of INTERVAL have increasing split values.

The splitting rule assigns observations with a value less than the first split value to the first branch, values greater or equal to the first split value and less than the second split value to the second branch, and so on. The rule assigns observations with a value greater than all split values to the last branch containing nonmissing values.

When STAT equals DECIMALS, NUMERIC_VALUE equals the precision specified in the INTERVALDECIMALS option to the PROC and INPUT statements. It applies only to INTERVAL inputs.

If the rule assigns missing values to a separate branch, CHARACTER_VALUE equals MISSING VALUES ONLY in the row in which STAT equals MISSING, and the number of split values equals the number of branches minus two. Otherwise, the number of split values equals one. No split value appears for a binary split in which the second branch is exclusively for observations with missing values.

Nominal Split

For a nominal input, STAT equals NOMINAL once for each formatted category explicitly assigned to some branch. The value of CHARACTER_VALUE is the formatted category, and the value of NUMERIC_VALUE is the branch number. Categories not explicitly mentioned are assigned to the same branch as missing values. Notice that a branch reserved for missing values only can have observations with nonmissing values in it when these values are not explicitly included in the splitting rule.

Ordinal Split

Specification of an ordinal rule is similar to an interval rule. For an ordinal input, STAT equals ORDINAL once for each but the last branch containing nonmissing values. The value of CHARACTER_VALUE equals the split category. Successive occurrences of ORDINAL have increasing split categories. The value of NUMERIC_VALUE contains a

branch number. The splitting rule assigns observations with ordinal values less than the first split category to the branch appearing in NUMERIC_VALUE. This is always the first branch for primary and competing rules, but not necessarily for surrogate rules. The rule assigns observations with values greater or equal to the first split category and less than the second split category to the branch appearing in NUMERIC_VALUE with the second split category. For primary and competing rules, this is the second branch. In all other respects, rules for ordinal inputs are like those for interval inputs. The value ORDINAL will not appear for binary splits in which the second branch is exclusively for missing values.

SCORE Statement OUT= Output Data Set

The OUT= option in the SCORE statement creates a data set by appending new variables to the data set specified in the DATA= option. Which new variables appear depends on other options in the SCORE statement, the level of measurement of the target variable, and whether a profit or loss function is specified in the DECISION statement.

Variable Names and Conditions for Their Creation

The names of all the possible new variables are listed in the following table.

New Variables in the OUT= Data Set

Variable	Description	Target	Other
Variables for Prediction			
F_name	actual formatted target category	yes	
I_name	predicted formatted target category	no	
P_namevalue	predicted value	no	
R_namevalue	residual from the prediction	yes	
U_name	predicted unformatted target category	no	
V_namevalue	predicted value computed with validation data	no	
WARN	indications of problems with the prediction	no	
Variables for Decisions			DECDATA= type
BL_name	best possible loss from any decision	yes	LOSS
BP_name_	best possible profit from any decision	yes	PROFIT, REVENUE
CL_name_	loss computed from the target value	yes	LOSS

Variable	Description	Target	Other
CP_name_	profit computed from the target value	yes	PROFIT, REVENUE
D_name_	label of the chosen decision alternative	no	any
EL_name_	expected loss from the chosen decision	no	LOSS
EP_name_	expected profit from the chosen decision	no	PROFIT, REVENUE
IC_name_	investment cost	no	REVENUE
ROI_name_	return on investment	yes	REVENUE
Variables for Leaf Assignment			Option
i	proportion of the observation in leaf i	no	DUMMY
LEAF	leaf identification number	no	LEAF
NODE	node identification number	no	LEAF

The names of most of these variables incorporate the name of the target variable. For a categorical target variable, namevalue represents the name of the target concatenated with a formatted target value. For example, a categorical target variable named Response, with values '0' and '1', will generate new variables, P_Response0 and P_Response1. For an interval target, namevalue represents the name of the target. For example, an interval target variable, Sales, will generate the variable P_Sales.

The NOPREDICTION option to the SCORE statement suppresses the creation of the prediction and decision variables. Otherwise, the conditions necessary for creating these variables are as follows. Variables P_namevalue and _WARN_ are always created. Variables I_name and U_name appear when the target is categorical. When ROLE=TRAIN, VALID, or TEST, the DATA= data set must contain the target variable, and the OUT= data set will contain R_namevalue and, for a categorical target, F_name. The V_namevalue variable is created if validation data was used during the creation of the tree.

