4 • The ARBORETUM Procedure
The ARBORETUM Procedure

Overview

A decision tree is a type of predictive model developed independently in the statistics and artificial intelligence communities. A tree partitions large amounts of data into segments called terminal nodes or leaves. The data in a leaf determine estimates of the value of a target variable, the dependent variable to be predicted. These estimates are subsequently applied to predict the target of a new observation assigned to the leaf. The ARBORETUM procedure searches for partitions that fit the training data, the data used to compute the estimates. If these estimates fit new data well, the tree is said to generalize well. Good generalization is the primary goal for predictive tasks. A tree might fit the training data well but generalize poorly.

Decision trees may also help prepare data for other predictive models by suggesting which variables to use, suggesting interactions of variables, and providing segments for stratified modeling.

Trees are popular because they seem easy to use and understand. They seem easy to use because trees accept interval, ordinal, and nominal variables and tolerate missing values well. No understanding of statistical distributions is necessary because tree construction relies on frequencies of values in the training data set. Trees seem easy to understand because small trees clearly depict how a few variables characterize a target variable.

However, trees have shortcomings. Small trees, though easy to understand, are too simplistic to represent complex relationships in data, and large trees have many partitions that, collectively, may be difficult to comprehend. Trees require a lot of data to discover complex relationships, and even to fit a simple linear relationship well. Trees are therefore less efficient and less intuitive than a simple regression when a linear relationship exists. Even when a tree provides a simple and accurate description, other equally simple and accurate descriptions may exist. The tree would then give the false impression that certain variables uniquely explain the variations in the target values, whereas different variables would suggest a different interpretation that might generalize just as well.

Note: this document describes syntax for the ARBORETUM procedure in SAS 9.1. The syntax is likely to change and be incompatible in future releases.
The ARBORETUM procedure uses recursive partitioning to create a decision tree. 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, thereby mixing 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 descendents.

The ARBORETUM 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. It is sometimes called the primary splitting rule when discussing alternative partitions of the same node. A competing rule refers to a partition considered on an input variable other than the one used in the primary rule. A leaf has no primary or competing rule, but may have a candidate rule ready for splitting the leaf. A surrogate rule is one chosen to emulate a primary rule, and is used when the primary rule cannot be used, most commonly when an observation is missing a value for the primary input variable.

The ARBORETUM procedure searches for a splitting rule that maximizes an association between the target variable and the node partitions. The splitting criterion defines the measure of worth of the rule.

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 measurement level of the variable is the property of being nominal, ordinal, or interval.

The ARBORETUM procedure uses normalized, formatted values of categorical variables, and considers two categorical values the same if the normalized values are the same. Normalization removes any leading blank spaces from a value, converts lower case characters to upper case, 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 PROC ARBORETUM 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.

The relative proportions of categorical target values in the training data may differ from the proportions in the data to which the tree will be applied. Estimates of these latter proportions should be specified with prior probabilities when the tree is created. If the prior probabilities are the same as the proportions of the target values in the training data, then the predicted probabilities for an observation, also called the posterior probabilities, equal the proportions of the target values in the training data in the leaf to which the observation is assigned. If the prior probabilities differ from the training data proportions, then the posterior probabilities will also.
For an interval target variable, the prediction of an observation is the average of the target values in the training data in the leaf to which the tree assigns the observation.

For one method of handling missing values, and in some nonstandard models based on recursive partitioning, an observation may be assigned to more than one leaf. For a categorical target, the posterior probabilities for the observation are a weighted average of the posterior probabilities associated with each leaf. For an interval target, the prediction is a weighted average of the predictions in each leaf.

Assigning a branch to an observation is often called a decision, and hence the term, decision tree. Unfortunately, the terms decision and decision tree have different meanings in the closely related discipline of decision theory. In decision theory, a decision refers to the decision alternative whose utility or profit function is largest for a given probability distribution of outcomes. The ARBORETUM procedure adopts this definition, and will assign a decision to each observation when decision alternatives and a profit or loss function are specified in the DECISION statement.

**Basic Features**

The ARBORETUM procedure provides the ability to mix tree-construction strategies advocated by Kass (CHAID) (1980) and by Breiman, Friedman, Olshen, and Stone (1984) to match the needs of the situation. It extends the p-value adjustments of Kass and the retrospective pruning and misclassification costs of Breiman et al.

The basic features of the ARBORETUM procedure include

- Nominal, ordinal, and interval input and target variables
- Several splitting criteria
  - Variance reduction for interval targets
  - F-test for interval targets
  - Gini or entropy reduction (Information Gain) for categorical targets
  - CHAID for nominal targets
- Binary or n-ary splits, for fixed or unspecified n
- Several missing values policies:
  - Use missing values in the split search
  - Assign missing values to most correlated branch
  - Distribute missing observations over all branches
- Surrogate rules for missing values and variable importance
- Cost-complexity pruning and reduced-error pruning with validation data
- Prior probabilities optionally used in training or assessment
- Misclassification cost matrix incorporating new decision alternatives
- Incorporation of nominal decision matrix in the split criterion
- Interactive training mode for specifying splits and nodes to prune
Variable importance computed separately with training and validation data
• Generation of SAS DATA step code with an indicator variable for each leaf
• Generation of PMML

Enterprise Miner Tree Desktop Application

The SAS Enterprise Miner Tree Desktop Application is a Microsoft Windows application enabling a user to

• view the results from the ARBORETUM procedure
• modify the tree created with the ARBORETUM procedure
• create a new tree

The Desktop Application is highly interactive, containing many tables and views that may be independently arranged. Clicking on a variable, node, or subtree in one view automatically selects corresponding items in others. The tree may print to a single page or across multiple pages.

The Desktop Application for SAS 9.1 runs on Windows NT, 2000, and XP. It may be executed on its own, or launched from Enterprise Miner 4.3 and 5.1. It is automatically installed with Enterprise Miner 4.3.

Getting Started

This section presents a simple example in three parts to introduce the syntax of the ARBORETUM procedure. The first part runs the procedure with a minimum number of statements. The procedure creates a sequence of an increasingly complicated subtrees. The second part of the example prints an assessment of each subtree, and selects one that is different from the one the procedure selected. The third part illustrates how to explicitly change a splitting rule. The section begins with an explanation of how the procedure statements may alternate between a training phase and an assessment and output phase.

Running the ARBORETUM Procedure

The ARBORETUM procedure runs in two or three phases:

• initialization
• interactive training (optional)
• model assessment and output

The initialization statements specify the training data, variable roles, and other options that may not be set more than once. The interactive training statements are optional. They allow complete control of the creation and modification of splitting
rules and of the deletion of nodes. The model assessment and output statements create a sequence of subtrees, evaluate and select subtrees, apply the predictions to new data, and save model estimates in SAS data sets.

The ARBORETUM procedure executes statements as soon as they are submitted. The initialization statements must appear first. If interactive training is intended, those statements would typically come next, followed by assessment and output statements. Interactive training statements may follow model assessment and output statements, which in turn may be repeated after interactive training.

Interactive training begins with an INTERACT statement. 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 ARBORETUM statement, the ARBORETUM procedure will automatically create a tree.

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

A Brief Example

The following SAS code creates and saves a decision tree:

```sas
proc arboretum data=sashelp.shoes;
  target sales;
  input region subsidiary product stores;
  save summary=sum1
    sequence=seq1
    model=tree1;
run;
proc print data=sum1 label;
```

The PROC ARBORETUM statement invokes the procedure. The DATA= option specifies the training data to be the SHOES data set that exists in the SAS library, SASHELP. The TARGET statement specifies SALES as the target variable. The INPUT statement specifies input variables from the SHOES data set. No LEVEL= option appears in the INPUT statement, and consequently the ARBORETUM procedure assumes that the character variables, REGION, SUBSIDIARY, and PRODUCT, have a nominal level of measurement, and the numeric variable, STORES, has an interval level of measurement.

The ARBORETUM procedure does not use the SAS Output Delivery System. Instead, it saves results in SAS data sets that may be printed, or, in the case of the MODEL= data set, may be input into a subsequent PROC ARBORETUM statement, eliminating the need to respecify the data set and variables, or may be input into the Enterprise Miner Tree Desktop Application to continue the analysis or just explore the results graphically.
The SA VE statement specifies the output data sets. The SUMMARY= option outputs summary statistics to the data set SUM1. Figure 1 shows the result of printing the SUM1 data set. The sum of square errors produces an $R$-square of 0.57.

![Table](image)

<table>
<thead>
<tr>
<th>Obs</th>
<th>STATISTIC</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N</td>
<td>395.00</td>
</tr>
<tr>
<td>2</td>
<td>AVERAGE</td>
<td>85700.17</td>
</tr>
<tr>
<td>3</td>
<td>AVE SQ ERR</td>
<td>7123521155.08</td>
</tr>
<tr>
<td>4</td>
<td>R SQUARED</td>
<td>0.57</td>
</tr>
</tbody>
</table>

**Figure 1.** Summary Statistics

### Selecting a Subtree

The SEQUENCE= option in the SAVE statement creates a SAS data set with statistics for subtrees of every possible size. The number of leaves is stored in variable `_NW_`. The variable `_NW_` is created in other SAS data mining procedures to represent the complexity of a model. `_NW_` is an abbreviation for *number of weights* in a neural network.

Figure 2 shows the output from printing the number of leaves and the assessment measure from the SEQUENCE= data set. The default assessment measure for an interval target is the average square error. The first observation shows the average square error in the training data, before applying the tree. The first five observations show that the average square error decreases quickly as the number of leaves in the subtree increases from 1 to 5. The error decreases more slowly as the number of leaves increases from 6 to 16.

```sas
proc print data=seq1 label;
  var _NW_ _ASSESS_
run;
```
Selecting a Subtree

![Table]

<table>
<thead>
<tr>
<th>Obs</th>
<th>Subtree</th>
<th>Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>16626478669.3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>13780468465.0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>9872157361.2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>9025518133.9</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>8288990878.8</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7633053644.4</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7427061309.9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>7238580477.6</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>7201752194.9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>717052669.6</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>7139810808.1</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>7132483315.7</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>7127223465.7</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>7124198471.6</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>7123655825.3</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>7123521155.1</td>
</tr>
</tbody>
</table>

Figure 2. Subtree Sequence

The following PROC ARBORETUM code selects the subtree with 5 leaves and saves the node statistics and splitting rules in SAS data sets:

```sas
proc arboretum inmodel=tree1;
  subtree nleaves=5;
  save model=tree2 summary=sum2
    nodestats=nodes2 rules=rules2;
run;
proc print data=sum2 label;
```

The INMODEL= option imports the information saved from the previous execution of the ARBORETUM procedure, eliminating the need to respecify the training data set or the variables or to re-create the tree. The SUBTREE statement selects the subtree with five leaves. The NODESTATS= option in the SAVE statement saves information about each node into data set NODES2. The RULES= option in the SAVE statement saves all the splitting rules into data set RULES2.

Figure 3 shows the contents of the SUM2 data set:

![Table]

<table>
<thead>
<tr>
<th>Obs</th>
<th>STATISTIC</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N</td>
<td>395.00</td>
</tr>
<tr>
<td>2</td>
<td>AVERAGE</td>
<td>85700.17</td>
</tr>
<tr>
<td>3</td>
<td>AVE SQ ERR</td>
<td>8288990878.80</td>
</tr>
<tr>
<td>4</td>
<td>R SQUARED</td>
<td>0.50</td>
</tr>
</tbody>
</table>

Figure 3. Summary Statistics for Subtree 5
The $R$-square for the subtree with 5 leaves is 0.50, compared to 0.57 for the tree with 16 leaves.

The NODES2 data set contains information about each node. The variable P_SALES contains the predicted sales amount for observations in the node. In this example, P_SALES equals the average sales among observations in the SHOES data set assigned to the node. The WHERE statement in the following code selects the leaf nodes for printing, excluding the nonterminal nodes. Figure 4 shows the output listing.

```sas
proc print data=nodes2;
  var node leaf n p_sales;
  where leaf ne .;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>NODE</th>
<th>LEAF</th>
<th>N</th>
<th>P_Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>202</td>
<td>33554.14</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>2</td>
<td>86</td>
<td>41364.47</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>3</td>
<td>38</td>
<td>146430.61</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td>4</td>
<td>16</td>
<td>386879.94</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>5</td>
<td>53</td>
<td>221921.57</td>
</tr>
</tbody>
</table>

Figure 4. Information about Each Leaf

The RULES2 data set contains all the splitting rules in the tree, including the unused competing rules, and the candidate rules in the leaves. The WHERE statement in the following code selects the primary rules for printing, and Figure 5 shows the output listing:

```sas
proc print data=rules2;
  where role = 'PRIMARY';
```
Figure 5. Primary Splitting Rules

Node 1 represents the entire training data. The rows in the listing for which column NODE is 1 describe the rule for partitioning node 1. The STAT column determines the type of information in the row. Rows with STAT equal to ‘VARIABLE’ and ‘LABEL’ identify the name and label of the input variable used by the splitting rule. In this example, the splitting rule is based on Number of Stores, creates two branches (STAT equals ‘BRANCHES’), and assigns data observations with Stores < 10.5 (STAT equals ‘INTERVAL’) to the first branch.

The splitting rule for node 3 assigns to the first branch observations in which Product equals ‘BOOT’, ‘SANDAL’, ‘SLIPPER’, or ‘SPORT SHOE’. In node 7, ‘MEN’S CASUAL’ are assigned to the first branch, and ‘MEN’S DRESS’, ‘WOMEN’S CASUAL’, and ‘WOMEN’S DRESS’ to the second branch.
Changing a Splitting Rule

The following PROC ARBORETUM code changes the rule for node 7 to assign ‘MEN’S DRESS’ to the first branch along with ‘MEN’S CASUAL’:

```plaintext
proc arboretum data=sashelp.shoes inmodel=tree2 ;
   interact pruned;
   prune node=7;
   split node=7 var=product /
      "Men's casual" "Men's dress",
      "Women's casual" "Women's dress"
   ;
   save summary=sum3
      node=7
      rules=rules3
   ;
run;
proc print data=rules3;
   where role = 'PRIMARY';
run;
```

The INTERACT statement declares the start of interactive training statements to modify the tree. The PRUNED option instructs the ARBORETUM procedure to use the five-leaf subtree and to delete the nodes that only occur in larger subtrees. The PRUNE statement removes the branches from node 7, converting node 7 to a leaf. The SPLIT statement specifies the new splitting rule for node 7. The values of Product appear after the slash (‘/’) character. The comma separates values assigned to different branches. The NODE= option to the SAVE statement modifies the RULES= option to only save the rules in node 7. Figure 6 shows the primary splitting rule saved using the RULES= option in the SAVE statement. The SUM3 data set (not shown) contains the $R^2$-square for the modified tree. It is 0.46, less than the $R^2$-square for the unmodified tree.

```
Obs NODE ROLE RANK STAT NUMERIC VALUE CHARACTER VALUE
   1 7 PRIMARY   1 VARIABLE   . Product
   2 7 PRIMARY   1 MISSING   1
   3 7 PRIMARY   1 BRANCHES  2
   4 7 PRIMARY   1 NOMINAL  1 MEN’S CASUAL
   5 7 PRIMARY   1 NOMINAL  1 MEN’S DRESS
   6 7 PRIMARY   1 NOMINAL  2 WOMEN’S CASUAL
   7 7 PRIMARY   1 NOMINAL  2 WOMEN’S DRESS
```

Figure 6. Splitting Rule for Node 7
Syntax

The following statements are available in PROC ARBORETUM:

PROC ARBORETUM < options >;
  DECISION DECDATA=SAS-data-set < options > ;
  FREQ variable ;
  INPUT variables < / options > ;
  TARGET variable < / options > ;
  PERFORMANCE < options > ;
  INTERACT < subtree > ;
  BRANCH < options > ;
  PRUNE NODES=list < options > ;
  SEARCH < options > ;
  SETRULE NODE=id VAR=variable < options > ;
  SPLIT < options > ;
  TRAIN < options > ;
  UNDO ;
  REDO ;
  ASSESS < options > ;
  CODE < options > ;
  DESCRIBE < options > ;
  MAKEMACRO NLEAVES=macname ;
  SAVE < options > ;
  SCORE < options > ;
  SUBTREE subtree ;

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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECISION</td>
<td>specify profits and prior probabilities</td>
</tr>
<tr>
<td>FREQ</td>
<td>specify a frequency variable</td>
</tr>
<tr>
<td>INPUT</td>
<td>specify input variables with common options</td>
</tr>
<tr>
<td>TARGET</td>
<td>specify the target variable</td>
</tr>
<tr>
<td>PERFORMANCE</td>
<td>specify memory size and where to locate data</td>
</tr>
<tr>
<td>INTERACT</td>
<td>declare start of interactive training</td>
</tr>
<tr>
<td>BRANCH</td>
<td>create branches from candidate splitting rules</td>
</tr>
<tr>
<td>PRUNE</td>
<td>prune the descendents of a node</td>
</tr>
<tr>
<td>SEARCH</td>
<td>search for candidate splitting rules</td>
</tr>
<tr>
<td>SETRULE</td>
<td>specify a candidate splitting rule</td>
</tr>
<tr>
<td>SPLIT</td>
<td>search for a splitting rule and create branches</td>
</tr>
<tr>
<td>TRAIN</td>
<td>find splitting rules and branch recursively</td>
</tr>
<tr>
<td>UNDO</td>
<td>undo the previous interactive training operation</td>
</tr>
<tr>
<td>REDO</td>
<td>redo the action the previous UNDO statement undid</td>
</tr>
<tr>
<td>ASSESS</td>
<td>evaluate subtrees and declare beginning of results</td>
</tr>
</tbody>
</table>
The ARBORETUM Procedure

Table 1. (continued)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
<td>generate SAS DATA step code for scoring new cases</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>print description of rules defining each leaf</td>
</tr>
<tr>
<td>MAKEMACRO</td>
<td>define a macro variable</td>
</tr>
<tr>
<td>SAVE</td>
<td>output data sets containing model results</td>
</tr>
<tr>
<td>SCORE</td>
<td>use the model to make predictions on new data</td>
</tr>
<tr>
<td>SUBTREE</td>
<td>specify which subtree to use</td>
</tr>
</tbody>
</table>

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

PROC ARBORETUM Statement

PROC ARBORETUM < options > ;

The PROC ARBORETUM statement starts the ARBORETUM procedure. Either the DATA= option or the INMODEL= option must appear. The DATA= option must appear to begin or resume training a model. The INMODEL= option specifies a previously saved model. Any option available in the TRAIN statement is available here also.

CRITERION=name

specifies the criterion for evaluating candidate splitting rules. Table 2 summarizes the criteria available for each level of measurement of the target variable.

Table 2. Split Search Criteria

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

The default criterion is PROBF for an interval target, PROBCHISQ for a nominal target, and GINI for an ordinal target. See the “Splitting Criteria” section beginning on page 38 for more information.

DATA= SAS-data-set

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

**DECSEARCH**

specifies that the split search should incorporate the profit or loss function specified in the DECISION statement. See the “Incorporating Decisions, Profit, and Loss” section on page 38 for more information. The DECSEARCH option only works with a categorical target.

**INMODEL= SAS-data-set**

names a data set created from the SAVE MODEL= option, or saved from the Enterprise Miner Tree Desktop Application. 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 tree (assuming the saved name of the training data is still valid).

**MISSING= policy**

specifies how a splitting rule handles an observation with missing values. Table 3 lists the available policies.

**Table 3. Missing Value Policies**

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGBRANCH</td>
<td>assign the observation to the largest branch</td>
</tr>
<tr>
<td>DISTRIBUTE</td>
<td>assign the observation to each branch with a fractional frequency proportional to the number of training observations in the branch</td>
</tr>
<tr>
<td>SMALLRESIDUAL</td>
<td>assign to the branch minimizing SSE among observations with missing values</td>
</tr>
<tr>
<td>USEINSEARCH</td>
<td>use missing values during the split search (default)</td>
</tr>
</tbody>
</table>

The default policy is USEINSEARCH. The MISSING= option in the INPUT statement assigns a policy to the variables listed in the statement, and supersedes the MISSING= option to the PROC ARBORETUM statement. See the “INPUT Statement” section on page 25.

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. Using the CODE statement for a tree containing a rule with MISSING=DISTRIBUTE is an error. See the “Missing Values” section on page 45 for a complete description of the missing value options.

**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.
The ARBORETUM Procedure

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

Specifying both CHAIDAFTER and CHAIDBEFORE is an error. Specifying NONE with any other method is an error. If the PADJUST= option is not specified, the CHAIDBEFORE and DEPTH methods are used. The PADJUST= option is ignored unless CRITERION= PROBCHISQ or PROBF. See the “Adjusting p-Values for the Number of Input Values and Branches” section on page 43 for more information.

PRIORSSEARCH requests that the prior probabilities defined in the DECISION statement be incorporated in the split search criterion for a categorical target. See the “Incorporating Prior Probabilities” section on page 37 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 ARBORETUM 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 TRAIN statement. Consequently, the ARBORETUM procedure may only search for rules using $m \leq N$ of the original $N$ input variables. The procedure will regard $\max((n/N)m, 1)$ of the $m$ variables as independent. See the “Adjusting p-Values for the Number of Input Variables” section on page 44 for more detail. The default number $n$ is 0, requesting no adjustment for the number of inputs.

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 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.

ASSESS Statement

ASSESS < options > ;

The ASSESS statement specifies a measure for evaluating trees, evaluates all subtrees (with the original root), chooses a best one for each possible number of leaves, and organizes the chosen ones in a sequence, beginning with the subtree consisting of the root only, and ending with the largest tree consisting of all the nodes. (For assessment measures LIFT and LIFTPROFIT, the subtrees are evaluated with measures ASE and PROFIT, respectively. See the section “Tree Assessment and the Subtree Sequence” on page 49.)

The ARBORETUM procedure selects the best subtree in the sequence consistent with the options in the ASSESS statement. A subsequent SUBTREE statement can change
the selection. The output statements, CODE, DESCRIBE, SAVE, and SCORE, use the subtree selected in the most recent ASSESS or SUBTREE statement, and ignore nodes not in the selected subtree.

The ASSESS, SUBTREE, and output statements end initialization and interactive training. If the SUBTREE statement or an output statement is the first statement after initialization or interactive training, an ASSESS statement is implied. The procedure computes a new subtree sequence using the most recently specified ASSESS statement options, and selects the best subtree before executing the SUBTREE or output statement.

If the ASSESS, SUBTREE, or output statement immediately follows initialization so that no interactive training statements appear, and the tree contains no more than the root node, then these statements will create a tree. Otherwise, if the root node is already split using information imported using the INMODEL= option in the PROC statement, then further split searches will not occur unless explicitly requested with interactive training statements.

Table 4 summarizes the options available in the ASSESS statement. An option remains in effect in subsequent occurrences of the ASSESS statement unless explicitly specified differently.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT=</td>
<td>specifies categorical target value for LIFT</td>
</tr>
<tr>
<td>MEASURE=</td>
<td>specifies the assessment measure</td>
</tr>
<tr>
<td>NOPRIORS</td>
<td>ignores prior probabilities in subtree search</td>
</tr>
<tr>
<td>PROPORTION=</td>
<td>specifies proportion of observations for LIFT</td>
</tr>
<tr>
<td>PRUNEDATA=</td>
<td>specifies training or validation data for choosing subtrees</td>
</tr>
<tr>
<td>PRIORS</td>
<td>incorporates prior probabilities in subtree search</td>
</tr>
<tr>
<td>VALIDATA=</td>
<td>specifies validation data set</td>
</tr>
<tr>
<td>NOVALIDATA</td>
<td>terminates a previous VALIDATA= option</td>
</tr>
</tbody>
</table>

The following list describes these options. See the “Tree Assessment and the Subtree Sequence” section beginning on page 49 for more detail.

**EVENT= category**
specifies a formatted value of a categorical target to use with the LIFT assessment measure. If the EVENT= option is absent in one ASSESS statement, the last value specified in any ASSESS statement is used. If the EVENT= option has never been specified, the least frequent target value in the training data is used. The EVENT option is ignored with an interval target and with other assessment measures.

**MEASURE=PROFIT | ASE | MISC | LIFT | LIFTPROFIT**
specifies the assessment measure. Table 5 summarizes the available measures.
Table 5. Assessment Measures

<table>
<thead>
<tr>
<th>Measure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASE</td>
<td>Average square error</td>
</tr>
<tr>
<td>LIFT</td>
<td>Average or proportion among the highest ranked</td>
</tr>
<tr>
<td></td>
<td>observations</td>
</tr>
<tr>
<td>LIFTPROFIT</td>
<td>Average profit or loss among the highest ranked</td>
</tr>
<tr>
<td></td>
<td>observations</td>
</tr>
<tr>
<td>MISC</td>
<td>Proportion misclassified</td>
</tr>
<tr>
<td>PROFIT</td>
<td>Average profit or loss from the decision function</td>
</tr>
</tbody>
</table>

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

For an interval target, the LIFT measure is the average target value among observations predicted to have the highest average. The PROPORTION= option specifies the proportion of observations to use. For a categorical target, the LIFT measure is the proportion of observations with the target value specified in the EVENT= option among observations with the highest posterior probability of the EVENT= target value.

**NOPRIORS | PRIORS**

specifies whether to ignore prior probabilities when creating the sequence of subtrees. The default is NOPRIORS, ignoring prior probabilities. The section “Formulas for Assessment Measures” on page 50 describes how prior probabilities enter into the formulae for evaluating subtrees.

**PROPORTION=value**

The PROPORTION= option specifies the proportion of observations to use with the LIFT and LIFTPROFIT assessment measures. The PROPORTION= option is ignored unless LIFT or LIFTPROFIT is specified. The value must be between 0 and 1. If absent, the most recent value specified with LIFT or LIFTPROFIT is used. Requesting LIFT or LIFTPROFIT without ever specifying the PROPORTION= option is an error.

**PRUNEDATA= VALID | TRAIN**

specifies whether to use training or validation data when evaluating subtrees for inclusion in the subtree sequence. The default is VALID. If PRUNEDATA=VALID and validation data exists, then the subtree chosen for a given number of leaves is one with the best assessment value using the validation data.

**NOVALIDATA | VALIDATA= SAS-data-set**

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

**BRANCH Statement**

```
BRANCH < options > ;
```

The BRANCH statement is an interactive training statement that splits leaves into branches using the primary candidate splitting rule defined in the leaves. The
SETRULE and SEARCH statements create candidate rules. The PRUNE statement converts primary and competing rules to candidate rules when converting a node to a leaf. The BRANCH statement will not split a leaf without a candidate rule.

**NODES=nodeids**
 restricts the creation of branches to leaves descendent to nodes in the list of node identifiers, nodeids.

**ONE**
 restricts branching to the one leaf with the best candidate splitting rule. If a list of nodes is specified in the NODES= option, the ONE option only considers the leaves descendent to nodes in the list.

---

**CODE Statement**

```sas
CODE < options > ;
```

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 the section “SCORE Statement OUT= Output Data Set” on page 59. Using the CODE statement for a tree 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:

1. A quoted string, the value of which is the name (including the extension, if any) of the file to be opened.
2. 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.

**DUMMY | NODUMMY**
 requests creation of a dummy variable for each leaf node. The variables have names, _i_, for i = 1, 2, ..., L, where L is the number of leaves. The value of the dummy variable _i_ is 1 for observations assigned exclusively to leaf i, and 0 for observations not in leaf i. For observations distributed over more than one leaf, _i_ equals the proportion of the observation assigned the leaf i. The default is NODUMMY, suppressing the creation of dummy variables.
The ARBORETUM Procedure

**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. The default format is BEST20.

**LINESIZE | LS=** n
specifies the line size for generated code. The default is 72. The permissible range is 64 to 254.

**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. The variables are created by default.

**NOPREDICTION**
suppresses the code for computing predicted variables, such as P_: The default is PREDICTION, requesting such code.

**PMML**
requests XML output instead of SAS DATA step 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 table 9 in the section “Variable Names and Conditions for Their Creation” on page 59. 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. The default is NORESIDUAL, suppressing the generation of the DATA step code for these variables.

### DECISION Statement

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

The DECISION statement specifies decision functions and prior probabilities for categorical targets. The ARBORETUM 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 tree 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 species the data set type. If the DECDATA= data set has no type, the ARBORETUM 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.
DECISION Statement

**PROFIT**

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

\[ d_i = \text{argmax}_d E_i(d) \]

**REVENUE**

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

\[ d_i = \text{argmax}_d E_i(d) \]

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

**LOSS**

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

\[ d_i = \text{argmin}_d E_i(d) \]

The decision functions do not affect the creation of the tree unless the DECSEARCH option is specified in the PROC ARBORETUM statement. However, the decision functions determine a profit or loss measure for assessing trees, and consequently may greatly affect what nodes are pruned and omitted from the final subtree. See the “Tree Assessment and the Subtree Sequence” section on page 49 for more information about retrospective pruning.

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 DECVARS= option. The first cost in the list corresponds to the first alternative in the DECVARS= list, the second cost with the second alternative, and so on. The number of costs must equal the number of alternatives specified in the DEVCARS= 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 ARBORETUM statement. The value of this variable for observation \(i\) is assigned to \(C_{id}\). The ARBORETUM procedure does not recognize abbreviations of lists of variables in the COST= option. For example, D1-D3, ABC-ZYX, and PQR: are invalid representations of lists of variables.
The ARBORETUM Procedure

**DECDATA**=SAS-data-set

specifies the input data set containing the decision coefficients \( A_{jd} \) and prior probabilities. The DECDATA= 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 ARBORETUM statement.

**DECVARS**=decision-alternatives

specifies the variables in the DECDATA= 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 DECDATA= data set that contains the prior probabilities of categorical target values. The “Terminology” section on page 6 defines prior probabilities. \( Pvar \) must have nonnegative numeric values. The ARBORETUM 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 tree unless the PRIORSSPLIT option to the PROC ARBORETUM statement is specified. Prior probabilities affect the posterior probabilities, and consequently affect the model predictions and assessment.

---

**DESCRIBE Statement**

```
DESCRIBE < options > ;
```

The DESCRIBE statement causes the ARBORETUM procedure to output a simple description of the rules that define each leaf, along with a few statistics. The description is much easier to understand than the equivalent information output using the CODE statement.

The options to the DESCRIBE statement have the same form and function as those in the CODE statement.

**CATALOG**= catname | **FILE**= filename

specifies where to output the description. See the “CODE Statement” section on page 21 for more information.

**FORMAT**= format

specifies the format to use in the description for numeric values that do not have a format from the input data set. The default format is BEST20.

**LINESIZE** | **LS**= n

specifies the line size for description. The default is 72. The permissible range is 64 to 254.
FREQ Statement

FREQ variable ;

The FREQ statement identifies a variable that contains the frequency of occurrence of each observation. The ARBORETUM 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 $n$ may 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 LEAFSIZE=, MINCATSIZE=, and SPLITSIZE= options in the TRAIN 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.

INPUT Statement

INPUT variables < / options > ;

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

LEVEL= INTERVAL | NOMINAL | ORDINAL

specifies the level of measurement, as defined in the “Terminology” section on page 6. 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 ARBORETUM 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 ARBORETUM statement applies to these variables.

ORDER= ASCENDING
ORDER= ASCFORMATTED
ORDER= DESCENDING
ORDER= DESFORMATTED
ORDER= DSORDER

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

<table>
<thead>
<tr>
<th>Value of ORDER=</th>
<th>Variable Values Sorted By</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCENDING</td>
<td>ascending order of unformatted values (default)</td>
</tr>
<tr>
<td>ASCFORMATTED</td>
<td>ascending order of formatted values</td>
</tr>
<tr>
<td>DESCENDING</td>
<td>descending order of unformatted values</td>
</tr>
<tr>
<td>DESFORMATTED</td>
<td>descending order of formatted values</td>
</tr>
<tr>
<td>DSORDER</td>
<td>order of appearance in the input data set</td>
</tr>
</tbody>
</table>
The “Terminology” section on page 6 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, and 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 have the same formatted value, the ARBORETUM 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, may be machine dependent. For more information on sorting order, see the chapter on the SORT procedure in the *SAS Procedures Guide*. The default sorting order is 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 ARBORETUM statement. The SPLITATDATUM option is the alternative.

**INTERACT Statement**

```
INTERACT  PRUNED | LARGEST | NLEAVES= nleaves ;
```

The INTERACT statement declares the start of interactive training statements. If more than one node exists in the largest subtree, then one of the options, PRUNED, LARGEST, or NLEAVES= must appear to specify which subtree to use. Nodes not in the specified subtree are permanently deleted. See the “Tree Assessment and the Subtree Sequence” section beginning on page 49 for more information about the subtree sequence.

The INTERACT statement is required before using any of the interactive statements: BRANCH, PRUNE, REDO, SEARCH, SETRULE, SPLIT, TRAIN, or UNDO. If an INTERACT statement appears before any assessment or output statement, then no training is performed before the INTERACT statement executes, and no split is created unless requested with an interactive statement (or unless a tree is already input using the INMODEL= option to the PROC statement).

**PRUNED** permanently deletes all nodes not in the selected subtree.

**LARGEST** maintains all the nodes in the largest tree.
**NLEAVES** = \( n_{leaves} \)

selects the subtree with \( n_{leaves} \) leaves.

---

**MAKEMACRO Statement**

```
MAKEMACRO NLEAVES=macname ;
```

The MAKEMACRO statement specifies the name of a macro variable to contain the number of leaves in the current subtree.

---

**PERFORMANCE Statement**

```
PERFORMANCE < options > ;
```

The PERFORMANCE statement specifies options affecting the speed of computations with little or no impact on the results. See the “Performance Considerations” section on page 53 for more information.

**DISK | MULTIPASS | RAM**

specifies where to put the working copy of the training data. The RAM option 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 option requests that the working copy be stored in a disk utility file. Storing the copy on disk may free a considerable amount of memory for calculations, possibly speeding up the program. The MULTIPASS option requests that the training data be read multiple times instead of copying it to memory or a disk utility file. MULTIPASS is slower than DISK because the DISK copy is converted to encodings directly usable in the calculations. The MULTIPASS option is only preferable when the training data will not fit in RAM or in a disk utility file.

**MEMSIZE** = \( \text{bytes} \)

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

The SAS MEMSIZE system option sets an upper limit to \( \text{bytes} \).

**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 may require more passes of the data, resulting in a longer running time. See the “Performance Considerations” section on page 53 for more detail.

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.
The default value is 10,000. See the “Within Node Training Sample” section on page 46 for more detail.

**PRUNE Statement**

```
PRUNE NODES=nodeids | LEAVES <KEEPRULES=n> < / DROPVARS=names>
```

The PRUNE statement is an interactive training statement that deletes all nodes descendant to any node in the list of node identifiers, `nodeids`. The splitting rules in the nodes remain available as candidate rules unless the `KEEPRULES=` or `DROPVARS=` option deletes them. A subsequent SEARCH, SPLIT, or TRAIN statement will use the remaining rules instead of performing a new search. The `NODES=LEAVES` option deletes no nodes, but deletes rules from the current leaves as the `KEEPRULES=` and `DROPVARS=` options specify.

**DROPVARS= names**
deletes rules based on any of the named input variables.

**KEEPRULES= n**
requests to keep only the top `n` ranked rules in nodes `nodeids`. By default, all the rules are kept. The PRUNE statement deletes any rules using input variables in the list `names` before before considering the `KEEPRULES=` option.

**REDO Statement**

```
REDO ;
```

The REDO statement is an interactive training statement that reverses a previous UNDO statement. These statements can work in series in that a series of REDO statements will reverse a series of UNDO statements. If a previous UNDO statement is followed by any statement other than an UNDO or REDO, then REDO does nothing.

**SAVE Statement**

```
SAVE < options > ;
```

The SAVE statement outputs tree information into SAS data sets. Unless otherwise stated, the information describes a subtree selected in the ASSESS or SUBTREE statement, which may omit nodes from the largest tree in the sequence.

**IMPORTANCE= SAS-data-set**
names the output data set to contain the variable importance. See the section “IMPORTANCE= Output Data Set” on page 54 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 the ARBORETUM procedure. The output data set may also be input to the Enterprise Miner Tree Desktop Application for a visual display. The section “Enterprise Miner Tree Desktop Application” on page 8 describes the Desktop Application.
**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 subtree.

**NODESTAT= SAS-data-set**
names the output data set to contain node information. See the section “NODESTATS= Output Data Set” on page 56 for more information.

**PATH= SAS-data-set**
names the output data set describing the path to nodes. See the section “PATH= Output Data Set” on page 57 for more information.

**RULES= SAS-data-set**
names the output data set describing the splitting rules. See the section “RULES= Output Data Set” on page 58 for more information.

**SEQUENCE= SAS-data-set**
names the output data set to contain statistics on each subtree in the sequence of subtrees. See the section “SEQUENCE= Output Data Set” on page 61 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 ( \( R^2 = 1 - \text{average squared error} / \text{sum of square errors from the prediction} \)).

---

**SCORE Statement**

```sas
SCORE <options> ;
```

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 may be repeated.

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

**DUMMY**
causes the OUT= data set to contain dummy variables, \( _i \), for \( i = 1, 2, ..., L \), where \( L \) is the number of leaves. The value of the dummy variable \( _i \) is 1 for observations assigned exclusively to leaf \( i \), and 0 for observations not in leaf \( i \). For observations distributed over more than one leaf, \( _i \) equals the proportion of the observation assigned the leaf \( i \). The default is NODUMMY, suppressing the creation of dummy variables.
The ARBORETUM Procedure

**NODES=**nodes
lists the nodes containing the observations to score. If an observation is not assigned
to any node in the list, it does not contribute to the fit statistics and is not output. The
default is to use all the observations.

**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. The
variables are created by default.

**NOPREDICTION**
suppresses the generation of prediction variables, such as P_*. The default is
PREDICTION, requesting prediction variables.

**OUT=**SAS-data-set
names the output data set to contain the scored data. If the OUT= option is absent,
the ARBORETUM procedure creates a data set name using the DATA\nconvention. Specify OUT=_NULL_ to avoid creating a scored data set. The “SCORE Statement
OUT= Output Data Set” section on page 59 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.

**SEARCH Statement**

```
SEARCH < options > ;
```

The search statement is an interactive training statement that searches for splitting
rules in leaves. It behaves like the TRAIN statement except no branches are formed.
The options for the SEARCH statement are the same as those for the TRAIN state-
ment.

**SETRULE Statement**

```
SETRULE NODE=id VAR=var < missing > < /var-values> ;
```

The SETRULE statement is an interactive training statement that specifies the pri-
mary candidate splitting rule for leaf node id. If the node already has a candidate rule
for the variable specified in the VAR= option, and the missing and var-values options
are omitted, then the candidate rule for the variable is set to the primary candidate
rule for the node. Otherwise, variable values must be assigned to branches using the
var-values option or the MISSONLY missing option. The SEARCH statement is use-
ful for finding good variable values for a splitting rule. The options in the SETRULE
statement are the same as in the SPLIT statement. Unlike the SPLIT statement, the
SETRULE statement does not search for a split, does not create branches, and re-
quires the VAR= option.
The SPLIT statement is an interactive training statement that specifies how to split a leaf node. Only `id` is required. The `VAR=` option specifies which input variable to use in the splitting rule and is required if any other option is specified. The `missing` option specifies which branch to assign missing values, and the `var-values` option specifies which branch to assign nonmissing values. The `missing` option requires the `var-values` option, and together they determine the number of branches.

`VAR=var` specifies which input variable to use in the splitting rule. If the `VAR=` option is omitted and the leaf contains a candidate split, then the ARBORETUM procedure will use the primary candidate rule to create branches. Otherwise, the procedure will search for a splitting rule and create the branches if a rule is found.

`missing` specifies which branch to assign an observation in which `var` is missing. The `missing` option may be one of the following:

- **MISSBRANCH=b** specifies branch `b` for missing values. If `b` is greater than the number of branches implied by the `var-values` option, then the last branch specified in the `var-values` option is used.
- **MISSDISTRIBUTE** specifies that observations with missing values be distributed over all branches. Using the `CODE` statement with a rule using `MISSDISTRIBUTE` is an error.
- **MISSONLY** reserves the last branch exclusively for missing values. The branch is added to the branches specified in the `var-values` option. If `var-values` is absent, a binary split is created, and observations with a nonmissing value of the variable are assigned to the first branch.

If the `var-values` option is omitted, the `missing` option is ignored.

If the `missing` option is omitted, then the ARBORETUM procedure will honor the `MISSING=` option in the INPUT statement for the variable. If `MISSING=USEINSEARCH` and the `var-values` option specifies the branches for nonmissing values, then the branch that creates the smallest residual square error among the observations with missing values in the within-node training sample is assigned to missing values as if `MISSING=SMALLRESIDUAL` were specified.

`var-values` specifies a list of values of the variable. The following Table 6 summarizes the form of the list appropriate for the different levels of measurement.

<table>
<thead>
<tr>
<th>Measurement Level</th>
<th>Form of List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominal</td>
<td>all values, using commas to separate branches</td>
</tr>
<tr>
<td>Ordinal</td>
<td>minimum branch values in increasing order</td>
</tr>
<tr>
<td>Interval</td>
<td>splitting values in increasing order</td>
</tr>
</tbody>
</table>
For nominal and ordinal variables, specify formatted values in quotes. For a nominal variable, specify a list of all values with a comma (‘,’) inserted to separate categories assigned to different branches. Categories appearing before the first comma are assigned to the first branch.

For an ordinal variable, specify the smallest value for each branch except the first branch. Only one value should appear for a binary split. Commas are prohibited.

For an interval variable, specify an increasing list of values for separating branches. A single value specifies a binary split. An observation with a value less than the first specified number is assigned to the first branch. An observation whose value equals the first number is assigned to the second branch. A list of \( n \) numbers specifies \( n+1 \) branches.

The `missing` and `var-values` options determine the number of branches, overriding the `MAXBRANCHES=` option in the `TRAIN` statement.

If the `var-values` option is not specified, a split is made on the candidate rule for `variable` stored in the leaf. If no candidate rule exists, the ARBORETUM procedure will search for a split using `variable` and create the branches if a split is found.

**SUBTREE Statement**

```plaintext
SUBTREE BEST | LARGEST | NLEAVES= nleaves ;
```

The `SUBTREE` statement selects a subtree from the sequence of subtrees. See the “Tree Assessment and the Subtree Sequence” section beginning on page 49 for more information.

- **BEST**
  - selects the smallest subtree with the best assessment value.

- **LARGEST**
  - selects the largest subtree. The largest subtree is the tree with all the nodes.

- **NLEAVES= nleaves**
  - selects the largest subtree with no more than \( nleaves \) leaves.

**TARGET Statement**

```plaintext
TARGET variable < / options > ;
```

The `TARGET` statement names the variable the model tries to predict. The “INPUT Statement” section (page 25) describes the `LEVEL` and `ORDER` options more completely.

- **LEVEL= INTERVAL | ORDINAL | NOMINAL | BINARY**
  - specifies the level of measurement of the target variable. The default is `INTERVAL` for a numeric variable, `NOMINAL` for a character variable.

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

specifies the ordering of the values of an ordinal target variable. The ORDER= option is only available 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.

---

**TRAIN Statement**

```
TRAIN < / options > ;
```

The TRAIN statement grows the tree by searching for splitting rules in leaves, applying the rules to create branches, and repeating the process in the newly formed leaves. Most options remain in effect for subsequent SEARCH, SPLIT, and TRAIN statements. The exceptions are MAXNEWDEPTH= and NODES=.

**ALPHA=** $p$

specifies a threshold $p$-value for the significance level of a candidate splitting rules, applicable for splitting criteria that depend on $p$-values, namely, CRITERION=PROBF and PROBCHISQ. The default value of $p$ is 0.20. For splitting criteria not based on $p$-values, the ARBORETUM procedure uses the value associated with the MINWORTH= option instead of $p$.

**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 the “Split Search Algorithm” section on page 47 for a more complete description. The default value of $n$ is 5,000.

**INTERVALBINS=** $n$

indirectly specifies the minimum allowable width between two successive candidate split points on an interval input. The width equals $(\max(x) - \min(x))/(n + 1)$, where $\max(x)$ and $\min(x)$ are the maximum and minimum of the input variable values in the within-node sample being searched. The width is computed separately for each input and each node. The INTERVALBINS= option may indirectly modify $p$-value adjustments. The search algorithm ignores the INTERVALBINS= option if the number of distinct input values in the node sample is less than $n$. The default value of $n$ is 100.

**LEAFSIZE=** $n$

specifies the smallest number of training observations a new branch may have. The default value equals the number of observations in the training data divided by 1,000, or 5, if 5 is larger, or 5,000, if 5,000 is smaller.

The number $n$ applies to the within-node training sample used during the split search, described in the “Within Node Training Sample” section on page 46. The LEAFSIZE= option does not use the values of the variable in the FREQ statement to adjust the count of observations in the leaf.
The ARBORETUM Procedure

**MAXBRANCH=n**

restricts the number of subsets a splitting rule can produce to \( n \) or fewer. Setting \( n \) to 2 will create a binary tree. Any integer from 2 through 50 is permitted. The default value of \( n \) is 2.

**MAXDEPTH=n | MAX**

specifies the maximum depth of a node that the TRAIN 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 TRAIN 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 BRANCH, SEARCH, and SPLIT statements will search for splitting rules and create branches in a leaf at any depth, regardless of the MAXDEPTH= and the MAXNEWDEPTH= options.

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 TRAIN statement finishes. The default value of \( n \) is six.

**MAXNEWDEPTH=n | MAX**

specifies the maximum number of new generations of nodes created from a leaf. Specify MAXNEWDEPTH=1 to create at most one split in a leaf. Other options such as the MAXDEPTH= option may prevent the creation of all \( n \) generations. Specify MAXNEWDEPTH=0 to specify other options to the TRAIN statement without performing a search for a splitting rule. The MAXNEWDEPTH=MAX option specifies \( n \) to be 50, the largest acceptable value, and also the default value. \( N \) is not retained from one TRAIN statement to the next, and will equal its default value of 50 unless explicitly changed even if it was changed in a previous TRAIN statement.

**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 ARBORETUM procedure to save all the available splitting rules for each node.

Saved rules may be displayed in results and may be output using the RULES option to the SAVE statement. Subsequent BRANCH, SEARCH, SETRULE, SPLIT, and TRAIN statements use candidate rules saved in leaves.

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 TRAIN statement. Other causes occur less often. For example, the MINCATSIZE= option in the TRAIN statement may prevent creation of a split on a categorical input \( X \) if few observations exist for any specific value of \( X \). As another example, the LEAFSIZE= option may 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 may contain fewer than \( n \) rules.

The amount of memory needed to save splitting rules, especially rules using nominal input variables with many values, may 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.) The default value of \( n \) is 3.

**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 the “Missing Values” section on page 45 for more information.

Surrogate rules enhance the importance of the variables they use. See the “IMPORTANCE= Output Data Set” section on page 54 for more detail.

**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. Refer to the “Missing Values” section (page 45) for more detail. The default value of \( n \) is 5.

**MINWORTH=**\( \text{worth} \)

specifies a threshold value for the worth of a candidate splitting rule, unless the CRITERION= option in the PROC ARBORETUM statement is specified as PROBCHISQ or PROBF. A candidate rule whose worth is less than \( \text{worth} \) is discarded. The default value is 0. When CRITERION=PROBCHISQ or PROBF, the MINWORTH= option is ignored and the ALPHA= option is used instead.

**NODES=**\( \text{nodelist} \)

specifies that training proceed from all leaves descendent from the nodes in the list \( \text{nodelist} \). By default, the training proceeds from all leaves.

**SEARCHBINS=**\( n \)

specifies the maximum number of consolidated input values to use in an exhaustive or heuristic split search. See the “Split Search Algorithm” section beginning on page 47 for a complete explanation. The default value of \( n \) is 15 times the value specified in the MAXBRANCH= option in the TRAIN or PROC ARBORETUM statement. If the MAXBRANCH= option is unspecified, the default value of the SEARCHBINS= option is 30.
The ARBORETUM Procedure

**SPLITSIZE=** *n*

specifies the requisite number of training observations a node must have for the ARBORETUM procedure to consider splitting it. By default, *n* is twice the value of the LEAFSIZE= option. For the LEAFSIZE=, MINCATSIZE=, and SPLITSIZE= options in the TRAIN 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.

**USEVARONCE**

specifies that no splitting rule will be based on an input variable used in a splitting rule of an ancestor node.

**UNDO Statement**

```
UNDO ;
```

The UNDO statement is an interactive training statement that undoes the most recent PRUNE, SETRULE, SPLIT, or TRAIN statement issued since the most recent INTERACT statement. The REDO statement may restore what the UNDO statement undoes.

**Details**

**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 may disregard the ordering and assign smaller values of the input to any branch.

Rules need not assign any training observations to a particular branch. The ARBORETUM procedure does not automatically generate such rules, but a user may specify them in the SETRULE or SPLIT interactive training statements.

**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, 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.
Incorporating Prior Probabilities

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 SPLITSEARCH option in the PROC ARBORETUM 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:

\[ p_j = \text{proportion}_j(\tau) = \frac{N_j(\tau)}{N(\tau)} \]

where \( N_j(\tau) \) is the number of training observations in \( \tau \) 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 \( \rho_j \) depends on whether \( p_j \) incorporates priors, profits, or losses. Table 7 defines \( \rho_j \) by type of quantity being incorporated.

**Table 7. \( \rho_j \) by Type of Incorporated Quantity**

<table>
<thead>
<tr>
<th>Incorporated Quantity</th>
<th>( \rho_j )</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nothing</td>
<td>( N_j(\text{root}) )</td>
<td>number of ( j ) observations in root</td>
</tr>
<tr>
<td>Prior Probabilities</td>
<td>( \pi_j )</td>
<td>prior probability</td>
</tr>
<tr>
<td>Profit or Loss</td>
<td>( \pi^a_j )</td>
<td>altered prior probability</td>
</tr>
</tbody>
</table>

**Incorporating Prior Probabilities**

The PRIORV AR 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 ARBORETUM 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, $\pi_j^a$, the altered prior probability defined below.

**Incorporating Decisions, Profit, and Loss**

The DECSEARCH option in the PROC ARBORETUM 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. The ARBORETUM 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 ARBORETUM 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 $\pi_j$ 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 $\pi_j^a$ changes by replacing $\pi_j$ with $N_j(\tau)/N(\tau)$, the simple proportion of observations having target value $j$:

$$\pi_j^a = \frac{a_jN_j(\tau)/N(\tau)}{\sum_i a_iN_i(\tau)/N(\tau)}$$

**Splitting Criteria**

The ARBORETUM procedure searches for rules that maximize the measure of worth associated with the splitting criterion specified in the CRITERION= option in the PROC ARBORETUM statement. Some measures are based on a node impurity measure, others on $p$-values of a statistical test. A $p$-value may 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 may incorporate prior probabilities. A measure for a nominal target may incorporate profit or loss functions, including unequal misclassification costs. A measure for ordinal targets must incorporate distances between target values. The ARBORETUM 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 \( \tau \) into \( B \) branches, creating nodes, \( \{ \tau_b : b = 1, 2, ..., B \} \). \( N(\tau) \) denotes the number of observations in node \( \tau \) used in the search for the rule \( s \).

**Reduction in Node Impurity**

The impurity \( i(\tau) \) of node \( \tau \) is a nonnegative number that equals zero if all observations in \( \tau \) have the same target value, and is large if the target values in \( \tau \) 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 \( \tau \), \( Y_i \) is the target value of observation \( i \), and \( \bar{Y} \) is the average of \( Y_i \) in \( \tau \).

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 \( \tau \), 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 measured 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 the split \( s \) defines, and \( p(\tau_b|\tau) \) is the proportion of observations in \( \tau \) assigned to branch \( b \).
**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. The ARBORETUM 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 using 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 \(\chi^2\) 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 \(\chi^2_\nu \geq \chi^2\), where \(\chi^2_\nu\) is a random variable from a \(\chi^2\) distribution with \(\nu = (B - 1)(J - 1)\) degrees of freedom.

The ARBORETUM 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 may depend on the branch, \(\tau_b\). The \(\chi^2\)-square test assumes that the difference between the actual and predicted number of observations for a given target value \(j\) in a given branch \(\tau_b\), \(N_j(\tau_b) - p(\tau_b|\tau)N_j(\tau)\), is normally distributed.

Normality is never checked in practice. Even if the distribution over all 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 \texttt{ALPHA=} option in the \texttt{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 \texttt{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 only computed 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 \( \alpha \) denote the significance level of a test; \( \alpha \) 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

\[
P(\text{one or more spurious splits}) = \sum_{k=1}^{m} (-1)^{k+1} \frac{m!}{k!(m-k)!} \alpha^k
\]

The Bonferroni approximation assumes that terms in \( \alpha^2 \) 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 \( \alpha \) equal to the \( p \)-value of the split. Let \( \kappa(\tau, v, b) \) denote the number of candidate splits of node \( \tau \) 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 \( \tau \), a split of node \( \tau \) 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 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, the ARBORETUM procedure may consolidate the remaining values, thereby reducing the number of candidate splits. See the “Split Search Algorithm” section beginning on page 47 for more information.

**Adjusting \( p \)-Values for Multiple Tests**

When specifying CRITERION=PROBF or CRITERION=PROBCHISQ, the ARBORETUM procedure may 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 ARBORETUM 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 \( \chi^2 \)-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 the chooses the one with the best adjusted or unadjusted \( p \)-value accordingly as PADJUST=CHAIDBEFORE is or is not specified.

If the PADJUST=CHAIDAFTER is specified, the final candidate split in the node for the variable is is multiplied by \( \kappa(\tau, v, b) \). If either the PVARS=n or PADJUST=DEPTH option is specified in the PROC ARBORETUM statement, the \( p \)-value is further multiplied by a factor to adjust for the number of variables or the depth of the node \( \tau \) 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 ARBORETUM statement, the candidate is discarded, and the procedure proposes no split of \( \tau \) 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 ARBORETUM procedure to multiply the \( p \)-value of the \( \chi^2 \) statistic computed for the PROBCHISQ criterion for a nominal target by a Bonferroni factor \( \kappa \) to adjust for using multiple significance tests. If \( \kappa p \) is larger than the \( p \)-value of an alternative conservative significance test called Gabriel’s, then Gabriel’s \( p \)-value is used instead of \( \kappa p \) 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” section beginning on page 47.

The Bonferroni factor \( \kappa \) 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 \( \kappa p \) with a \( p \)-value from a more conservative test. Unless the PADJUST=NOGABRIEL is specified,

\[
p = \min(\kappa Pr(\chi^2_{(B-1,J-1)} > \chi^2), Pr(\chi^2_{(c-1,J-1)} > \chi^2))
\]
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 ARBORETUM 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 $\tau$ is the product of the number of branches in each ancestor node:

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

**Adjusting $p$-Values for the Number of Input Variables**

The PVARS=m option in the PROC statement requests the ARBORETUM procedure to adjust the $p$-value to account for multiple significance tests with independent input variables. Let $M(\text{root})$ denote the number of input variables, and $M(\tau)$ denote the number of input variables for which the ARBORETUM procedure searches for a splitting rule in a specific node. ($M(\tau)$ may be less than $M(\text{root})$ because the ARBORETUM procedure does not search on variables that are constant in $\tau$, 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.) The ARBORETUM procedure multiplies the $p$-value by $\max((m/M(\text{root}))M(\tau), 1)$ 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, the ARBORETUM 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 DECDATA= 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 $\geq 0$. By default, $A_{jk} = |k - j|$.

The ARBORETUM 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 $\tau$ minimizing the loss, $\sum_j A_{jk}p_j$. For CRITERION=ENTROPY, define the impurity measure,

$$i(\tau) = -\sum_{j=1}^{J} (A_{j\hat{k}(\tau)} + 1)p_j \log_2 p_j$$
For CRITERION=GINI, define the impurity measure,

\[ i(\tau) = \sum_{j=1}^{J} (A_{jk(\tau)} + 1)p_j(1 - p_j) \]

### 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 the ARBORETUM procedure treats the observation. If the option is omitted from the INPUT statement, then the MISSING= option in the PROC ARBORETUM statement determines the policy for \( X \). If the option is omitted from 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 may assign missing values to any single branch. Specifying MISSING=USEINSEARCH may 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 \( X \), \( \delta_{ij} \) 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 the “Posterior and Within Node Probabilities” section beginning on page 36.
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), and 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 ARBORETUM 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 if 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 \( J_{\text{rare}} \) denote the number of these categories, and let \( n_{\text{rare}} \) denote the total number of observations in the node with these infrequent
categories. \( J - J_{\text{rare}} \) is the number of remaining categories. The sampling algorithm selects 
\[ s = \frac{(n - n_{\text{rare}})}{(J - J_{\text{rare}})} \]
observations from each of the \( J - J_{\text{rare}} \) remaining categories. If \( s \) is not an integer, then the sample will contain one more observation from some of \( J - J_{\text{rare}} \) categories than others so that the total equals \( (n - n_{\text{rare}}) \).

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

The ARBORETUM 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.

In 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= option in the TRAIN statement, then the split points will be at least \( (\max(X) - \min(X))/(n+1) \) 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, and 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, and 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 the “Missing Values” section on page 45.
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.

The ARBORETUM 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.

The ARBORETUM 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.

Tree Assessment and the Subtree Sequence

The ASSESS statement declares an assessment measure with which to evaluate trees. The ARBORETUM procedure anticipates creating more nodes than are worthwhile, and therefore considers subtrees obtainable by choosing which nodes to be leaves and then deleting their descendents. For every feasible number of leaves, the procedure chooses a subtree with the best sequencing assessment measure (discussed in the next paragraph), and then organizes these subtrees into a sequence, beginning with the subtree consisting only of the original root node, and ending with the largest tree consisting of all the nodes.

The sequencing assessment measure is the same as the assessment measure in the ASSESS statement except when the latter is LIFT or LIFTPROFIT, in which case the sequencing measure is ASE or PROFIT, respectively. The ARBORETUM procedure only uses the sequencing measure to create the sequence of subtrees.
Retrospective Pruning

The ASSESS statement and the SUBTREE statement select one subtree in the sequence. Tree results output from the CODE, DESCRIBE, SAVE, and SCORE statements reflect the selected subtree, and ignore any nodes not in the subtree.

This strategy of intentionally creating more nodes than will be used is called retrospective pruning, and seeks a subtree that generalizes well, that is, a subtree that predicts better on new data than the tree with all the nodes would. A primary purpose of developing a predictive model after all is said and done is to make a prediction on an observation for which the target value is not already known. Predicting validation data, observations withheld from training, is worthwhile for estimating how well the model will do when predicting observations with unknown target value, but is not itself an important objective because the target values in the validation data are already known.

Splitting rules are usually more effective on the training data than on new data; the target values of training data are more homogeneous within a branch and more different between branches than the target values of validation data. The effect accumulates. A plot of the assessment value of each subtree in the sequence versus the number of leaves portrays the growth of unfounded optimism as an increasing gap between the curve computed with the training data and the curve computed with the validation data. Larger trees may overfit, producing more errors when applied to new data than would occur with a smaller tree. Proponents of retrospective pruning contend that the right number of nodes for a tree cannot be known while the nodes are being created.

Retrospective pruning originated with cost–complexity pruning, described in Breiman et al. (1984). Cost-complexity pruning uses the training data to find a nested sequence of subtrees, each of which evaluates best among other subtrees with the same number of leaves. Nesting requires each subtree in the sequence to be a subtree of all larger subtrees in the sequence. A best subtree with three leaves might not be a subtree of a best subtree with four leaves. Consequently, a nested sequence of best subtrees may have to omit subtrees for some specific numbers of leaves. The nesting restriction, however, makes possible a fast method of finding the subtrees. The ARBORETUM procedure foregoes the nesting restriction to find a best subtree for every possible number of leaves. The PRUNEDATA=TRAIN option in the ASSESS statement chooses the same subtrees for the sequence that cost–complexity pruning would find, as well as non-nested subtrees for all the tree sizes that cost–complexity pruning would omit.

Retrospective pruning based entirely on validation data was introduced with reduced–error pruning in Quinlan (1987). Reduced–error pruning finds the subtree that is best for a validation data set. It creates no sequence of subtrees. The PRUNEDATA=VALID option in the ASSESS statement finds the same best subtree and provides an entire sequence.

Formulas for Assessment Measures

All assessment measures are of the form

$$\sum_{\tau \in \Lambda} \omega(\tau, \chi) \lambda(\tau, \chi) \psi(\tau, \chi)$$
where \( \Lambda \) denotes the set of leaves, \( \chi \) indicates either \textit{training} or \textit{validation} data, \( \omega(\tau, \chi) \) is a weight for the node \( \tau \), \( \lambda(\tau, \chi) \) is an inclusion function for cumulative lift measures, and \( \psi(\tau, \chi) \) is a node statistic.

The node weight, \( \omega(\tau, \chi) \), equals the proportion of observations in data set \( \chi \) in \( \tau \) unless the assessment measure incorporates prior probabilities, in which case,

\[
\omega(\tau, \chi) = \sum_j \pi_j N_j(\tau, \chi) / N_j(\text{root}, \chi)
\]

where \( \pi_j \) denotes the prior probability of target value \( j \), and \( N_j \) denotes the number of observations with target value \( j \) in data set \( \chi \) in \( \tau \).

The inclusion function, \( \lambda(\tau, \chi) \), equals 1 unless MEASURE= LIFT or LIFTPROFIT. These measures only use a proportion \( \gamma \) of the data to compute the assessment. The ARBORETUM procedure orders the leaves \( \tau \) by descending values of \( \psi(\tau, \text{training}) \). The first leaf has the largest value of \( \psi(\tau, \text{training}) \), not the smallest. The cumulative lift measures use observations in leaves with large values of \( \psi \).

Let the relation \( \tau' < \tau \) stand for \( \psi(\tau', \text{training}) > \psi(\tau, \text{training}) \). Define

\[
\Omega(\tau, \chi) = \sum_{\tau' < \tau} \omega(\tau', \chi)
\]

Intuitively, \( \Omega(\tau, \chi) \) is the number of observations in the \( \chi \) data set in leaves \( \tau' \) such that \( \psi(\tau', \text{training}) > \psi(\tau, \text{training}) \).

For fixed \( \chi \) and \( 0 < \gamma < 1 \), there exists a unique \( \tau^* \) such that \( \Omega(\tau^*, \chi) \geq \gamma \), and \( \Omega(\tau^* - 1, \chi) < \gamma \). Define the inclusion function to be

\[
\lambda(\tau^*, \chi) = \begin{cases} 
1 & \tau < \tau^* \\
\frac{(\gamma - \Omega(\tau^* - 1, \chi))}{\omega(\tau^*, \chi)} & \tau = \tau^* \\
0 & \tau > \tau^*
\end{cases}
\]

Note that \( 0 < \lambda(\tau^*, \chi) \leq 1 \). Intuitively, \( \lambda(\tau, \chi) \) selects which leaves to include in the cumulative lift measure, and will select a fraction of one particular leaf, \( \tau^* \), if the required number of observations, \( \gamma N(\text{root}, \chi) \), does not equal the number of observations in a set of whole leaves.

In the definition of \( \psi(\tau, \chi) \) below, \( p_j(\tau, \chi) \) denotes the proportion of observations with target value \( j \) in data set \( \chi \) in \( \tau \). If the assessment measure incorporates prior probabilities,

\[
p_j(\tau, \chi) = \frac{\pi_j N_j(\tau, \chi) / N_j(\text{root}, \chi)}{\sum_i \pi_i N_i(\tau, \chi) / N_i(\text{root}, \chi)}
\]

The remaining sections define \( \psi(\tau, \chi) \) for the different assessment measures.
Formula for Profit and Loss

For an interval target with MEASURE=PROFIT,

\[ \psi(\tau, \chi) = \sum_{i=1}^{N(\tau,\chi)} \frac{E_i(\tau)}{N(\tau,\chi)} \]

where \( E_i(\tau) \) is the estimated profit or loss for observation \( i \) in \( \tau \).

For a categorical target,

\[ \psi(\tau, \chi) = \sum_j A_{jd} \hat{p}_j(\tau, \chi) \]

where \( \hat{d} \) is the node decision, and \( A_{jd} \) is the coefficient in the decision matrix for target value \( j \), decision \( \hat{d} \). \( \psi(\tau, \chi) \) represents profit, revenue, or loss according to whether the DECDATA= data set in the DECISION statement has type PROFIT, REVENUE, or LOSS. Note that \( \psi(\tau, \chi) \) does not incorporate decision costs, and therefore does not represent profit if the DECDATA= data set has type REVENUE.

Formula for Misclassification Rate

\[ \psi(\tau, \chi) = \sum_{j \neq \hat{j}} p_j(\tau, \chi) \]

where \( \hat{j} \) is the predicted target value in the node.

Formula for Average Square Error and Gini

For an interval target using MEASURE=ASE,

\[ \psi(\tau, \chi) = \sum_{i=1}^{N(\tau,\chi)} \frac{(y_i - \hat{y}(\tau))^2}{N(\tau,\chi)} \]

where \( \hat{y}(\tau) \) is the average of the target variable among the training observations in node \( \tau \).

For a categorical target using MEASURE=ASE,

\[ \psi(\tau, \chi) = \sum_{i=1}^{N(\tau,\chi)} \sum_{j=1}^{J} \frac{(\delta_{ij} - \hat{p}_j(\tau))^2}{N(\tau,\chi)} \]

where \( \delta_{ij} \) equals 1 if observation \( i \) has target value \( j \), and equals 0 otherwise, and \( \hat{p}_j(\tau) = p_j(\tau, \text{training}) \), the predicted probability of target value \( j \) for observations in \( \tau \). A simpler, equivalent expression is

\[ \psi(\tau, \chi) = (1 - 2 \sum_j p_j(\tau, \chi)\hat{p}_j(\tau) + \sum_j \hat{p}_j^2(\tau)) \]
If the assessment measure incorporates prior probabilities (if any) and \( \chi \) represents the training data, then the expression reduces to the \textit{Gini index},

\[
\psi(\tau, \text{training}) = (1 - \sum_j p_j^2(\tau, \text{training}))
\]

\textbf{Formula for Lift}

For MEASURE=LIFT,

\[
\psi(\tau, \chi) = p_{\text{event}}(\tau, \chi)
\]

For MEASURE=LIFTPROFIT, \( \psi(\tau, \chi) \) is the same as defined for MEASURE=PROFIT.

\textbf{Performance Considerations}

When the ARBORETUM 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.

\textbf{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 approximately equal 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 may 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 only necessary 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 may reduce the number of data passes, but may not 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 ARBORETUM 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.

IMPORTANCE= Output Data Set

The IMPORTANCE= option in the SAVE statement specifies the output data set to contain the measure of relative importance of 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.

Variable Importance

The relative importance of input variable \( v \) in subtree \( T \) is computed as

\[
I(v; T) \propto \sqrt{\sum_{\tau \in T} a(s_v, \tau) \Delta SSE(\tau)}
\]
where the sum is over nodes $\tau$ 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 $\tau$:

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

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

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

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

where

$B(\tau)$ = set of branches from $\tau$

$\tau_b$ = child node of $\tau$ in branch $b$

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

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

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

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

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

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

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

Variables in the Data Set

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.

## 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 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_: U_: V_: variables output in the OUT= option in the SCORE statement

The variables VN, VNPRIORS, and _VRASE_ only occur if validation data is specified. The variables NPRIORS and VNPRIORS only occur for categorical targets. The variables _RASE_ and _VRASE_ only occur for interval targets. The colon in a name expression such as, I_: refers to all variables whose name begins with, I_. The section “Variable Names and Conditions for Their Creation” on page 59 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

\[ N_j = N \sum_i P_{namei} N_i(root) / \pi_i \]

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

---

**PATH= Output Data Set**

The PATH= option in the SAVE statement specifies the output data set describing the observations 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 MISSING or NOT MISSING. The relation MISSING indicates that missing values of the variable are accepted in the node. The relation NOT MISSING indicates that missing values of the variable are excluded from the node. If the relation is not MISSING or NOT MISSING, than 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 may be infinite, or the lower end may be infinitely negative, but at least one end will be finite (otherwise RELATION would equal NOT MISSING). 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 only contains 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, and ‘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. Table 8 summaries the possible values of STAT.

<table>
<thead>
<tr>
<th>STAT</th>
<th>NUMERIC_VALUE</th>
<th>CHARACTER_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLE</td>
<td></td>
<td>Variable name</td>
</tr>
</tbody>
</table>
### Variable Names and Conditions for Their Creation

The names of all the possible new variables are listed in Table 9.

#### Table 9. New Variables in the OUT= Data Set

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Target</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Variables for Prediction</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F_name</td>
<td>actual, formatted category</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>I_name</td>
<td>predicted, formatted category</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>P_namevalue</td>
<td>predicted value</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>R_namevalue</td>
<td>residual from the prediction</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>U_name</td>
<td>predicted, unformatted category</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>V_namevalue</td>
<td>predicted value computed with validation data</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td><em>WARN</em></td>
<td>indications of problems with the prediction</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td><strong>Variables for Decisions</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BL_name</td>
<td>best possible loss from any decision</td>
<td>yes</td>
<td>LOSS</td>
</tr>
<tr>
<td>BP_name</td>
<td>best possible profit from any decision</td>
<td>yes</td>
<td>PROFIT, REVENUE</td>
</tr>
<tr>
<td>CL_name</td>
<td>loss computed from the target value</td>
<td>yes</td>
<td>LOSS</td>
</tr>
<tr>
<td>CP_name</td>
<td>profit computed from the target value</td>
<td>yes</td>
<td>PROFIT, REVENUE</td>
</tr>
<tr>
<td>D_name</td>
<td>label of the chosen decision alternative</td>
<td>no</td>
<td>any</td>
</tr>
<tr>
<td>EL_name</td>
<td>expected loss from the chosen decision</td>
<td>no</td>
<td>LOSS</td>
</tr>
<tr>
<td>EP_name</td>
<td>expected profit from the chosen decision</td>
<td>no</td>
<td>PROFIT, REVENUE</td>
</tr>
<tr>
<td>IC_name</td>
<td>investment cost</td>
<td>no</td>
<td>REVENUE</td>
</tr>
<tr>
<td>ROI_name</td>
<td>return on investment</td>
<td>yes</td>
<td>REVENUE</td>
</tr>
<tr>
<td><strong>Variables for Leaf Assignment</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>i</strong></td>
<td>proportion of the observation in leaf i</td>
<td>no</td>
<td>DUMMY</td>
</tr>
<tr>
<td><strong>LEAF</strong></td>
<td>leaf identification number</td>
<td>no</td>
<td>LEAF</td>
</tr>
<tr>
<td><strong>NODE</strong></td>
<td>node identification number</td>
<td>no</td>
<td>LEAF</td>
</tr>
</tbody>
</table>
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` simply 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_name` 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 `DECVARS=` option in the DECISION statement, the variable `D_name` is created, as is either `EL_name` or `EP_name` depending on whether or not 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 `DECVARS=` 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.
Example 1. Prior Probabilities with Biased Samples

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 the leaf \( i \).

SEQUENCE= Output Data Set

The SEQUENCE= option in the SAVE statement specifies a data set to contain fit statistics for all subtrees in the subtree sequence. See the “Tree Assessment and the Subtree Sequence” section beginning on page 49 for an explanation of the subtree sequence. Each observation describes a subtree with a different number of leaves. The variables are

- \( \_ASSESS\_ \), the assessment value
- \( \_VASSESS\_ \), the assessment value based on validation data
- \( \_SEQUENCE\_ \), the assessment value used for creating the subtree sequence if different from \( \_ASSESS\_ \)
- \( \_VSEQUENCE\_ \), the validation assessment value used for creating the subtree sequence if different from \( \_VASSESS\_ \)
- fit statistics variables output by the OUTFIT= option of the SCORE statement

Examples

Example 1. Prior Probabilities with Biased Samples

This example illustrates the need for prior probabilities when the training data contains different proportions of categorical target values than does the data to which the model is intended to apply. A common situation is that of a binary target in which one value occurs infrequently. Some analysts will remove a portion of the observations with the more frequent target value from the training data. One reason is to reduce the volume of data without changing the predictions very much. Another stems from the belief that the algorithm performs better when starting with equal proportions of the target values.

This example compares four approaches to analyzing data in which one value of a binary target dominates:

- ORIGINAL: no sampling or use of prior probabilities
- BIASED: sampling of observations with the majority target value
- PRIORS: using prior probabilities in prediction and assessment
Prior probabilities convert the predictions from the BIASED analysis to predictions that would obtain from analyzing the original data. Using prior probabilities in the split search produces splits similar to those found in the original data, undermining any attempt to train the tree on roughly equal amounts of the two target values.

Suppose variables COIN1 and COIN2 represent the outcomes of tossing two coins in which heads and tails are equally likely. Let the variable HEADS be 1 if both COIN1 and COIN2 are heads, and 0 otherwise. Evidently, the probability that HEADS equals 1 is 1/4.

COIN1 and COIN2 completely determine HEADS, and a decision tree should predict HEADS perfectly, regardless of what prior probabilities are specified or what proportion of target values are in the training data, as long as at least one instance of each of the four possible combinations of COIN1 and COIN2 are in the training data.

Prior probabilities become necessary when the input variables influence the target without completely determining it. The SAS DATA step below generates a random variable X1 equal to COIN1 75 percent of the time, and generates a random variable X2 similarly. When COIN1 and COIN2 are both heads, the chance that both X1 and X2 are heads is 0.75 times 0.75, which equals 0.56.

```sas
data original;
  keep heads x1 x2;
  call streaminit(9754321);
  do i = 1 to 10000;
    coin1 = rand('bernoulli', 0.5);
    coin2 = rand('bernoulli', 0.5);
    heads = coin1 eq 1 and coin2 eq 1;
    x1 = coin1;
    if rand('bernoulli', 0.25) ne 0 then x1 = 1 - x1;
    x2 = coin2;
    if rand('bernoulli', 0.25) ne 0 then x2 = 1 - x2;
    output;
  end;
```

The proportion of observations in the data set ORIGINAL for which HEADS equals 1 is about 1/4. By removing every third observation in which HEADS is not equal to 1, the DATA step below creates a data set named BIASED, in which the proportion of observations with HEADS equal to one is about 1/2.

```sas
data biased;
  retain n_tails 0;
  drop n_tails;
  set original;
  if heads eq 0 then do;
    n_tails + 1;
  end;
```
In the following code, the ARBORETUM procedure creates a tree from the data set ORIGINAL and saves three data sets containing results:

- ORIGINAL_PATH: description of each path to a leaf
- ORIGINAL_NODES: counts and predictions in each leaf
- ORIGINAL_SEQ: assessment of each subtree

The KEEP= and RENAME= data set options specify which variables to keep and what to name them. Selecting and renaming variables now will make it easy to merge and print results from different runs of the procedure later. The procedure is also run using the BIASED data set. The code is not shown because it is identical to that below except that ‘biased’ replaces ‘original’ everywhere.

```plaintext
proc arboretum data=original;
   input x1 x2 / level=binary;
   target heads / level=binary;
   assess;
   save path = original_path,
             nodestats =original_nodes(keep = leaf n p_heads1
                                          where =(leaf ne .)
                                          rename=(n = n_original
                                                  npriors = np_original
                                                  p_heads1 = p_original)
             sequence =original_seq(keep=_assess_ rename=(_assess_ = original));
```

The following DATA step creates a variable, PRIOR, equal to the prior probability of the corresponding value of HEADS.

```plaintext
data priors;
   input heads prior;
datalines;
0  0.75
1  0.25
;
```

The ARBORETUM procedure is run a second time on the BIASED data set, this time with a DECISION statement to include the prior probabilities specified in the PRIOR variable of the PRIORS data set. The procedure uses the priors to adjust the posterior probabilities, but not to adjust the overall evaluation of a subtree unless explicitly requested. The “Incorporating Prior Probabilities in the Tree Assessment” section on page 66 compares the two approaches using the assessment values saved from the two ASSESS statements here. The first ASSESS statement computes the subtree
sequence without using priors, and the second ASSESS statement computes it using priors. The two SAVE statements save the assessment values for each subtree. The first SAVE statement saves the values for the first subtree in the variable PRIORS, and the second SAVE statement saves the values in the variable PASSESS, indicating that the overall assessment uses priors.

```sas
proc arboretum data=biased;
  input x1 x2 / level=binary;
  target heads / level=binary;
  decision decdata=priors priorvar=prior;
  assess;
  save path = priors_path
    nodestats = priors_nodes(keep = leaf n npriors p_heads1
      where =(leaf ne .)
      rename=(n = n_priors
        npriors = np_priors
        p_heads1 = p_priors))
    sequence = priors_seq(keep=_assess_ rename=_assess_ = priors)
    rules = priors_rules;
  assess priors;
  save sequence = assess_seq(keep=_assess_ rename=_assess_ = passess);
run;
```

The three runs of the ARBORETUM procedure have produced the same splitting rules depicted in the ORIGINAL_PATH data set shown below. (The BIASED_PATH and PRIORS_PATH data sets are the same and are not shown.)

```sas
proc print data=original_path;
  var leaf variable character_value;
  where relation eq ’=’;
```

**Output 1.1. Path to Each Leaf**

<table>
<thead>
<tr>
<th>Obs</th>
<th>LEAF</th>
<th>VARIABLE</th>
<th>CHARACTER_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>x1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>x2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>x1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>x2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>x2</td>
<td>0</td>
</tr>
</tbody>
</table>

The output describes three leaves defined as follows:

1. X1 equals 0 and X2 equals 1
2. Both X1 and X2 equal 1
3. X2 equals 0
Table 10 shows the proportion of observations expected to appear in each leaf, as well as the probability that HEADS equals 1.

Table 10. Expected Statistics for Each Leaf

<table>
<thead>
<tr>
<th>Leaf</th>
<th>Proportion of N</th>
<th>Prob(HEADS=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25</td>
<td>0.1875 (= 0.25 * 0.75)</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0.5625 (= 0.75 * 0.75)</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>0.1250 (= 0.25 * 0.50)</td>
</tr>
</tbody>
</table>

These expected numbers will be compared with the actual numbers from the three different ARBORETUM runs. The DATA step below merges the leaf statistics from the three runs.

```plaintext
data nodes;
  set original_nodes;
  set biased_nodes;
  set priors_nodes;
```

The following code creates Output 1.2 showing the actual count of observations in each leaf. Only counts in the first and last columns are in the expected proportions. The first column results from training on the ORIGINAL data set. The last two columns show the variables N and NPRIORS, respectively, from the execution of the ARBORETUM procedure with prior probabilities on the BIASED training data set. The counts in the last column (variable NPRIORS) incorporate the prior probabilities to adjust the counts of the observations displayed in the previous column.

```plaintext
proc print data=nodes;
  var leaf n_: np_:;
```

Output 1.2. Count in Each Leaf for Each ARBORETUM Run

<table>
<thead>
<tr>
<th>Obs</th>
<th>LEAF</th>
<th>n_original</th>
<th>n_biased</th>
<th>n_priors</th>
<th>np_priors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2529</td>
<td>1152</td>
<td>1152</td>
<td>1250.30</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2468</td>
<td>1722</td>
<td>1722</td>
<td>1223.63</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5003</td>
<td>2072</td>
<td>2072</td>
<td>2472.07</td>
</tr>
</tbody>
</table>

Output 1.3 shows the predicted probability that HEADS equals one for each leaf in each ARBORETUM run. The column labeled, P_ORIGINAL, uses the ORIGINAL training data, the next column uses the BIASED, and the last uses the BIASED data with prior probabilities. Only the P_ORIGINAL and the last column agree with the expected probabilities.
The ARBORETUM Procedure

```sas
proc print data=nodes;
  var leaf p_;
```

**Output 1.3.** Prob(HEADS Equals 1) by Leaf and ARBORETUM Run

<table>
<thead>
<tr>
<th>Obs</th>
<th>LEAF</th>
<th>p_original</th>
<th>p_biased</th>
<th>p_priors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.18</td>
<td>0.40</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.55</td>
<td>0.79</td>
<td>0.57</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.12</td>
<td>0.29</td>
<td>0.12</td>
</tr>
</tbody>
</table>

**Incorporating Prior Probabilities in the Tree Assessment**

The ARBORETUM procedure uses the misclassification rate by default for assessing how well a tree fits data with a categorical target. After creating a tree, the procedure creates a sequence of subtrees such that each subtree in the sequence has the lowest misclassification rate among all subtrees with the same number of leaves. The SEQUENCE= option to the SAVE statement creates a data set containing statistics about each subtree.

The following DATA step code combines the subtree sequence statistics from the three ARBORETUM runs. The third run computes the misclassification rates in two different ways: first without and then with incorporating prior probabilities into the misclassification counts. The DATA step therefore merges four sequences.

```sas
data sequence;
  set original_seq;
  set biased_seq;
  set priors_seq;
  set assess_seq;
```

**Output 1.4** shows the data set of four subtree sequences. The column labeled ORIGINAL results from the ARBORETUM run using the ORIGINAL training data set, while the other columns are based on the BIASED sample. Prior probabilities were specified in the run producing last two columns, PRIORS and PASSESS. Only for the last column, PASSESS, were the misclassification counts adjusted with prior probabilities.

The rows represent subtrees with 1, 2, 3, and 4 subtrees respectively. The first row represents the consequence of predicting each observation to have HEADS equal to 0. The manner of generating the data results in an expected proportion of observations with HEADS equal to 0 of 3/4, so that the expected misclassification rate is 1/4. Only the first and last columns match this expected proportion. In fact, the first and last columns agree for every subtree.

```sas
proc print data=sequence;
```
**Incorporating Prior Probabilities in the Split Search**

The worth of a splitting rule for a categorical target depends on the relative proportions of the different target values within each branch (and within the node being split). When the PRIORSSEARCH option to the PROC ARBORETUM statement is specified, the ARBORETUM procedure uses the prior probabilities to modify the proportions and consequently modify the worth of a splitting rule. This section compares the worth computed for the same split in four situations:

- **ORIGINAL**: 4,946 unbiased observations from the ORIGINAL data
- **BIASED**: the BIASED subsample, no priors
- **PRIORS**: the BIASED subsample with prior probabilities
- **PSEARCH**: the BIASED subsample with the PRIORSSEARCH option

The worth of a split depends on the number of observations. To compare the split worth, the number of observations in the ORIGINAL training data is reduced here to the same number (4,946) as in the BIASED data. All candidate splitting rules are saved in a data set named ORIGINAL_RULES.

```plaintext
proc arboretum data=original(obs=4946);
  input x1 x2 / level=binary;
  target heads / level=binary;
  assess;
  save rules =original_rules ;
```

The splitting rules for the BIASED and PRIORS run have already been saved in the data sets BIASED_RULES and PRIORS_RULES, respectively.

The following code uses the PRIORSSEARCH option and saves the splitting rules in the data set PSEARCH_RULES.

```plaintext
proc arboretum data=biased priorssearch ;
  input x1 x2 / level=binary;
  target heads / level=binary;
  decision decdata=priors priorvar=prior;
  assess priors;
  save rules =psearch_rules;
```
The WORTHDS macro extracts the worth of the best splitting rule on \( X_1 \) and on \( X_2 \) in the root node as saved in the RULES= option in the SAVE statement of the ARBORETUM procedure.