When decision alternatives are specified in the DECVAR= option in the DECISION statement, the variable D_name_ is created, as is either EL_name_ or EP_name_ depending on whether the type of the DECDATA= data set is LOSS or PROFIT, respectively. If the type is REVENUE, then variables IC_name_ and ROI_name_ are also created. When ROLE=TRAIN, VALID, or TEST, either the variables BL_name_ and CL_name_, or the variables BP_name_ and CP_name_, are created.

Decision Variables

The labels of the variables specified in the DECVAR= option in the DECISION statement are the names of the decision alternatives. For a variable without a label, the name of the decision alternative is the name of the variable. The variable D_name_ in the OUT= data set contains the name of the decision alternative assigned to the observation.

Leaf Assignment Variables

Each node is uniquely identified with a positive integer. Once an identification number is assigned to a node, the number is never reassigned to another node, even after the node is pruned. Consequently, most subtrees in the subtree sequence will not have consecutive node identifiers.

Each leaf has a leaf identification number in addition to the node identifier. The leaf identifiers range from 1 to the number of leaves. The leaf numbers are reassigned whenever a new subtree is selected from the subtree sequence.

For an observation in the OUT= data set assigned to a single leaf, the variables `_NODE_` and `_LEAF_` contain the node and leaf identification numbers, respectively. For an observation assigned to more than one leaf, the variables `_NODE_` and `_LEAF_` contain missing values. An observation is assigned to more than one leaf when the observation is missing a value required by one of the splitting rules, and the `MISSING=DISTRIBUTE` option in the INPUT statement for the required variable dictates that the observation be distributed among the branches.

The `DUMMY` option in the SCORE statement specifies that the OUT= data set contain numeric variables `_i_` for integers `i` ranging from 1 to the number of leaves. The value of `_i_` equals the proportion of the observation assigned to the leaf with leaf identification number `i`. The sum of these variables equals one for each observation. Unless the `MISSING=DISTRIBUTE` option is specified in some INPUT statement or in the PROC statement, exactly one of the variables `_i_` equals one, and the rest are zero. When the `MISSING=DISTRIBUTE` option is specified, observations are distributed over more than one leaf, and `_i_` equals the proportion of the observation assigned to the leaf `i`.

SCORE Statement OUTFIT= Output Data Set

The `OUTFIT=` option in the SCORE statement creates a data set of statistics measuring how well the model fits the data specified in the `DATA=` option of the SCORE statement. The recursive partitioning procedures compute the statistics listed in the following table. The predictive modeling procedures in Enterprise Miner compute most of them also, facilitating model comparison. All these procedures follow the convention of modifying the name when the `ROLE=` option to the SCORE statement is `VALID` or `TEST`: `_X_` becomes `_VX_` if `ROLE=VALID`, or `_TX_` if `ROLE=TEST`.

Variables in the OUTFIT= Data Set

Variable	Description
Variables for Prediction	
<code>_ALOSS_</code>	average loss
<code>_APROF_</code>	average profit
<code>_ASE_</code>	average square error
<code>_DFT_</code>	<code>_NOBS_</code> times (number of target categories - 1)
<code>_DIV_</code>	<code>_SUMW_</code>
<code>_LOSS_</code>	total loss
<code>_NOBS_</code>	sum of frequencies of the observations
<code>_NW_</code>	Number of leaves in the model
<code>_MAX_</code>	maximum error
<code>_MISC_</code>	misclassification rate

Variable	Description
<code>_RASE_</code>	square root of the average square error
<code>_PROF_</code>	total profit
<code>_PASE_</code>	average square error using prior probabilities
<code>_PMISC_</code>	misclassification rate using prior probabilities
<code>_SSE_</code>	sum of square errors
<code>_SUMW_</code>	<code>NOBS_</code> times number of target categories

`_PROF_` and `_APROF_` appear only when a profit or revenue function is specified in the `DECISION` statement. Similarly, `_LOSS_` and `_ALOSS_` appear only with a loss function. `_MISC_` occurs only with a categorical target, and `_PASE_` and `_PMISC_` occur only when prior probabilities are present.

The partitioning procedures that create a family of models can output the fit statistics for every model using a `SAVE` statement option: `SEQUENCE=` for `ARBOR`, `FIT=` for `TREEBOOST`.