```sas
%macro worthds( prefix);
  data &prefix._worth;
    length var $ 2;
    length variable $ 2;
    keep &prefix variable;
    set &prefix._rules end=the_end;
    retain x1 x2 var;
    if node eq 1 and stat='VARIABLE' then
        var = left(trim(character_value));
    if node eq 1 and stat='WORTH' then do;
        if var eq 'x1' then x1 = numeric_value;
        else x2 = numeric_value;
    end;
    if the_end ne 0 then do;
        variable = 'x1';
        &prefix = x1;
        output;
        variable = 'x2';
        &prefix = x2;
        output;
    end;
%mend;
```

The following code extracts the split worth from each of four ARBORETUM runs and combines them into one data set, which is then printed.

```sas
%worthds( original);
%worthds( biased);
%worthds( priors);
%worthds( psearch);

data worth;
  set original_worth;
  set biased_worth;
  set priors_worth;
  set psearch_worth;
```

Output 1.5. Split Worth for Four ARBORETUM Runs

<table>
<thead>
<tr>
<th>Obs</th>
<th>variable</th>
<th>original</th>
<th>biased</th>
<th>priors</th>
<th>psearch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x1</td>
<td>88.7930</td>
<td>127.151</td>
<td>127.151</td>
<td>93.5635</td>
</tr>
<tr>
<td>2</td>
<td>x2</td>
<td>97.8053</td>
<td>127.467</td>
<td>127.467</td>
<td>93.4600</td>
</tr>
</tbody>
</table>

Output 1.5 shows the worth of the same split on X1 in the four situations, and similarly for X2. X1 and X2 produce splits of equal worth, as expected, except in the truncated ORIGINAL training data, where, presumably, chance variation in the sample resulted in a better split for X2. Using the BIASED training data, only the column from the ARBORETUM run with the PRIORSSEARCH option produced a split worth comparable to that of the ORIGINAL data set.

References


Subject Index

A
ancestor node, 6
assessing trees, 18

B
branch, 6

C
candidate rule, 6
child node, 6
competing rule, 6

decision, 7
decision matrix
  incorporating in search, 17
decision theory, 7
decision tree, 7
descendent node, 6
desktop application, 8
dummy variable, 29, 61

E
event, 19

F
formatted values, 6

I
input variable, 6
internal node, 6
interval variable, 6

L
leaf identification number, 60
leaves, 6
level of measurement, 6

M
missing values, 45
  none in training data, 46

N
node identification number, 60
nominal variable, 6
normalized values, 6

O
ordinal variable, 6

P
p-value
  adjusting for values and branches, 17
  adjusting for variables, 18, 44
  Gabriel’s adjustment, 43
  posterior probability, 6
  primary rule,
    See splitting rule
  prior probability, 6
  incorporating in search, 18

R
recursive partitioning, 6
root node, 6

S
sequence of subtrees, 18
splitting criteria, 16
splitting criterion, 6
splitting rule, 6
  candidate, 6
  competing, 6
  primary, 6
  surrogate, 6
subtree
  sequence, 18
  surrogate rule, 6

T
target variable, 5
terminal node,
  See leaves
Syntax Index

A
ALPHA= option
  TRAIN statement, 33
ASSESS statement, 18
  EVENT= option, 19
  MEASURE= option, 19
  NOPRIORS option, 20
  NOVALIDATA, 20
  PRIORS option, 20
  PROPORTION= option, 20
  PRUNEDATA= option, 20
  VALIDATA= option, 20

B
BEST option
  SUBTREE statement, 32
BIGBRANCH
  MISSING= option, 46
BRANCH statement, 20
  NODES= option, 21
  ONE option, 21

C
CATALOG= option
  CODE statement, 21
  DESCRIBE statement, 24
CODE statement, 21
  CATALOG= option, 21
  DUMMY option, 21
  FORMAT= option, 22
  NODES= option, 21
  RESIDUAL option, 22
COST= option
  DECISION statement, 23
CRITERION= option
  PROC ARBORETUM statement, 16

D
DATA= option
  PROC ARBORETUM statement, 16
  SCORE statement, 29
DECDATA= option
  DECISION statement, 24
  DECISION statement, 22
  DECVARS= option, 24
  PRIORVARS= option, 24
  DECSEARCH option
    PROC ARBORETUM statement, 17
  DECVARS= option
    DECISION statement, 24
  DESCRIBE statement, 24
    CATALOG= option, 24
    FILE= option, 24
    FORMAT= option, 24
DISK option
  PERFORMANCE statement, 27
DISTRIBUTE
  MISSING= option, 46
DROPVARS= option
  PRUNE statement, 28
DUMMY option
  CODE statement, 21
  SCORE statement, 29

E
EVENT= option
  ASSESS statement, 19
EXHAUSTIVE= option
  TRAIN statement, 33

F
FILE= option
  DESCRIBE statement, 24
FORMAT= option
  CODE statement, 22
  DESCRIBE statement, 24
  FREQ statement, 25

I
IMPORTANCE= option
  SAVE statement, 28
INMODEL= option
  PROC ARBORETUM statement, 17
INPUT statement, 25
  LEVEL= option, 25
  MISSING= option, 25
  ORDER= option, 25
  SPLITATDATUM, 26
  SPLITBETWEEN, 26
INTERACT statement, 26
LARGEST option, 26, 32
NLEAVES= option, 27, 32
PRUNED option, 26
INTERVALBINS= option
TRAIN statement, 33

K
KEEPRULES= option
PRUNE statement, 28

L
LARGEST option
INTERACT statement, 26, 32
LEAFSIZE= option
TRAIN statement, 33
LEVEL= option
INPUT statement, 25
TARGET statement, 32
LINESIZE option
CODE statement, 22, 24

M
MAKEMACRO statement, 27
MAXBRANCH= option
TRAIN statement, 34
MAXDEPTH= option
TRAIN statement, 34
MAXNEWDEPTH= option
TRAIN statement, 34
MAXRULES= option
TRAIN statement, 34
MAXSURROGATES= option
TRAIN statement, 35
MEASURE= option
ASSESS statement, 19
MEMSIZE= option
PERFORMANCE statement, 27
MINCATSIZE= option
TRAIN statement, 35
MINWORTH= option
TRAIN statement, 35
MISSBRANCH= option
SPLIT statement, 31
MISSING= option
BIGBRANCH, 46
DISTRIBUTE, 46
INPUT statement, 25
PROC ARBORETUM statement, 17
SMALLRESIDUAL, 45
USEINSEARCH, 45
MISSONLY option
SPLIT statement, 31
MODEL= option
SAVE statement, 28
MULTIPASS option
PERFORMANCE statement, 27

N
NLEAVES= option
INTERACT statement, 27, 32
NODES= option
BRANCH statement, 21
SAVE statement, 29
SCORE statement, 30
TRAIN statement, 35
NODESIZE= option
PERFORMANCE statement, 27
NODESTAT= option
SAVE statement, 29
NODUMMY option
CODE statement, 21
NOLEAFID option
CODE statement, 22
SCORE statement, 30
NOPREDICTION option
CODE statement, 22
SCORE statement, 30
NOPRIORS option
ASSESS statement, 20
NOVALIDATA
ASSESS statement, 20

O
ONE option
BRANCH statement, 21
ORDER= option
INPUT statement, 25
TARGET statement, 33
OUT= option
SCORE statement, 30
OUTFIT= option
SCORE statement, 30

P
PADJUST= option
PROC ARBORETUM statement, 17
PATH= option
SAVE statement, 29
PERFORMANCE statement, 27
DISK option, 27
MEMSIZE= option, 27
MULTIPASS= option, 27
NODESIZE= option, 27
RAM option, 27
PMML option
CODE statement, 22
PRIORS option
ASSESS statement, 20
PRIORSSEARCH option
PROC ARBORETUM statement, 18
PRIORVAR= option
DECISION statement, 24
PROC ARBORETUM statement, 16
CRITERION= option, 16
DATA= option, 16
DECSEARCH option, 17
The ASSOC Procedure

Overview
Procedure Syntax
PROC ASSOC Statement
CUSTOMER Statement
TARGET Statement
Details
Example
References

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Overview

Association discovery is the identification of items that occur together in a given event or record. This technique is also known as market basket analysis. Online transaction processing systems often provide the data sources for association discovery. Associations rules are based on frequency counts of the number of times items occur alone and in combination in the database. The rules are expressed as "if item A is part of an event then item B is also part of the event X percent of the time." The rules should not be interpreted as a direct causation but as an association between two or more items. Identifying creditable associations can help the business technologist make decisions such as when to distribute coupons, when to put a product on sale, or how to layout items in a store.

Hypothetical association discovery rules include: If a customer buys shoes, then 10% of the time he also buys socks. A grocery chain may find that 80% of all shoppers are apt to buy a jar of salsa when they also purchase a bag of tortilla chips. When "do-it-yourselfers" buy latex paint they, also buy rollers 85% of the time. Forty percent of investors holding an equity index fund will have a growth fund in their portfolio.

An association rule has a left side (antecedent) and a right side (consequent). Both sides of the rule can contain more than one item. The confidence factor, level of support, and lift are three important evaluation criteria of association discovery. The strength of an association is defined by its confidence factor, which is the percentage of cases in which a consequent appears with a given antecedent. The level of support is how frequently the combination occurs in the market basket (data base). Lift is equal to the confidence factor divided by the expected confidence. A creditable rule has a large relative confidence factor, a relatively large level of support, and a value of lift greater than 1. Rules having a high level of confidence but little support should be interpreted with caution.

The maximum number of items in an association determines the maximum size of the item set to be considered. For example, the default of 4 items indicates that up to 4-way associations are performed.
The ASSOC Procedure

Procedure Syntax

PROC ASSOC <option(s)>;

CUSTOMER variable-list;

TARGET variable;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The ASSOC Procedure

PROC ASSOC Statement

Invokes the ASSOC procedure.

PROC ASSOC <option(s)>;

Required Argument

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

Specifies the output data set that contains the following variables: SET_SIZE, COUNT, ITEM1, ITEM2,...ITEMn (where n is the maximum number of variables). See Details for more information.

**SET_SIZE:**

Variable that contains the total number of transactions in the data set. The first observation has the SET_SIZE equal to 0. SET_SIZE is labeled as Relations in the Results Browser.

**COUNT:**

Contains the number of transactions meeting the rule.

**ITEM1, ITEM2,...ITEMn:**

Contains the individual items forming the rule including the arrow.

**Tip:**

The OUT= data set created by PROC ASSOC is input to the RULEGEN and SEQ procedures. Run PROC ASSOC and PROC RULEGEN to perform association discovery. Run PROC ASSOC and PROC SEQ to perform sequence discovery.

Options

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

Identifies the input data source. To perform association discovery, the input data set must have a separate observation for each product purchased by each customer. You must also assign the ID model role to a variable and the TARGET model role to another variable in the Input Data Source.

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

Identifies the data catalog of the input data source.

**ITEMS=**integer

Specifies the maximum number of events or transactions to chain (or associate) together.

**SUPPORT=**integer

Specifies the minimum number of transactions that must be considered in order for a rule to be
accepted. Rules that do not meet this support level are rejected. The level of support represents how frequently the combination occurs in the market basket (input data source).

| **Default:** | 5% of the largest item frequency count |

*Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.*
CUSTOMER Statement

Specifies the customer(s) to be analyzed.

Alias: CUST

CUSTOMER variable-list;

Required Argument

variable-list

  Specifies one or more names of customers to be analyzed.
TARGET Statement

Specifies the target to be analyzed.

TARGET variable;

Required Argument

variable

Specifies the NOMINAL variable, which contains items usually ordered by customers.
Details

The input to the ASSOC procedure has the following role variables: ID and TARGET. All records with the same ID values form a transaction. Every transaction has a unique ID value and one or more TARGET values.

You may have more than one ID variable. However, associations analysis can only be performed on one target variable at a time. When there are multiple ID variables, PROC ASSOC concatenates them into a single identifier value during computation.

For numeric target variables, missing values constitute a separate item or target level and show up in the rules as a period (.). For character target variables, completely blank values constitute a separate item (target level) and show up in the rules as a period (.). All records with missing ID values are considered a single valid transaction.

Output Processing

PROC ASSOC makes a pass through the data and obtains transaction counts for each item. It outputs these counts with a SET_SIZE of 1 and the items listed under ITEM1. Items that do not meet the support level are discarded. By default, the support level is set to 5% of the largest item count.

PROC ASSOC then generates all potential 2-item sets, makes a pass through the data and obtains transaction counts for each of the 2-item sets. The sets that meet the support level are output with SET_SIZE of 2 and items listed under ITEM1 and ITEM2.

The entire process is repeated for up to n-item sets. The output from PROC ASSOC is saved as SAS data sets. The data sets enable you to define your own evaluation criteria and/or reports.

Note that the ordering of n-items within an n-item set is not important. Any individual transaction, where each of the n-items occurs in any order, qualifies for a count to that particular set. The support level, once set, remains constant throughout the process.

Caution: The theoretical potential number of item sets can grow very quickly. For example, with 50 different items, you have 1225 potential 2-item sets and 19,600 3-item sets. With 5,000 items, you have over 12 million of the 2-item sets, and a correspondingly large number of 3-item sets.

Processing an extremely large number of sets could cause your system to run out of disk and/or memory resources. However, by using a higher support level, you can reduce the item sets to a more manageable number.
Example

PROC ASSOC must be executed before PROC RULEGEN or PROC SEQUENCE is run.

Please see the RULEGEN and SEQUENCE procedures syntax for examples of PROC ASSOC code.
References


The DECIDE Procedure

Overview

Procedure Syntax

- PROC DECIDE Statement
- CODE Statement
- DECISION Statement
- FREQ Statement
- POSTERIORS Statement
- PREDICTED Statement
- TARGET Statement

Details

Example

Example 1: Using the DECIDE Procedure Following the DISCRIM Procedure

References

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DECIDE Procedure

Overview

The DECIDE procedure produces optimal decisions based on a user-supplied decision matrix, prior probabilities, and output from a modeling procedure. The output from the modeling procedure may be either posterior probabilities for the classes of a categorical target variable or predicted values of an interval target variable. The DECIDE procedure can also adjust posterior probabilities for changes in the prior probabilities. Background and formulas for decision processing are given in the chapter on "Predictive Modeling."

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

For a categorical target variable, there should be one row for each category. The value in the decision matrix that is located at a given row and column specifies the consequence of making the decision corresponding to column when the target value corresponds to the row.

For an interval target variable, each row defines a knot in a piecewise linear spline function. The consequence of making a decision is computed by interpolating in the corresponding column of the decision matrix. If the predicted target value is outside the range of knots in the decision matrix, the consequence of a decision is computed by linear extrapolation.

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

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

The DECIDE procedure will choose the optimal decision for each observation. If the decision data set is of TYPE=PROFIT or REVENUE, the decision that produces the maximum expected or estimated profit is chosen. If the decision data set is of TYPE=LOSS the decision that produces the minimum expected or estimated loss is chosen.

If the actual value of the target variable is known, the DECIDE procedure will calculate:
● The consequence of the chosen decision for the actual target value for each case.
● The best possible consequence for each case.
● Summary statistics giving the total and average profit or loss.

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

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
PROC DECIDE <option(s)>;

CODE <option(s)>;

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

FREQ variable;

POSTERIORS variable(s);

PREDICTED variable;

TARGET variable;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DECIDE Procedure

**PROC DECIDE Statement**

Invokes the DECIDE procedure.

**Discussion:** The PROC DECIDE statement runs the DECIDE procedure and identifies the input and output data sets. You also need the following statements:
- A DECISION statement
- Either a POSTERIORS or a PREDICTED statement
- A TARGET statement

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

**Options**

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

Specifies the input data set that contains the output from a modeling procedure.

**Default:**_LAST_

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

Specifies the output data set that contains the following variables:
- the variables from the input data set
- the chosen decision (prefix D_)
- the expected consequence of the chosen decision (prefix EL_ or EP_)

If the target value is in the input data set, the output data set also contains the following variables:
- the consequence of the chosen decision computed from the target value (prefix CL_ or CP_)
- the consequence of the best possible decision knowing the target value (prefix BL_ or BP_)

If PRIORVAR= and OLDPRIORVAR= variables are specified, then the output data will contain the recalculated posteriors.

**Note:** If you want to create a permanent SAS data set, you must specify a two-level name. For more information on this topic, see "SAS Files" and "DATA Step Concepts " in *SAS Language Reference: Concepts.*

**Default:**

If the OUT= option is omitted, PROC DECIDE creates an output data set and names it according to the DATAn convention, just as if you had omitted a data set name in a DATA statement.
OUTFIT=<libref.> SAS-data-set
Specifies the output data set that contains statistics including the total and average profit or loss. The OUTFIT= option may not be specified with ROLE=SCORE.

Default: None

ROLE=TRAIN|VALID|VALIDATION|TEST|SCORE
Specifies whether the DATA= data set is a training set, validation set, test set, or scoring set. The ROLE= option affects the names of the variables in the OUTFIT= data set.

Default: TEST
The DECIDE Procedure

CODE Statement

Generates SAS DATA step code that can be used to score data sets.

Tip: If neither FILE= nor METABASE= are specified, then the SAS code is written to the SAS log. You may specify both FILE= and METABASE= to write code to both locations. The TARGET variable must appear in the DATA= data set as well as the DECDATA= data set.

```
CODE <code-option(s)>;
```

Code Options

**FILE='filename'**

Specifies a path for writing the code to an external file. For example:
FILE="c:\mydir\scorecode.sas".

Default: None

**FORMAT=**

Specifies the numeric format to be used when printing numeric constants. For example, FORMAT=BEST20.

Default: FORMAT=BEST12

**GROUP=**

Specifies the group identifier (up to four characters) for group processing.

Default: GROUP=_

**METABASE=**

Specifies a catalog entry to which the code is written. For example, METABASE=mylib.mycat.myentry.

Default: None

**RESIDUAL**

Specifies that variables that depend on the target variable, such as the BL_, BP_, CL_, and CP_ variables, are to be computed in the code.
DECISION PROCEDURE

DECISION Statement

Specifies information used for decision processing in the DECIDE, DMREG, NEURAL, and SPLIT procedures. This documentation applies to all four procedures.

Tip: The DECISION statement is required in the DECIDE procedure, but not in the DMREG, NEURAL, or SPLIT procedures.

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

DECDATA= <libref.> SAS-data-set

Specifies the input data set that contains the decision matrix and/or prior probabilities. The DECDATA= data set must also contain the target variable.

The DECDATA= data set may contain decision variables specified by means of the DECVARS= option, or prior probability variable(s) specified by means of the PRIORVAR= option and/or the OLDPRIORVAR= option, or both.

The target variable is specified by means of the TARGET statement in the DECIDE, NEURAL, and SPLIT procedures or by using the MODEL statement in the DMREG procedure.

For a categorical target variable, there should be one row for each class. The value in the decision matrix located at a given row and column specifies the consequence of making the decision corresponding to column when the target class corresponds to the row. If any class appears twice or more in the DECDATA= data set, an error message is printed and the procedure terminates. For the DMREG, NEURAL, and SPLIT procedures, all class values in the training set must also appear in the DECDATA= data set, but it is allowed to have class values in the DECDATA= data set that are not in the training set. For the DECIDE procedure, any class value in the DATA= data set that is not found in the DECDATA= data set is treated in the same way as a missing class value; it is allowed to have class values in the DECDATA= data set that are not in the DATA= data set, but note that the classes in the DECDATA= data set must correspond exactly with the variables in the POSTERIORS statement.

For an interval target variable, each row defines a knot in a piecewise linear spline function. The consequence of making a decision is computed by interpolating in the corresponding column of the decision matrix. If the predicted target value is outside the range of knots in the decision matrix, the consequence of a decision is computed by linear extrapolation. If the target values are in nondecreasing or nonincreasing order, any interior target value is allowed to appear twice in the data set so you can specify discontinuities. The end points (that is, the minimum and maximum target values in the data set) may not appear more than once. No target value is allowed to appear more than twice. If the target values are not in nondecreasing or nonincreasing order, the target
values are sorted by the procedure, and no target value may appear more than once.

**Tip:** The DECDATA= data set may be of TYPE=LOSS, PROFIT, or REVENUE. If unspecified, TYPE=PROFIT is assumed by default. TYPE= is a data set option that can be specified in parenthesis following the data set name when the data set is created or when the data set is used.

---

### Options

**DECVARS=**`decision-variable(s)`

Specifies the numeric decision variables in the DECDATA= data set that contain the target-specific consequences for each decision. The decision variables may not contain missing values. If DECVARS= is not specified, the procedure does not make any decisions or output any variables that depend on making a decision.

**Default:** None

**COST=**`cost-option(s)`

Specifies numeric constants that give the cost of a decision, or numeric variables in the DATA= data set that contain the case-specific costs, or any combination of constants and variables. There must be the same number of cost constants and variables as there are decision variables in the DECVARS= option. In the COST= option, you may not use abbreviated variable lists such as D1-D3, ABC--XYZ, or PQR:. For any case where a cost variable is missing, the results for that case are set to missing.

**Default:** All costs are assumed to be 0.

**Note:** The COST= option may only be specified when the DECDATA= data set is of TYPE=REVENUE.

**PRIORVAR=**`variable`

Specifies the numeric variable in the DECDATA= data set that contains the prior probabilities to use for making decisions. Prior probabilities are also used to adjust the total and average profit or loss. Prior probabilities may not be missing or negative, and there must be at least one positive prior probability. The priors are not required to sum to 1; if they do not sum to 1, they are automatically multiplied by a constant to do so. If PRIORVAR= is not specified, no adjustment for prior probabilities is applied to the posteriors.

**Default:** None

**OLDPRIORVAR=**`variable`

Specifies the numeric variable in the DECDATA= data set that contains the prior probabilities that were used when originally fitting the model. The OLDPRIORVAR= option is used only by the DECIDE procedure. In the DMREG, NEURAL, and SPLIT procedures, the procedure automatically supplies the values of the old priors if PRIORVAR= is specified.

**Note:** If OLDPRIORVAR= is specified, PRIORVAR= must also be specified.
The DECIDE Procedure

FREQ Statement

Specifies a numeric variable whose values represent the frequency of the observation.

Discussion: If one variable in the input data set represents the frequency of occurrence for other values in the observation, specify the variable's name in a FREQ statement.

Alias: FREQUENCY

FREQ variable;

Required Argument

variable

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

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The POSTERIORS Statement

Specifies which variables in the DATA= data set contain the estimated posterior probabilities that correspond to the categories of the target variable.

Discussion: The POSTERIORS statement may be specified only with a categorical target variable. You may not use both a POSTERIORS statement and a PREDICTED statement.

POSTERIORS variable(s);

variable(s)

Specifies the numeric variable(s) in the DATA= data set that contain the estimated posterior probabilities corresponding to the classes (that is, the categories of the target variable). The results for a case are set to missing and the P flag is set in the _WARN_ variable for any case where a posterior probability is missing, negative, or greater than one, or there is a non-0 posterior corresponding to a 0 prior, or there is not at least one valid positive posterior probability.

CAUTION: The order of the variables must correspond exactly to the order of the classes in the DECDATA= data set.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
## PREDICTED Statement

Specifies which variable in the DATA= data set contains the predicted values of an interval target variable.

**Discussion:** The PREDICTED statement may be specified only with an interval target variable. You may not use both a PREDICTED statement and a POSTERIORS statement.

```
PREDICTED variable;
```

*variable*

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

---

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DECIDE Procedure

TARGET Statement

Specifies which variable in the DECDATA= data set contains values for the target variable.

**Discussion:** The DECIDE procedure will search for a target variable with the same name in the DATA= data set. If none is found, then the DECIDE procedure will assume that the actual target values are unknown. For a categorical target, the target variables in the DATA= and DECDATA= data sets need not be of the same type because the normalized formatted values are used for comparisons. For an interval target, both variables must be numeric. If scoring code is generated by a CODE statement, the code will format the target variable using the format and length from the DATA= data set.

**Tip:** The TARGET statement is required.

```
TARGET variable;
```

*variable*

Specifies the variable in the DECDATA= data set that contains the values for the target variable.

---

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Details

**Note:** Formulas for adjusting posterior probabilities and for decision processing are given in the chapter on "Predictive Modeling."
Example

Example 1: Using the DECIDE Procedure Following the DISCRIM Procedure

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Example 1: Using the DECIDE Procedure Following the DISCRIM Procedure

This example shows how to use the DECIDE procedure to adjust posterior probabilities from the DISCRIM procedure, and how to make decisions using a revenue matrix and cost constants.

In a population of men who consult urologists for prostate problems, 70% have benign enlargement of the prostate, 25% have an infection, and 5% have cancer. A sample of 100 men is taken, and two new diagnostic measures, X and Y, are made on each patient. The training set also includes the diagnosis made by reliable, conventional methods. For each patient, two treatments are available: 1) Antibiotics are effective against infection, but may have moderately bad side effects. Antibiotics have no effect on benign enlargement or cancer. 2) Surgery is effective for all diseases but has potentially severe side effects such as impotence. There is also the option of doing nothing.

Note: This example is purely fictional. Any resemblance to actual medical conditions or treatments is coincidental.

data prostate;
  length dx $10;
  dx='Benign';
  mx=30; sx=10;
  my=30; sy=10;
  n=70;
  link generate;

  dx='Infection';
  mx=70; sx=20;
  my=35; sy=15;
  n=25;
  link generate;

  dx='Cancer';
  mx=50; sx=10;
  my=50; sy=15;
  n=5;
  link generate;
stop;

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

run;

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

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

title2 'Classification with equal priors';
proc plot data=outdis; plot y*x=_into_; run;

data rx(type=revenue);
   input dx $10. eqprior prior nothing antibiot surgery;
   datalines;
Benign            .33      70     0        0        5
Infection         .33      25     0       10       10
Cancer            .33       5     0        0      100
;

proc decide data=outdis out=outdec outstat=sumdec;
   target dx;
   posteriors benign infectio cancer;
   decdata=rx
   decision
      oldpriorvar=eqprior priorvar=prior
      decvars=nothing antibiot surgery
      cost=     0       5       20;
run;

title2 'Treatment: Cost of surgery=20';
proc print data=sumdec label; run;
proc plot data=outdec; plot y*x=d_rx; run;
proc decide data=outdis out=outdec2 outstat=sumdec2;
target dx;
  posterior benign infection cancer;
decision decdata=rx
  oldpriorvar=eqprior priorvar=prior
  decvars=nothing antibiot surgery
  cost=  0     5     50;
run;

title2 'Treatment: Cost of surgery=50';
proc print data=sumdec2 label; run;
proc plot data=outdec2; plot y*x=d_rx; run;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Use DISCRIM to see how well inputs X and Y can classify each patient according to disease.

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

   title2 'Classification with equal priors';
   proc plot data=outdis; plot y*x=_into_; run;
```
The following DATA step creates a decision data set containing prior probabilities and a revenue matrix. The revenue matrix indicates the benefit of each treatment. The costs of each treatment (such as bad side effects) will be specified later in a DECISION statement.

The variables are: EQPRIOR = The prior probabilities used by DISCRIM (equal, by default), PRIOR = The known proportions from the population, NOTHING = The benefit of doing nothing (0), ANTIBIOT = The benefit of using antibiotics (cures infection, no benefit for other diseases), and SURGERY = The benefit of surgery (cures all diseases).

data rx(type=revenue);
  input dx $10. eqprior prior nothing antibiot surgery;
datalines;
Benign .33  70  0  0  5
Infection .33  25  0 10  10
Cancer .33  5  0  0 100
;
Use PROC DECIDE to assign a treatment to each patient.

The cost associated with each treatment is: NOTHING = 0 cost, ANTIBIOT = 5 for possible bad side effects, and SURGERY = 20 for possible severe side effects.

```
proc decide data=outdis out=outdec outstat=sumdec;
  target dx;
  posteriors benign infectio cancer;
  decision decdata=rx
    oldpriorvar=eqprior priorvar=prior
    decvars=nothing antibiot surgery
    cost=     0       5       20;
run;
```

title2 'Treatment: Cost of surgery=20';
proc print data=sumdec label; run;
proc plot data=outdec; plot y*x=d_rx; run;
```
References


The DMDB Procedure

Overview

Procedure Syntax

PROC DMDB Statement
CLASS Statement
FREQ Statement
ID Statement
TARGET Statement
VARIABLE Statement
WEIGHT Statement

Details

Examples

Example 1: Getting Started with the DMDB Procedure
Example 2: Specifying a FREQ Variable

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DMDB Procedure

Overview

SAS/Enterprise Miner architecture is based on the creation of a data mining database (DMDB) that is a snapshot of the original data. PROC DMDB creates this DMDB from the input data source. It also compiles and computes metadata information about the input data based on variable roles and stores it in a metadata catalog. The DMDB and the associated metadata catalog facilitate subsequent data mining activities. They are both designed for processing and storage efficiencies.

Note: The DMBCAT= argument is required for the following procedures: ASSOC, DMDB, DMINE, DMSPLIT, SPLIT, and SEQ. It is optional in the NEURAL and DMREG procedures. It is not a valid argument in the RULEGEN and STDIZE procedures.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
PROC DMDB <option(s)>;

CLASS variable(s) <ORDER=order-option(s)>;

ID variable(s);

FREQ variable;

TARGET variable(s);

VARIABLE variable(s) </WEIGHT=weight-variable>;

WEIGHT variable;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DMDB Procedure

PROC DMDB Statement

Invokes the DMDB procedure.

PROC DMDB <option(s)>;

Required Arguments

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

Names the SAS data set containing the information that you want added to the data mining database.

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

Names the metadata catalog that is created or updated by PROC DMDB.

Refer to Details for more information.

Options

**BATCH | NOMETAIN**

Specifies the creation of a new metadata catalog as specified in the DMDBCAT= option, instead of updating an existing metadata catalog. Any existing catalog with this name will be overridden and replaced by system-generated information.

**MAXLEVEL=**integer

Specifies the maximum number of class levels to be processed.

<table>
<thead>
<tr>
<th>Default:</th>
<th>MACINT. (If an integer greater than MACINT is specified, the integer specified for MAXLEVEL is ignored and MACINT is used.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range:</td>
<td>integer $\geq$ 3</td>
</tr>
</tbody>
</table>

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

Names the data mining database (DMDB) that you want created. The new DMDB contains each of the ID, VAR, and FREQ variables - copied 'as is' from the DATA= data set. The DMDB also contains each of the CLASS variables written out as their corresponding integer class level value in 5-bytes (sizeof (float)+1).

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

Names the previously created data mining database that you want updated with the observations given in the DATA= data set. At the same time, these observations update the information maintained in the DMDBCAT= option, reflecting the revised statistics from the additional input
records.

**VARDEF=divisor**

Specifies the divisor to use in the calculation of the variance and standard deviation. The following table shows the possible values for VARDEF:

**Values for VARDEF=**

<table>
<thead>
<tr>
<th>Value</th>
<th>Divisor</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF</td>
<td>degrees of freedom</td>
<td>n - 1</td>
</tr>
<tr>
<td>N</td>
<td>number of observations</td>
<td>n</td>
</tr>
<tr>
<td>WDF</td>
<td>sum of weights minus one</td>
<td>((\sum w_i)) - 1</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>sum of weights</td>
<td>(\sum w_i)</td>
</tr>
</tbody>
</table>

**Default:** DF (degrees of freedom)

**Tip:** When you use the WEIGHT statement, you can specify VARDEF=WDF to get an estimate of the variance of an observation using the average observation weight.

**CAUTION:**

**Using a Divisor with the WEIGHT statement.** If you use a WEIGHT statement and you calculate the variance or standard deviation, you may want to specify VARDEF=WEIGHT. Use WEIGHT as the divisor to compute an approximate estimate of the variance of an observation using the average observation weight. Use VARDEF=DF to compute the standard error of the weighted mean.
The DMDB Procedure

CLASS Statement

Specifies the variables whose values define subgroup combinations for the analysis.

INTERACTION: CLASS, ID, and VAR statements are mutually exclusive.

CLASS variable(s) <ORDER=order-option(s)>;

Required Argument

variable(s)

Specifies one or more categorical variables to be used in the analysis. For each CLASS variable, the metadata contains information on each of the following: its class level value, its frequency, and its ordering information.

Range: variable(s) can be character or numeric.

Options

ORDER

 Specifies the order to use when considering the levels of classification variables (specified in the CLASS statement) to be sorted. order-option(s) may be one of the following:

ASCENDING | ASC

Class levels are arranged in lowest-to-highest order of unformatted values.

DESCENDING | DESC

Class levels are arranged in highest-to-lowest order of unformatted values.

ASCFORMATTED | ASCFMT

Class levels are arranged in ascending order by their formatted values.

DESFORMATTED | DESFMT

Class levels are arranged in descending order by their formatted values.

DSORDER | DATA

Class levels are arranged according to the order of their appearance in the input data set.

Default: ASCENDING

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DMDB Procedure

FREQ Statement

Specifies a numeric variable that contains the frequency of each observation.

FREQ variable;

Required Argument

variable

Specifies a numeric variable whose value represents the frequency of the observation.

Default: If variable is 0 or missing, then the observation is omitted in the DMDB and is not included in statistical calculations.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
ID Statement

Includes additional variables in the output data set.

**INTERACTION:** CLASS, ID, and VAR statements are mutually exclusive.

ID variable(s);

**Required Argument**

`variable(s)`

Identifies one or more variables from the input data set whose maximum values for groups of observations are included in the output data set by PROC DMDB.

**Range:** `variable(s)` can be character or numeric.
TARGET Statement

Specifies variables to be created for the output data set.

TARGET variable(s);

Required Argument

variable(s)

Identifies TARGET variables. Variables must also be specified in the VAR or CLASS variable lists.

Range: variable(s) can be character or numeric.
**VARIABLE Statement**

Identifies the analysis variables and their order in the results.

**INTERACTION:** CLASS, ID, and VAR statements are mutually exclusive.

**Alias:** VAR

VARIABLE variable(s) */ WEIGHT=weight-variable */;

**Required Argument**

variable(s)

Identifies the analysis variables and specifies their order in the results.

The metadata catalog contains the following statistics for the VAR variables: N, NMISS, MIN, MAX, SUM, SUMWGT, CSS, USS, STD, SKEWNESS, and KURTOSIS.

**Default:** If you omit the VAR statement, PROC DMDB analyzes all numeric variables not listed in the other statements.

**Range:** variable(s) are numeric only.

**Options**

**WEIGHT=weight-variable**

Specifies a numeric variable whose values weight the variables specified in the VAR statement.

**Tip:** When you use the WEIGHT= option, consider which value of the VARDEF= option is appropriate. See the [VARDEF= option](#) for more information.
WEIGHT Statement

Specifies weights for observations in the statistical calculations.

Interaction: If you use the WEIGHT statement, you cannot compute SKEWNESS and KURTOSIS.

WEIGHT variable;

Required Argument

variable

Specifies a numeric variable whose value weights the analysis variables in each observation. The variable value does not have to be an integer.

Default: If the value of variable is less than 0 or is missing, the procedure uses a value of 0.

Range: variable can be an integer or non-integer number.
Details

The data mining database (DMDB) is maintained as a SAS data set. The metadata information associated with the DMDB is maintained in a SAS catalog. Metadata includes overall data set information as well as statistical information for the variables according to their roles. For each CLASS variable, the metadata contains information on each of the following: its class level value, its frequency, and its ordering information. In the DMDB, the CLASS variables are stored as integers 0, 1, 2, ..., which can be mapped into different class level values.

For each VAR variable, the metadata catalog contains the following statistics:

- **N**
  - The number of observations with nonmissing values of the variable
- **NMISS**
  - The number of observations with missing values of the variable
- **MIN**
  - The minimum
- **MAX**
  - The maximum
- **SUM**
  - The sum of all the nonmissing values of the variable
- **SUMWGT**
  - The sum of weights
- **CSS**
  - The corrected sum of squares
- **USS**
  - The uncorrected sum of squares
- **STD**
  - The standard deviation
- **SKEWNESS**
  - Measure of the tendency for the distribution of values to be more spread out on one side of the mean than on the other
- **KURTOSIS**
  - Measure of the "heaviness of the tails"

(Refer to the SAS Procedures Guide, Chapter 1 for formulas and other details.)

DMDBs are only created for training data and should not be used for validation or test during modeling.
Examples

The following examples were executed using the HP-UX version 10.20 operating system and the SAS software release 6.12TS045.

Example 1: Getting Started with the DMDB Procedure

Example 2: Specifying a FREQ Variable

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Example 1: Getting Started with the DMDB Procedure

Features:
- Specifying the Output DMDB Data Set and Catalog
- Defining the Numeric Variables in a VAR Statement
- Defining the Class Variables in a Class Statement
- Setting the Order of the Class Variables
- Defining the Target Variable in a Target Statement

This example demonstrates how to create a data mining database (DMDB) data set and catalog. The example uses the fictitious mortgage data set name SAMSIO.HMEQ. The data set contains 5,960 cases. Each case represents an applicant for a home equity loan. All applicants have an existing mortgage. The binary target BAD indicates whether or not an applicant eventually defaulted or was ever seriously delinquent. There are ten numeric inputs and two class inputs available for subsequent modeling.

Program

```sas
proc dmdb batch data=sampsio.hmeq
  out=dmhmeq
dmdbcat=cathmeq;

var loan derog mortdue value yoj delinq clage ninq clno debtinc;

class bad(desc)
  reason(ascending)
  job;

target bad;
run;
```
**Log**

```
proc dmdb batch data=sampsio.hmeq
   out=dmhmeq
dmdbcat=cathmecq;

var loan derog mortdue value yoj delinq
   clage ninq clno debtinc;

class bad(desc)
   reason(ascending)
   job;

target bad;
run;
```

Records processed= 5960  Mem used = 511K.
NOTE: The PROCEDURE DMDB used 0:00:08.30 real 0:00:02.85 cpu.
The PROC DMDB statement invokes the procedure. The BATCH option requests the creation of a new DMDB catalog. The DATA= option specifies the input data set.

```plaintext
proc dmdb batch data=sampsio.hmeq
```
The OUT= option specifies the name of the output DMDB data set. The DMDBCAT= option specifies the name of the output DMDB catalog.

```
out=dmhmeq
dmdbcat=cathmeq;
```
The VAR statement identifies the numeric analysis variables. If you omit the VAR statement, PROC DMDB analyzes all numeric variables not listed in other statements.

```
var loan derog mortdue value yoj delinq
   clage ning clno deptinc;
```
The CLASS statement specifies the categorical variables to be used in the analysis. The ORDER option specifies the order to use when considering the levels of the classification variables. Valid ORDER options include ASCENDING (ASC), DESCENDING (DESC), ASCFORMATTED (ASCFMT), DESFORMATTED (DESFMT), or DSORDER (DATA). The default for the ORDER option is set to ASCENDING.

class bad(desc)
  reason(ascending)
  job;

The TARGET statement identifies the target (response) variable.

    target bad;
run;
Example 2: Specifying a FREQ Variable

This example demonstrates how to define a FREQ variable in the DMDB data set and catalog. A FREQ variable represents the frequency of occurrence for other values in each observation of the input data set. The DATA step required to create the WORK.FREQEX input data set is provided.

**Program**

```plaintext
data freqex;
  input  count X1 X2 X3 Y ;
datalines;
3   -0.17339  -0.04926  -0.61599  0
2   -1.51586   0.31526  -1.65430  1
1    1.04348   0.64517  -0.06878  0
1   -1.74298   0.02592  -0.71203  1
1    0.07806   1.45284  -0.39064  1
4    0.20073   0.22533  -0.44507  0
1   -0.08887  -1.24641  -0.73156  0
1    0.10309   0.88542  -1.63595  1
2   -0.57030  -1.35613  -1.58209  0
1   -1.39170  -1.22333   1.98124  1
2    0.51356  -0.36128   0.77962  0
1   -0.89216  -0.01054  -0.76720  0
1   -0.09882   1.43263   0.53820  0
3    0.03225  -0.17737   0.25381  0
1   -0.14203  -1.64183  -0.34028  0
1   -0.24436  -0.83537  -2.00245  0
2   -0.78277   0.00284  -0.75016  0
1    0.77732  -0.28847  -0.77437  0
1   1.55172  -0.21167  -0.53833  0
2   -0.74054  -1.23276   0.11452  1
run;

proc dmdb batch data=freqex out=dmfout dmdbcat=outfcat;
  var x1 x2 x3;
  class y(desc);
  target y;
```

---

**Features**

- Specifying a FREQ variable with the FREQ Statement
freq count;
run;

Log

1  data freqex;
2   input  count X1 X2 X3 Y ;
3   datalines;
NOTE: The data set WORK.FREQEX has 20 observations and 5 variables.
NOTE: The DATA statement used 0:00:00.44 real 0:00:00.15 cpu.
24  run;
25  proc dmdb batch data=freqex out=dmfout dmdbcat=outfcat;
26     var x1 x2 x3;
27     class y(desc);
28     target y;
29
30     freq count;
31  run;
Records processed=      20  Mem used = 511K.
NOTE: The PROCEDURE DMDB used 0:00:00.92 real 0:00:00.27 cpu.
The FREQ statement specifies the numeric variable that contains the frequency of each observation.

\[
\text{freq count;}
\]
\[
\text{run;}
\]
The DMINE Procedure

Overview

Procedure Syntax

PROC DMINE Statement
FREQ Statement
TARGET Statement
VARIABLES Statement
WEIGHT Statement

Details

Examples

Example 1: Modeling a Continuous Target with the DMINE Procedure (Simple Selection Settings)
Example 2: Including the AOV16 and Grouping Variables into the Analysis (Detailed Selection Settings)
Example 3: Modeling a Binary Target with the DMINE Procedure

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Overview

Many data mining databases have hundreds of potential model inputs (independent variables). The DMINE procedure enables you to quickly identify the input variables that are useful for predicting the target variable(s) based on a linear models framework. The procedure facilitates ordinary least squares or logistic regression methods. (Logistic regression is a form of regression analysis in which the response variable represents a binary or ordinal-level response.)

PROC DMINE and PROC DMSPLIT are underlying procedures for the Variable Selection node.
PROC DMINE <option(s)>;

FREQ variable;

TARGET variable;

VARIABLES variable-list;

WEIGHT variable;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DMINE Procedure

PROC DMINE Statement

Invokes the DMINE procedure.

PROC DMINE <option(s)>;

Required Arguments

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

Identifies the input data set generated by PROC DMDB. The data set is associated with a valid catalog specified by the DMDBCAT= option. This option must be specified; no default is permitted. The DATA= data set must contain interval scaled variables and CLASS variables in a specific form written by PROC DMDB.

**Default:** none

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

Identifies an input catalog of meta information generated by PROC DMDB. The information is associated with a valid data set specified by the DATA= option. The catalog contains important information (for example, the range of variables, number of missing values of each variable, moments of variables) that is used by many other Enterprise Miner procedures that require a DMDB data set. The DMDBCAT= catalog and the DATA= data set must be appropriately related to each other in order to obtain proper results.

**Default:** None.

Options

**NOAOV16**

By default, the DMINE procedure creates the AOV16 variables, calculates their R-squares with the target variable, and then uses the remaining significant variables in the final forward stepwise selection process. The interval scaled variables are grouped into categories to create the AOV16 variables. The range of interval scaled variables can be equally divided into 16 categories and each observation (value) of the variable is then mapped into one of these categories. The NOAOV16 option prevents the procedure from including the AOV16 variables in the final stepwise selection process. Note that the R-square value is calculated for each AOV16 variable even if you specify the NOAOV16 option.
**DMINE procedure**

The DMINE procedure organizes numeric variables into 16 equally-spaced groups or bins called AOV16 variables. The AOV16 variables are created to help identify non-linear relationships with the target. Bins that have zero observations are eliminated; therefore, an AOV16 variable can have fewer than 16 bins.

**Default:** Create the AOV16 variables. Note that there is not an AOV16 option, only a NOAOV16 option to prevent these variables from being used in the final forward stepwise selection process.

---

**NOINTER**

Specifies not to consider interactions between categories (that is, a two-way interaction) of CLASS variables in the process of variable selection.

**Definition:** A two-way interaction measures the effect of a classification input variable across the levels of another classification variable. For example, credit worthiness may not be consistent across job classifications. The lack of uniformity in the response may signify a credit worthiness by job interaction.

**Default:** Two-way interactions between categories of the class variables are considered in the variable selection process. Note that the two-way interactions can dramatically increase the processing time of the DMINE procedure.

---

**MAXROWS=value**

Specifies the upper bound for the number of independent variables selected for the model. This is an upper bound for the number of rows and columns of the X'X matrix of the regression problem.

**Default:** 3000. This means that for most models, the MINR2 and STOPR2 settings will determine the number of selected independent variables. The X'X matrix used for the stepwise regression requires \( n \times \left( \frac{n+1}{2} \right) \) double precision values storage in RAM, where \( n \) is the number of rows in the matrix. (This corresponds to 3000 * 1500 * 8 bytes (which is about 36 megabytes) of RAM needed for storage.)

---

**MINR2=value**

Specifies a lower bound for the individual R-square value of a variable to be eligible for the model selection process. Variables with R-square values greater than or equal to value are included in the selection process.

**Definition:** R-square is the ratio of the model sum of squares (SS) to the total sum of squares. It measures the sequential improvement in the model as input variables are selected.

**Default:** \( 10^{-5} \)
NOMONITOR
Suppresses the output of the status monitor that indicates the progress made in the computations.

**Default:** The output of the status monitor is displayed.

NOPRINT
Suppresses all output printed in the output window.

**Default:** The output is printed to the output window.

STOPR2=value
Specifies a lower value for the incremental model R-square value at which the variable selection process is stopped.

**Default:** $5 \times 10^{-5}$

USEGROUPS
PROC DMINE automatically tries to reduce the levels of each class variable to a group variable based on the relationship with the target. By doing so, observations of class variables with many categories (for example, ZIP codes) can be mapped into groups of fewer categories. If you specify the USEGROUPS option, and a class variable can be reduced to a group variable, then only the group version of the variable is considered in the model. If you omit the USEGROUPS option, then both the group variable and the original class variable are allowed in the model.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
FREQ Statement

Alias: FREQUENCY

Tip: Specify the FREQ variable in PROC DMDB so that the information is saved in the catalog and so that the variable is automatically used as a FREQ variable in PROC DMINE. This also ensures that the FREQ variable is automatically used by all other Enterprise Miner procedures in the project.

FREQ variable;

Required Argument

variable

Specifies one numeric (interval-scaled) FREQUENCY variable.

Range: Any integer. A noninteger value is truncated.
TARGET Statement

```plaintext
TARGET variable;
```

Required Argument

`variable`

Specifies the output variable. One variable name can be specified identifying the target (response) variable for the two regressions.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
VARIABLES Statement

Alias: VAR

VARIABLES variable-list;

Required Argument

variable-list

Specifies all the variables (numeric and categorical, that is, INTERVAL and CLASS) that can be used for independent variables in the prediction or modeling of the target variable.
WEIGHT Statement

Alias: WEIGHTS

Tip: Specify the WEIGHT variable in PROC DMDB so that the information is saved in the catalog and so that the variable is used automatically as a WEIGHT variable in PROC DMINE.

WEIGHT variable;

Required Argument

variable

Specifies one numeric (interval-scaled) variable that is used to weight the input variables.
The DMINE Procedure

Details

PROC DMINE performs the following two tasks:

1. PROC DMINE first computes a forward stepwise least-squares regression. In each step, an independent variable is selected, which contributes maximally to the model R-square value. Two parameters, MINR2 and STOPR2, can be specified to guide the variable selection process.
   - **MINR2**
     
     If a variable has an individual R-square value smaller than MINR2, the variable is not considered for selection into the model.
   - **STOPR2**
     
     A second test is performed using the STOPR2 value: the remaining independent variable with the largest contribution to the model R-square is added to the model. If the resulting global R-square value changes from its former value by less than the STOPR2 value, then the stepwise regression is terminated.

2. For a binary target (CLASS response variable), a fast algorithm for (approximate) logistic regression is computed in the second part of PROC DMINE. The independent variable is the prediction from the former least squares regression. Since only one regression variable is used in the logistic regression, only two parameters are estimated, the intercept and slope. The range of predicted values is divided into a number of equidistant intervals (knots), on which the logistic function is interpolated.

   If NOPRINT is not specified, a table is printed indicating the accuracy of the prediction of the target.

Missing Values

Missing values are handled differently, depending on the type of variable.

- Missing values in categorical variables are replaced with a new category that represents missing values.
- Missing values in noncategorical variables are replaced with the mean.
- Observations with missing target values are dropped from the data.
Examples

The following examples were executed on the Windows NT operating system; the version of the SAS System was 6.12TS045.

Example 1: Modeling a Continuous Target with the DMINE Procedure (Simple Selection Settings)

Example 2: Including the AOV16 and Grouping Variables into the Analysis (Detailed Selection Settings)

Example 3: Modeling a Binary Target with the DMINE Procedure
Example 1: Modeling a Continuous Target with the DMINE Procedure (Simple Selection Settings)

Features:
- Setting the MINR2= and STOPR2= cutoff values.
- Specifying the target and input variables.
- Excluding the AOV16 variables by specifying the NOAOV16 option.
- Excluding the two-way class interactions by specifying the NOINTER option.

As a marketing analyst at a catalog company, you want to quickly identify the inputs that best predict the dollar amount that customers will purchase from your new fall outerwear catalog. The fictitious catalog mailing data set is named SAMPSIO.DMEXA1 (stored in the sample library). The data set contains 1,966 customer cases. The interval target AMOUNT contains the purchase amount in dollars.

There are 48 input variables available for predicting the target. Note that PURCHASE is a binary target that is modeled in "Example 3: Modeling a Binary Target with the DMINE Procedure". ACCTNUM is an id variable, which is not a suitable input variable.

Program

```
proc dmdb batch data=sampsio.dmexa1 out=dmbexa1 dmdbcat=catexa1;
  id  acctnum;
  var  amount income homeval frequent recency age
domestic apparel leisure promo7 promo13 dpm12
county return mensware flatware homeacc lamps
linens blankets towels outdoor coats wcoat
wappar hhappar jewelry custdate numkids travtime job;
class purchase(desc) marital ntitle gender telind
  aprtmnt snglmom mobile kitchen luxury dishes tmktord
  statecod race origin heat numcars edlevel;
run;
```

```
proc dmine data=dmbexa1 dmdbcat=catexa1
  minr2=0.020 stopr2=0.0050
  noaov16
  nointer;
```
var income homeval frequent recency age
domestic apparel leisure promo7 promo13 dpm12
county return mensware flatware homeacc lamps
linens blankets towels outdoor coats wcoat
wappar hhappar jewelry custdate numkids travtime job
marital ntitle gender telind aprtmmt snglmom mobile
kitchen luxury dishes tmktord statecod race origin heat
numcars edlevel;

target amount;
title 'DMINE: Continuous Target';
run;

Output

DMINE Status Monitor

When you invoke the DMINE procedure, the Dmine Status Monitor window appears. This window monitors the
execution time of the procedure. To suppress the display of this window, specify the NOMONITOR option on the PROC
DMINE statement.
Partial Listing of R-Squares for Target Variable

This section of the output ranks all model effects by their R-square values. The degrees of freedom (DF) associated with each effect is also listed. Effects that have an R-square value less than the MINR2 = cutoff value are not chosen for the model. These effects are labeled as "R2 < MINR2" in the table. The remaining significant variables are analyzed in a subsequent forward selection regression.

There are four types of model effects:

- **Class** effects are estimated for each class variable and all possible two-factor interactions. The R-square statistic is calculated for each class effect using a one-way analysis of variance. Two-factor interaction effects are constructed by combining all possible levels of each class variable into one term. Because the NOINTER option was specified, the two-factor interactions are not used in the final forward stepwise regression. The degrees of freedom for a class effect is equal to: (the number of unique factor levels minus 1). For two-factor interactions, the degrees of freedom is equal to: (the number of levels in factor A multiplied by the number of levels in factor B minus 1).

- **Group** effects are created by reducing each class effect through an analysis of means. The degrees of freedom for each group effect is equal to the number of levels.

- **VAR** effects are estimated from interval variables as standard regression inputs. A simple linear regression is performed to determine the R2 statistic for interval inputs. The degrees of freedom is always equal to 1.

- **AOV16** effects are calculated as a result of grouping numeric variables into a maximum of 16 equally spaced buckets. AOV16 effects may account for possible non-linearity in the target variable AMOUNT. The degrees of freedom are calculated as the number of groups. Because the NOAOV16 option was specified, the AOV16 variables are not used in the final forward stepwise regression.

Note that the original input LEISURE has the largest R-square statistic with the target. Several AOV16 and group variables have large R-square values, but these effects are not used in the final forward stepwise regression.

### DMINE: Continuous Target

R-Squares for Target variable: AMOUNT

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var: LEISURE</td>
<td>1</td>
<td>0.4827</td>
</tr>
<tr>
<td>AOV16: APPAREL</td>
<td>12</td>
<td>0.4762</td>
</tr>
<tr>
<td>Class: KITCHEN*STA</td>
<td>8</td>
<td>0.4268</td>
</tr>
<tr>
<td>Group: KITCHEN*STATECOD</td>
<td>9</td>
<td>0.4210</td>
</tr>
<tr>
<td>Class: KITCHEN*LUXURY</td>
<td>16</td>
<td>0.4019</td>
</tr>
<tr>
<td>Var: APPAREL</td>
<td>1</td>
<td>0.4001</td>
</tr>
<tr>
<td>Group: KITCHEN*LUXURY</td>
<td>5</td>
<td>0.3959</td>
</tr>
<tr>
<td>AOV16: DOMESTIC</td>
<td>15</td>
<td>0.3869</td>
</tr>
<tr>
<td>Var: DOMESTIC</td>
<td>1</td>
<td>0.3652</td>
</tr>
<tr>
<td>AOV16: FREQUENT</td>
<td>11</td>
<td>0.3418</td>
</tr>
<tr>
<td>Class: LUXURY*STATECOD</td>
<td>101</td>
<td>0.3335</td>
</tr>
<tr>
<td>Group: LUXURY*STATECOD</td>
<td>6</td>
<td>0.3284</td>
</tr>
<tr>
<td>Class: LUXURY*TMKTORD</td>
<td>7</td>
<td>0.3212</td>
</tr>
<tr>
<td>Group: LUXURY*TMKTORD</td>
<td>4</td>
<td>0.3177</td>
</tr>
<tr>
<td>Class: KITCHEN*DISHERS</td>
<td>42</td>
<td>0.3128</td>
</tr>
<tr>
<td>Class: MARITAL*KITCHEN</td>
<td>17</td>
<td>0.3084</td>
</tr>
<tr>
<td>Group: KITCHEN*DISHERS</td>
<td>5</td>
<td>0.3066</td>
</tr>
<tr>
<td>Var: FRENCH</td>
<td>1</td>
<td>0.3048</td>
</tr>
<tr>
<td>Class: TMKTORD*STATECOD</td>
<td>110</td>
<td>0.3046</td>
</tr>
<tr>
<td>Group: MARITAL*KITCHEN</td>
<td>5</td>
<td>0.3033</td>
</tr>
</tbody>
</table>
Group: TMKTORD*STATECOD 8 0.2995
Class: KITCHEN*TMKTORD 26 0.2921

Group: KITCHEN*TMKTORD 5 0.2870
AOV16: DPM12 12 0.2770
Class: NTITLE*KITCHEN 28 0.2717
Group: NTITLE*KITCHEN 6 0.2690
Class: KITCHEN*RACE 25 0.2594
Class: KITCHEN*EDLEVEL 26 0.2572
Group: KITCHEN*RACE 5 0.2568
Class: LUXURY*DISHES 15 0.2547
Group: KITCHEN*EDLEVEL 6 0.2533
Group: LUXURY*DISHES 3 0.2504
Class: LUXURY*RACE 8 0.2455
Group: LUXURY*RACE 2 0.2447
Class: KITCHEN*ORIGIN 35 0.2438
Class: LUXURY*ORIGIN 11 0.2419
Class: NTITLE*LUXURY 7 0.2419
Group: KITCHEN*ORIGIN 6 0.2407
Group: NTITLE*LUXURY 2 0.2406
Class: LUXURY*NUMCARS 6 0.2394
Class: APRTMNT*LUXURY 3 0.2389
Class: TELIND*LUXURY 3 0.2386
Class: LUXURY*EDLEVEL 7 0.2384

Additional effects are not listed

R-Squares for Target variable: AMOUNT

Group: DISHES*NUMCARS 7 0.0288
Class: SNGLMOM*DISHES 13 0.0283
Group: SNGLMOM*DISHES 5 0.0279
Class: DISHES 9 0.0271
Group: DISHES 4 0.0268
Var: COATS 1 0.0228
AOV16: FLATWARE 10 0.0221
Var: FLATWARE 1 0.0196 R2 < MINR2
AOV16: RETURN 2 0.0195 R2 < MINR2
Class: NTITLE*RACE 17 0.0148 R2 < MINR2
Group: NTITLE*RACE 6 0.0146 R2 < MINR2
Var: RETURN 1 0.0146 R2 < MINR2
AOV16: WCOAT 10 0.0122 R2 < MINR2
Class: ORIGIN*EDLEVEL 21 0.0118 R2 < MINR2
Class: RACE*EDLEVEL 14 0.0116 R2 < MINR2
Group: ORIGIN*EDLEVEL 8 0.0116 R2 < MINR2
Group: RACE*EDLEVEL 5 0.0114 R2 < MINR2
Var: WCOAT 1 0.0107 R2 < MINR2
AOV16: CUSTDATE 15 0.0106 R2 < MINR2
Class: RACE*HEAT 14 0.0103 R2 < MINR2
Group: RACE*HEAT 5 0.0102 R2 < MINR2
Class: ORIGIN*HEAT 22 0.0093 R2 < MINR2
SS and R2 Portion for Effects Chosen for the Target

This section lists the chosen input variables from the forward stepwise regression. The table is divided into the following five columns:

- **Effect** lists the sequentially selected effects, which are ranked by the R2 statistic.
- **DF** shows the degrees of freedom associated with each model effect.
- **R2** measures the sequential improvement in the model as input variables are selected. Multiply the R2 statistic by 100 to express it as a percentage. You can interpret the R2 statistic for the LEISURE effect as: "48.27% of the variation in the target AMOUNT is explained by its linear relationship with LEISURE". The R2 statistic for APPAREL indicates that this effect accounts for an additional 13.44% of the target variation.
- **SS** lists the sums of squares for each model effect.
- **EMS** lists the Error Mean Square, which measures variation due to either random error or to other inputs that are not in the model. The EMS should get smaller as important inputs are added to the model. Note that although STATECOD has an R2 value greater the STOPR2 cutoff value of 0.005, the error mean square becomes larger when this effect enters the model.

### SS and R2 portion for Effects chosen for target: AMOUNT

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
<th>SS</th>
<th>EMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var: LEISURE</td>
<td>1</td>
<td>0.4827</td>
<td>532690709</td>
<td>290726.1</td>
</tr>
<tr>
<td>Var: APPAREL</td>
<td>1</td>
<td>0.1344</td>
<td>148302146</td>
<td>215325.5</td>
</tr>
<tr>
<td>Class: LUXURY</td>
<td>1</td>
<td>0.0912</td>
<td>100683600</td>
<td>164118.4</td>
</tr>
<tr>
<td>Var: DOMESTIC</td>
<td>1</td>
<td>0.0489</td>
<td>53918925.2</td>
<td>136706.5</td>
</tr>
<tr>
<td>Var: DPM12</td>
<td>1</td>
<td>0.0101</td>
<td>11190481.5</td>
<td>131066.8</td>
</tr>
<tr>
<td>Class: KITCHEN</td>
<td>9</td>
<td>0.0086</td>
<td>9486959.66</td>
<td>126808.8</td>
</tr>
<tr>
<td>Class: STATECOD</td>
<td>54</td>
<td>0.0051</td>
<td>5659245.24</td>
<td>127435.3</td>
</tr>
</tbody>
</table>

**Note:** Note that the AOV16, GROUP, and two-way class interaction effects are not considered in the forward stepwise regression. Including these effects may produce a better model, but it will also increase the execution time of the DMINE procedure. To learn how to include these effects into the analysis, see Example 2.

Final ANOVA Table for the Target

The ANOVA table is divided into the following four columns:

- **Effect** labels the source of variation as Model, Error, or Total.
- **DF** lists the degrees of freedom for each source of variation.
- **R2** is the model R2, which is the ratio of the model sums of squares (SS) to the total sums of squares. In this
example, the inputs collectively explain 78.10% of the total variability in the target AMOUNT.

- SS partitions the total target variation into portions that can be attributed to the model inputs and to error.

```
DMINE: Continuous Target

The final ANOVA table for target: AMOUNT

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>68</td>
<td>0.7810</td>
<td>861932067</td>
</tr>
<tr>
<td>Error</td>
<td>1897</td>
<td></td>
<td>241744735</td>
</tr>
<tr>
<td>Total</td>
<td>1965</td>
<td></td>
<td>1103676802</td>
</tr>
</tbody>
</table>
```

**SS and R2 portion for Effects not chosen for Target**

The final section lists the sums of squares and the R-square values for the effects that are not chosen in the final model.

```
SS and R2 portion for Effects not chosen for target: AMOUNT

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var: HHAPPAR</td>
<td>1</td>
<td>0.0022</td>
<td>2405347</td>
</tr>
<tr>
<td>Var: TOWELS</td>
<td>1</td>
<td>0.0002</td>
<td>213272</td>
</tr>
<tr>
<td>Var: LINENS</td>
<td>1</td>
<td>0.0000</td>
<td>19937</td>
</tr>
<tr>
<td>Var: HOMEACC</td>
<td>1</td>
<td>0.0000</td>
<td>8551</td>
</tr>
<tr>
<td>Var: LAMPS</td>
<td>1</td>
<td>0.0003</td>
<td>351453</td>
</tr>
<tr>
<td>Var: PROMO7</td>
<td>1</td>
<td>0.0012</td>
<td>1342694</td>
</tr>
<tr>
<td>Var: MENSWARE</td>
<td>1</td>
<td>0.0027</td>
<td>3016894</td>
</tr>
<tr>
<td>Var: WAPPAR</td>
<td>1</td>
<td>0.0023</td>
<td>2489091</td>
</tr>
<tr>
<td>Var: BLANKETS</td>
<td>1</td>
<td>0.0005</td>
<td>517624</td>
</tr>
<tr>
<td>Var: PROMO13</td>
<td>1</td>
<td>0.0006</td>
<td>641861</td>
</tr>
<tr>
<td>Class: DISHES</td>
<td>9</td>
<td>0.0018</td>
<td>1991598</td>
</tr>
<tr>
<td>Var: COATS</td>
<td>1</td>
<td>0.0015</td>
<td>1706268</td>
</tr>
</tbody>
</table>
```
Before you analyze the data using the DMINE procedure, you must create the DMDB encoded data set and catalog. For more information about how to do this, see "Example 1: Getting Started with the DMDB Procedure" in the DMDB procedure documentation.

``` Sas
proc dmdb batch data=sampsio.dmexal out=dmbexal dmdbcat=catexal;
   id acctnum;
   var amount income homeval frequent recency age
domestic apparel leisure promo7 promo13 dpm12
county return menswear flatware homeacc lamps
linens blankets towels outdoor coats wcoat
wappar hhappr jewelry custdate numkids travtime job;
   class purchase(desc) marital ntitle gender telind
aprtmnt snglmom mobile kitchen luxury dishes tmktord
statecod race origin heat numcars edlevel;
run;
```
The PROC DMINE statement invokes the procedure. The DATA= option identifies the DMDB encoded training data set that is used to fit the model. The DMDBCAT= option identifies the DMDB training data catalog.

```
proc dmine data=dmbexa1 dmdbcat=catexa1
```
The MINR2= option specifies a lower bound for the individual R-square value to be eligible for the model selection process. Variables with R2 values less than the MINR2 cutoff are not entered into the model. The STOPR2 specifies a lower value for the incremental model R-square value at which the forward selection process is stopped.

minr2=0.020 stopr2=0.0050
The NOAOV16 option prevents the DMINE procedure from including the AOV16 variables in the final forward stepwise selection process.
The NOINTER option prevents the use of two-way interactions between categories of class variables in the selection process.

nointer;
The VAR statement lists the numeric and categorical inputs (independent variables).

```
var income homeval frequent recency age
domestic apparel leisure promo7 promo13 dpm12
county return mensware flatware homeacc lamps
linens blankets towels outdoor coats wcoat
wappar hhappar jewelry custdate numkids travtime job
marital ntitle gender telind aprtmnt snglmom mobile
kitchen luxury dishes tmktord statecod race origin heat
numcars edlevel;
```
The TARGET statement defines the target (response) variable.

    target amount;
    title 'DMINE: Continuous Target';
run;
Example 2: Including the AOV16 and Grouping Variables into the Analysis (Detailed Selection Settings)

Features:

- Omitting the NOAOV16 option to include the AOV16 variables into the analysis.
- Specifying the USEGROUPS option to include only the group variables in the final model. If the original class variable can be reduced into a group variable that contains fewer levels, then only the group variable is considered in the final model.
- Omitting the NOINTER option to include the two-way class interactions into the analysis.
- Specifying the NOMONITOR option to suppress the monitor window, which displays the execution time of the procedure.

This example expands on the previous example by including the AOV16, GROUP, and two-way class interaction effects into the final forward stepwise analysis. Including these effects into the analysis may produce a better model, but it will also increase the execution time of the DMINE procedure.

If you have not already done so, you should submit the PROC DMDB step from Example 1 before you submit the example PROC DMINE step.

Program

```plaintext
proc dmine data=WORK.dmbexa1 dmdbcat=catexa1
  minr2=0.020 stopr2=0.0050
  usegroups
  nomonitor;

var income homeval frequent recency age
domestic apparel leisure promo7 promo13 dpm12
county return mensware flatware homeacc lamps
linens blankets towels outdoor coats wcoat
wappar hhapar jewelry custdate numkids travtime job
marital ntitle gender telind aprtmnt snglmom mobile
kitchen luxury dishes tmktord statecod race origin heat
numcars edlevel;

target amount;
title 'DMINE: Continuous Target';
title2 'Add AOV16, GROUP, and 2-Way Interactions Effects';
run;
```

Output
Partial Listing of the R-Squares for the Target Variable

DMINE: Continuous Target
Add AOV16, GROUP, and 2-Way Interactions Effects

R-Squares for Target variable: AMOUNT

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var: LEISURE</td>
<td>1</td>
<td>0.4827</td>
</tr>
<tr>
<td>AOV16: APPAREL</td>
<td>12</td>
<td>0.4762</td>
</tr>
<tr>
<td>Class: KITCHEN*STATECOD</td>
<td>197</td>
<td>0.4271</td>
</tr>
<tr>
<td>AOV16: LEISURE</td>
<td>8</td>
<td>0.4268</td>
</tr>
<tr>
<td>Group: KITCHEN*STATECOD</td>
<td>9</td>
<td>0.4210</td>
</tr>
<tr>
<td>Class: KITCHEN*LUXURY</td>
<td>16</td>
<td>0.4019</td>
</tr>
<tr>
<td>Var: APPAREL</td>
<td>1</td>
<td>0.4001</td>
</tr>
<tr>
<td>Group: KITCHEN*LUXURY</td>
<td>5</td>
<td>0.3959</td>
</tr>
<tr>
<td>AOV16: DOMESTIC</td>
<td>15</td>
<td>0.3869</td>
</tr>
<tr>
<td>Var: DOMESTIC</td>
<td>1</td>
<td>0.3652</td>
</tr>
<tr>
<td>AOV16: FREQUENT</td>
<td>11</td>
<td>0.3418</td>
</tr>
<tr>
<td>Class: LUXURY*STATECOD</td>
<td>101</td>
<td>0.3335</td>
</tr>
<tr>
<td>Group: LUXURY*STATECOD</td>
<td>6</td>
<td>0.3284</td>
</tr>
<tr>
<td>Class: LUXURY*TMKTORD</td>
<td>7</td>
<td>0.3212</td>
</tr>
<tr>
<td>Group: LUXURY*TMKTORD</td>
<td>4</td>
<td>0.3177</td>
</tr>
<tr>
<td>Class: KITCHEN*DISHES</td>
<td>42</td>
<td>0.3128</td>
</tr>
<tr>
<td>Class: MARITAL*KITCHEN</td>
<td>17</td>
<td>0.3084</td>
</tr>
<tr>
<td>Group: KITCHEN*DISHES</td>
<td>5</td>
<td>0.3066</td>
</tr>
<tr>
<td>Var: FREQUENT</td>
<td>1</td>
<td>0.3048</td>
</tr>
<tr>
<td>Class: TMKTORD*STATECOD</td>
<td>110</td>
<td>0.3046</td>
</tr>
<tr>
<td>Group: MARITAL*KITCHEN</td>
<td>5</td>
<td>0.3033</td>
</tr>
<tr>
<td>Group: TMKTORD*STATECOD</td>
<td>8</td>
<td>0.2995</td>
</tr>
<tr>
<td>Class: KITCHEN*TMKTORD</td>
<td>26</td>
<td>0.2921</td>
</tr>
<tr>
<td>Group: KITCHEN*TMKTORD</td>
<td>5</td>
<td>0.2870</td>
</tr>
<tr>
<td>AOV16: DPM12</td>
<td>12</td>
<td>0.2770</td>
</tr>
<tr>
<td>Class: NTITLE*KITCHEN</td>
<td>28</td>
<td>0.2717</td>
</tr>
<tr>
<td>Group: NTITLE*KITCHEN</td>
<td>6</td>
<td>0.2690</td>
</tr>
<tr>
<td>Class: KITCHEN*RACE</td>
<td>25</td>
<td>0.2594</td>
</tr>
<tr>
<td>Class: KITCHEN*EDLEVEL</td>
<td>26</td>
<td>0.2572</td>
</tr>
<tr>
<td>Group: KITCHEN*RACE</td>
<td>5</td>
<td>0.2568</td>
</tr>
<tr>
<td>Class: LUXURY*DISHES</td>
<td>15</td>
<td>0.2547</td>
</tr>
</tbody>
</table>

Additional effects are not listed

DMINE: Continuous Target
Add AOV16, GROUP, and 2-Way Interactions Effects

R-Squares for Target variable: AMOUNT

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var: LEISURE</td>
<td>1</td>
<td>0.4827</td>
</tr>
<tr>
<td>AOV16: APPAREL</td>
<td>12</td>
<td>0.4762</td>
</tr>
<tr>
<td>Class: KITCHEN*STATECOD</td>
<td>197</td>
<td>0.4271</td>
</tr>
<tr>
<td>AOV16: LEISURE</td>
<td>8</td>
<td>0.4268</td>
</tr>
<tr>
<td>Group: KITCHEN*STATECOD</td>
<td>9</td>
<td>0.4210</td>
</tr>
<tr>
<td>Class: KITCHEN*LUXURY</td>
<td>16</td>
<td>0.4019</td>
</tr>
<tr>
<td>Var: APPAREL</td>
<td>1</td>
<td>0.4001</td>
</tr>
<tr>
<td>Group: KITCHEN*LUXURY</td>
<td>5</td>
<td>0.3959</td>
</tr>
<tr>
<td>AOV16: DOMESTIC</td>
<td>15</td>
<td>0.3869</td>
</tr>
<tr>
<td>Var: DOMESTIC</td>
<td>1</td>
<td>0.3652</td>
</tr>
<tr>
<td>AOV16: FREQUENT</td>
<td>11</td>
<td>0.3418</td>
</tr>
<tr>
<td>Class: LUXURY*STATECOD</td>
<td>101</td>
<td>0.3335</td>
</tr>
<tr>
<td>Group: LUXURY*STATECOD</td>
<td>6</td>
<td>0.3284</td>
</tr>
<tr>
<td>Class: LUXURY*TMKTORD</td>
<td>7</td>
<td>0.3212</td>
</tr>
<tr>
<td>Group: LUXURY*TMKTORD</td>
<td>4</td>
<td>0.3177</td>
</tr>
<tr>
<td>Class: KITCHEN*DISHES</td>
<td>42</td>
<td>0.3128</td>
</tr>
<tr>
<td>Class: MARITAL*KITCHEN</td>
<td>17</td>
<td>0.3084</td>
</tr>
<tr>
<td>Group: KITCHEN*DISHES</td>
<td>5</td>
<td>0.3066</td>
</tr>
<tr>
<td>Var: FREQUENT</td>
<td>1</td>
<td>0.3048</td>
</tr>
<tr>
<td>Class: TMKTORD*STATECOD</td>
<td>110</td>
<td>0.3046</td>
</tr>
<tr>
<td>Group: MARITAL*KITCHEN</td>
<td>5</td>
<td>0.3033</td>
</tr>
<tr>
<td>Group: TMKTORD*STATECOD</td>
<td>8</td>
<td>0.2995</td>
</tr>
<tr>
<td>Class: KITCHEN*TMKTORD</td>
<td>26</td>
<td>0.2921</td>
</tr>
<tr>
<td>Group: KITCHEN*TMKTORD</td>
<td>5</td>
<td>0.2870</td>
</tr>
<tr>
<td>AOV16: DPM12</td>
<td>12</td>
<td>0.2770</td>
</tr>
<tr>
<td>Class: NTITLE*KITCHEN</td>
<td>28</td>
<td>0.2717</td>
</tr>
<tr>
<td>Group: NTITLE*KITCHEN</td>
<td>6</td>
<td>0.2690</td>
</tr>
<tr>
<td>Class: KITCHEN*RACE</td>
<td>25</td>
<td>0.2594</td>
</tr>
<tr>
<td>Class: KITCHEN*EDLEVEL</td>
<td>26</td>
<td>0.2572</td>
</tr>
<tr>
<td>Group: KITCHEN*RACE</td>
<td>5</td>
<td>0.2568</td>
</tr>
<tr>
<td>Class: LUXURY*DISHES</td>
<td>15</td>
<td>0.2547</td>
</tr>
</tbody>
</table>
SS and R2 Portion for Effects Chosen for the Target

As in the Example 1 analysis, the original input LEISURE appears to be the most important predictor of how much a customer spends. Notice that the remaining chosen effects are either AOV16 variables or group variables.

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
<th>SS</th>
<th>EMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var: LEISURE</td>
<td>1</td>
<td>0.4827</td>
<td>532690709</td>
<td>290726.1</td>
</tr>
<tr>
<td>Group: KITCHEN*LUXURY</td>
<td>5</td>
<td>0.1590</td>
<td>175505155</td>
<td>201879.0</td>
</tr>
<tr>
<td>AOV16: APPAREL</td>
<td>12</td>
<td>0.1111</td>
<td>122618569</td>
<td>140145.0</td>
</tr>
<tr>
<td>AOV16: DOMESTIC</td>
<td>15</td>
<td>0.0415</td>
<td>45758285.2</td>
<td>117548.7</td>
</tr>
<tr>
<td>AOV16: WAPPAR</td>
<td>7</td>
<td>0.0122</td>
<td>13463471</td>
<td>110982.1</td>
</tr>
<tr>
<td>Group: LUXURY*STATECOD</td>
<td>6</td>
<td>0.0096</td>
<td>10557346.7</td>
<td>105827.7</td>
</tr>
<tr>
<td>AOV16: LEISURE</td>
<td>6</td>
<td>0.0074</td>
<td>8168060.98</td>
<td>101889.8</td>
</tr>
</tbody>
</table>
Final ANOVA Table for the Target

The R-square value of 0.8234 is slightly larger than the 0.7810 value obtained from the initial analysis, which did not include the AOV16 or group variables.

The final ANOVA table for target: AMOUNT

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>52</td>
<td>0.8234</td>
<td>908761597</td>
</tr>
<tr>
<td>Error</td>
<td>1913</td>
<td></td>
<td>194915205</td>
</tr>
<tr>
<td>Total</td>
<td>1965</td>
<td></td>
<td>1103676802</td>
</tr>
</tbody>
</table>

Partial Listing of the SS and R2 portion for Effects Not Chosen for Target Table

SS and R2 portion for Effects not chosen for target: AMOUNT

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group: KITCHEN*STATECOD</td>
<td>7</td>
<td>0.0040</td>
<td>4434939</td>
</tr>
<tr>
<td>Var: APPAREL</td>
<td>1</td>
<td>0.0024</td>
<td>2672342</td>
</tr>
<tr>
<td>Var: DOMESTIC</td>
<td>1</td>
<td>0.0035</td>
<td>3889555</td>
</tr>
<tr>
<td>AOV16: FREQUENT</td>
<td>11</td>
<td>0.0031</td>
<td>3395145</td>
</tr>
<tr>
<td>Group: LUXURY*TMKTORD</td>
<td>4</td>
<td>0.0007</td>
<td>811183</td>
</tr>
<tr>
<td>Group: KITCHEN*DISHES</td>
<td>4</td>
<td>0.0012</td>
<td>1286988</td>
</tr>
<tr>
<td>Var: FREQUENT</td>
<td>1</td>
<td>0.0007</td>
<td>762668</td>
</tr>
<tr>
<td>Group: MARITAL*KITCHEN</td>
<td>4</td>
<td>0.0011</td>
<td>1159219</td>
</tr>
<tr>
<td>Group: TMKTORD*STATECOD</td>
<td>6</td>
<td>0.0010</td>
<td>1089776</td>
</tr>
<tr>
<td>Group: KITCHEN*TMKTORD</td>
<td>5</td>
<td>0.0017</td>
<td>1880850</td>
</tr>
<tr>
<td>AOV16: DPM12</td>
<td>11</td>
<td>0.0042</td>
<td>4623270</td>
</tr>
<tr>
<td>Group: NTITLE*KITCHEN</td>
<td>6</td>
<td>0.0009</td>
<td>947581</td>
</tr>
<tr>
<td>Group: KITCHEN*RACE</td>
<td>5</td>
<td>0.0006</td>
<td>655343</td>
</tr>
<tr>
<td>Group: KITCHEN*EDLEVEL</td>
<td>6</td>
<td>0.0010</td>
<td>1084391</td>
</tr>
<tr>
<td>Group: LUXURY*DISHES</td>
<td>3</td>
<td>0.0002</td>
<td>258116</td>
</tr>
<tr>
<td>Group: LUXURY*RACE</td>
<td>2</td>
<td>0.0004</td>
<td>419452</td>
</tr>
<tr>
<td>Group: KITCHEN*ORIGIN</td>
<td>6</td>
<td>0.0010</td>
<td>1117402</td>
</tr>
</tbody>
</table>

Additional effects not listed
The USEGROUPS option specifies to retain only the reduced group variables (the original class variables are not included in the model selection process). Since the NOAOV16 option was not specified, the AOV16 variables that have an R-square value less than the MINR2 cutoff are used in the final forward stepwise regression. Because the NOINTER option is omitted, the two-factor class interactions are also evaluated in the forward stepwise regression.
The NOMONITOR option suppresses the monitor that displays the execution status of the DMINE procedure.

nomonitor;
Example 3: Modeling a Binary Target with the DMINE Procedure

Features:
- Setting the MINR2= and STOPR2= cutoff values.
- Specifying the target and input variables.

As a marketing analyst at a catalog company, you want to determine the inputs that best predict whether or not a customer will make a purchase from your new fall outerwear catalog. The fictitious catalog mailing data set is named SAMPSON.DMEXA1 (stored in the sample library). The data set contains 1,966 customer cases. The binary target (PURCHASE) contains a formatted value of "Yes" if a purchase was made and a formatted value of "No" if a purchase was not made.

There are 48 input variables available for predicting the target. Note that AMOUNT is an interval target that is modeled in Examples 1 and 2 of this chapter. ACCTNUM is an id variable, which is not a suitable input variable.

Program

```sql
proc dmdb batch data=sampson.dmexa1 out=dmbexa1 dmdbcat=catexa1;
  id acctnum;
  var  amount income homeval frequent recency age
domestic apparel leisure promo7 promo13 dpm12
county return mensware flatware homeacc lamps
linens blankets towels outdoor coats wcoat
wappar hhappar jewelry custdate numkids travtime job;
  class purchase(desc) marital ntitle gender telind
  aprtmnt snglmom mobile kitchen luxury dishes tmktord
  statecod race origin heat numcars edlevel;
run;
```