Performance Considerations

When the partitioning procedure begins, it reserves memory in the computer for the calculations necessary for growing the tree. Later the procedure will read the entire training data and perform as many tasks as the reserved memory can accommodate, postponing other tasks for a subsequent pass of the data. Typically, the procedure spends most of its time accessing the data, and therefore reducing the number of passes of the data will also reduce the execution time.

Passes Over the Data

Each of the following tasks for a node require a pass over the entire training data:

- compute node statistics
- search for a split on an input variable
- determine a rule for missing values for a specified split
- search for a surrogate rule on an input variable

If only one task were done at a time, the number of passes over the training data would equal approximately the number of nodes times the number of input variables. Surrogate splits would require more passes. The number of additional passes equals the number of inputs minus one. The actual number is typically less for three reasons. First, if no split on an input variable is found in a node, then no search is attempted on that input in any descendent node. See the description of the `MAXRULES=` option in the `TRAIN` statement for some situations in which no split exists on an input. Second, the procedure does not search for any splits in nodes at the depth specified in the `MAXDEPTH=` option in the `TRAIN` statement. Third, given sufficient memory, the procedure can perform several tasks during the same pass.

The procedure computes node statistics before beginning a split search in that node. Consequently, creating a node and finding a split requires at least two passes of the data. The procedure will search for a split in a node on every input variable in one pass of the data if enough memory is available. The search for surrogate splits begins after establishing the primary splitting rule. Consequently, creating a node, finding a split,

and finding surrogate splits requires at least three passes of the data. A separate search for a rule for missing values (and hence a separate pass) is necessary only for splits that are defined in the SPLIT statement and for which the rule for missing values is omitted. If the rule for missing values is present in the SPLIT statement, no pass is needed for a split search in the node for any input.

The number of bytes needed for each search task is approximately equal to the within-node sample size specified in the NODESIZE= option in the PERFORMANCE statement, times 3, times the number of bytes in a double word, which is 8 on most computers.

Memory Considerations

Reserving more memory can reduce the number of data passes, but cannot reduce the execution time if a large proportion of the memory is virtual memory swapped to disk. A computer operating system allocates more memory to software programs running on the system than is physically available. When the operating system detects that no program is using an allocated section of *physical* memory, the system copies the contents of the section to disk, an action commonly called *swapping-out*, and then reassigns the section to satisfy another request for memory. When the program that created the original contents tries to access it, the operating system finds another dormant section of physical memory, swaps that section to disk, and swaps the original contents to the new section of physical memory. The programs appear to have access to more memory than physically exists. The apparent amount of memory is called *virtual* memory.

By default, the partitioning procedure estimates the amount of memory it will need for tree construction tasks, asks the operating system how much physical memory is available, and then allocates just enough to perform the tasks, or all of physical memory, whichever is smaller. The estimate of the amount of memory assumes that all split searches in a node are done in the same pass. The MEMSIZE= option to the PERFORMANCE statement overrides the default process. The SAS MEMSIZE option sets limits on the amount of physical memory available to the tasks.

References

- Buntine, W. 1992. "Learning Classification Trees." *Statistics and Computing* 2, 63–73.
- Breiman, L. 1996. "Bagging Predictors." *Machine Learning* 24, 123–140.
- Breiman, L., et al. 1984. *Classification and Regression Trees*. Belmont, CA: Wadsworth, Inc.
- Friedman, J.H. 1999. "Greedy Function Approximation: A Gradient Boosting Machine." Available at <http://www-stat.stanford.edu/jhf>.
- Friedman, J.H. 1999. "Stochastic Gradient Boosting." Available at <http://wwwstat.stanford.edu/jhf>.
- Friedman, J.H. and B.E. Popescu. 2005. "Predictive Learning via Rule Ensembles." Available at <http://www-stat.stanford.edu/jhf>.

Hawkins, D.M., ed. 1982. *Topics in Applied Multivariate Analysis*. Cambridge: Cambridge University Press.

Hawkins, D.M. and G.V. Kass. 1982. "Automatic Interaction Detection." in Hawkins (1982).

Kass, G.V. 1980. "An Exploratory Technique for Investigating Large Quantities of Categorical Data." *Applied Statistics* 29, 119–127.

Kohavi, R., and Kunz, C. 1997. "Option Decision Trees with Majority Votes." *Machine Learning: Proceedings of the Fourteenth International Conference*, San Francisco: Morgan Kaufmann.

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

Quinlan, R.J. 1993. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, Inc.