```sql
proc dmine data=WORK.dmbexa1 dmdbcat=catexa1

  minr2=0.020 stopr2=0.0050;

  var income homeval frequent recency age
domestic apparel leisure promo7 promo13 dpm12
county return mensware flatware homeacc lamps
linens blankets towels outdoor coats wcoat
wappar hhappar jewelry custdate numkids travtime job
marital ntitle gender telind aprtmnt snglmom mobile
kitchen luxury dishes tmktord statecod race origin heat
numcars edlevel;
```
Output

DMINE Status Monitor

When you invoke the DMINE procedure, the Dmine Status Monitor window appears. This window monitors the execution time of the procedure.

Partial Listing of the R-Squares for the Target Variable

This section of the output ranks all model effects by their R-square values. The degrees of freedom (DF) associated with each effect is also listed. The significant variables are analyzed in a subsequent forward stepwise regression. Non-significant variables are labeled as having an R2 value less than the MINR2 cutoff; these variables are not chosen in the final model.

There are four types of model effects:

- **Class** effects are estimated for each class variable and all possible two-factor class interactions. The R-square statistic is calculated for each class effect using a one-way analysis of variance. Two-factor interaction effects are constructed by combining all possible levels of each class variable into one term. The degrees of freedom for a class effect is equal to: (the number of unique factor levels minus 1). For two-factor interactions, the degrees of freedom is equal to: (the number of levels in factor A multiplied by the number of levels in factor B minus 1). You can omit the two-factor interaction effects from the final stepwise analysis by specifying the NOINTER option on the PROC DMINE.
Group effects are created by reducing each class effect through an analysis of means. The degrees of freedom for each group effect is equal to the number of levels. Since the USEGROUPS option was not specified in the PROC DMINE statement, the group effects and the original class effects can be used in the final model.

VAR effects are estimated from interval variables as standard regression inputs. A simple linear regression is performed to determine the R2 statistic for interval inputs. The degrees of freedom is always equal to 1.

AOV16 effects are calculated as a result of grouping numeric variables into a maximum of 16 equally spaced buckets. AOV16 effects may account for possible non-linearity in the target variable PURCHASE. The degrees of freedom are calculated as the number of groups. The AOV16 variables can be expensive to compute. You can prevent these variables from being evaluated in the forward stepwise regression by specifying the NOAOV16 option in the PROC DMINE statement.

For this example, the class KITCHEN*STATECOD interaction has the largest R2 with the target PURCHASE. Class and group interactions composed of the same terms have very similar R2 values. Of all the AOV16 variables, FREQUENT has the largest R2 value with the target.

DMINE: Binary Target

R-Squares for Target variable: PURCHASE

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class: KITCHEN*STATECOD</td>
<td>197</td>
<td>0.1045</td>
</tr>
<tr>
<td>Group: KITCHEN*STATECOD</td>
<td>8</td>
<td>0.1026</td>
</tr>
<tr>
<td>Class: NTITLE*STATECOD</td>
<td>191</td>
<td>0.0955</td>
</tr>
<tr>
<td>Group: NTITLE*STATECOD</td>
<td>9</td>
<td>0.0940</td>
</tr>
<tr>
<td>Class: STATECOD*ORIGIN</td>
<td>166</td>
<td>0.0917</td>
</tr>
<tr>
<td>Group: STATECOD*ORIGIN</td>
<td>8</td>
<td>0.0899</td>
</tr>
<tr>
<td>Class: DISHES*STATECOD</td>
<td>169</td>
<td>0.0875</td>
</tr>
<tr>
<td>Group: DISHES*STATECOD</td>
<td>8</td>
<td>0.0862</td>
</tr>
<tr>
<td>Class: STATECOD*EDLEVEL</td>
<td>146</td>
<td>0.0736</td>
</tr>
<tr>
<td>Group: STATECOD*EDLEVEL</td>
<td>9</td>
<td>0.0723</td>
</tr>
<tr>
<td>Class: STATECOD*HEAT</td>
<td>141</td>
<td>0.0643</td>
</tr>
<tr>
<td>Group: STATECOD*HEAT</td>
<td>9</td>
<td>0.0633</td>
</tr>
<tr>
<td>Class: LUXURY*STATECOD</td>
<td>101</td>
<td>0.0607</td>
</tr>
<tr>
<td>Class: STATECOD*NUMCARS</td>
<td>121</td>
<td>0.0597</td>
</tr>
<tr>
<td>Group: LUXURY*STATECOD</td>
<td>10</td>
<td>0.0597</td>
</tr>
<tr>
<td>AOV16: FREQUENT</td>
<td>11</td>
<td>0.0592</td>
</tr>
<tr>
<td>Group: STATECOD*NUMCARS</td>
<td>9</td>
<td>0.0585</td>
</tr>
<tr>
<td>Class: STATECOD*RACE</td>
<td>105</td>
<td>0.0575</td>
</tr>
<tr>
<td>Class: TMKTORD*STATECOD</td>
<td>110</td>
<td>0.0571</td>
</tr>
<tr>
<td>Group: STATECOD*RACE</td>
<td>9</td>
<td>0.0566</td>
</tr>
<tr>
<td>Group: TMKTORD*STATECOD</td>
<td>8</td>
<td>0.0562</td>
</tr>
<tr>
<td>Class: MARITAL*STATECOD</td>
<td>107</td>
<td>0.0537</td>
</tr>
<tr>
<td>Group: MARITAL*STATECOD</td>
<td>10</td>
<td>0.0526</td>
</tr>
<tr>
<td>Group: GENDER*STATECOD</td>
<td>104</td>
<td>0.0514</td>
</tr>
<tr>
<td>Group: GENDER*STATECOD</td>
<td>10</td>
<td>0.0505</td>
</tr>
<tr>
<td>Var: FREQUENT</td>
<td>1</td>
<td>0.0498</td>
</tr>
</tbody>
</table>

Additional Effects Are Not Listed

Group: KITCHEN*EDLEVEL       | 7   | 0.0216|
Class: KITCHEN*RACE          | 25  | 0.0214|
Class: TELIND*KITCHEN        | 14  | 0.0213|
Group: TELIND*KITCHEN        | 5   | 0.0211|
### SS and R2 Portion for Effects Chosen for Target

This section lists the chosen input variables from the forward stepwise regression. The table is divided into the following five columns:

- **Effect** lists the sequentially selected effects, which are ranked by the R-square statistic.
- **DF** shows the degrees of freedom associated with each model effect.
- **R2** measures the sequential improvement in the model as input variables are selected. Multiply the R2 statistic by 100 to express it as a percentage. You can interpret the R2 statistic for the KITCHEN*STATECOD interaction as "10.45% of the variation in the target PURCHASE is explained by its linear relationship with this effect". The R2 statistic for NTITLE*STATECOD indicates that this two-factor interaction accounts for an additional 6.38% of the target variation.
- **SS** lists the sums of squares for each model effect.
- **EMS** lists the Error Mean Square, which measures variation due to either random error or to other inputs that are not in the model. The EMS should get smaller as important inputs are added to the model.

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
<th>SS</th>
<th>EMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class: KITCHEN*STATECOD</td>
<td>197</td>
<td>0.1045</td>
<td>51.35546</td>
<td>0.2488769</td>
</tr>
<tr>
<td>Class: NTITLE*STATECOD</td>
<td>132</td>
<td>0.0683</td>
<td>33.55930</td>
<td>0.2484444</td>
</tr>
<tr>
<td>Class: STATECOD*ORIGIN</td>
<td>106</td>
<td>0.0660</td>
<td>32.43976</td>
<td>0.2444544</td>
</tr>
<tr>
<td>Class: DISHES*STATECOD</td>
<td>100</td>
<td>0.0473</td>
<td>23.21811</td>
<td>0.2453127</td>
</tr>
</tbody>
</table>

Additional Effects Are Not Listed
The Final Anova Table for the Target

The ANOVA table is divided into the following four columns:

- **Effect** labels the source of variation as Model, Error, or Total.
- **DF** lists the degrees of freedom for each source of variation.
- **R2** is the model R2, which is the ratio of the model sums of squares (SS) to the total sums of squares. In this example, the selected inputs collectively explain 49.83% of the total variability in the target PURCHASE.
- **SS** partitions the total target variation into portions that can be attributed to the model inputs and to error.

The final ANOVA table for target: PURCHASE

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>880</td>
<td>0.4983</td>
<td>244.85936</td>
</tr>
<tr>
<td>Error</td>
<td>1085</td>
<td></td>
<td>246.51043</td>
</tr>
<tr>
<td>Total</td>
<td>1965</td>
<td></td>
<td>491.36979</td>
</tr>
</tbody>
</table>

SS and R2 portion for Effects Not Chosen for the Target: PURCHASE

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>R2</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class: GENDER*STATECOD</td>
<td>0</td>
<td>0.0000</td>
<td>0.00000</td>
</tr>
<tr>
<td>Var: FREQUENT</td>
<td>1</td>
<td>0.0010</td>
<td>0.50991</td>
</tr>
<tr>
<td>Var: DOMESTIC</td>
<td>1</td>
<td>0.0012</td>
<td>0.57574</td>
</tr>
</tbody>
</table>

Estimating Logistic

When the target is binary, predicted values or SUPERX's are computed from the forward stepwise regression. The SUPERX's are then grouped into 256 equally spaced intervals, which are used as the independent variable in a final logistic regression analysis. The logistic regression helps you decide the cutoff of the binary response. Since there is one input, only two parameters are estimated (the intercept and the slope).

The first table shows the iteration history for estimating the intercept (alpha) and the slope (beta) for the approximate logistic regression.

The second table contains the predicted values, which are bucketed into the 256 equally sized sub-intervals. The table
contains the following columns:

- **N0** - number of observations that have an observed value of PURCHASE = 'No'
- **N1** - number of observations that have an observed value of PURCHASE = 'Yes'.
- **Nmiss** - number of missing values.
- **X** - center value of the corresponding interval.
- **P** - predicted value for the center value X of the sub-interval.

Estimating logistic

<table>
<thead>
<tr>
<th>Iter</th>
<th>Alpha</th>
<th>Beta</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-2.0520</td>
<td>4.0998</td>
</tr>
<tr>
<td>1</td>
<td>-3.1242</td>
<td>6.1777</td>
</tr>
<tr>
<td>2</td>
<td>-3.6975</td>
<td>7.2727</td>
</tr>
<tr>
<td>3</td>
<td>-3.8236</td>
<td>7.5127</td>
</tr>
<tr>
<td>4</td>
<td>-3.8284</td>
<td>7.5219</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N0</th>
<th>N1</th>
<th>Nmiss</th>
<th>X</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>1</td>
<td>0</td>
<td>-0.135</td>
<td>0.0078</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-0.130</td>
<td>0.0081</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-0.125</td>
<td>0.0084</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-0.120</td>
<td>0.0087</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>-0.115</td>
<td>0.0091</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>-0.110</td>
<td>0.0094</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-0.105</td>
<td>0.0098</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-0.100</td>
<td>0.0101</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-0.095</td>
<td>0.0105</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>-0.090</td>
<td>0.0109</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>-0.085</td>
<td>0.0113</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>-0.080</td>
<td>0.0118</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-0.075</td>
<td>0.0122</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-0.070</td>
<td>0.0127</td>
</tr>
</tbody>
</table>

Additional subintervals are not listed.

<table>
<thead>
<tr>
<th>Iter</th>
<th>N0</th>
<th>N1</th>
<th>Nmiss</th>
<th>X</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td></td>
<td>1.085</td>
<td>0.9870</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
<td></td>
<td>1.090</td>
<td>0.9875</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td></td>
<td>1.095</td>
<td>0.9880</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>1.100</td>
<td>0.9884</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td></td>
<td>1.105</td>
<td>0.9888</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td></td>
<td>1.110</td>
<td>0.9892</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>1.115</td>
<td>0.9896</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>1.125</td>
<td>0.9904</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td></td>
<td>1.130</td>
<td>0.9907</td>
</tr>
<tr>
<td>0</td>
<td>23</td>
<td>0</td>
<td></td>
<td>1.140</td>
<td>0.9914</td>
</tr>
</tbody>
</table>

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Before you analyze the data using the DMINE procedure, you must create the DMDB encoded data set and catalog. For more information about how to do this, see "Example 1: Getting Started with the DMDB Procedure" in the DMDB procedure documentation. Since the (DESCENDING) ORDER option is specified for the target PURCHASE on the CLASS statement, the DMINE procedure reads this encoded information from the metadata and then models the probability that a customer will make a purchase (PURCHASE = 'Yes'). The default ORDER is set to ASCENDING for all class variables.

```
proc dmdb batch data=sampsio.dmexa1 out=dmbexa1 dmdbcat=catexa1;
  id  acctnum;
  var  amount income homeval frequent recency age
domestic apparel leisure promo7 promo13 dpm12
county return mensware flatware homeacc lamps
linens blankets towels outdoor coats wcoat
wappar hhappar jewelry custdate numkids travtime job;
class purchase(desc) marital ntitle gender telind
  aprtmnt snglmom mobile kitchen luxury dishes tmktord
statecod race origin heat numcars edlevel;
run;
```
The PROC DMINE statement invokes the procedure. The DATA= option identifies the DMDB encoded training data set that is used to fit the model. The DMDBCAT= option identifies the DMDB training data catalog.

```plaintext
proc dmine data=WORK.dmbexa1 dmdbcat=catexa1
```
The MINR2= option specifies a lower bound for the individual R-square value to be eligible for the model selection process. Variables with R2 values less than the MINR2 cutoff are not entered into the model. The STOPR2 specifies a lower value for the incremental model R-square value at which the forward selection process is stopped.

\[ \text{minr2}=0.020 \quad \text{stopr2}=0.0050; \]
The VAR statement specifies the numeric and categorical inputs (independent variables).

```
var income homeval frequent recency age
domestic apparel leisure promo7 promo13 dpm12
county return mensware flatware homeacc lamps
linens blankets towels outdoor coats wcoat
wappar hhapppar jewelry custdate numkids travtime job
marital ntitle gender telind aprtmnt snglmom mobile
kitchen luxury dishes tmktord statecod race origin heat
numcars edlevel;
```
PROC DMNEURL: Approximation to PROC NEURAL

Purpose of PROC DMNEURL

In its current form, PROC DMNEURL tries to establish a nonlinear model for the prediction of a binary or interval scaled response variable (called target in data mining terminology). The approach will soon be extended to nominal and ordinal scaled response variables.

The algorithm used in DMNEURL was developed to overcome some problems of PROC NEURAL for data mining purposes, especially when the data set contains many highly collinear variables:

1. The nonlinear estimation problem in common Neural Networks is seriously underdetermined yielding to highly rankdeficient Hessian matrices and resulting in extremely slow convergence (close to linear) of nonlinear optimization algorithms.
   \[\implies\text{Full-rank estimation.}\]

2. Each function call in PROC NEURAL corresponds to a single run through the entire (training) data set and normally many function calls are needed for convergent nonlinear optimization with rankdeficient Hessians.
   \[\implies\text{Optimization of discrete problem with all data incore.}\]

3. Because the zero eigenvalues in a Hessian matrix correspond to long and very flat valleys in the shape of the objective function, the traditional Neural Net approach has serious problems to decide when an estimate is close to an appropriate solution and the optimization process can be terminated.
   \[\implies\text{Quadratic convergence.}\]

4. For the same reasons, the common Neural Net algorithms suffer from a high sensibility toward finding local rather than global optimal solutions and the optimization result often is very sensitive w.r.t. the starting point of the optimization.
   \[\implies\text{Good starting point.}\]

With PROC DMNEURL we deal with specified optimization problems (with full rank Hessian matrices) which have not many parameters and for which good starting points can be obtained. The convergence of the nonlinear optimizer is normally very fast, resulting mostly in less than 10 iterations per optimization. The function and derivative calls during the optimization do not need any passes through the data set, however, the search for obtaining good starting points and the final evaluations of the solutions (scoring of all observations) need passes through the data, as well as a number of preliminary tasks. In PROC DMNEURL we fit separately an entire
set of about 8 activation functions and select the best result. Since the optimization processes for different activation functions do not depend on each other, the computer time could be reduced greatly by parallel processing.

Except for applications where PROC NEURAL would hit a local solution much worse than the global solution, it is not expected that PROC DMNEURL can beat PROC NEURAL in the precision of the prediction. However, for the applications we have run until now we found the results of PROC DMNEURL very close to those of PROC NEURAL. PROC DMNEURL will be faster than PROC NEURAL only for very large data sets. For small data sets, PROC NEURAL could be much faster than PROC DMNEURL, especially for an interval target. The most efficient application of PROC DMNEURL is the analysis of a binary target variable without FREQ and WEIGHT statement and without COST variables in the input data set.

Application: HMEQ Data Set:
Binary Target BAD

To illustrate the use of PROC DMNEURL we choose the HMEQ data set:

```
libname sampsio '/sas/a612/dmine/sampsio';
proc dmdb batch data=sampsio.hmeq out=dmdbout dmdbcat=outcat;
  var LOAN MORTDUE VALUE YOJ DELINQ CLAGE NINQ CLNO DEBTINC;
  class BAD(DESC) REASON(ASC) JOB(ASC) DEROG(ASC);
  target BAD;
run;
```

When selecting the binary target variable BAD a typical run of PROC DMNEURL would be the following:

```
proc dmneurl data=dmdbout dmdbcat=outcat
  outclass=oclass outest=estout out=dsout outfit=ofit
  ptable maxcomp=3 maxstage=5;
  var LOAN MORTDUE VALUE REASON JOB YOJ DEROG DELINQ
       CLAGE NINQ CLNO DEBTINC;
  target BAD;
run;
```

The number of parameters $p$ estimated in each stage of the optimization is $p = 2 \cdot c + 1$, where $c$ is the number of components that is selected at the stage. Since here $c = 3$ is specified with the MAXCOMP= option each optimization process estimates only $p = 7$ parameters.

First some general information is printed and the four moments of the numeric data set variables involved in the analysis:

**The DMNEURL Procedure**

<table>
<thead>
<tr>
<th>Binary Target</th>
<th>BAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number Observations</td>
<td>5960</td>
</tr>
<tr>
<td>NOBS w/o Missing Target</td>
<td>5960</td>
</tr>
</tbody>
</table>
Purpose of PROC DMNEURL

Link Function: LOGIST
Selection Criterion: SSE
Optimization Criterion: SSE
Estimation Stages: 5
Max. Number Components: 3
Minimum R2 Value: 0.000050
Number Grid Points: 17

Response Profile for Target: BAD

<table>
<thead>
<tr>
<th>Level</th>
<th>Nobs</th>
<th>Frequency</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1189</td>
<td>1189</td>
<td>1189.000000</td>
</tr>
<tr>
<td>0</td>
<td>4771</td>
<td>4771</td>
<td>4771.000000</td>
</tr>
</tbody>
</table>

Variable | Mean  | Std Dev | Skewness  | Kurtosis |
---------|-------|---------|-----------|----------|
LOAN     | 18608 | 11207   | 2.02378   | 6.93259  |
MORTDUE  | 67350 | 44458   | 1.81448   | 6.48187  |
VALUE    | 99863 | 57386   | 3.05334   | 24.36280 |
YOJ      | 8.15130 | 7.57398 | 0.98846   | 0.37207  |
DELINQ   | 0.40570 | 1.12727 | 4.02315   | 23.56545 |
CLAGE    | 170.47634 | 85.81009 | 1.34341   | 7.59955  |
NINQ     | 1.08456 | 1.72867 | 2.62198   | 9.78651  |
CLNO     | 20.50285 | 10.13893 | 0.77505   | 1.15767  |
DEBTINC  | 26.59885 | 8.60175 | 2.85235   | 50.50404 |

For the first stage we select three eigenvectors corresponding to the 4th, 11th, and 2nd largest eigenvalues. Obviously, there is no relationship between

- the $R^2$ value which measures the prediction of the response (target) variable by each eigenvector
- and the eigenvalue corresponding to each eigenvector which measures the variance explained in the $X^TX$ data matrix.

Therefore, the eigenvalues are not used in the analysis of PROC DMNEURL and are printed only for curiosity.

Component Selection: SS(y) and R2 (SS_total=4771)

<table>
<thead>
<tr>
<th>Comp</th>
<th>Eigval</th>
<th>R-Square</th>
<th>F Value</th>
<th>p-Value</th>
<th>SSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>9397.769045</td>
<td>0.017419</td>
<td>105.640645</td>
<td>&lt;.0001</td>
<td>4687.893424</td>
</tr>
<tr>
<td>11</td>
<td>6327.041282</td>
<td>0.006317</td>
<td>38.550835</td>
<td>&lt;.0001</td>
<td>4657.755732</td>
</tr>
<tr>
<td>2</td>
<td>13164</td>
<td>0.005931</td>
<td>36.408247</td>
<td>&lt;.0001</td>
<td>4629.461194</td>
</tr>
</tbody>
</table>

The optimization history indicates a maximum of 11 iterations for the activation function LOGIST:
PROC DMNEURL: Approximation to PROC NEURAL

--- Optimization Cycle (Stage=0) ---
--- Activation= SQUARE (Stage=0) ---

NOTE: ABSGCONV convergence criterion satisfied.
SQUARE: Iter=5 Crit=0.06782364: SSE=808.457819 Acc= 81.6443
--- Activation= TANH (Stage=0) ---

NOTE: ABSGCONV convergence criterion satisfied.
TANH: Iter=4 Crit=0.06802595: SSE=810.869323 Acc= 81.6275
--- Activation= ARCTAN (Stage=0) ---

NOTE: ABSGCONV convergence criterion satisfied.
ARCTAN: Iter=5 Crit=0.06795346: SSE=810.005204 Acc= 81.6611
--- Activation= LOGIST (Stage=0) ---

NOTE: ABSGCONV convergence criterion satisfied.
LOGIST: Iter=11 Crit=0.06802943: SSE=810.910850 Acc= 81.6107
--- Activation= GAUSS (Stage=0) ---

NOTE: ABSGCONV convergence criterion satisfied.
GAUSS: Iter=10 Crit=0.07727582: SSE=921.127726 Acc= 80.2517
--- Activation= SIN (Stage=0) ---

NOTE: ABSGCONV convergence criterion satisfied.
SIN: Iter=5 Crit=0.06811774: SSE=811.963450 Acc= 81.6611
--- Activation= COS (Stage=0) ---

NOTE: ABSGCONV convergence criterion satisfied.
COS: Iter=9 Crit=0.07419096: SSE=884.356261 Acc= 81.1913
--- Activation= EXP (Stage=0) ---

NOTE: ABSGCONV convergence criterion satisfied.
EXP: Iter=9 Crit=0.06798656: SSE=810.399740 Acc= 81.5436

The following approximate accuracy rates are based on the discrete values of the predictor ($x$) variables:

Approximate Goodness-of-Fit Criteria (Stage 0)

<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>Criterion</th>
<th>SSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>0.067824</td>
<td>808.457819</td>
<td>81.644295</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>0.067953</td>
<td>810.005204</td>
<td>81.661074</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>0.067987</td>
<td>810.399740</td>
<td>81.543624</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>0.068026</td>
<td>810.869323</td>
<td>81.627517</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>0.068029</td>
<td>810.910850</td>
<td>81.610738</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td>0.068118</td>
<td>811.963450</td>
<td>81.661074</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>0.074191</td>
<td>884.356261</td>
<td>81.191275</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>0.077276</td>
<td>921.127726</td>
<td>80.251678</td>
</tr>
</tbody>
</table>

After running through the data set we obtain the correct accuracy tables:

Classification Table for CUTOFF = 0.5000

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Observed</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
The activation function SQUARE seems to be most appropriate for the first stage (stage=0) of estimation. However, TANH yields an even higher accuracy rate:

**Goodness-of-Fit Criteria (Ordered by SSE, Stage 0)**

<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>805.19026</td>
<td>0.367558</td>
<td>81.610738</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>805.89106</td>
<td>0.367718</td>
<td>81.778523</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>806.66533</td>
<td>0.367895</td>
<td>81.593960</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>807.30313</td>
<td>0.368040</td>
<td>81.778523</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>807.72088</td>
<td>0.368135</td>
<td>81.778523</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td>809.31533</td>
<td>0.368499</td>
<td>81.291946</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>881.68579</td>
<td>0.384622</td>
<td>81.359060</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>949.21059</td>
<td>0.399078</td>
<td>80.167785</td>
</tr>
</tbody>
</table>

The following is the start of the second stage of estimation (stage=1). It starts with selecting three eigenvectors which may predict the residuals best:

**Component Selection: SS(y) and R² (Stage=1)**

<table>
<thead>
<tr>
<th>Comp Value</th>
<th>Eigval</th>
<th>R-Square</th>
<th>F Value</th>
<th>p-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>4763.193233</td>
<td>0.023292</td>
<td>142.109442</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>21</td>
<td>5192.070258</td>
<td>0.018366</td>
<td>114.178467</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>24</td>
<td>4514.317020</td>
<td>0.017493</td>
<td>110.756118</td>
<td>&lt;.0001</td>
</tr>
</tbody>
</table>

When fitting the first order residuals the average value of the objective function dropped from 0.068 to 0.063. For time reasons the approximate accuracy rates are not computed after the first stage:
The best accuracy went up from 81.61 to 83.28, the (1,1) entry from 229 to 319 counts (which is still less than that of TANH in the first stage):

<table>
<thead>
<tr>
<th>Activation</th>
<th>Accuracy</th>
<th>Observed</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQUARE</td>
<td>83.288591</td>
<td>319.0</td>
<td>870.0</td>
</tr>
<tr>
<td></td>
<td>741.977606</td>
<td>126.0</td>
<td>4645.0</td>
</tr>
<tr>
<td>EXP</td>
<td>83.187919</td>
<td>304.0</td>
<td>885.0</td>
</tr>
<tr>
<td></td>
<td>747.249371</td>
<td>117.0</td>
<td>4654.0</td>
</tr>
<tr>
<td>SIN</td>
<td>83.120805</td>
<td>306.0</td>
<td>883.0</td>
</tr>
<tr>
<td></td>
<td>748.284252</td>
<td>123.0</td>
<td>4648.0</td>
</tr>
<tr>
<td>TANH</td>
<td>83.221477</td>
<td>309.0</td>
<td>880.0</td>
</tr>
<tr>
<td></td>
<td>748.436926</td>
<td>120.0</td>
<td>4651.0</td>
</tr>
<tr>
<td>ARCTAN</td>
<td>83.255034</td>
<td>310.0</td>
<td>879.0</td>
</tr>
<tr>
<td></td>
<td>748.614897</td>
<td>119.0</td>
<td>4652.0</td>
</tr>
<tr>
<td>LOGIST</td>
<td>83.171141</td>
<td>305.0</td>
<td>884.0</td>
</tr>
<tr>
<td></td>
<td>749.309756</td>
<td>119.0</td>
<td>4652.0</td>
</tr>
<tr>
<td>COS</td>
<td>82.583893</td>
<td>309.0</td>
<td>880.0</td>
</tr>
<tr>
<td></td>
<td>781.342717</td>
<td>158.0</td>
<td>4613.0</td>
</tr>
<tr>
<td>GAUSS</td>
<td>82.567114</td>
<td>309.0</td>
<td>880.0</td>
</tr>
</tbody>
</table>
Goodness-of-Fit Criteria (Ordered by SSE, Stage 1)

<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>741.97761</td>
<td>0.352835</td>
<td>83.288591</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>747.24937</td>
<td>0.354087</td>
<td>83.187919</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td>748.28425</td>
<td>0.354332</td>
<td>83.120805</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>748.43693</td>
<td>0.354368</td>
<td>83.221477</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>748.61490</td>
<td>0.354410</td>
<td>83.255034</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>749.30976</td>
<td>0.354575</td>
<td>83.171141</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>781.34272</td>
<td>0.362074</td>
<td>82.583893</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>782.48499</td>
<td>0.362339</td>
<td>82.567114</td>
</tr>
</tbody>
</table>

Here starts the third stage (stage=2):

Component Selection: SS(y) and R2 (Stage=2)

<table>
<thead>
<tr>
<th>Comp</th>
<th>Eigval</th>
<th>R-Square</th>
<th>F Value</th>
<th>p-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15337</td>
<td>0.006514</td>
<td>39.068994</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>5</td>
<td>8117.555354</td>
<td>0.005566</td>
<td>33.564983</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>7</td>
<td>7371.205837</td>
<td>0.005429</td>
<td>32.918782</td>
<td>&lt;.0001</td>
</tr>
</tbody>
</table>

When fitting the second order residuals the average value of the objective function dropped from 0.063 to 0.061.
COS: Iter=3 Crit=0.06277476: SSE=741.349891 Acc= 83.2886
------------- Activation= EXP (Stage=2) -------------
NOTE: ABSGCONV convergence criterion satisfied.
EXP: Iter=4 Crit=0.06131052: SSE=721.679089 Acc= 83.4396

The best accuracy increased from 83.29 to 83.47 and the (1,1) entry from 319 to 343 counts:

\[
\text{Classification Table for CUTOFF = 0.5000}
\]

<table>
<thead>
<tr>
<th>Activation</th>
<th>Accuracy</th>
<th>Observed</th>
<th>Predicted</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQUARE</td>
<td>83.473154</td>
<td>1</td>
<td>343.0</td>
<td>846.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>721.118524</td>
<td>0</td>
<td>139.0</td>
<td>4632.0</td>
<td></td>
</tr>
<tr>
<td>EXP</td>
<td>83.422819</td>
<td>1</td>
<td>337.0</td>
<td>852.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>721.301910</td>
<td>0</td>
<td>136.0</td>
<td>4635.0</td>
<td></td>
</tr>
<tr>
<td>LOGIST</td>
<td>83.607383</td>
<td>1</td>
<td>337.0</td>
<td>852.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>724.687746</td>
<td>0</td>
<td>125.0</td>
<td>4646.0</td>
<td></td>
</tr>
<tr>
<td>TANH</td>
<td>83.607383</td>
<td>1</td>
<td>340.0</td>
<td>849.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>725.553808</td>
<td>0</td>
<td>128.0</td>
<td>4643.0</td>
<td></td>
</tr>
<tr>
<td>ARCTAN</td>
<td>83.607383</td>
<td>1</td>
<td>341.0</td>
<td>848.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>725.724668</td>
<td>0</td>
<td>129.0</td>
<td>4642.0</td>
<td></td>
</tr>
<tr>
<td>SIN</td>
<td>83.607383</td>
<td>1</td>
<td>340.0</td>
<td>849.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>725.889780</td>
<td>0</td>
<td>128.0</td>
<td>4643.0</td>
<td></td>
</tr>
<tr>
<td>GAUSS</td>
<td>83.372483</td>
<td>1</td>
<td>317.0</td>
<td>872.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>741.471407</td>
<td>0</td>
<td>119.0</td>
<td>4652.0</td>
<td></td>
</tr>
<tr>
<td>COS</td>
<td>83.322148</td>
<td>1</td>
<td>316.0</td>
<td>873.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>741.668156</td>
<td>0</td>
<td>121.0</td>
<td>4650.0</td>
<td></td>
</tr>
</tbody>
</table>

Even though SQUARE shows the best SSE, the accuracy rates for some other functions (e.g. LOGIST) are slightly better:

\[
\text{Goodness-of-Fit Criteria (Ordered by SSE, Stage 2)}
\]

<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>721.11852</td>
<td>0.347841</td>
<td>83.473154</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>721.30191</td>
<td>0.347885</td>
<td>83.422819</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>724.68775</td>
<td>0.348700</td>
<td>83.607383</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>725.55381</td>
<td>0.348909</td>
<td>83.607383</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>725.72467</td>
<td>0.348950</td>
<td>83.607383</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td>725.88978</td>
<td>0.348989</td>
<td>83.607383</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>741.47140</td>
<td>0.352715</td>
<td>83.372483</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>741.66816</td>
<td>0.352762</td>
<td>83.322148</td>
</tr>
</tbody>
</table>

Component selection w.r.t. the residuals of the stage 2 starts the estimation of stage 3. Note, that the $R^2$ values become smaller and smaller.
Component Selection: SS(y) and R2 (Stage=3)

<table>
<thead>
<tr>
<th>Comp</th>
<th>Eigval</th>
<th>R-Square</th>
<th>F Value</th>
<th>p-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>6938.083228</td>
<td>0.005571</td>
<td>33.383374</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>20</td>
<td>5345.603436</td>
<td>0.004223</td>
<td>25.409312</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>12</td>
<td>6136.575271</td>
<td>0.004059</td>
<td>24.517995</td>
<td>&lt;.0001</td>
</tr>
</tbody>
</table>

Also the size of the objective function at the optimization results decreases:

------------- Optimization Cycle (Stage=3) --------------
------------- Activation= SQUARE (Stage=3) -------------
NOTE: ABSGCONV convergence criterion satisfied.
SQUARE: Iter=1 Crit=0.06049339: SSE=710.516275 Acc= 83.7081
------------- Activation= TANH (Stage=3) -------------
NOTE: ABSGCONV convergence criterion satisfied.
TANH: Iter=4 Crit=0.06052425: SSE=710.396136 Acc= 83.7752
------------- Activation= ARCTAN (Stage=3) -------------
NOTE: ABSGCONV convergence criterion satisfied.
ARCTAN: Iter=3 Crit=0.06052607: SSE=710.489715 Acc= 83.7081
------------- Activation= LOGIST (Stage=3) -------------
NOTE: ABSGCONV convergence criterion satisfied.
LOGIST: Iter=6 Crit=0.06055936: SSE=711.054572 Acc= 83.6577
------------- Activation= GAUSS (Stage=3) -------------
NOTE: ABSGCONV convergence criterion satisfied.
GAUSS: Iter=6 Crit=0.06111674: SSE= 719.41694 Acc= 83.3725
------------- Activation= SIN (Stage=3) -------------
NOTE: ABSGCONV convergence criterion satisfied.
SIN: Iter=3 Crit=0.06051959: SSE=710.308709 Acc= 83.8087
------------- Activation= COS (Stage=3) -------------
NOTE: ABSGCONV convergence criterion satisfied.
COS: Iter=6 Crit=0.06117044: SSE=719.262211 Acc= 83.3725
------------- Activation= EXP (Stage=3) -------------
NOTE: ABSGCONV convergence criterion satisfied.
EXP: Iter=2 Crit=0.06051088: SSE=710.810558 Acc= 83.7081

The accuracy of the best fit improves slightly from 83.47 to 83.79 and the size of the (1,1) entry increases from 343 to 364.

Classification Table for CUTOFF = 0.5000

<table>
<thead>
<tr>
<th>Activation</th>
<th>Accuracy</th>
<th>Observed</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIN</td>
<td>83.791946</td>
<td>1</td>
<td>364.0</td>
</tr>
<tr>
<td>SIN</td>
<td>709.778632</td>
<td>0</td>
<td>141.0</td>
</tr>
<tr>
<td>TANH</td>
<td>83.758389</td>
<td>1</td>
<td>363.0</td>
</tr>
</tbody>
</table>
Goodness-of-Fit Criteria (Ordered by SSE, Stage 3)

<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>SIN</td>
<td>709.77863</td>
<td>0.345095</td>
<td>83.791946</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>709.88296</td>
<td>0.345122</td>
<td>83.758389</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>710.03639</td>
<td>0.345157</td>
<td>83.708054</td>
</tr>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>710.07520</td>
<td>0.345167</td>
<td>83.724832</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>710.21216</td>
<td>0.345167</td>
<td>83.741611</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>710.82265</td>
<td>0.345348</td>
<td>83.691275</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>718.94491</td>
<td>0.347316</td>
<td>83.288591</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>719.26996</td>
<td>0.347394</td>
<td>83.288591</td>
</tr>
</tbody>
</table>

Now the residuals are computed and components are selected for the last estimation stage:

Component Selection: SS(y) and R2 (Stage=4)

<table>
<thead>
<tr>
<th>Comp</th>
<th>Eigval</th>
<th>R-Square</th>
<th>F Value</th>
<th>p-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>1195.710958</td>
<td>0.003997</td>
<td>23.916548</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>27</td>
<td>3456.490592</td>
<td>0.001822</td>
<td>10.919693</td>
<td>0.0010</td>
</tr>
<tr>
<td>25</td>
<td>3935.018952</td>
<td>0.001803</td>
<td>10.824185</td>
<td>0.0010</td>
</tr>
</tbody>
</table>

There are no problems with the optimization processes:

------------ Optimization Cycle (Stage=4) ------------
------------ Activation= SQUARE (Stage=4) ------------
NOTE: ABSGCONV convergence criterion satisfied.
SQUARE: Iter=1 Crit=0.05983921: SSE=703.669268 Acc= 83.6913
------------ Activation= TANH (Stage=4) ------------
NOTE: ABSGCONV convergence criterion satisfied.
TANH: Iter=5 Crit=0.06015823: SSE=706.476969 Acc= 83.6074
---------- Activation= ARCTAN (Stage=4) ----------
NOTE: ABSGCONV convergence criterion satisfied.
ARCTAN: Iter=3 Crit=0.06013359: SSE=706.212332 Acc= 83.7081
---------- Activation= LOGIST (Stage=4) ----------
NOTE: ABSGCONV convergence criterion satisfied.
LOGIST: Iter=3 Crit=0.06017552: SSE=706.851414 Acc= 83.7919
---------- Activation= GAUSS (Stage=4) ----------
NOTE: ABSGCONV convergence criterion satisfied.
GAUSS: Iter=4 Crit=0.06032127: SSE=708.571854 Acc= 83.8255
---------- Activation= SIN (Stage=4) ----------
NOTE: ABSGCONV convergence criterion satisfied.
SIN: Iter=3 Crit=0.06014411: SSE=706.402904 Acc= 83.6745
---------- Activation= COS (Stage=4) ----------
NOTE: ABSGCONV convergence criterion satisfied.
COS: Iter=4 Crit=0.06007575: SSE=707.016805 Acc= 83.8087
---------- Activation= EXP (Stage=4) ----------
NOTE: ABSGCONV convergence criterion satisfied.
EXP: Iter=3 Crit=0.05983526: SSE=703.933766 Acc= 83.6074

The accuracy of the result is no longer improved and drops from 83.79 to 83.72, and also the (1,1) entry was decreased from 365 to 363. This can happen only when the discretization error becomes too large in relation to the goodness of fit of the nonlinear model. Perhaps the specification of larger values for MAXCOMP= and NPOINT= could improve the solution. However, in most applications we would see this behavior as a sign that no further improvement of the model fit is possible.

Classification Table for CUTOFF = 0.5000

<table>
<thead>
<tr>
<th>Activation</th>
<th>Accuracy</th>
<th>Observed</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQUARE</td>
<td>83.724832</td>
<td>702.899794</td>
<td>363</td>
<td>826</td>
</tr>
<tr>
<td></td>
<td>83.691275</td>
<td>703.295564</td>
<td>361</td>
<td>828</td>
</tr>
<tr>
<td>ARCTAN</td>
<td>83.775168</td>
<td>705.243085</td>
<td>364</td>
<td>825</td>
</tr>
<tr>
<td>SIN</td>
<td>83.691275</td>
<td>705.508160</td>
<td>363</td>
<td>826</td>
</tr>
<tr>
<td>TANH</td>
<td>83.708054</td>
<td>705.634506</td>
<td>362</td>
<td>827</td>
</tr>
<tr>
<td>LOGIST</td>
<td>83.708054</td>
<td>705.732595</td>
<td>360</td>
<td>829</td>
</tr>
<tr>
<td>COS</td>
<td>83.842282</td>
<td>707.292433</td>
<td>364</td>
<td>825</td>
</tr>
<tr>
<td>GAUSS</td>
<td>83.791946</td>
<td>708.659944</td>
<td>138</td>
<td>463</td>
</tr>
</tbody>
</table>

Classification Table for CUTOFF = 0.5000

<table>
<thead>
<tr>
<th>Activation</th>
<th>Accuracy</th>
<th>Observed</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQUARE</td>
<td>83.724832</td>
<td>702.899794</td>
<td>363</td>
<td>826</td>
</tr>
<tr>
<td></td>
<td>83.691275</td>
<td>703.295564</td>
<td>361</td>
<td>828</td>
</tr>
<tr>
<td>ARCTAN</td>
<td>83.775168</td>
<td>705.243085</td>
<td>364</td>
<td>825</td>
</tr>
<tr>
<td>SIN</td>
<td>83.691275</td>
<td>705.508160</td>
<td>363</td>
<td>826</td>
</tr>
<tr>
<td>TANH</td>
<td>83.708054</td>
<td>705.634506</td>
<td>362</td>
<td>827</td>
</tr>
<tr>
<td>LOGIST</td>
<td>83.708054</td>
<td>705.732595</td>
<td>360</td>
<td>829</td>
</tr>
<tr>
<td>COS</td>
<td>83.842282</td>
<td>707.292433</td>
<td>364</td>
<td>825</td>
</tr>
<tr>
<td>GAUSS</td>
<td>83.791946</td>
<td>708.659944</td>
<td>138</td>
<td>463</td>
</tr>
</tbody>
</table>

The accuracy of the result is no longer improved and drops from 83.79 to 83.72, and also the (1,1) entry was decreased from 365 to 363. This can happen only when the discretization error becomes too large in relation to the goodness of fit of the nonlinear model. Perhaps the specification of larger values for MAXCOMP= and NPOINT= could improve the solution. However, in most applications we would see this behavior as a sign that no further improvement of the model fit is possible.
Even though accuracy did not improve, the SSE value still dropped from 710 to 703 during the last stage.

**Goodness-of-Fit Criteria (Ordered by SSE, Stage 4)**

<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>702.89979</td>
<td>0.343418</td>
<td>83.724832</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>703.29556</td>
<td>0.343515</td>
<td>83.691275</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>705.24309</td>
<td>0.343990</td>
<td>83.775168</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td>705.50816</td>
<td>0.344055</td>
<td>83.691275</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>705.63451</td>
<td>0.344086</td>
<td>83.708054</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>705.73259</td>
<td>0.344110</td>
<td>83.708054</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>707.29243</td>
<td>0.344490</td>
<td>83.842282</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>708.65994</td>
<td>0.344823</td>
<td>83.791946</td>
</tr>
</tbody>
</table>

The following summary table shows the improvements in SSE and Accuracy rates across the 5 stages:

**Summary Table Across Stages**

<table>
<thead>
<tr>
<th>Stage</th>
<th>Activation</th>
<th>Link</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
<th>AIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SQUARE</td>
<td>LOGIST</td>
<td>805.1902</td>
<td>0.36756</td>
<td>81.61074</td>
<td>-11730</td>
</tr>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>IDENT</td>
<td>741.9776</td>
<td>0.35284</td>
<td>83.28859</td>
<td>-12018</td>
</tr>
<tr>
<td>2</td>
<td>SQUARE</td>
<td>IDENT</td>
<td>721.1185</td>
<td>0.34784</td>
<td>83.47315</td>
<td>-11988</td>
</tr>
<tr>
<td>3</td>
<td>SIN</td>
<td>IDENT</td>
<td>709.7786</td>
<td>0.34510</td>
<td>83.79195</td>
<td>-11882</td>
</tr>
<tr>
<td>4</td>
<td>SQUARE</td>
<td>IDENT</td>
<td>702.8998</td>
<td>0.34342</td>
<td>83.72483</td>
<td>-11740</td>
</tr>
</tbody>
</table>

All 40 optimizations were very efficient with about 5 iterations per optimization and less than 10 function calls per optimization:

*** Total Number of Runs through Data : 27  
*** Total Number of NL Optimizations : 40  
*** Total Number of Iterations in NLP : 219  
*** Total Number Function Calls in NLP: 392

In this application those solutions were selected which had the smallest Sum-of-Squares Error. By specifying the `selcrit=acc` option we can instead select the solutions with the largest accuracy rate:

```plaintext
proc dmneurl data=dmdbout dmdbcat=outcat  
   outclass=oclass outest=estout out=dsout outfit=ofit  
   ptable maxcomp=3 maxstage=5 selcrit=acc;  
   var LOAN MORTDUE VALUE REASON JOB YOJ DEROG DELINQ  
      CLAGE NINQ CLNO DEBTINC;  
   target BAD;  
run;
```
The following output only shows the summary table. For this example, the total accuracy was slightly increased in all stages except the second. However, this behavior must not always be true for other examples.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Activation</th>
<th>Link</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
<th>AIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ARCTAN</td>
<td>LOGIST</td>
<td>805.8911</td>
<td>0.36772</td>
<td>81.77852</td>
<td>-11725</td>
</tr>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>IDENT</td>
<td>746.6223</td>
<td>0.35394</td>
<td>83.27181</td>
<td>-11981</td>
</tr>
<tr>
<td>2</td>
<td>LOGIST</td>
<td>IDENT</td>
<td>730.7282</td>
<td>0.35015</td>
<td>83.85906</td>
<td>-11909</td>
</tr>
<tr>
<td>3</td>
<td>EXP</td>
<td>IDENT</td>
<td>714.0782</td>
<td>0.34614</td>
<td>84.02685</td>
<td>-11846</td>
</tr>
<tr>
<td>4</td>
<td>COS</td>
<td>IDENT</td>
<td>711.5807</td>
<td>0.34553</td>
<td>83.97651</td>
<td>-11667</td>
</tr>
</tbody>
</table>

Application: HMEQ Data Set: Interval Target LOAN

Now we show the specification and results of PROC DMNEURL for the interval target LOAN. First we have to obtain the DMDB data set and catalog from the raw data set:

```sas
libname sampsio '/sas/a612/dmine/sampsio';
proc dmdb batch data=sampsio.hmeq out=dmdbout dmdbcat=outcat;
    var LOAN MORTDUE VALUE YOJ DELINQ CLAGE NINQ CLNO DEBTINC;
    class BAD(ASC) REASON(ASC) JOB(ASC) DEROG(ASC);
    target LOAN;
run;
```

The PROC DMNEURL call is very similar, but here 5 stages with each 3 components ($p = 7$) are specified:

```sas
proc dmneurl data=dmdbout dmdbcat=outcat
    outclass=oclass outest=estout out=dsout outfit=ofit
    ptable maxcomp=3 maxstage=6;
    var BAD MORTDUE VALUE REASON JOB YOJ DEROG DELINQ CLAGE NINQ CLNO DEBTINC;
    target LOAN;
run;
```

The link function for interval target is by default specified as the identity:

```
Purpose of PROC DMNEURL

<table>
<thead>
<tr>
<th>Summary Table Across Stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

The DMNEURL Procedure

<table>
<thead>
<tr>
<th>Interval Target</th>
<th>LOAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number Observations</td>
<td>5960</td>
</tr>
<tr>
<td>NOBS w/o Missing Target</td>
<td>5960</td>
</tr>
<tr>
<td>Target Range</td>
<td>[ 1100, 89900]</td>
</tr>
<tr>
<td>Link Function</td>
<td>IDENT</td>
</tr>
<tr>
<td>Selection Criterion</td>
<td>SSE</td>
</tr>
<tr>
<td>Optimization Criterion</td>
<td>SSE</td>
</tr>
</tbody>
</table>
```

The link function for interval target is by default specified as the identity:
Estimation Stages 6
Max. Number Components 3
Minimum R2 Value 0.000050
Number Grid Points 17

Variable	Mean	Std Dev	Skewness	Kurtosis
LOAN	18608	11207	2.02378	6.93259
MORTDUE	67350	44458	1.81448	6.48187
VALUE	99863	57386	3.05334	24.36280
YOJ	8.15130	7.57398	0.98846	0.37207
DELINQ	0.40570	1.12727	4.02315	23.56545
CLAGE	170.47634	85.81009	1.34341	7.59955
NINQ	1.08456	1.72867	2.62198	9.78651
CLNO	20.50285	10.13893	0.77505	1.15767
DEBTINC	26.59885	8.60175	2.85235	50.50404

For an interval target the percentiles of the response (target) variable are computed as an aside of the preliminary runs through the data. (Note, that the values of the response $y$ are not all stored in RAM.)

**Percentiles of Target LOAN in [1100 : 89900]**

<table>
<thead>
<tr>
<th>Nobs</th>
<th>Y Value</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>596</td>
<td>7600.000000</td>
</tr>
<tr>
<td>2</td>
<td>1192</td>
<td>10000</td>
</tr>
<tr>
<td>3</td>
<td>1788</td>
<td>12100</td>
</tr>
<tr>
<td>4</td>
<td>2384</td>
<td>14400</td>
</tr>
<tr>
<td>5</td>
<td>2980</td>
<td>16300</td>
</tr>
<tr>
<td>6</td>
<td>3576</td>
<td>18800</td>
</tr>
<tr>
<td>7</td>
<td>4172</td>
<td>21700</td>
</tr>
<tr>
<td>8</td>
<td>4768</td>
<td>25000</td>
</tr>
<tr>
<td>9</td>
<td>5364</td>
<td>30500</td>
</tr>
<tr>
<td>10</td>
<td>5960</td>
<td>89900</td>
</tr>
</tbody>
</table>

The first estimation stage starts with the selection of the best predictor components (eigenvectors):

**Component Selection: SS(y) and R2 (SS_total=326.60303927)**

<table>
<thead>
<tr>
<th>Comp</th>
<th>Eigval</th>
<th>R-Square</th>
<th>F Value</th>
<th>p-Value</th>
<th>SSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>14414</td>
<td>0.015964</td>
<td>96.672480</td>
<td>&lt;.0001</td>
<td>321.389163</td>
</tr>
<tr>
<td>28</td>
<td>1232.230727</td>
<td>0.005739</td>
<td>34.949673</td>
<td>&lt;.0001</td>
<td>319.514886</td>
</tr>
<tr>
<td>11</td>
<td>6335.576701</td>
<td>0.005490</td>
<td>33.620923</td>
<td>&lt;.0001</td>
<td>317.721686</td>
</tr>
</tbody>
</table>

A maximum of 8 iterations is needed for convergence:
For interval target $y$ the accuracy is computed as the Goodman-Kruskal $\gamma$ coefficient for observed-predicted frequency table using the percentiles of row and column definitions. (Note, that the Goodman-Kruskal $\gamma$ can have negative values for extrem bad fit.)

### Approximate Goodness-of-Fit Criteria (Stage 0)

<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>Criterion</th>
<th>SSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>0.007196</td>
<td>676373814905</td>
<td>32.748384</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>0.007245</td>
<td>681021814295</td>
<td>29.901149</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>0.007294</td>
<td>685560807525</td>
<td>29.242251</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>0.007296</td>
<td>685783817673</td>
<td>28.772685</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>0.007305</td>
<td>686651267193</td>
<td>29.242724</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td>0.007325</td>
<td>688526431720</td>
<td>29.039929</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>0.007532</td>
<td>708006119992</td>
<td>15.418007</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>0.007539</td>
<td>708601735521</td>
<td>22.553358</td>
</tr>
</tbody>
</table>

The Root-Mean-Squared-Estimate RMSE for the first stage is 10589:
<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>6.68237E11</td>
<td>10589</td>
<td>33.925841</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>6.74156E11</td>
<td>10635</td>
<td>30.776431</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>6.74885E11</td>
<td>10641</td>
<td>30.492000</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>6.77111E11</td>
<td>10659</td>
<td>30.503925</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>6.78503E11</td>
<td>10670</td>
<td>29.832932</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td>6.81758E11</td>
<td>10695</td>
<td>29.402115</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>7.02918E11</td>
<td>10860</td>
<td>21.174016</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>7.04934E11</td>
<td>10876</td>
<td>20.022492</td>
</tr>
</tbody>
</table>

The second stage (stage=1) starts with selecting the best principal components for predicting the residual:

**Component Selection: SS(y) and R2 (Stage=1)**

<table>
<thead>
<tr>
<th>Comp</th>
<th>Eigval</th>
<th>R-Square</th>
<th>F Value</th>
<th>p-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16197</td>
<td>0.023135</td>
<td>141.126005</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>3</td>
<td>12170</td>
<td>0.017130</td>
<td>106.340108</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>20</td>
<td>5623.081574</td>
<td>0.012121</td>
<td>76.193667</td>
<td>&lt;.0001</td>
</tr>
</tbody>
</table>

Now a maximum of 5 iterations is needed for convergence:

---

**Optimization Cycle (Stage=1)**

**NOTE:** ABSGCONV convergence criterion satisfied.

**SQUARE: Iter=1 Crit=0.00675824: SSE=6.35237E11 Acc=39.9782**

**NOTE:** ABSGCONV convergence criterion satisfied.

**TANH: Iter=4 Crit=0.00677155: SSE=6.36489E11 Acc=40.1296**

**NOTE:** ABSGCONV convergence criterion satisfied.

**ARCTAN: Iter=4 Crit=0.00675928: SSE=6.35335E11 Acc=41.0832**

**NOTE:** ABSGCONV convergence criterion satisfied.

**LOGIST: Iter=5 Crit=0.00676491: SSE=6.35864E11 Acc=41.1768**

**NOTE:** ABSGCONV convergence criterion satisfied.

**GAUSS: Iter=5 Crit=0.00701114: SSE=6.59009E11 Acc=37.5715**

**NOTE:** ABSGCONV convergence criterion satisfied.

**SIN: Iter=4 Crit=0.00676857: SSE=6.36208E11 Acc=40.1257**

**NOTE:** ABSGCONV convergence criterion satisfied.

**COS: Iter=5 Crit=0.00699351: SSE=6.57351E11 Acc=36.1925**

**NOTE:** ABSGCONV convergence criterion satisfied.

**EXP: Iter=1 Crit=0.00676817: SSE=6.3617E11 Acc=40.3710**

**NOTE:** ABSGCONV convergence criterion satisfied.
The RMSE dropped from 10589 to 10294:

**Goodness-of-Fit Criteria (Ordered by SSE, Stage 1)**

<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>6.31521E11</td>
<td>10294</td>
<td>41.275893</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>6.32669E11</td>
<td>10303</td>
<td>40.218692</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>6.32795E11</td>
<td>10304</td>
<td>40.947613</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td>6.32908E11</td>
<td>10305</td>
<td>40.499603</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>6.3331E11</td>
<td>10308</td>
<td>40.464591</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>6.33355E11</td>
<td>10309</td>
<td>40.566497</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>6.56699E11</td>
<td>10497</td>
<td>36.381117</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>6.58666E11</td>
<td>10513</td>
<td>37.785522</td>
</tr>
</tbody>
</table>

The third stage starts with selecting the best eigenvectors for prediction of the residuals of the last stage:

**Component Selection: SS(y) and R2 (Stage=2)**

<table>
<thead>
<tr>
<th>Comp</th>
<th>Eigval</th>
<th>R-Square</th>
<th>F Value</th>
<th>p-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>4811.081772</td>
<td>0.011805</td>
<td>71.186100</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>28</td>
<td>1232.230727</td>
<td>0.007479</td>
<td>45.434565</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>18</td>
<td>5865.674624</td>
<td>0.006724</td>
<td>41.123895</td>
<td>&lt;.0001</td>
</tr>
</tbody>
</table>

Now, the maximum of iterations is four!

--- Optimization Cycle (Stage=2) ---

NOTE: ABSGCONV convergence criterion satisfied.
SQUARE: Iter=1 Crit=0.00652258: SSE=6.13086E11 Acc= 42.4729

--- Activation= TANH (Stage=2) ---

NOTE: ABSGCONV convergence criterion satisfied.
TANH: Iter=3 Crit=0.00650972: SSE=6.11878E11 Acc= 41.5654

--- Activation= ARCTAN (Stage=2) ---

NOTE: ABSGCONV convergence criterion satisfied.
ARCTAN: Iter=3 Crit=0.00650403: SSE=6.11342E11 Acc= 42.2974

--- Activation= LOGIST (Stage=2) ---

NOTE: ABSGCONV convergence criterion satisfied.
LOGIST: Iter=4 Crit=0.00641908: SSE=6.03358E11 Acc= 43.3779

--- Activation= GAUSS (Stage=2) ---

NOTE: ABSGCONV convergence criterion satisfied.
GAUSS: Iter=2 Crit=0.00671802: SSE=6.31456E11 Acc= 41.2660

--- Activation= SIN (Stage=2) ---

NOTE: ABSGCONV convergence criterion satisfied.
SIN: Iter=4 Crit=0.00651654: SSE=6.12519E11 Acc= 41.9809

--- Activation= COS (Stage=2) ---

NOTE: ABSGCONV convergence criterion satisfied.
COS: Iter=4 Crit=0.00671738: SSE=6.31396E11 Acc= 41.2783
-------------- Activation= EXP (Stage=2) ---------------
NOTE: ABSCONV convergence criterion satisfied.
EXP: Iter=0 Crit=0.00656615: SSE=6.17182E11 Acc= 41.6251

The RMSE dropped from 10294 to 10035:

Goodness-of-Fit Criteria (Ordered by SSE, Stage 2)

<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>6.00171E11</td>
<td>10035</td>
<td>44.076324</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>6.11611E11</td>
<td>10130</td>
<td>42.722853</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>6.12574E11</td>
<td>10138</td>
<td>42.454925</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td>6.12902E11</td>
<td>10141</td>
<td>42.618536</td>
</tr>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>6.14545E11</td>
<td>10154</td>
<td>43.452922</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>6.17964E11</td>
<td>10183</td>
<td>42.588823</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>6.31415E11</td>
<td>10293</td>
<td>41.153349</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>6.31533E11</td>
<td>10294</td>
<td>41.028769</td>
</tr>
</tbody>
</table>

In stage 3 components are selected w.r.t. the residuals from stage 2:

Component Selection: SS(y) and R2 (Stage=3)

<table>
<thead>
<tr>
<th>Comp</th>
<th>Eigval</th>
<th>R-Square</th>
<th>F Value</th>
<th>p-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8108.233368</td>
<td>0.008115</td>
<td>48.751302</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>7</td>
<td>7678.598513</td>
<td>0.004569</td>
<td>27.574638</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>27</td>
<td>3496.302840</td>
<td>0.003929</td>
<td>23.802006</td>
<td>&lt;.0001</td>
</tr>
</tbody>
</table>

---------- Optimization Cycle (Stage=3) ----------
---------- Activation= SQUARE (Stage=3) ----------
NOTE: ABSCONV convergence criterion satisfied.
SQUARE: Iter=1 Crit=0.00627437: SSE=5.89756E11 Acc= 46.5664
---------- Activation= TANH (Stage=3) ----------
NOTE: ABSCONV convergence criterion satisfied.
TANH: Iter=2 Crit=0.00628152: SSE=5.90428E11 Acc= 46.2812
---------- Activation= ARCTAN (Stage=3) ----------
NOTE: ABSCONV convergence criterion satisfied.
ARCTAN: Iter=2 Crit=0.00628195: SSE=5.90469E11 Acc= 46.2125
---------- Activation= LOGIST (Stage=3) ----------
NOTE: ABSCONV convergence criterion satisfied.
LOGIST: Iter=5 Crit=0.00628127: SSE=5.90404E11 Acc= 45.8040
---------- Activation= GAUSS (Stage=3) ----------
NOTE: ABSCONV convergence criterion satisfied.
GAUSS: Iter=2 Crit=0.00637032: SSE=5.98775E11 Acc= 45.3250
---------- Activation= SIN (Stage=3) ----------
NOTE: ABSCONV convergence criterion satisfied.
SIN: Iter=6 Crit=0.00627884: SSE=5.90176E11 Acc= 46.0972
-------- Activation= COS (Stage=3) --------
NOTE: ABSGCONV convergence criterion satisfied.
COS: Iter=5 Crit=0.00637738: SSE=5.99438E11 Acc= 44.6439
-------- Activation= EXP (Stage=3) --------
NOTE: ABSGCONV convergence criterion satisfied.
EXP: Iter=1 Crit=0.00628498: SSE=5.90753E11 Acc= 46.4650

The RMSE dropped from 10035 to 9939:

Goodness-of-Fit Criteria (Ordered by SSE, Stage 3)

<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>5.88794E11</td>
<td>9939.361833</td>
<td>46.874168</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td>5.89308E11</td>
<td>9943.696786</td>
<td>46.045231</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>5.89532E11</td>
<td>9945.590341</td>
<td>45.936035</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>5.89658E11</td>
<td>9946.655850</td>
<td>45.621683</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>5.89714E11</td>
<td>9947.122017</td>
<td>46.462470</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>5.89994E11</td>
<td>9949.489480</td>
<td>46.590155</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>5.98777E11</td>
<td>10023</td>
<td>45.338954</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>5.99432E11</td>
<td>10029</td>
<td>44.563112</td>
</tr>
</tbody>
</table>

Again, the new stage 4 starts with component selection w.r.t. the residuals of the last stage3:

Component Selection: SS(y) and R2 (Stage=4)

<table>
<thead>
<tr>
<th>Comp</th>
<th>Eigval</th>
<th>R-Square</th>
<th>F Value</th>
<th>p-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>5977.581155</td>
<td>0.004044</td>
<td>24.196310</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>24</td>
<td>4589.938565</td>
<td>0.002803</td>
<td>16.817296</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>8</td>
<td>7098.575517</td>
<td>0.002425</td>
<td>14.583721</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

A maximum of 7 iterations is needed for convergence:

-------- Optimization Cycle (Stage=4) --------
-------- Activation= SQUARE (Stage=4) --------
NOTE: ABSGCONV convergence criterion satisfied.
SQUARE: Iter=1 Crit=0.00618628: SSE=5.81476E11 Acc= 46.9112
-------- Activation= TANH (Stage=4) --------
NOTE: ABSGCONV convergence criterion satisfied.
TANH: Iter=5 Crit= 0.0061812: SSE=5.80998E11 Acc= 45.5545
-------- Activation= ARCTAN (Stage=4) --------
NOTE: ABSGCONV convergence criterion satisfied.
ARCTAN: Iter=3 Crit=0.00618984: SSE= 5.8181E11 Acc= 45.7876
-------- Activation= LOGIST (Stage=4) --------
NOTE: ABSGCONV convergence criterion satisfied.
LOGIST: Iter=2 Crit= 0.0062313: SSE= 5.85708E11 Acc= 47.2978
The RMSE dropped from 9939 to 9889:

**Goodness-of-Fit Criteria (Ordered by SSE, Stage 4)**

<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>5.82844E11</td>
<td>9889.013804</td>
<td>46.529424</td>
</tr>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>5.82906E11</td>
<td>9889.541858</td>
<td>47.202054</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>5.83336E11</td>
<td>9893.184747</td>
<td>47.340247</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>5.83553E11</td>
<td>9895.031798</td>
<td>47.907199</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>5.8489E11</td>
<td>9906.353245</td>
<td>46.238301</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>5.86142E11</td>
<td>9916.953006</td>
<td>47.605286</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>5.88716E11</td>
<td>9938.707816</td>
<td>46.013181</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td>6.12035E11</td>
<td>10134</td>
<td>45.185468</td>
</tr>
</tbody>
</table>

For space reasons we are skipping the results of stage 5 except the following table which shows that the RMSE dropped again.

**Goodness-of-Fit Criteria (Ordered by SSE, Stage 5)**

<table>
<thead>
<tr>
<th>Run</th>
<th>Activation</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQUARE</td>
<td>5.78114E11</td>
<td>9848.803178</td>
<td>47.337721</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
<td>5.78394E11</td>
<td>9851.192061</td>
<td>47.342324</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
<td>5.78507E11</td>
<td>9852.150383</td>
<td>47.269327</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
<td>5.79057E11</td>
<td>9856.832196</td>
<td>46.609720</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
<td>5.8133E11</td>
<td>9876.166691</td>
<td>46.144529</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
<td>5.82144E11</td>
<td>9883.077740</td>
<td>46.792103</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td>5.82405E11</td>
<td>9885.287904</td>
<td>46.540469</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td>6.10243E11</td>
<td>10119</td>
<td>45.311873</td>
</tr>
</tbody>
</table>

This is a summary table for the first six estimation stages:

**Summary Table Across Stages**

<table>
<thead>
<tr>
<th>Stage</th>
<th>Activation</th>
<th>Link</th>
<th>SSE</th>
<th>RMSE</th>
<th>Accuracy</th>
<th>AIC</th>
</tr>
</thead>
</table>

---

**Activation= GAUSS (Stage=4)**

NOTE: ABSGCONV convergence criterion satisfied.

GAUSS: Iter=3 Crit=0.00617962: SSE= 5.8085E11 Acc= 46.3454

**Activation= SIN (Stage=4)**

NOTE: ABSGCONV convergence criterion satisfied.

SIN: Iter=7 Crit=0.00614634: SSE=5.7772E11 Acc= 45.3470

**Activation= COS (Stage=4)**

NOTE: ABSGCONV convergence criterion satisfied.

COS: Iter=2 Crit=0.00619158: SSE=5.8197E11 Acc= 47.5278

**Activation= EXP (Stage=4)**

NOTE: ABSGCONV convergence criterion satisfied.

EXP: Iter=2 Crit=0.00620074: SSE=5.8283E11 Acc= 47.1760
The six stages took 48 optimizations (each with 7 parameters) and 33 runs through the data. In average less than 4 iterations and about 7 function calls are needed for each optimization:

*** Total Number of Runs through Data : 33
*** Total Number of NL Optimizations : 48
*** Total Number of Iterations in NLP : 159
*** Total Number Function Calls in NLP: 348

Missing Values

Observations with missing values in the target variable (response or dependend variable) are not included in the analysis. Those observations are, however, scored, i.e. predicted values are computed.

Observations with missing values in the predictor variables (independend variables) are processed depending on the scale type of the variable:

- For numeric variables, missing values are replaced by the (weighted) mean of the variable.
- For class variables, missing values are treated as an additional category.

Syntax of PROC DMNEURL

```
PROC DMNEURL options;  required statement
VAR variables;
TARGET variables;
FUNCTION names;
LINK name;
FREQ variables;
WEIGHT variables;
DECISION options;
```

Overview of PROC DMNEURL Options

```
PROC DMNEURL options;
```
This statement invokes the DMNEURL procedure. The options available with the PROC DMNEURL statement are:
DATA=SASdataset :
specifies an input data set generated by PROC DMDB which is associated with a valid catalog specified by the DMDBCAT= option. This option must be specified, no default is permitted. The DATA= data set must contain interval scaled variables and CLASS variables in a specific form written by PROC DMDB.

DMDBCAT=SAScatalog :
specifies an input catalog of meta information generated by PROC DMDB which is associated with a valid data set specified by the DATA= option. The catalog contains important information (e.g. range of variables, number of missing values of each variable, moments of variables) which is used by many other procedures which require a DMDB data set. That means, that both, the DMDBCAT= catalog and the DATA= data set must be InSync to obtain proper results! This option must be specified, no default is permitted.

TESTDATA=SASdataset :
specifies a second input data set which is by default NOT generated by PROC DMDB, which however must contain all variables of the DATA= input data set which are used in the model. The variables not used in the model may be different. The order of variables is not relevant. If TESTDATA= is specified, you can specify a TESTOUT= output data set (containing predicted values and residuals) which relates to the TESTDATA= input data set the same as the OUT= data set relates to the DATA= input training data set. When specifying the TESTDMDB option you may use a data set generated by PROC DMDB as the TESTDATA= input data set.

OUTCLASS=SASdataset :
specifies an output data set generated by PROC DMNEURL which contains the mapping inbetween compound variable names and the names of variables and categories of CLASS variables used in the model. The compound variable names are used to denote dummy variables which are created for each category of a CLASS variable with more than two categories. Since the compound names of dummy variables are used for variable names in other data sets the user must know to which category each compound name corresponds. The OUTCLASS= data set has only three character variables

(NAME) contains compound name used as variable names in other output data sets

(VAR) contains variable name used in DATA= input data set

(LEVEL) contains level name of variable as used in DATA= input data set.

Note, if the DATA= input data set does not contain any CLASS variables the OUTCLASS= data set is not written.

OUTEST=SASdataset :
specifies an output data set generated by PROC DMNEURL which contains all the model information necessary for scoring additional cases or data sets.

Variables of the output data set:

(TARGET) (character) name of the target
_TYPE_  (character) type of observation
_NAME_  (character) name of observation
_STAGE_  number of stage
_MEAN_  contains different numeric information
_STDEV_  contains different numeric information

Variables in the model variables: the first variables correspond to CLASS (categorical) the remaining variables are continuously (interval or ratio) scaled. Note, that for nonbinary CLASS (nominal or ordinal categorical) variables a set of binary dummy variables is created. In those cases the prefix of variable names used for a group of variables in the data set may be the same for a successive group of variables which differs only by a numeric suffix.

This data set contains all the model information necessary to compute the predicted model values (scores).

1. The _TYPE_=_V_MAP_ and _TYPE_=_C_MAP_ observations contain the mapping indices between the variables used in the model and the number of the variable in the data set.
   - The _MEAN_ variable contains the number of index mappings.
   - The _STDEV_ variable contains the index of the target (response) variable in the data set for the _TYPE_=_V_MAP_ observation. For _TYPE_=_C_MAP_ it contains the level (category) number of a categorical target variable that corresponds to missing values.

2. The _TYPE_=_EIGVAL_ observation contains the sorted eigenvalues of the $X'X$ matrix. Here, the _MEAN_ variable contains the number of model variables (rows/columns of the model $X'X$ matrix) and the _STDEV_ variable contains the number $c$ of model components.

3. For each stage of the estimation process two groups of observations are written to the OUTEST= data set:
   (a) The _TYPE_=_EIGVEC_ observations contain a set of $c$ principal components which are used as predictor variables for the estimation of the original target value $y$ (in stage 0) or for the prediction of the stage $i$ residual. Here, the _MEAN_ variable contains the value for the criterion used to include the component into the model which is normally the $R^2$ value. The _STDEV_ variable contains the eigenvalue number to which the eigenvector corresponds.

<table>
<thead>
<tr>
<th></th>
<th>Activation Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQUARE</td>
</tr>
<tr>
<td>2</td>
<td>TANH</td>
</tr>
<tr>
<td>3</td>
<td>ARCTAN</td>
</tr>
<tr>
<td>4</td>
<td>LOGIST</td>
</tr>
<tr>
<td>5</td>
<td>GAUSS</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
</tr>
<tr>
<td>8</td>
<td>EXP</td>
</tr>
</tbody>
</table>

The _NAME_ variable reports the corresponding name of the best activation function found.
(b) The _TYPE_=PARMS_ observations contain for each activation function the \( p = 2c + 1 \) parameter estimates. Here, the _MEAN_ variable contains the value for the optimization criterion and the _STDEV_ variable contains the accuracy value of the prediction.

**OUT=SASdataset:** specifies an output data set generated by PROC DMNEURL which contains the predicted values (posteriors) and residuals for all observations in the DATA= input data set.

Variables of the output data set:

- idvarnam_i values of all ID variables
- _TARGET_ (character) name of the target
- _STAGE_ number of stage
- _P_ predicted value \( (\hat{y}) \)
- _R_ residual \( (y - \hat{y}) \)

The following variables are added if a DECISION statement is used:

- _BSTDEC_
- _CONSEQ_
- _EVALUE_ expected profit or cost value

The number of observations in the OUT= data set agrees with that of the DATA= input data set.

**TESTOUT=SASdataset:**

specifies an output data set which is in structure identical to the OUT= output data set but relates to the information given in the TESTDATA= input data set rather than that of the DATA= input data set used in the OUT= output data set.

The number of observations in the TESTOUT= data set agrees with that of the TESTDATA= input data set.

**OUTFIT=SASdataset:**

specifies an output data set generated by PROC DMNEURL which contains a number of fit indices for each stage and for the final model estimates. For a binary target (response variable) it also contains the frequencies of the \( 2 \times 2 \) accuracy table of the best fit at the final stage. The same information is additionally provided if a TESTDATA= input data set is specified.

Variables of the output data set:

- _TARGET_ (character) name of the target
- _DATA_ (character) specifies the data set to which the fit criteria correspond:
  - =TRAINING: fit criteria belong to DATA= input data set
  - =TESTDATA: fit criteria belong to TESTDATA= input data set
- _TYPE_ (character) describes type of observation
  - _TYPE_=FITIND_ for fit indices;
  - _TYPE_=ACCTAB_ for frequencies of accuracy table (only for binary target)
Purpose of PROC DMNEURL

_STAGE_ number of stages in the estimation process
_SSE_ sum-of-squared error of solution
_RMSE_ root mean squared error of solution
_ACCU_ percentage of accuracy of prediction (only for categorical target)
_AIC_ Akaike information criterion
_SBC_ Schwarz’ information criterion

The following variables are added if a DECISION statement is used:

_PROF_
_APROF_
_LOSS_
_ALOSS_
_IC_
_ROI_

OUTSTAT=SASdataset:
specifies an output data set generated by PROC DMNEURL which contains all
eigenvalues and eigenvectors of the $X'X$ matrix. When this option is specified,
no other computations are performed and the procedure terminates after writing
this data set.

Variables of the OUTSTAT= output data set:

_TYPE_ (character) type of observation
_EIGVAL_ contains different numeric information

varname$_i$ variables in the model; the first variables correspond to CLASS
(categorical) the remaining variables are continuously (interval or ratio)
scaled. Note, that for nonbinary CLASS (nominal or ordinal categorical)
variables a set of binary dummy variables is created. In those cases the
prefix of variable names varname$_i$ used for a group of variables in the
data set may be the same for a successive group of variables which differs
only by a numeric suffix.

Observations of the OUTSTAT= output data set:

1. The first three observations, _TYPE_=_V_MAP_ and _TYPE_=_C_MAP_,
contain the mapping indices between the variables used in the model and
the number of the variables in the data set. The _EIGVAL_ variable
contains the number of index mappings. This is the same information
as in the first observation of the OUTEST= data set, except that here
the _TYPE_=_EIGVAL_ variables replaces the _TYPE_=_MEAN_
variable in the OUTEST= data set.

2. The _TYPE_=_EIGVAL_ observation contains the sorted eigenvalues of
the $X'X$ matrix.

3. The _TYPE_=_EIGVEC_ observations contain a set of $n$ eigenvectors
of the $X'X$ matrix. Here, the _EIGVAL_ variable contains the eigen-
value to which the eigenvector corresponds.
**ABSGCONV, ABSGTOL** : $r \geq 0$

specifies an absolute gradient convergence criterion for the default (OPTCRIT=SSE) optimization process. See the document of PROC NLP in SAS/OR for more details. Default is ABSGCONV=5e-4 in general and ABSCONV=1e-3 for FUNCTION=EXP.

**CORRDF** : specifies that the correct number of degrees of freedom is used for the values of RMSE, AIC, and SBC. Without specifying CORRDF the error degrees of freedom are computed as $W - p$, where $W$ is the sum of weights (if the WEIGHT statement is not used, each observation has a weight of 1 assigned, and $W$ is the total number of observations) and $p$ is the number of parameters. When CORRDF is specified the value $p$ is replaced by the rank of the joint Jacobian.

**COV, CORR** : specifies that a covariance or correlation matrix is used for computing eigenvalues and eigenvectors compatible with the PRINCOMP procedure. The COV and CORR options are valid only if an OUTSTAT= data set is specified. If neither COV nor CORR are specified, the eigenvalues and eigenvectors of the cross product matrix $X^T X$ are computed and written to the OUTSTAT= data set.

**CRITWGT=r** : $r > 0$

specifies a positive weight for a weighted least squares fit. Currently this option is valid only for binary target. Values of $r > 1$ will enforce a better fit of the (1,1) entry in the accuracy table which may be useful for fitting rare events. Values of $0 < r < 1$ will enforce a better fit of the (0,0) entry in the accuracy table. Note, that values for $r$ which are far away from $r = 1$ will reduce the fit quality of the remaining entries in the frequency table. At this time values of either $1 < r < 2$ or $.5 < r < 1$ are preferred.

**CUTOFF=r** : $0 < r < 1$

specifies a cutoff threshold for deciding when a predicted value of a binary response is classified as 0 or 1. The default is $cutoff = .5$. If the value of the posterior, $(\hat{y}_i)$, for observation $i$ is smaller than the specified cutoff value, the observation is counted in the first column of the accuracy table (i.e. as 0), otherwise it is counted in the second column (i.e. as 1). For nonbinary target the cutoff= value is not used.

**GCONV, GTOL** : $r \geq 0$

specifies a relative gradient convergence criterion for the optimization process. See the document of PROC NLP in SAS/OR for more details. Default is GCONV=1e-8.

**FCRIT** specifies that the probability of the $F$ test is being used for the selection of principal components rather than the default $R^2$ criterion.

**MAXCOMP=i** : $2 \leq i \leq 8$

specifies an upper bound for the number of components selected for predicting the target in each stage. Good values for MAXCOMP are in between 3 and 5. Note, that the computer time and core memory will increase superlinear for...
larger values than 5. There is one memory allocation which takes $n^m$ long integer values, where $n$ is the value specified with the NPOINT= option and $m$ is the value specified by the MAXCOMP= option. The following table lists values of $4n^m/1000000$ for specific combinations of $(n, m)$. This is the actual memory requirement in Megabytes assuming that a long integer takes 4 bytes storage.

<table>
<thead>
<tr>
<th>$n$</th>
<th>m=3</th>
<th>m=4</th>
<th>m=5</th>
<th>m=6</th>
<th>m=7</th>
<th>m=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>23*</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>19*</td>
<td>172</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>7*</td>
<td>78</td>
<td>857</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>2*</td>
<td>19</td>
<td>250</td>
<td>3263</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0*</td>
<td>3</td>
<td>46</td>
<td>683</td>
<td>10252</td>
</tr>
<tr>
<td>17</td>
<td>0*</td>
<td>0</td>
<td>6</td>
<td>97</td>
<td>1641</td>
<td>27903</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>188</td>
<td>3575</td>
<td>67934</td>
</tr>
</tbody>
</table>

The trailing asterisk indicates the default number of points for a given number of components. Therefore, values larger than 8 for $i$ in MAXCOMP=i are reduced to this upper range. It seems to be better to increase the value $i$ of the MAXSTAGE=i option when higher precision is requested.

**MAXFUNC=i : $i \geq 0$**

specifies an upper bound for the number of function calls in each optimization. The default is MAXFUNC=500. Normally the default number of function calls will be sufficient to reach convergence. Larger values should be used if the iteration history indicates that the optimization process was close to a promising solution but would have needed more than the specified number of function calls. Smaller values should be specified when a faster but suboptimal solution may be sufficient.

**MAXITER=i : $i \geq 0$**

specifies an upper bound for the number of iterations in each optimization. The default is MAXITER=200. Normally the default number of iterations will be sufficient to reach convergence. Larger values should be used if the iteration history indicates that the optimization process was close to a promising solution but would have needed more than the specified number of iterations. Smaller values should be specified when a faster but suboptimal solution may be sufficient.

**MAXROWS=i : $i \geq 1$**

specifies an upper bound for the number of independent variables selected for the model. More specific, this is an upper bound for the rows and columns of the $X'X$ matrix of the regression problem. The default is $\text{maxrows} = 3000$. Note, that the $X'X$ matrix used for the stepwise regression takes $n_{\text{rows}}(n_{\text{rows}} + 1)/2$ double precision values storage in RAM. For the default maximum size of $n_{\text{rows}} = 3000$ you will need more than $3000 \times 1500 \times 8$ bytes RAM, which is slightly more than 36 megabytes.

**MAXSTAGE=i : $i \geq 1$**

specifies an upper bound for the number of stages of estimation. If
MAXSTAGE is not specified, the default is MAXSTAGE=5. When a missing value is specified, the multistage estimation process is terminated

- if the sum-of-squares residual in the component selection process changes by less than 1%
- or when an upper range of 100 stages are processed.

That means, not specifying MAXSTAGE= or specifying a missing value are treated differently. Large values for MAXSTAGE= may result in numerical problems: the discretization error may be too large and the fit criterion does no longer improve and can actually become worse. In such a case the stagewise process is terminated with the last good stage.

**MAXSTPT=i : i ≥ 1**

specifies the number of values of the objective function inspected for the start of the optimization process. Larger values than the default value may improve the result of the optimization especially when more than three components are used. The default is MAXSTPT=250.

**MAXVECT=i : i ≥ 2**

specifies an upper bound for the number of eigenvectors made available for selection. The default is MAXVECT=400. Smaller values should be used only if there are memory problems for storing the eigenvectors when too many variables are included in the analysis. The specified value for MAXVECT= cannot be smaller than that for MINCOMP=. If the specified value of MAXVECT= is larger than the value for MAXROWS= it is reduced to the value of MAXROWS=.

**MEMSIZ=i : i ≥ 1**

For interval targets and in a multiple stage process some memory consuming operations are being performed. For very large data sets the computations may significantly depend on the size of the available RAM memory for those computations. By default MEMSIZ=8 specifies the availability of 8 mb of RAM for such operations. Since other operations need additional memory not more than 25 percent of the total amount of memory should be specified here. If you are running out of memory during the DMNEURL run, you may actually specify a smaller amount than the default 8 mb.

**MINCOMP=i : 2 ≤ i ≤ 8**

specifies a lower bound for the number of components selected for predicting the target in each stage. The default is MINCOMP=2. The specified value for MINCOMP= cannot be larger than that for MAXCOMP=. The MINCOMP= specification may permit the selection of components which otherwise would be rejected by the STOPR2= option. PROC DMNEURL may override the specified value when the rank of the $X'X$ matrix is less than the specified value.

**NOMONITOR**

supresses the output of the status monitor indicating the progress made in the computations.
NOPRINT:
supresses all output printed in the output window.

NPOINT=i : $5 \leq i \leq 19$
number of discretization points (should be even inbetween 5 and 19). By de-
fault NPOINT= is selected depending on the number of components selected
in the model using the MINCOMP= and MAXCOMP= options.

OPTCRIT=SSE|ACC|WSSE:
specifies the criterion for the optimization:

OPTCRIT=SSE the sum-of-squares error is minimized.

OPTCRIT=ACC a measure of the accuracy rate is maximized. (For interval
target the Goodman-Kruskal $\gamma$ is applied on a frequency table defined by
deciles of the actual target value.)

OPTCRIT=WSSE a weighted sum-of-squares criterion is minimized.
When this option is specified the weight must be specified using the
CRITWGT= option. Currently this option is valid only for binary target.

PALL:

- If an OUTSTAT= data set is specified, i.e. only principal components are
being computed, the following table illustrates the output options:

<table>
<thead>
<tr>
<th>Output</th>
<th>PSHORT</th>
<th>default</th>
<th>PALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Stat</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Eigenvalues</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

If PMATRIX is specified, the $X'X$, the covariance, or the correlation
matrix is also printed (depending on COV and CORR option).

- If no OUTSTAT= data set is specified, i.e. a nonlinear model based on
activation and link functions is being optimized, the following table illus-
trates the output options:

<table>
<thead>
<tr>
<th>Output</th>
<th>NOPRINT</th>
<th>PSHORT</th>
<th>default</th>
<th>PALL</th>
</tr>
</thead>
</table>
| PMATRIX:

This option is valid only if an OUTSTAT= data set is specified, i.e. when
DMNEURL is used only for computing eigenvalues and eigenvectors of the
$X'X$, covariance, or correlation matrix. If PMATRIX is specified, this matrix
is being printed. Since this matrix may be very large its printout is not included
by that of the PALL option.

POPTHIS:
print the detailed histories of all optimization processes. The PALL option
includes only the summarized forms of the history output (header and result).

PSHORT:
see the PALL option for the amount of output being printed.

PTABLE:
specifies the output of accuracy tables. This option is invoked automatically if
the PALL option is specified.
SELCRIT=SSE|ACC|WSSE:
specifies the criterion for selecting the best result among all of the activation functions:

SELCRIT=SSE select solution with smallest sum-of-squares error.
SELCRIT=ACC select solution with largest accuracy rate. (For interval target the Goodman-Kruskal \( \gamma \) is applied on a frequency table defined by deciles of the actual target value.)
SELCRIT=WSSE select solution with smallest weighted sum-of-squares error. This option is valid only for binary target. When this option is specified the weight must be specified using the CRITWGT= option.

SINGULAR=r:
specifies a criterion for the singularity test. The default is \( r = 1.e-8 \) and should not be changed if there are no significant reasons to do so.

STOPR2=r:
specifies a lower value for the incremental model \( R^2 \) value at which the variable selection process is stopped. The STOPR2= criterion is used only for the R2 values of the components selected in the range specified by the MINCOMP= and MAXCOMP= values. The default is \( r = 5e-5 \).

TESTDMDB:
permits the use of a data set generated by PROC DMDB to be specified as a TESTDATA= input data set. If this option is not specified, the data set specified with TESTDATA= must be a normal SAS data set.

DECISION Statement
For the syntax of the DECISION statement see the document of PROC DECIDE.

FUNCTION and LINK Statement
An activation function \( f \) and a link function \( g \) may be specified for the mapping inbetween the component scores \( s_{ij} \) and the values \( y_i \) of the response variable (stage=0) (or the residuals in stage > 0),

\[
\hat{y}_i = g(f^{(k)}(s_{ij}, \theta_j)), \quad i = 1, \ldots, N, j = 1, \ldots, p
\]

for each activation function \( f^{(k)}, k = 1, \ldots, K \). The FUNCTION and LINK statement can be used to specify the functions \( f^{(k)} \) and \( g \):

FUNCTION statement One or more of the following activation functions \( f \) can be specified
If more than one function $f^{(k)}$ is specified, each of the specified functions is evaluated during the estimation process and the best result w.r.t. to the sum-of-squares residual or accuracy (see SELCRIT= option) is selected. By default all available activation functions are used.

**LINK statement** Currently only one of the following link functions can be used for the outer function $g$:

<table>
<thead>
<tr>
<th>Function</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDENT</td>
<td>$x$</td>
</tr>
<tr>
<td>LOGIST</td>
<td>$\exp(x)/(1. + \exp(x))$</td>
</tr>
<tr>
<td>RECIPR</td>
<td>$1/x$</td>
</tr>
</tbody>
</table>

By default, the LOGIST function is used for a binary target and the IDENT(ity) function is used for interval target. In a parallelized version of PROC DMNEURL, multiple functions $g$ could be feasible.

**TARGET Statement**

```plaintext
TARGET onevar ;
```

One variable name may be specified identifying the target (response) variable for the two regressions. Note, that one or more target variables may be specified already with the PROC DMDB run. If a target is specified in the PROC DMDB run, it must not be specified in the PROC DMNEURL call.

**VAR or VARIABLES Statement**

```plaintext
VAR varlist ;
VARIABLES varlist ;
```

All variables, numeric (interval) and categorical (CLASS) variables which may be used for independent variables are specified with the VAR statement.

**FREQ or FREQUENCY Statement**

```plaintext
FREQ onevar ;
FREQUENCY onevar ;
```

One numeric (interval scaled) variable may be specified as a FREQ variable. Note, that a rational value is truncated to the next integer. It is recommended to specify the FREQ variable already in the PROC DMDB run. Then the information is saved in the catalog and that variable is used automatically as a FREQ variable in PROC DMNEURL. This also ensures that the FREQ variable is being used automatically by all other PROCs in the EM project.
**WEIGHT or WEIGHTS Statement**

```plaintext
WEIGHT onevar;
WEIGHTS onevar;
```

One numeric (interval scaled) variable may be specified as a WEIGHT variable. It is recommended to specify the WEIGHT variable already in the PROC DMDB invocation. Then the information is saved in the catalog and that variable is used automatically as a FREQ variable in PROC DMNEURL.

**Scoring the Model Using the OUTEST= Data set**

The score value \( \hat{y}_i \) is computed for each observation \( i = 1, \ldots, N_{\text{obs}} \) with nonmissing value of the target (response) variable \( y \) of the input data set. All information needed for scoring an observation of the DMDB data set is contained in the output of the OUTEST= data set. First an observation from the input data set is mapped into a vector \( v \) of \( n \) new values in which

1. CLASS predictor variables with \( K \) categories are replaced by \( K + 1 \) or \( K \) dummy (binary) variables, depending on the fact whether the variable has missing values or not.

2. Missing values in interval predictor variables are replaced by the mean value of this variable in the DMDB data set. This mean value is taken from the catalog of the DMDB data set.

3. The values of a WEIGHT or FREQ variable are multiplied into the observation.

4. For an interval target variable \( y \) its value is transformed into the interval \([0,1]\) by the relationship

\[
\hat{y}_i^{\text{new}} = \frac{y_i - y_{\text{min}}}{y_{\text{max}} - y_{\text{min}}}
\]

5. All predictor variables are transformed into values with zero mean and unit standard deviation by

\[
x_{ij}^{\text{new}} = \frac{x_{ij} - \text{Mean}(x_j)}{\text{StDev}(x_j)}
\]

The values for \( \text{Mean}(x_j) \) and \( \text{StDev}(x_j) \) are listed in the OUTEST= data set.

This means, that in the presence of CLASS variables the n-vector \( v \) has more entries than the observation in the data set.

The scoring is additive across the stages. The following information is available for scoring each stage:

- \( c \) components (eigenvectors) \( z_t \) each of dimension \( n \)
- the best activation function \( f \) and a specified link function \( g \)
- the \( p = 2c + 1 \) optimal parameter estimates \( \theta_j \)
For each component $z_l$ we compute the component score $u_l$,

$$u_l = \sum_{j=1}^{n} z_{lj} v_j$$

similar to principal component analysis. With those values $u_l$ the model can be expressed as

$$\hat{y} = \sum_{i=1}^{n_{\text{stage}}} g(f(u, \theta))$$

where $f$ is the best activation function and $g$ is the specified link function. In other words, this means, that given the $u_l$ the value $w$ is computed from

$$w = \theta_0 + \sum_{l} f(u_l, a_l, b_l)$$

where $a_l$ and $b_l$ are two of the $p = 2 * c + 1$ optimal parameters $\theta$ and $f$ is defined as

<table>
<thead>
<tr>
<th>Activation Function</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQUARE</td>
<td>$w = (a + b * u) * u$</td>
</tr>
<tr>
<td>TANH</td>
<td>$w = a * \tanh(b * u)$</td>
</tr>
<tr>
<td>ARCTAN</td>
<td>$w = a * \arctan(b * u)$</td>
</tr>
<tr>
<td>LOGIST</td>
<td>$w = \exp(a * u) / (1. + \exp(b * u))$</td>
</tr>
<tr>
<td>GAUSS</td>
<td>$w = a * \exp(-(b * u)^2)$</td>
</tr>
<tr>
<td>SIN</td>
<td>$w = a * \sin(b * u)$</td>
</tr>
<tr>
<td>COS</td>
<td>$w = a * \cos(b * u)$</td>
</tr>
<tr>
<td>EXP</td>
<td>$w = a * \exp(b * u)$</td>
</tr>
</tbody>
</table>

For the first component $a_1 = \theta_1$ and $b_1 = \theta_2$, for the second component $a_2 = \theta_3$ and $b_2 = \theta_4$, and for the last component $a_c = \theta_{p-1}$ and $b_c = \theta_p$ are used.

The link function $g$ is applied on $w$ and yields to $h$

<table>
<thead>
<tr>
<th>Link Function</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDENT</td>
<td>$h = w$</td>
</tr>
<tr>
<td>LOGIST</td>
<td>$h = \exp(w) / (1. + \exp(w))$</td>
</tr>
<tr>
<td>RECIPR</td>
<td>$h = 1/w$</td>
</tr>
</tbody>
</table>

Across all stages the values of $h$ are added to the predicted value (posterior) $\hat{y}$. 
The DMREG Procedure

Overview

Procedure Syntax

PROC DMREG Statement
CLASS Statement
CODE Statement
DECISION Statement
FREQ Statement
MODEL Statement
NLOPTIONS Statement
REMOTE Statement
SCORE Statement

Details

Examples

Example 1: Linear and Quadratic Logistic Regression with an Ordinal Target (Rings Data)
Example 2: Performing a Stepwise OLS Regression (DMREG Baseball Data)
Example 3: Comparison of the DMREG and LOGISTIC Procedures when Using a Categorical Input Variable

References

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Overview

DMREG enables you to fit both linear and logistic regression models. Linear regression attempts to predict the value of a continuous target as a linear function of one or more independent inputs. Logistic regression attempts to predict the probability that a categorical (binary, ordinal, or nominal) target will acquire the event of interest as a function of one or more independent inputs. The procedure supports forward, backward, and stepwise selection methods. It also allows you to score data sets or generate SAS DATA step code to score a data set.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DMREG Procedure

Procedure Syntax

PROC DMREG < option(s)>;
   MODEL dependent=independent(s) </model-option(s)>;
   CLASS variable(s);
   CODE code-option(s);
   DECISION DECDATA=<libref.:SAS-data-set<DECVARS=decision-variable(s)><option(s)>;
   FREQ variable;
   NLOPTIONS nonlinear-option(s);
   REMOTE remote-option(s);
   SCORE scoring-option(s);

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DMREG Procedure

PROC DMREG Statement

Invokes the DMREG procedure.

PROC DMREG <option(s)>;

Required Arguments

DATA=<libref.> SAS-data-set

Identifies the training data set.

DMDBCAT=<libref.> SAS-catalog

Identifies the training data catalog.

Options

COVOUT

Specifies that the OUTEST= data set is to include the variance-covariance matrix of the parameter estimates.

DESCENDING

Specifies that the order of categorical target is to be reversed.

ESTITER=n

Specifies that the OUTEST= data set contains parameter estimates and fit statistics (for training, test, and validation data) for every $n$th iteration.

Default: 0. Only the parameter estimates of the final iteration are output.

INEST=<libref.> SAS-data-set

Identifies the data set that contains initial estimates.

MINIMAL

Specifies the use of minimal resources to fit a logistic regression model. Memory for the Hessian matrix is not needed. The optimization defaults to the conjugate gradient technique and standard errors of the regression parameters are not computed. Model selection is disabled when this option is specified. This option does not apply to the normal error regression models.

NAMELEN=n

Specifies the length of effect names in the printed output to be $n$ characters, where $n$ is a value between 20 and 200. The default length is 20 characters.

OUTEST=<libref.> SAS-data-set
Identifies the output data set containing estimates and fit statistics. See for more information.

**NOPRINT**

Suppresses all printed output.

**SIMPLE**

Prints simple descriptive statistics of the input variables.

**TESTDATA=**) SAS-data-set**

Identifies the data set containing test data.

**VALIDATA=**) SAS-data-set**

Identifies the data set containing validation data.

*Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.*
CLASS Statement

Specifies one or more categorical variables to be used in the analysis.

CLASS variable(s);

Required Argument

variable(s)

Specifies a list of categorical variables to be used in the analysis. You must specify the target variable if it has a categorical (binary, ordinal, or nominal) measurement level.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DMREG Procedure

CODE Statement

Controls the creation of SAS code that can be used to score data sets.

**Tip:** If neither FILE= nor METABASE= is specified, then the SAS code is written to the SAS log.

```sas
CODE <code-option(s)>;
```

**CODE Options**

**ERROR**

Specifies that the error function is to be computed.

**FILE=**

Specifies the path for writing the code to an external file. For example,

FILE="c:\mydir\scorecode.sas".

**FORMAT=**

Specifies numeric formats for printing the estimated parameters.

**GROUP=**

Specifies the group identifier (up to four characters) for group processing.

**METABASE=mylib.mycat.myentry**

Specifies the code catalog entry to which the results are written.

**RESIDUAL**

Specifies that residuals are to be computed.
**DECISION Statement**

Specifies information used for decision processing in the DECIDE, DMREG, NEURAL, and SPLIT procedures. *This documentation applies to all four procedures.*

**Tip:** The DECISION statement is required for the DMREG and NEURAL procedures. It is optional for PROC SPLIT.

```
DECISION DECDATA=<libref.> SAS-data-set <DECVARS=decision-variable(s)> <option(s)>
```

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

Specifies the input data set that contains the decision matrix. The DECDATA= data set must contain the target variable.

**Note:** The DECDATA= data set may also contain decision variables specified by means of the DECVARS= option, and prior probability variable(s) specified by means of the PRIORVAR= option.

The target variable is specified by means of the TARGET statement in the DECIDE, NEURAL, and SPLIT procedures or by using the MODEL statement in the DMREG procedure. If the target variable in the DATA= data set is categorical then the target variable of the DECDATA= data set should contain the category values, and the decision variables will contain the common consequences of making those decisions for the corresponding target level. If the target variable is interval, then each decision variable will contain the value of the consequence for that decision at a point specified in the target variable. The unspecified regions of the decision function are interpolated by a piecewise linear spline.

**Tip:** The DECDATA= data set may be of TYPE=LOSS, PROFIT, or REVENUE. If unspecified, TYPE=PROFIT is assumed by default. TYPE= is a data set option that should be specified when the data set is created.

**DECVARS=decision-variable(s)**

Specifies the decision variables in the DECDATA= data set that contain the target-specific consequences for each decision.

**Default:** None

**COST=cost-option(s)**

Specifies numeric constants that gives the cost of a decision, or variables in the DATA= data set that contain the case-specific costs, or any combination of constants and variables. There must be the same number of cost constants and variables as there are decision variables in the DECVARS= option.
option. In the COST= option, you may not use abbreviated variable lists such as D1-D3, ABC--XYZ, or PQR:

| Default          | All costs are assumed to be 0. |

**CAUTION:**

The COST= option may only be specified when the DECDATA= data set is of TYPE=REVENUE.

**PRIORVAR=variable**

Specifies the variable in the DECDATA= data set that contains the prior probabilities to use for making decisions.

| Default          | None |
The DMREG Procedure

FREQ Statement

Specifies the variable that contains frequencies for training data.

FREQ <variable> ;

variable

Specifies the frequency variable. If specified, the FREQ variable overrides whatever is in the DMDB metadata. If the FREQ statement contains no name, then a FREQ variable is not used.

CAUTION:

If there is a frequency variable in the DMDB, it is not advisable to use another variable as a frequency variable because the training data does not contain observations with invalid values in the FREQ variable specified in the DMDB. For example, if the frequency variable specified in the DMDB contains a 0 or negative value, then that observation is discarded even if the FREQ variable that you specified in the FREQ statement of the DMREG procedure contains valid frequency values.

Default:

If the FREQ statement is not specified, the frequency variable in the DMDB is used. If the FREQ statement is specified without a variable, a frequency of 1 is used for all observations.

Range:

The frequency variable can contain integer or non-integer values.
The DMREG Procedure

MODEL Statement

Specifies modeling options.

Requirements: Model statement is required.

```
MODEL dependent=independent(s) / model-option(s);
```

Required Argument

dependent=independent(s)

where the arguments are defined as follows:

dependent

Specifies the response variable (target).

independents

Specifies the explanatory variables or effects (inputs). The syntax of effects is described in .

Options

model-options(s)

Specifies options that affect the fit, confidence intervals, variable selection, and specification of the model as follows:

MODEL Options - Fitting Options

MISCCONV=n

Specifies the critical misclassification rate at which to stop iterations.

Default: \( n = 0 \)

Range: \( 0 - 1 \)

STARTMISC=n

Specifies the number of iterations to be processed before checking misclassification rate.
<table>
<thead>
<tr>
<th>Default:</th>
<th>Depends on the optimization technique:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$n = 3$</td>
</tr>
<tr>
<td></td>
<td>TECHNIQUE = NEWRAP, NRRIDG, TRUREG</td>
</tr>
<tr>
<td></td>
<td>$n = 5$</td>
</tr>
<tr>
<td></td>
<td>TECHNIQUE = QUANEW, DBLDOG</td>
</tr>
<tr>
<td></td>
<td>$n = 10$</td>
</tr>
<tr>
<td></td>
<td>TECHNIQUE = CONGRA</td>
</tr>
<tr>
<td>Alias:</td>
<td>STATMISC</td>
</tr>
</tbody>
</table>

**MODEL Options - Miscellaneous Options**

**ALPHA=$n$**

Specifies the significance level of confidence intervals for regression parameters.

**Default:** 0.05

**CLPARM**

Specifies the computation of confidence intervals for parameters.

**CORRB**

Specifies that the correlation matrix is to be printed.

**COVB**

Specifies that the covariance matrix is to be printed.

**MODEL Options - Selection Options**

**CHOOSE=AIC | NONE | SBC | TDECDATA | VDECDATA | VERROR | VMISC | XDECDATA | XERROR | XMISC**

Specifies the criterion for the selection of the model.

**AIC**

Represents the Akaike Information Criterion. The model with the smallest criterion value is chosen.

**NONE**

Chooses standard variable selection based on the entry and/or stay $P$-values.

**SBC**

 Represents the Schwarz Bayesian Criterion. The model with the smallest criterion value is chosen.

**TDECDATA**

Represents the total profit/loss for the training data. The model with the largest profit or the smallest loss is chosen.
VDECDATA
Represents the total profit/loss for the VALIDATA= data set. The model with the largest profit or the smallest loss is chosen.

VERROR
Represents the error rate for the VALIDATA= data set. The error is the sum of square errors for least-square regression and negative log-likelihood for logistic regression. The model with the smallest error rate is chosen.

VMISC
Represents the misclassification rate for the VALIDATA= data set. The model with the smallest misclassification rate is chosen.

XDECDATA
Represents the total profit/loss for cross-validation of the training data. The model with the largest profit or the smallest loss is chosen.

XERROR
Represents the error rate for cross validation. The error is the sum of square errors for least-square regression and negative log-likelihood for logistic regression. The model with the smallest error rate is chosen.

XMISC
Represents the misclassification rate for cross validation. The model with the smallest misclassification rate is chosen.

Default: If decision processing is specified, the default is CHOOSE=TDECDATA; if the VALIDATA= data set is also specified, the default is CHOOSE=VDECDATA.

DETAILS
Prints details at each model selection step.

HIERARCHY=ALL | CLASS
Specifies how containment is to be applied.

ALL
Specifies that all independent variables that meet hierarchical requirements are included in the model.

CLASS
Specifies that only CLASS variables that meet hierarchical requirements are included in the model.

Default: ALL

INCLUDE=n
Specifies that the first \( n \) effects in the model are to be included in each model.

Default: 0
MAXSTEP=n
Specifies the maximum number of steps for the STEPWISE variable selection method.

| Default:       | Two times the number of effects specified in the MODEL statement. |

NODESIGNPRINT
Suppresses the display of the coding of the CLASS inputs.

| ALIAS:         | NODP |

RULE=MULTIPLE | SINGLE | NONE
Specifies the rule for inclusion of effects for SELECTION=FORWARD, BACKWARD, or STEPWISE.
MULTIPLE
One or more effects can be considered for entry or removal at the same time provided the hierarchical rule is observed. For example, if main effects A and B and interactions A*B are not in the model, effects that can be considered for entry in a single step are A alone, or B alone, or A, B, and A*B together.

SINGLE
A single effect is considered for entry into the model only if its lower order effects are already in the model; a single effect is considered for removal from the model only if its higher order effects are not in the model.

NONE
Effects are included or excluded one at a time without preservation of any hierarchical order.

| Default:       | RULE=NONE |


SELECTION= FORWARD | BACKWARD | STEPWISE | NONE
Specifies the variable selection methods.
FORWARD
 Begins with no inputs in the model and then, systematically, adds inputs that are related to the target.
BACKWARD
 Begins with all inputs in the model and then, systematically, removes inputs that are not related to the target.
STEPWISE
Systematically adds and deletes inputs from the model. Stepwise selection is similar to forward selection except that stepwise may remove an input after it has entered the model.
and replace it with another input.

NONE

All inputs are used to fit the model.

<table>
<thead>
<tr>
<th>Default:</th>
<th>NONE</th>
</tr>
</thead>
</table>

**SEQUENTIAL**

Specifies the addition or deletion of variables in sequential order, as specified in the MODEL statement.

**SLEnTRY=n**

Specifies the significance level for addition of variables.

<table>
<thead>
<tr>
<th>Default:</th>
<th>.05</th>
</tr>
</thead>
</table>

**SLSTAY=n**

Specifies the significance level for removal of variables.

<table>
<thead>
<tr>
<th>Default:</th>
<th>.05</th>
</tr>
</thead>
</table>

**START=n**

Specifies that the first *n* effects be included in the starting model.

| Default: | 0 - for the FORWARD or the STEPWISE method
|          | s (the total number of effects in the MODEL statement) - for the BACKWARD method |
| Range:   | The value of *n* ranges from 0 to *s*, where *s* is the total number of effects in the MODEL statement. |

**STOP=n**

Specifies the maximum (FORWARD method) or minimum (BACKWARD method) number of effects to be included in the final model. The variable selection process is stopped when *n* effects are added or deleted. The STOP= option has no effect when SELECTION=NONE or STEPWISE.

| Range:   | The value of *n* ranges from 0 to *s*, where *s* is the total number of effects in the MODEL statement. |
| Default: | *s* - for the FORWARD method
|          | 0 - for the BACKWARD method |
CODING= DEVIATION | GLM
  Specifies design variable coding for CLASS inputs.
  DEVIATION
    Deviation from mean coding, which is also known as effect coding.
  GLM
    Non-full rank GLM coding as used in the GLM procedure.
  Default: CODING=DEVIATION

LEVEL=INTERVAL | NOMINAL | ORDINAL
  Specifies the measurement level of the target variable.
  INTERVAL
    Interval variable.
  NOMINAL
    Nominal variable.
  ORDINAL
    Ordinal variable.
  Default: ORDINAL for a categorical target; INTERVAL for a numerical target.

ERROR=MBERNOULLI | NORMAL
  Specifies the error distribution.
  MBERNOULLI
    Multinomial distribution with on trial. This includes the binomial distribution with on trial. MBERNOULLI is not available if the target measurement level is interval.
    Alias: BINOMAIL or MULTINOMIAL
  NORMAL
    Normal distribution. NORMAL is not allowed if the target measurement level is nominal.
  Default: ERROR=NORMAL (for LEVEL=INTERVAL), ERROR=MBERNOULLI (otherwise).

LINK= CLOGLOG | IDENTITY | LOGIT | PROBIT
  Specifies the link function that represents the expected values of the target to the linear predictors.
  CLOGLOG
    Specifies the complementary log-log function, which is the inverse of the extreme value distribution function. The CLOGLOG function is available for ordinal or binary targets.
  IDENTITY
    Specifies the identity function. The IDENTITY function can only be used for the linear
regression analysis (ERROR=NORMAL).

**LOGIT**

Specifies the logit function, which is the inverse of the logistic distribution function. The LOGIT function is available for nominal, ordinal, or binary targets.

**PROBIT**

Specifies the probit function, which is the inverse of the standard normal distribution function. The PROBIT function is available for ordinal or binary targets.

<table>
<thead>
<tr>
<th>Default:</th>
<th>LOGIT (for ERROR=MBERNOULLI), IDENTITY (for ERROR=NORMAL).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IDENTITY (for ERROR=NORMAL)</td>
</tr>
</tbody>
</table>

**Tip:**

The CLOGLOG, LOGIT, and PROBIT link functions are used for a logistic regression analysis. The IDENTITY link function is used for a linear regression analysis.

**NOINT**

Suppresses the intercept for the binary target model or the normal error linear regression model.

**SINGULAR= n**

Specifies the tolerance for testing singularity.

| Default:          | \( 1e^{-6} \)                                             |
The DMREG Procedure

NLOPTIONS Statement

Specifies options for nonlinear optimizations. These options only apply to logistic regression models.

\[ \text{NLOPTIONS nonlinear-option(s);} \]

Nonlinear-Options

ABSCONV= number

Specifies an absolute function convergence criterion. ABSCONV= is a function of the log-likelihood for the intercept-only model. The optimization is to maximize the log-likelihood.

<table>
<thead>
<tr>
<th>Default</th>
<th>The default value is 1e-3 times the log-likelihood of the null model (intercept-only model).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>number &gt; 0</td>
</tr>
</tbody>
</table>

ABSFCONV= number

Specifies an absolute function convergence criterion.

<table>
<thead>
<tr>
<th>Default</th>
<th>10^{-3} times the log-likelihood of the intercept-only model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>number &gt; 0</td>
</tr>
</tbody>
</table>

ABSGCONV= number

Specifies the absolute gradient convergence criterion.

<table>
<thead>
<tr>
<th>Default</th>
<th>1E-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>number &gt; 0</td>
</tr>
</tbody>
</table>

ABSXCONV= number

Specifies the absolute parameter convergence criterion.

<table>
<thead>
<tr>
<th>Default</th>
<th>1E-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>number &gt; 0</td>
</tr>
</tbody>
</table>

DAMPSTEP= number

Specifies that the initial step size value for each line search used by the QUANEW, CONGRA, or NEWRAP techniques cannot be larger than the product of number and the step size value used in the previous iteration.
**DIAHES**

Forces the optimization algorithm (TRUREG, NEWRAP, or NRRIDG) to take advantage of the diagonality.

**FCONV= number**

Specifies a function convergence criterion.

<table>
<thead>
<tr>
<th>Default:</th>
<th>$10^{-F\text{DIGITS}}$, where FDIGITS is the value of the FDIGITS= option.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range:</td>
<td>$number &gt; 0$</td>
</tr>
</tbody>
</table>

**FCONV2= number**

Specifies another function convergence criterion.

<table>
<thead>
<tr>
<th>Default:</th>
<th>$10^{-4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range:</td>
<td>$number &gt; 0$</td>
</tr>
</tbody>
</table>

**FDIGITS= number**

Specifies the number of accurate digits in evaluations of the objective function.

<table>
<thead>
<tr>
<th>Default:</th>
<th>$-\log_{10}(\varepsilon)$, where (\varepsilon) is the machine precision.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range:</td>
<td>$number &gt; 0$</td>
</tr>
</tbody>
</table>

**FSIZE= number**

Specifies the parameter of the relative function and relative gradient termination criteria.

<table>
<thead>
<tr>
<th>Default:</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range:</td>
<td>$number \geq 0$</td>
</tr>
</tbody>
</table>

**GCONV= number**

Specifies the relative gradient convergence criterion.

<table>
<thead>
<tr>
<th>Default:</th>
<th>$10^{-6}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range:</td>
<td>$number &gt; 0$</td>
</tr>
</tbody>
</table>

**GCONV2= number**

Specifies another relative gradient convergence criterion.

<table>
<thead>
<tr>
<th>Default:</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>**HESCAL= 0</td>
<td>1</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Specifies the scaling version of the Hessian or cross-product Jacobian matrix used in NRRIDG, TRUREG, LEVMAR, NEWRAP, or DBLDOG optimization.</td>
<td></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>1 - for LEVMAR minimization technique</td>
</tr>
<tr>
<td></td>
<td>0 - for all others</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>INHESSIAN= number</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies how to define the initial estimate of the approximate Hessian for the quasi-Newton techniques QUANEW and DBLDOG.</td>
</tr>
<tr>
<td><strong>Range:</strong></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>INSTEP= number</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies a larger or smaller radius of the trust region used in the TRUREG, DBLDOG, and LEVMAR algorithms.</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Range:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>LINESEARCH= number</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the line-search method for the CONGRA, QUANEW, and NEWRAP optimization techniques.</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
</tr>
<tr>
<td><strong>Range:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>LSPRECISION= number</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the degree of accuracy that should be obtained by the second and third line-search algorithms.</td>
</tr>
</tbody>
</table>
### Table of Line-Search Precision Values

<table>
<thead>
<tr>
<th>TECHNIQUE=</th>
<th>UPDATE=</th>
<th>LSPRECISION VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUANEW</td>
<td>DBFGS, BFGS</td>
<td>0.4</td>
</tr>
<tr>
<td>QUANEW</td>
<td>DDFP, DFP</td>
<td>0.06</td>
</tr>
<tr>
<td>CONGRA</td>
<td>all</td>
<td>0.1</td>
</tr>
<tr>
<td>NEWRAP</td>
<td>no update</td>
<td>0.9</td>
</tr>
</tbody>
</table>

**Range:** \( \textit{number} > 0 \)

### MAXFUNC= number

Specifies the maximum number of function calls in the optimization process. The objective function that is minimized is the negative log-likelihood.

**Default:**
- 125 for TRUREG, NRRIDG, and NEWRAP.
- 500 for QUANEW and DBLDOG.
- 1000 for CONGRA.

**Range:** \( \textit{number} > 0 \)

### MAXITER= number

Specifies the maximum number of iterations in the optimization process.

**Default:**
- 50 for TRUREG, NRRIDG and NEWRAP
- 200 for QUANEW and DBLDOG
- 400 for CONGRA

**Range:** \( \textit{number} > 0 \)

### MAXSTEP= number

Specifies the upper bound for the step length of the line-search algorithms.

**Default:** The largest double precision value

**Range:** \( \textit{number} > 0 \)
MAXTIME= *number*

Specifies the upper limit of CPU time for the optimization process. It is measured in seconds.

| Default: | 7 days, that is, MAXTIME=604800 seconds |
| Range:   | *number* > 0 |

NOPRINT

Suppresses all output printed and only ERRORs, WARNINGs, and NOTEs are printed on the log file.

PALL

Prints all optional output except the output generated by the PSTDERR, LIST, or LISTCODE options.

PHISTORY

Prints the optimization history. If PSUMMARY or NOPRINT are not specified, then the PHISTORY option is set automatically. The iteration history is printed by default.

PSUMMARY

Restricts the amount of default printed output to a short form of iteration history and NOTEs, WARNINGs, and ERRORs.

RESTART= *number*

Specifies that the QUANEW or CONGRA algorithm is restarted with a steepest descent/ascent search direction after the *number* of iterations has been completed.

| Default: | For TECHNIQUE=CONGRA, and UPDATE=PB, restart is done automatically, so *number* is not used; |
|          | For TECHNIQUE=CONGRA, and UPDATE not=PB, *number* is the number of parameters. |
|          | For TECHNIQUE=QUANEW, *number* is the largest integer available. |
| Range:   | *number* > 1 |

SINGULAR= *number*

Specifies an absolute singularity criterion for the computation of the inertia of Hessian and cross-product Jacobian and their projected forms.

| Default: | 1E-8 |
| Range:   | *number* > 0 |

TECHNIQUE= *method*

where *method* is one of the following:

NONE
Specifies no method; no optimization is performed.

**TRUREG**

Specifies the Trust-Region optimization technique.

**NEWRAP**

Specifies the Newton-Raphson with Line Search optimization technique.

**NRRIDG**

Specifies the Newton-Raphson with Ridging optimization technique. This is the default when the number of parameters to be estimated is $n \leq 40$.

**DBLDOG**

Specifies the Double-Dogleg optimization technique.

**QUANEW**

Specifies the quasi-Newton optimization technique. This is the default when the number of convergence parameters to be estimated is in the range: $40 < n \leq 400$.

**CONGRA**

Specifies the Conjugate Gradient optimization technique. This is the default when the number of convergence parameters to be estimated is $n \geq 400$.

**Default:** The default technique is either NRRIDG, QUANEW, or CONGRA, depending on the value of the number of convergence parameters to be estimated.

See for more information.

**UPDATE=** *update-type*

where *update-type* is one of the following:

**BFGS**

For TECHNIQUE=QUANEW, performs the BFGS (Broyden-Fletcher-Goldfarb-Shanno) update of the Cholesky factor of the Hessian matrix.

**CD**

For TECHNIQUE=CONGRA, performs a conjugate descent update of Fletcher.

**DBFGS**

For TECHNIQUE=DBLDOG or QUANEW, performs the dual BFGS (Broyden-Fletcher-Goldfarb-Shanno) update of the Cholesky factor of the Hessian matrix. This is the default for TECHNIQUE=QUANEW and DBLDOG.

**DDFP**

For TECHNIQUE=DBLDOG or QUANEW, performs the dual DFP (Davidson-Fletcher-Powell) update of the Cholesky factor of the Hessian matrix.

**DFP**

For TECHNIQUE=QUANEW, performs the original DFP (Davidson-Fletcher-Powell) update of the inverse Hessian matrix.
For TECHNIQUE=CONGRA, performs the Fletcher-Reeves update.

For TECHNIQUE=CONGRA, performs the automatic restart update method of Powell and Beale. This is the default for TECHNIQUE=CONGRA.

For TECHNIQUE=CONGRA, performs the Polak-Ribiere update.

**VERSION= 1 | 2 | 3**

Specifies the version of the hybrid quasi-Newton optimization technique or the version of the quasi-Newton optimization technique with nonlinear constraints.

| Default: | 2 |

**XCONV= number**

Specifies the relative parameter convergence criterion.

| Default: | 1E-8 |
| Range: | number > 0 |

**XSIZE= number**

Specifies the number of successive iterations for which the criterion must be satisfied before the optimization process can be terminated.

| Default: | 0 |
| Range: | number ≥ 0 |
REMOTE Statement

Overview

The REMOTE statement is implemented in the NEURAL, DMREG, and DMVQ procedures in Enterprise Miner 4.1. You can use it to communicate with an MFC monitor (an external process on a Window client) to observe the progress of the iterative algorithm or to interrupt the iterative process. The monitor has a Graph tab and a Status tab as shown below:

![SAS/EM Monitor](image-url)
The Graph tab displays the iteration history: objection function versus iteration number and maximum absolute gradient versus iteration number. Click [Stop Current] or [Stop All] to stop the current or all optimization process. The Status tab displays the objective function and the maximum absolute element of the gradient vectors for each iteration.

**REMOTE remote-option(s);**

### Options

remote-options can be the following:

**SOCKET=socket-reference**

establishes a TCP/IP socket connection to an MFC monitor on the Window client to receive the report of the ongoing optimization. The socket reference contains the IP address and the port number and can be defined by using the following FILENAME statement:

FILENAME <socket-reference> SOCKET '<ip_address:portnum>';

where ip_address is the IP address of the Window client and portnum is the socket port number. The socket port number os any number that you use to invoke the MFC monitor.

**PLOTFILE=fileref | ' external-file'**

Specifies the external file that contains the iterative history (for example, the iteration number, the objective function, and the maximum absolute gradient). You can specify the path of the external file in quotes or you can use the FILENAME statement to specify a file reference. This option is obsolete if you can take advantage of the SOCKET= option.

**STOPFILE**

Specifies an external file that the iterative process will be terminated if this file exists. This is useful when you run a project with a large data set. To stop the process, you must create the external file. The DMREG procedure stops the iterative process when it detects this file. The file does not have to have any content. You can specify the path of an external file in quotes or use the LIBNAME statement to specify the file reference. This option is obsolete if you can take advantage of the SOCKET= option.

Example:

FILENAME abc SOCKET 'd6026.us.sas.com:12234';
PROC DMREG DATA=SAMPSIO.DMDCENS DMDBCAT=SAMPSIO.DMDCENS;
    REMOTE SOCKET=abs;
    CLASS CLASS WORKCLAS MARTAL OCCUPATN RELATION RACE SEX COUNTRY;
    MODEL CLASS=AGE FNLWGT EDUC_NUM CAP_GAIN CAP_LOSS HOURWEEK
        WORKCLAS MARITAL OCCUPATN RELATION RACE SEX COUNTRY
        / SELECTION=F CHOOSE=AIC;
RUN;

You can invoke the monitor any time by using the port number (1234) that you choose. After the socket connect is made you can see the display of the iteration history of the ongoing optimization.
SCORE Statement

Specifies options for scoring data.

SCORE scoring-option(s);

Options

Scoring-options can be the following:

**ADDITIONALRESIDUALS**

Specifies that the OUT= data set contains additional residuals such as: RS_ for logistic regressions and RS_, RT_, RD_, RDS_, RDT_ for normal error regression. See for more detail.

Alias: ADDRES | AR

**ALPHA=number**

Specifies the significance level \( p \) for the construction of 100(1-p)% confidence interval for the posterior probabilities. This number must be between 0 and 1.

Default: 0.05

**CLP**

Specifies that the OUT= data set contains the confidence limits for the posterior probabilities. The significance level is controlled by the ALPHA= option.

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

Specifies the input data set that contains inputs and optionally targets.

Default: The default is the same as the DATA= data set in the PROC statement.

**DMDB | NODMDB**

Specifies whether an explicit DATA= data set has been DMDB-encoded or if the data set contains raw data.

Default: If the DATA= option is not specified in the SCORE statement, the training data is used and the NODMDB option is invalid.

Caution: If the DATA= in the SCORE statement specifies a data set other than the training data, either DMDB or NODMDB must be specified in the SCORE statement.

**OUT=<libref.> SAS-data-set**
Specifies the output data set with outputs.

**Default:** \( \text{DATA}n \)

Names for computed variables are normally taken from the data dictionary. If necessary, names for these variables can be generated by concatenating a prefix to the name of the corresponding target variable according to the rules in the following tables:

**Statistics Generated in the**

**OUT=SAS-data-set for Normal ErrorRegression**

<table>
<thead>
<tr>
<th>NAME</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_targetname</td>
<td>Predicted: targetname</td>
</tr>
<tr>
<td>E_targetname</td>
<td>Error Function: targetname</td>
</tr>
<tr>
<td>R_targetname</td>
<td>Residuals: targetname</td>
</tr>
<tr>
<td>RD_targetname</td>
<td>Deviance Residuals: targetname</td>
</tr>
</tbody>
</table>

If the target is declared as a categorical variable, the OUT=SAS-data-set also includes:

<table>
<thead>
<tr>
<th>NAME</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_targetname</td>
<td>From: targetname</td>
</tr>
<tr>
<td>I_targetname</td>
<td>Into: targetname</td>
</tr>
</tbody>
</table>

If the ADDITIONALRESIDUALS option is also specified, the OUT=SAS-data-set includes:
RS_targetname  Standardized Residuals:

RT_targetname  Studentized Residuals:

RDS_targetname  Standardized Deviance Residuals:

RDT_targetname  Studentized Deviance Residuals:

**Note:** In the table above, targetname is the name of the target variable. For example, if PURCHASE is the targetname, the predicted value statistic is named P_PURCHA and the studentized deviance residual is named RDT_PURC. [If the constructed names are longer than the maximum of eight characters allowed for SAS variable names, they are truncated to eight characters.]

**Statistics Generated in the OUT= SAS-data-set for Binomial or Multinomial Regression**

<table>
<thead>
<tr>
<th>NAME</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_targetname&amp;value</td>
<td>Predicted: targetname=targetvalue</td>
</tr>
<tr>
<td>F_targetname</td>
<td>From: targetname</td>
</tr>
<tr>
<td>I_targetname</td>
<td>Into: targetname</td>
</tr>
<tr>
<td>E_targetname</td>
<td>Error Function:</td>
</tr>
<tr>
<td>R_targetname&amp;value</td>
<td>Residual: targetname=targetvalue</td>
</tr>
</tbody>
</table>
If the ADDITIONALRESIDUALS option is specified, the OUT=SAS-data-set includes:

```
RS_targetname&value  Standardized Residual:
tagename=targetvalue
```

**Note:** In the table above, targetname&value is a combination of the target name (targetname) and target value (targetvalue). For example, if PURCHASE is the targetname and "YES" and "NO" are the two values possible for targetvalue, the predicted value statistics are named P_PURYES and P_PURN0.

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

Specifies the output data set with fit statistics. For more information, see .

**OUTSTEP**

Scores the data for each model selection step.

**ROLE=role-value**

Specifies the role of the DATA= data set. The ROLE= option primarily affects which fit statistics are computed and what their names and labels are.

**Role-value** can be:

- **TRAIN**
  
  This value is the default when the same data set name is used in the DATA= option in both the PROC and SCORE statements. Specifying TRAIN with any data set other than the actual training set is an error.

- **VALID | VALIDATION**
  
  This value is the default when the DATA= data set name in the SCORE statement is the same as the data set in the VALIDDATA= in the PROC statement.

- **TEST**
  
  This value is the default when the DATA= data set name in the SCORE statement is the same as the data set name in the TESTDATA= option of the PROC statement.

- **SCORE**
  
  Predicted values are produced but residuals, error functions, and other fit statistics are not produced.

---

*Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.*
Details

Input

The input to the DMREG procedure can be assigned one of these roles:

Training

The DATA= data set is used to fit the initial model.

Validation

The VALIDATA= data set is used to compute assessment statistics and to fine-tune the model during stepwise selection.

Test

The TESTDATA= data set is an additional "hold out" data set that you can use to compute assessment statistics.

Score

The DATA= data set in the SCORE statement is used for predicting target values for a new data set that may not contain the target.

Specification of Effects

Different types of effects can be used in the DMREG procedure. In the following list, assume that A, B, and C are class variables and that X1, X2, and Y are continuous variables:

1. Regressor effects are specified by writing continuous variables individually:
   X1 X2
2. Polynomial effects are specified by joining two or more continuous variables with asterisks:
   X1*X1 X1*X2
3. Main effects are specified by writing class variables individually:
   AC
4. Crossed effects (interactions) are specified by joining class variables with asterisks:
   A*BB*CA*B*C
5. Continuous-by-class effects are written by joining continuous variables and class variables with asterisks:
   X1*A.

Note: Nested effects are not supported.
Optimization Methods

The following table provides a list of the general nonlinear optimization methods and the default maximum number of iterations and function calls for each method.

**Optimization Methods for the Regression node.**

<table>
<thead>
<tr>
<th>Optimization Method</th>
<th>Maximum Iterations</th>
<th>Maximum Function Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conjugate Gradient</td>
<td>400</td>
<td>1000</td>
</tr>
<tr>
<td>Double Dogleg</td>
<td>200</td>
<td>500</td>
</tr>
<tr>
<td>Newton-Raphson with Line Search</td>
<td>50</td>
<td>125</td>
</tr>
<tr>
<td>Newton-Raphson with Ridging</td>
<td>50</td>
<td>125</td>
</tr>
<tr>
<td>Quasi-Newton</td>
<td>200</td>
<td>500</td>
</tr>
<tr>
<td>Trust-Region</td>
<td>50</td>
<td>125</td>
</tr>
</tbody>
</table>

You should set the optimization method based on the size of the data mining problem, as follows:

1. Small-to-medium problems - The Trust-Region, Newton-Raphson with Ridging, and Newton-Raphson with Line Search methods are appropriate for small and medium sized optimization problems (number of model parameters up to 40) where the Hessian matrix is easy and cheap to compute. Sometimes, Newton-Raphson with Ridging can be faster than Trust-Region, but Trust-Region is numerically more stable. If the Hessian matrix is not singular at the optimum, then the Newton-Raphson with Line Search can be a very competitive method.

2. Medium Problems - The quasi-Newton and Double Dogleg methods are appropriate for medium optimization problems (number of model parameters up to 400) where the objective function and the gradient are much faster to compute than the Hessian. Quasi-Newton and Double Dogleg require more iterations than does the Trust-Region or the Newton-Raphson methods, but each iteration is much faster.

3. Large Problems - The Conjugate Gradient method is appropriate for large data mining problems (number of model parameters greater than 400) where the objective function and the gradient are much faster to compute than the Hessian matrix, and where they need too much memory to store the approximate Hessian matrix.

**Note:** To learn about these optimization methods, see the SAS/OR Technical Report: The NLP Procedure (1997).

The underlying "Default" optimization entry method depends on the number of parameters in the model.
If the number of parameters is less than or equal to 40, then the default method is set to Newton-Raphson with Ridging. If the number of parameters is greater than 40 and less than 400, then the default method is set to quasi-Newton. If the number of parameters is greater than 400, then Conjugate Gradient is the default method.

**Fit Statistics for OUTEST and OUTFIT Data Sets**

The OUTEST= data set in the PROC DMREG statement contains fit statistics for the training, test, and/or validation data. Depending on the ROLE= option in the SCORE statement, the OUTFIT= data set contains fit statistics for either the training, test, or validation data.

*Fit Statistics for the Training Data*

<table>
<thead>
<tr>
<th>Fit Statistic</th>
<th>Training Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>AIC</em></td>
<td>Train: Akaike's Information Criterion</td>
</tr>
<tr>
<td><em>ASE</em></td>
<td>Train: Average Squared Error</td>
</tr>
<tr>
<td><em>AVERR</em></td>
<td>Train: Average Error Function</td>
</tr>
<tr>
<td><em>DFE</em></td>
<td>Train: Degrees of Freedom for Error</td>
</tr>
<tr>
<td><em>DFM</em></td>
<td>Train: Model Degrees of Freedom</td>
</tr>
<tr>
<td><em>DFT</em></td>
<td>Train: Total Degrees of Freedom</td>
</tr>
<tr>
<td><em>DIV</em></td>
<td>Train: Divisor for ASE</td>
</tr>
<tr>
<td><em>ERR</em></td>
<td>Train: Error Function</td>
</tr>
<tr>
<td><em>FPE</em></td>
<td>Train: Final Prediction Error</td>
</tr>
<tr>
<td>Fit Statistic</td>
<td>Test Data</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
</tr>
<tr>
<td><em>TASE</em></td>
<td>Test: Average Squared Error</td>
</tr>
<tr>
<td>Test Code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>TASEL</td>
<td>Test: Lower 95% Confidence Limit for TASE</td>
</tr>
<tr>
<td>TASEU</td>
<td>Test: Upper 95% Confidence Limit for TASE</td>
</tr>
<tr>
<td>TAVERR</td>
<td>Test: Average Error Function</td>
</tr>
<tr>
<td>TDIV</td>
<td>Test: Divisor for TASE</td>
</tr>
<tr>
<td>TERR</td>
<td>Test: Error Function</td>
</tr>
<tr>
<td>TMAX</td>
<td>Test: Maximum Absolute Error</td>
</tr>
<tr>
<td>TMSE</td>
<td>Test: Mean Square Error</td>
</tr>
<tr>
<td>TNOBS</td>
<td>Test: Sum of Frequencies</td>
</tr>
<tr>
<td>TRASE</td>
<td>Test: Root Average Squared Error</td>
</tr>
<tr>
<td>TRMSE</td>
<td>Test: Root Mean Square Error</td>
</tr>
<tr>
<td>TSSE</td>
<td>Test: Sum of Square Errors</td>
</tr>
<tr>
<td>TSUMW</td>
<td>Test: Sum of Case Weights Times Frequency</td>
</tr>
<tr>
<td>TMISC</td>
<td>Test: Misclassification Rate</td>
</tr>
</tbody>
</table>
**Test: Lower 95% Confidence Limit for TMISC**

**Test: Upper 95% Confidence Limit for TMISC**

---

**Fit Statistics for the Validation Data**

<table>
<thead>
<tr>
<th>Fit Statistic</th>
<th>Validation Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>VASE</em></td>
<td>Valid: Average Squared Error</td>
</tr>
<tr>
<td><em>VAVERR</em></td>
<td>Valid: Average Error Function</td>
</tr>
<tr>
<td><em>VDIV</em></td>
<td>Valid: Divisor for VASE</td>
</tr>
<tr>
<td><em>VERR</em></td>
<td>Valid: Error Function</td>
</tr>
<tr>
<td><em>VMAX</em></td>
<td>Valid: Maximum Absolute Error</td>
</tr>
<tr>
<td><em>VMSE</em></td>
<td>Valid: Mean Square Error</td>
</tr>
<tr>
<td><em>VNOBS</em></td>
<td>Valid: Sum of Frequencies</td>
</tr>
<tr>
<td><em>VRASE</em></td>
<td>Valid: Root Average Squared Error</td>
</tr>
<tr>
<td><em>VRMSE</em></td>
<td>Valid: Root Mean Square Error</td>
</tr>
<tr>
<td><em>VSSE</em></td>
<td>Valid: Sum of Square Errors</td>
</tr>
</tbody>
</table>
The DMREG Procedure

Examples

The following examples were executed using the HP-UX version 10.20 operating system and the SAS software release 6.12TS045.

Example 1: Linear and Quadratic Logistic Regression with an Ordinal Target (Rings Data)

Example 2: Performing a Stepwise OLS Regression (DMREG Baseball Data)

Example 3: Comparison of the DMREG and LOGISTIC Procedures when Using a Categorical Input Variable
Example 1: Linear and Quadratic Logistic Regression with an Ordinal Target (Rings Data)

This example demonstrates how to perform both a linear and a quadratic logistic regression with an ordinal target. The example DMDB training data set SAMPSIO.DMDRING contains an ordinal target with 3 levels (C= 0, 1, or 2) and two continuous inputs (X and Y). There are 180 observations in the data set. The SAMPSIO.DMSRING data set is scored using the scoring formula from the trained models. Both data sets are stored in the sample library.

Linear-Logistic Program

```plaintext
proc gplot data=sampsio.dmdring;
    plot y*x=c /haxis=axis1 vaxis=axis2;
    symbol c=black i=none v=dot;
    symbol2 c=red i=none v=square;
    symbol3 c=green i=none v=triangle;
    axis1 c=black width=2.5  order=(0 to 30 by 5);
    axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
    title 'Plot of the Rings Training Data';
run;

proc dmreg data=sampsio.dmdring dmdbcat=sampsio.dmdring;
    class c;
    model c = x y;
    score out=out outfit=fit;
    score data=sampsio.dmsring nodmdb out=gridout;
    title 'Linear-Logistic Regression with Ordinal Target';
run;

proc print data=fit noobs label;
    var _aic_ _max_ _rfpe_ _misc_;
    title2 'Fit Statistics for the Training Data Set';
run;

proc freq data=out;
    tables f_c*i_c;
    title2 'Misclassification Table: Training Data';
run;
```
proc gplot data=out;
    plot y*x=i_c / haxis=axis1 vaxis=axis2;
    symbol  c=black i=none v=dot;
    symbol2 c=red i=none v=square;
    symbol3 c=green i=none v=triangle;
    axis1 c=black width=2.5 order=(0 to 30 by 5);
    axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
    title2 'Classification Results';
run;

proc gcontour data=gridout;
    plot y*x=p_c1 / pattern ctext=black coutline=gray;
    plot y*x=p_c2 / pattern ctext=black coutline=gray;
    plot y*x=p_c3 / pattern ctext=black coutline=gray;
    title2 'Posterior Probabilities';
    pattern v=msolid;
    legend frame;
run;

Linear-Logistic Output

PROC GLOT Plot of the Rings Training Data

DMREG Summary Profile Information

PROC DMREG first lists background information about the fitting of the linear-logistic model. Included are the name of the input data set, the response variable, the number of response levels, the number of observations used, the error distribution, and the link function.
Linear-Logistic Regression with Ordinal Target

The DMREG Procedure

Training Data Set: SAMPS10.DMDRING
DMDB Catalog: SAMPS10.DMDRING
Target Variable: c
Target Measurement Level: Ordinal
Number of Target Categories: 3
Error: MBernoulli
Link Function: Logit
Number of Model Parameters: 4
Number of Observations: 180

DMREG Response Profile

The Response Profile table lists the target categories, their ordered values, and their total frequencies for the given data.

<table>
<thead>
<tr>
<th>Ordered Value</th>
<th>Total c Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>59</td>
</tr>
<tr>
<td>3</td>
<td>113</td>
</tr>
</tbody>
</table>

DMREG Optimization Table

The Optimization table provides a summary of the Newton-Raphson Ridge optimization results.
Newton-Raphson Ridge Optimization
Without Parameter Scaling
Parameter Estimates 4

Optimization Start
Active Constraints 0
Max Abs Gradient Element 2.5948717949

DMREG Model Fitting Information and Testing Global Null Hypothesis Beta=0

The Model Fitting Information and Testing Global Null Hypothesis Beta=0 table contains the negative of twice the log likelihood (-2 LOG L) for the fitted model. Results of the likelihood ratio test and the efficient score test for testing the joint significance of the explanatory inputs are also printed in the table.

DMREG Analysis of Maximum Likelihood Estimates

The Analysis of Maximum Likelihood Estimates table lists the parameter estimates, their standard errors, and the results of the Wald test for the individual parameters. A standardized estimate for each slope parameter and the odds ratio for each estimate is also printed. An odds ratio is obtained by exponentiating the corresponding parameter estimate.
DMREG Odds Ratio Estimates

The Odd Ratio Estimates table lists the odd ratios for the explanatory inputs. The odd ratio estimates provide the change in odds for a unit increase in each input.

### Linear-Logistic Regression with Ordinal Target

**The DMREG Procedure**

**Odds Ratio Estimates**

<table>
<thead>
<tr>
<th>Input</th>
<th>Odds Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0.984</td>
</tr>
<tr>
<td>y</td>
<td>1.023</td>
</tr>
</tbody>
</table>

### PROC PRINT Report of Selected Fit Statistics for the Training Data Set

The misclassification rate for the training data set is only 37.22%.

### Linear-Logistic Regression with Ordinal Target

**Fit Statistics for the Training Data Set**

<table>
<thead>
<tr>
<th>Train: Akaike's Information Criterion</th>
<th>Train: Maximum Absolute Error</th>
<th>Train: Final Prediction Error</th>
<th>Train: Root Mean Square Classification Error</th>
<th>Train: Misclassification Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>293.703</td>
<td>0.95656</td>
<td>0.41066</td>
<td>0.37222</td>
<td></td>
</tr>
</tbody>
</table>

### PROC FREQ Misclassification Table for the Training Data

All observations in the training data are classified into the C=3 level. The linear model is not adequate.
**PROC GPLOT Plot of the Classification Results**

The target classes are not linearly separable.

### Table of F_c by l_c

<table>
<thead>
<tr>
<th></th>
<th>F_c(From: c)</th>
<th>l_c(Into: c)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Frequency</td>
<td>Percent</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>4.44</td>
<td>8.8</td>
</tr>
<tr>
<td></td>
<td>100.00</td>
<td>4.44</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>59</td>
<td>32.78</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>100.00</td>
<td>32.78</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>113</td>
<td>62.78</td>
<td>113</td>
</tr>
<tr>
<td></td>
<td>100.00</td>
<td>62.78</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>180</td>
<td>100.00</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>100.00</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>
PROC GCONTOUR Plots of the Posterior Probabilities
Quadratic-Logistic Program
proc dmreg data=sampsio.dmdring dmdbcat=sampsio.dmdring;
   class c;
   model c=x|x|y|y @2;
   score out=qout outfit=qfit;
   score data=sampsio.dmsring nodmdb out=qgridout;
   title1 'Quadratic-Logistic Regression with Ordinal Target';
   run;

proc print data=qfit noobs label;
   var _aic_ _max_ _rfpe_ _misc_;
   title2 'Fit Statistics for the Training Data Set';
   run;

proc freq data=qout;
   tables f_c*i_c;
   title2 'Misclassification Table: Training Data';
   run;

proc gplot data=qout;
   plot y*x=i_c / haxis=axis1 vaxis=axis2;
   symbol c=black i=none v=dot;
   symbol2 c=red i=none v=square;
   symbol3 c=green i=none v=triangle;
   axis1 c=black width=2.5 order=(0 to 30 by 5);
   axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
   title2 'Classification Results';
   run;

proc gcontour data=qgridout;
   plot y*x=p_c1 / pattern ctext=black coutline=gray;
   plot y*x=p_c2 / pattern ctext=black coutline=gray;
   plot y*x=p_c3 / pattern ctext=black coutline=gray;
   title2 'Posterior Probabilities';
   pattern v=msolid;
   legend frame;
   run;

---

**Quadratic-Logistic Output**

**DMREG Output**
Quadratic-Logistic Regression with Ordinal Target

The DMREG Procedure

Training Data Set: SAMP10.DMDRING
DMDB Catalog: SAMP10.DMDRING
Target Variable: c
Target Measurement Level: Ordinal
Number of Target Categories: 3
Error: MBernoulli
Link Function: Logit
Number of Model Parameters: 7
Number of Observations: 180

Target Profile

<table>
<thead>
<tr>
<th>Ordered Value</th>
<th>Total c</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>59</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>113</td>
</tr>
</tbody>
</table>

Newton-Raphson Ridge Optimization
Without Parameter Scaling

Parameter Estimates 7

Optimization Start

Active Constraints 0
Max Abs Gradient Element 8.4022792023

<table>
<thead>
<tr>
<th>Iter</th>
<th>Restarts</th>
<th>Function Calls</th>
<th>Active Constraints</th>
<th>Objective Function</th>
<th>Objective Change</th>
<th>Max Abs Gradient Element</th>
<th>Ridge</th>
<th>Predicted Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>50.31687</td>
<td>33.0103</td>
<td>8.4596</td>
<td>0</td>
<td>1.095</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>26.02018</td>
<td>24.2957</td>
<td>2.9573</td>
<td>0</td>
<td>1.292</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>13.98860</td>
<td>12.0316</td>
<td>1.1136</td>
<td>0</td>
<td>1.306</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>7.43545</td>
<td>6.5532</td>
<td>0.5352</td>
<td>0</td>
<td>1.312</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>3.75004</td>
<td>3.6854</td>
<td>0.3031</td>
<td>0</td>
<td>1.310</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>1.75436</td>
<td>1.9957</td>
<td>0.1649</td>
<td>0</td>
<td>1.301</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0.75995</td>
<td>0.9944</td>
<td>0.0776</td>
<td>0</td>
<td>1.290</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0.30934</td>
<td>0.4506</td>
<td>0.0323</td>
<td>0</td>
<td>1.280</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0.12182</td>
<td>0.1875</td>
<td>0.0125</td>
<td>0</td>
<td>1.274</td>
</tr>
</tbody>
</table>

Optimization Results

Iterations 9
Hessian Calls 11
Objective Function 0.121823985
Max Abs Gradient Element 0.0125231148
Ridge 0

ABSCONV convergence criterion satisfied.

NOTE: At least one element of the (projected) gradient is greater than 1e-3.
PROC PRINT Report of Selected Fit Statistics for the Training Data

Note that the training misclassification rate is 0. All cases are correctly classified by the quadratic-logistic model.

Analysis of Maximum Likelihood Estimates

<table>
<thead>
<tr>
<th>Parameter</th>
<th>DF</th>
<th>Estimate</th>
<th>Standard Error</th>
<th>Wald Chi-square</th>
<th>Pr &gt; Chi-square</th>
<th>Standardized Estimate</th>
<th>exp(Est)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>1</td>
<td>-232.2</td>
<td>236.4</td>
<td>0.96</td>
<td>0.3260</td>
<td></td>
<td>0.000</td>
</tr>
<tr>
<td>Intercept</td>
<td>2</td>
<td>-197.4</td>
<td>188.0</td>
<td>1.10</td>
<td>0.2939</td>
<td></td>
<td>0.000</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>18.4897</td>
<td>16.3812</td>
<td>1.27</td>
<td>0.2590</td>
<td>70.701785</td>
<td>999.000</td>
</tr>
<tr>
<td>x*x</td>
<td>1</td>
<td>-0.6584</td>
<td>0.4552</td>
<td>2.09</td>
<td>0.1481</td>
<td></td>
<td>0.518</td>
</tr>
<tr>
<td>y</td>
<td>1</td>
<td>22.5413</td>
<td>31.5097</td>
<td>0.51</td>
<td>0.4744</td>
<td>51.803360</td>
<td>999.000</td>
</tr>
<tr>
<td>x*y</td>
<td>1</td>
<td>0.0469</td>
<td>0.7657</td>
<td>0.00</td>
<td>0.9512</td>
<td></td>
<td>1.048</td>
</tr>
<tr>
<td>y*y</td>
<td>1</td>
<td>-1.2320</td>
<td>1.3963</td>
<td>0.78</td>
<td>0.3776</td>
<td></td>
<td>0.292</td>
</tr>
</tbody>
</table>

PROC PRINT Report of Selected Fit Statistics for the Training Data

Note that the training misclassification rate is 0. All cases are correctly classified by the quadratic-logistic model.

Quadratic-Logistic Regression with Ordinal Target
Fits Statistics for the Training Data Set

<table>
<thead>
<tr>
<th>Train:</th>
<th>Akaike’s Information Criterion</th>
<th>Train: Final</th>
<th>Train: Root</th>
<th>Train: Absolute Prediction Error</th>
<th>Train: Misclassification Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.2436</td>
<td>0.023582</td>
<td>0.002403151</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PROC FREQ Misclassification Table for the Training Data
PROC GPLOT Plot of the Classification Results

**Quadratic-Logistic Regression with Ordinal Target**
**Misclassification Table: Training Data**

*The FREQ Procedure*

**Table of F_c by I_c**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_c</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>4.44</td>
<td>0.00</td>
<td>0.00</td>
<td>4.44</td>
</tr>
<tr>
<td></td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
<td>100.00</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>59</td>
<td>0</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>0.00</td>
<td>32.78</td>
<td>0.00</td>
<td>32.78</td>
</tr>
<tr>
<td></td>
<td>0.00</td>
<td>100.00</td>
<td>0.00</td>
<td>100.00</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>113</td>
<td>113</td>
</tr>
<tr>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>62.78</td>
<td>62.78</td>
</tr>
<tr>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Total</td>
<td>8</td>
<td>59</td>
<td>113</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>4.44</td>
<td>32.78</td>
<td>62.78</td>
<td>100.00</td>
</tr>
</tbody>
</table>

PROC GPLOT Plot of the Classification Results
PROC GCONTOUR Plots of the Posterior Probabilities

Quadratic—Logistic Regression with Ordinal Target

Posterior Probabilities

Predicted: c=1

Predicted: c=2

Graphs showing the posterior probabilities for quadratic-logistic regression with ordinal target.
PROC GLOT creates a scatter plot of the Rings training data.

proc gplot data=sampsio.dmdring;
  plot y*x=c /haxis=axis1 vaxis=axis2;
  symbol c=black i=none v=dot;
  symbol2 c=red i=none v=square;
  symbol3 c=green i=none v=triangle;
  axis1 c=black width=2.5  order=(0 to 30 by 5);
  axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
  title 'Plot of the Rings Training Data';
run;
The PROC DMREG statement invokes the procedure. The DATA= option identifies the DMDB encoded training data set that is used to fit the model. The DMDBCAT= option identifies the DMDB training data catalog. You can create DMDB encoded data sets and catalogs with the DMDB procedure.

proc dmreg data=sampsio.dmdring dmdbcat=sampsio.dmdring;
The CLASS statement identifies the target C as a categorical variable.

class c;
The MODEL statement specifies the linear-logistic model.

model c = x y;
The SCORE statement scores the training data set and outputs fit statistics to the OUTFIT= data set. A note is printed in the log that indicates the training data set is scored when the DATA= option is omitted.

```
score out=out outfit=fit;
```
The second SCORE statement scores the SAMPSIO.DMSRING data set. The NODMDB option specifies that the score data set contains raw values instead of DMDB encoded data.

```
score data=sampsio.dmsring nodmdb out=gridout;
   title 'Linear-Logistic Regression with Ordinal Target';
run;
```
PROC PRINT report of selected fit statistics for the training data.

proc print data=fit noobs label;
    var _aic_ _max_ _rfpe_ _misc_ ;
    title2 'Fit Statistics for the Training Data Set';
run;
PROC FREQ report of the misclassification rate for the training data set. The F_C variable is the actual target value for each case and the I_C variable is the target value into which the case is classified.

```
proc freq data=out;
    tables f_c*i_c;
    title2 'Misclassification Table: Training Data';
run;
```
PROC GPLOT produces a plot of the classification results for the training data.

```plaintext
proc gplot data=out;
  plot y*x=i_c / haxis=axis1 vaxis=axis2;
  symbol  c=black i=none v=dot;
  symbol2 c=red   i=none v=square;
  symbol3 c=green i=none v=triangle;
  axis1 c=black width=2.5 order=(0 to 30 by 5);
  axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
  title2 'Classification Results';
run;
```
PROC GCONTOUR produces plots of the posterior probabilities.

proc gcontour data=gridout;
  plot y*x=p_c1 / pattern ctext=black coutline=gray;
  plot y*x=p_c2 / pattern ctext=black coutline=gray;
  plot y*x=p_c3 / pattern ctext=black coutline=gray;
  title2 'Posterior Probabilities';
  pattern v=msolid;
  legend frame;
run;
The model statement specifies the quadratic-logistic model. The vertical bars indicate that interactions of the specified inputs should be generated. "$@2$" indicates that only interactions up to the second order should be used.

```
proc dmreg data=sampsio.dmdring dmdbcat=sampsio.dmdring;
  class c;
  model c=x|x|y|y @2;
  score out=qout outfit=qfit;
  score data=sampsio.dmsring nodmdb out=qgridout;
  title1 'Quadratic-Logistic Regression with Ordinal Target';
run;
```
PROC PRINT produces a report of selected fit statistics for the training data.

proc print data=qfit noobs label;
  var _aic_ _max_ _rfpe_ _misc_;  
  title2 'Fit Statistics for the Training Data Set';  
run;
PROC FREQ creates a report of the misclassification matrix for the training data set.

```plaintext
proc freq data=qout;
    tables f_c*i_c;
    title2 'Misclassification Table: Training Data';
run;
```
PROC GPLOT plots the classification results for the training data set.

```
proc gplot data=qout;
  plot y*x=i_c / haxis=axis1 vaxis=axis2;
  symbol  c=black i=none v=dot;
  symbol2 c=red i=none v=square;
  symbol3 c=green i=none v=triangle;
  axis1 c=black width=2.5 order=(0 to 30 by 5);
  axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
  title2 'Classification Results';
run;
```
PROC GCONTOUR plots the posterior probabilities.

proc gcontour data=qgridout;
   plot y*x=p_c1 / pattern ctext=black coutline=gray;
   plot y*x=p_c2 / pattern ctext=black coutline=gray;
   plot y*x=p_c3 / pattern ctext=black coutline=gray;
   title2 'Posterior Probabilities';
   pattern v=msolid;
   legend frame;
run;
run;
Example 2: Performing a Stepwise OLS Regression (DMREG Baseball Data)

Features
- Stepwise Regression using the SBC selection criterion
- Scoring a Test Data Set with the Score statement
- Outputting Fit Statistics
- Creating Diagnostic Plots

This example demonstrates how to perform a stepwise OLS regression using the DMREG procedure. The example DMDB training data set SAMSPO.DMBASE (baseball data set) contains performance measures and salary levels for regular hitters and leading substitute hitters in major league baseball for the year 1986 (Collier 1987). There is one observation per hitter. The continuous response variable is the log of the players salary (logsalar). The SAMSPO.DMTBASE data set is a test data set which is scored using the scoring formula from the trained model. The SAMSPO.DMBASE and SAMSPO.DMTBASE data sets and the SAMSPO.DMDBASE data mining catalog are stored in the sample library.

Program

```
proc dmreg data=sampsio.dmdbase dmdbcat=sampsio.dmdbase
testdata=sampsio.dmtbase outest=regest;
   class league division position;
   model logsalar = no_atbat no_hits no_home no_runs no_rbi no_bb
   yr_major cr_atbat cr_hits cr_home cr_runs
   cr_rbi cr_bb league division position no_outs
   no_assts no_error
   / error=normal
   choose=sbc
   selection=stepwise
   slentry=0.25 slstay=0.25;
   score data=sampsio.dmtbase nodmdb
   out=regout(rename=(p_logsal=predict r_logsal=residual));
title 'Output from the DMREG Procedure';
run;
```
proc print data=regest noobs label;
  var _step_ _chosen_ _sbc_ _mse_ _averr_ _tmse_ _taverr_
  where _type_ = 'PARMS';
  title 'Partial Listing of the OUTEST= Data Set';
run;

proc gplot data=regout;
  plot logsalar*predict / haxis=axis1 vaxis=axis2 frame;
  symbol c=black i=none v=dot h=3 pct;
  axis1 c=black width=2.5;
  axis2 c=black width=2.5;
  title 'Diagnostic Plots for the Scored Baseball Data';

  plot residual*predict / haxis=axis1 vaxis=axis2;
run;
quit;

---

### Output

#### Summary Profile Information

The first section of the output lists the two-level data set name, the response variable, the number of observations, the error distribution, and the link function.

---

<table>
<thead>
<tr>
<th>Output from the DMREG Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>The DMREG Procedure</td>
</tr>
</tbody>
</table>

| Training Data Set:               | SAMPS10.DMDBASE |
| DMDB Catalog:                    | SAMPS10.DMDBASE |
| Target Variable:                 | logsalar (Log Salary) |
| Target Measurement Level:        | Interval |
| Error:                           | Normal |
| Link Function:                   | Identity |
| Number of Model Parameters:      | 38 |
| Number of Observations:          | 163 |

---

### Design Matrix For Classification Effects

The DMREG procedure uses a deviation from the means method to generate the design matrix for the classification inputs. Each row of the design matrix is generated by a unique combination of the nominal input values. Each column of the design matrix corresponds to a model parameter.

If a nominal variable SWING has k levels (3), then its main effect has k-1 (2) degrees of freedom, and the design matrix has k-1 (2) columns that correspond to the first k-1 levels. The ith column contains a 1 in the ith row, a -1 in the last row, and 0 everywhere else. If \( \alpha_i \) denotes the parameter that corresponds to the ith level of variable SWING, then k-1 columns yield estimates of the independent parameter \( \alpha_1, \alpha_2, \ldots, \alpha_{k-1} \). The last parameter is not needed because DMREG constrains the k parameters to sum to 0. Crossed effects, such as SWING*LEAGUE, are formed by the horizontal direct product of main effects.

---

**Design Matrix Classification Table**
### Data Levels for SWING

<table>
<thead>
<tr>
<th></th>
<th>Design Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>1 0</td>
</tr>
<tr>
<td>Right</td>
<td>0 1</td>
</tr>
<tr>
<td>Switch</td>
<td>-1 -1</td>
</tr>
</tbody>
</table>

The printing of the design matrix can be suppressed by using the MODEL statement option NODESIGNPRINT.

### Input Class Level Information

#### Design Variables

<table>
<thead>
<tr>
<th>Class</th>
<th>Value</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>league</td>
<td>American</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>National</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>division</td>
<td>East</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>West</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>position</td>
<td>1B</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2B</td>
<td>0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2S</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3B</td>
<td>0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3D</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3S</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CD</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CF</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DH</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DO</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LF</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D1</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DF</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OS</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S3</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SS</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UT</td>
<td>-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Model Fitting Information for Each Subset Model of the Stepwise Selection Process

For brevity, only steps number 5 and 8 from the stepwise selection process are listed in the following output. Step number 5 contains the model that has the smallest SBC statistic. This model is used to score the test data set. Because no other inputs met the condition for removal from the model and no other variables met the criterion for addition to the model, the stepwise algorithm terminates after step number 8.

For each model subset of the stepwise modeling process, DMREG provides:

1. An analysis of variance table which lists degrees of freedom, sums of squares, mean squares, the Model F, and its associated p-value.

2. Model fitting information which contains the following statistics that enable you to assess the fit of each stepwise model:

   - **R-square** - which is calculated as \( 1 - \frac{SSE}{SST} \), where SSE is the error sums of squares and SST is the total sums of squares. The \( R^2 \) statistic ranges from 0 to 1. Models that have large values of \( R^2 \) are preferred. For step number 8, the regression equation explains 60.17% of the variability in the target.

   - **Adj R-sq** - the Adj-\( R^2 \) is an alternative criterion to the \( R^2 \) statistic that is adjusted for the number of parameters in the model. This statistic is calculated as \( 1 - \left( \left( \frac{n}{n - i} \right) \left( 1 - R^2 \right) \right) / \left( n - p \right) \), where \( n \) is the number cases, and \( i \) is an indicator variable that is 1 if the model includes an intercept and 0, otherwise. Large differences between the \( R^2 \) and the Adj-\( R^2 \) values for a given model can indicate that you have used too many inputs in the model.

   - **AIC** - Akaike's Information Criterion, which is a goodness-of-fit statistic that you can use to compare one model to another. Lower values indicate a more desirable model. It is calculated as \( (n) \ln \left( \frac{SSE}{n} \right) + 2p \), where \( n \) is the number of cases, SSE is the error sums of squares, and \( p \) is the number of model parameters.

   - **BIC** - Bayesian Information Criterion is another goodness-of-fit statistic that is calculated as \( (n) \ln \left( \frac{SSE}{n} \right) + (p + 2)q - 2q^2 \), where \( q = \frac{MSE}{SSE/n} \) (MSE is obtained from the full model). Smaller BIC values are preferred.

   - **SBC** - Schwarz's Bayesian Criterion is another goodness-of-fit statistic that is calculated as \( (n) \ln \left( \frac{SSE}{n} \right) + (p) \ln \left( n \right) \). Models that have small SBC values are preferred. Because the CHOOSE=SBC option was specified, DMREG selects the model that has the smallest SBC value.

   - **C(p)** - Mallow's Cp Statistic enables you to determine if your model is under or overspecified. This statistic is calculated as \( \left( \frac{SSE(p)}{MSE} \right) - (n - 2p) \), where \( SSE(p) \) is the error sums of squares for the subset model with \( p \) parameters including the intercept if any, \( MSE \) is the error mean square for the full model, and \( n \) is the number of cases. For any subset model \( C(p) > p \), there is evidence of bias due to an incompletely specified model (your model may not contain enough inputs). However, if there are values of \( C(p) < p \), the full model is said to be overspecified. When the right model is chosen, the parameter estimates are unbiased, and this is reflected in \( Cp < p \) or at least near \( p \).

3. Analysis of effects and parameter estimates that contains the effect, degrees of freedom, parameter estimate, standard error, type II sums of squares, F-value and the corresponding p-value.
Step 5. Effect no_bb entered:

Analysis of Variance

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>Mean Square</th>
<th>F Value</th>
<th>Pr&gt;F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>5</td>
<td>72.833938</td>
<td>14.566786</td>
<td>43.41</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>Error</td>
<td>157</td>
<td>52.678742</td>
<td>0.335533</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>Corrected Total</td>
<td>162</td>
<td>125.512680</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

Model Fitting Information

R-square 0.5803  Adj R-sq 0.5669
AIC -172.1147  BIC -169.6060
SBC -153.5522  C(p) 5.3257

Type III Analysis of Effects

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>Type III SS</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_hits</td>
<td>1</td>
<td>5.6348</td>
<td>16.7935</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>no_bb</td>
<td>1</td>
<td>1.7104</td>
<td>5.0976</td>
<td>0.0253</td>
</tr>
<tr>
<td>cr_hits</td>
<td>1</td>
<td>27.5518</td>
<td>82.1135</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>no_outs</td>
<td>1</td>
<td>2.5542</td>
<td>7.6123</td>
<td>0.0065</td>
</tr>
<tr>
<td>no_error</td>
<td>1</td>
<td>2.0405</td>
<td>6.0812</td>
<td>0.0147</td>
</tr>
</tbody>
</table>

Analysis of Parameter Estimates

| Parameter  | DF | Estimate | Standard Error | t Value | Pr>|t| |
|------------|----|----------|----------------|---------|------|
| Intercept  | 1  | 4.5835   | 0.1290         | 35.52   | <.0001 |
| no_hits    | 1  | 0.00562  | 0.00137        | 4.10    | <.0001 |
| no_bb      | 1  | 0.00602  | 0.00267        | 2.26    | 0.0253 |
| cr_hits    | 1  | 0.000701 | 0.000077       | 9.06    | <.0001 |
| no_outs    | 1  | 0.000453 | 0.000164       | 2.76    | 0.0065 |
| no_error   | 1  | -0.0180  | 0.00729        | -2.47   | 0.0147 |
Summary of the Stepwise Selection Process

The Summary of Stepwise Procedure section provides the step number, the explanatory input or inputs entered or removed at each step, the $F$ statistic, and the corresponding $p$-value in which the entry or removal of the input is based. For this example, 8 of the 19 original inputs met the 0.25 entry and stay probability values.

Analysis of Variance

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>Mean Square</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>8</td>
<td>75.525299</td>
<td>9.440662</td>
<td>29.08</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>Error</td>
<td>154</td>
<td>49.987381</td>
<td>0.324593</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>Corrected Total</td>
<td>162</td>
<td>125.512680</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

Model Fitting Information

| R-square | 0.6017 | Adj R-sq | 0.5810 |
| AIC       | -174.6627 | BIC | -170.9032 |
| SBC       | -146.8189 | C(p)     | 3.3390 |

Type III Analysis of Effects

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>Type III SS</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_hits</td>
<td>1</td>
<td>1.3988</td>
<td>4.3095</td>
<td>0.0396</td>
</tr>
<tr>
<td>no_rbi</td>
<td>1</td>
<td>0.4606</td>
<td>1.4191</td>
<td>0.2354</td>
</tr>
<tr>
<td>no_bb</td>
<td>1</td>
<td>2.1274</td>
<td>6.5541</td>
<td>0.0114</td>
</tr>
<tr>
<td>yr_major</td>
<td>1</td>
<td>1.6517</td>
<td>5.0885</td>
<td>0.0255</td>
</tr>
<tr>
<td>cr_hits</td>
<td>1</td>
<td>1.6766</td>
<td>5.1651</td>
<td>0.0244</td>
</tr>
<tr>
<td>cr_bb</td>
<td>1</td>
<td>0.7491</td>
<td>2.3077</td>
<td>0.1308</td>
</tr>
<tr>
<td>no_outs</td>
<td>1</td>
<td>2.3525</td>
<td>7.2475</td>
<td>0.0079</td>
</tr>
<tr>
<td>no_error</td>
<td>1</td>
<td>1.3915</td>
<td>4.2869</td>
<td>0.0401</td>
</tr>
</tbody>
</table>

Analysis of Parameter Estimates

| Parameter    | DF | Estimate | Standard Error | t Value | Pr>|t| |
|--------------|----|----------|----------------|---------|--------|
| Intercept    | 1  | 4.2938   | 0.1834         | 23.41   | <.0001 |
| no_hits      | 1  | 0.00424  | 0.00204        | 2.08    | 0.0396 |
| no_rbi       | 1  | 0.00362  | 0.00304        | 1.19    | 0.2354 |
| no_bb        | 1  | 0.00880  | 0.00344        | 2.56    | 0.0114 |
| yr_major     | 1  | 0.0580   | 0.0257         | 2.26    | 0.0255 |
| cr_hits      | 1  | 0.100571 | 0.000251       | 2.27    | 0.0244 |
| cr_bb        | 1  | -0.00084 | 0.000551       | -1.52   | 0.1308 |
| no_outs      | 1  | 0.000439 | 0.000163       | 2.69    | 0.0079 |
| no_error     | 1  | -0.0152  | 0.00733        | -2.07   | 0.0401 |

NOTE: No (additional) effects met the 0.25 significance level for entry into the model.
Summary of Stepwise Procedure

| Step | Effect | Entered | DF | Ind | F   | Prob>|F |
|------|--------|---------|----|-----|-----|------|
| 1    | cr_hits| 1       | 1  | 97.1611 | <.0001 |
| 2    | no_hits| 1       | 2  | 49.7845 | <.0001 |
| 3    | no_outs| 1       | 3  | 8.7109  | 0.0036 |
| 4    | no_error| 1     | 4  | 6.4182  | 0.0123 |
| 5    | no_bb| 1       | 5  | 5.0376  | 0.0253 |
| 6    | yr_major| 1    | 6  | 4.3023  | 0.0379 |
| 7    | cr_bb| 1       | 7  | 2.4313  | 0.1210 |
| 8    | no_rbi| 1       | 8  | 1.4191  | 0.2354 |

List Report of Selected Variables in the OUTEST= data set

The example PROC PRINT report of the OUTEST= data set lists selected fit statistics for the training and test data sets. The default OUTEST data set contains two observations for each step number. These observations are distinguished by value of the _TYPE_ variable:
- _TYPE_ = "PARMS" - contains parameter estimate statistics
- _TYPE_ = "T" - contains the t-value for the parameter estimate

Because a WHERE statement was used to select only values of TYPE = "PARMS", this report contains one observation per step number. An additional observation is displayed that identifies the model chosen based on the SBC criterion (CHOOSE="SBC").

Partial Listing of the OUTEST= Data Set

<table>
<thead>
<tr>
<th>Model Selection Number</th>
<th>Train: Selection Criterion</th>
<th>Train: Mean Criterion</th>
<th>Train: Average Criterion</th>
<th>Test: Mean Criterion</th>
<th>Test: Average Criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-37.505</td>
<td>0.77477</td>
<td>0.77002</td>
<td>0.81858</td>
<td>0.81858</td>
</tr>
<tr>
<td>1</td>
<td>-109.377</td>
<td>0.46618</td>
<td>0.48021</td>
<td>0.42598</td>
<td>0.42598</td>
</tr>
<tr>
<td>2</td>
<td>-148.441</td>
<td>0.37312</td>
<td>0.36625</td>
<td>0.34632</td>
<td>0.34632</td>
</tr>
<tr>
<td>3</td>
<td>-152.041</td>
<td>0.35597</td>
<td>0.34723</td>
<td>0.37438</td>
<td>0.37438</td>
</tr>
<tr>
<td>4</td>
<td>-153.438</td>
<td>0.34424</td>
<td>0.33368</td>
<td>0.38553</td>
<td>0.38553</td>
</tr>
<tr>
<td>5</td>
<td>-153.552</td>
<td>0.33553</td>
<td>0.32318</td>
<td>0.37563</td>
<td>0.37563</td>
</tr>
<tr>
<td>6</td>
<td>-152.974</td>
<td>0.32846</td>
<td>0.31435</td>
<td>0.34713</td>
<td>0.34713</td>
</tr>
<tr>
<td>7</td>
<td>-150.418</td>
<td>0.32547</td>
<td>0.30950</td>
<td>0.35231</td>
<td>0.35231</td>
</tr>
<tr>
<td>8</td>
<td>-146.819</td>
<td>0.32459</td>
<td>0.30667</td>
<td>0.35978</td>
<td>0.35978</td>
</tr>
<tr>
<td>5</td>
<td>SBC</td>
<td>-153.552</td>
<td>0.33553</td>
<td>0.32318</td>
<td>0.37563</td>
</tr>
</tbody>
</table>

GPLOT Diagnostic Plots for the Scored Baseball Test Data

Plot of the log of salary versus the predicted log of salary.
Plot of the residual values versus the predicted log of salary.
Diagnostic Plots for the Scored Baseball Data

Residual: logsolar

Predicted: logsolar

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The PROC DMREG statement invokes the procedure. The DATA= option identifies the training data set that is used to fit the model. The DMDBCAT= option identifies the training data catalog.

```plaintext
c proc dmreg data=sampsio.dmdbase dmdbcat=sampsio.dmdbase
```
The TESTDATA= option identifies the test data set. The OUTEST= option creates the output data set containing estimates and fit statistics.

testdata=sampsio.dmtbase outest=regest;
The CLASS statement specifies the categorical variables to be used in the regression analysis.

    class league division position;
The MODEL statement specifies the linear model. The ERROR=normal model option specifies to use the normal error distribution. The CHOOSE=SBC model option specifies to choose the model subset with the smallest Schwarz Bayesian criterion.

model logsalar = no_atbat no_hits no_home no_runs no_rbi no_bb
    yr_major cr_atbat cr_hits cr_home cr_runs
    cr_rbi cr_bb league division position no_outs
    no_assts no_error

/ error=normal
    choose=sbc
The MODEL option `SELECTION=STEPWISE` specifies to use the stepwise variable selection method. Stepwise selection systematically adds and deletes inputs from the model based on the `SLENTRY=` and `SLSTAY=` significance levels. The subset models are created based on the `SLENTRY` and `SLSTAY` significance levels, but the model that is chosen is based solely on the subset model that has the smallest SBC criterion.

```plaintext
selection=stepwise
slentry=0.25 slstay=0.25;
```
The SCORE statement specifies the data set that you want to score in conjunction with training. The DATA= option identifies the score data set (for this example, the test data set).

```
score data=sampsio.dmtbase nodmdb
```
The OUT=option identifies the output data set that contains estimates and fit statistics for the scored data set. The RENAME=option enables you to rename variables in the OUT= data set.

```plaintext
out=regout(rename=(p_logsal=predict r_logsal=residual));
title 'Output from the DMREG Procedure';
run;
```
PROC PRINT produces a report of selected variables from the OUTEST= data set.

```sas
proc print data=regest noobs label;
   var _step_ _chosen_ _sbc_ _mse_ _averr_ _tmse_ _taverr_;
   where _type_ = 'PARMS';
   title 'Partial Listing of the OUTEST= Data Set';
run;
```
PROC GPLOT produces diagnostic plots of the scored test data. The first PLOT statement plots the response versus the predicted values.

```plaintext
proc gplot data=regout;
  plot logsalar*predict / haxis=axis1 vaxis=axis2 frame;
  symbol c=black i=none v=dot h=3 pct;
  axis1 c=black width=2.5;
  axis2 c=black width=2.5;
  title 'Diagnostic Plots for the Scored Baseball Data';
```
The second PLOT statement plots the residuals versus the predicted values.

```
plot residual*predict / haxis=axis1 vaxis=axis2;
run;
quit;
```
Example 3: Comparison of the DMREG and LOGISTIC Procedures when Using a Categorical Input Variable

This example provides a comparison of the DMREG and LOGISTIC procedures when using a categorical input to model a binary target. The example data set SAMPsIO.HMEQ contains fictitious mortgage data where each case represents an applicant for a home equity loan. All applicants have an existing mortgage.

The binary target BAD represents whether or not an applicant eventually defaulted or was ever seriously delinquent. There are nine continuous inputs available for modeling. JOB is the only categorical input used to predict the target BAD.

When you compare the output from the DMREG and LOGISTIC procedures code, you must take into consideration how each procedure handles the categorical variables. By default, DMREG uses a deviations from the means coding to code the classification variables. The design matrix for the class effects has values of 0, 1, and -1 for the reference levels. This coding is sometimes referred to as "effects", "center-point", and "full-rank" coding. The parameters for these categorical indicators measure the difference from each level to the average across levels.

Because the LOGISTIC procedure does not enable you to specify class inputs directly in the MODEL statement, you must first create an input data set that contains the design matrix for the class variables. To create the design matrix data set for input to the LOGISTIC procedure, you can use a SAS DATA step, a TRANSREG procedure, or a GENMOD procedure. If you use the deviations from the means coding method to code the class variables, then the LOGISTIC output will automatically match the output generated from the DMREG run. If you use the GLM non-full rank coding (0, 1) to code the class variables, you must set the DMREG CODE= MODEL statement option in GLM. In this case, both procedures will generate the same output.

Program: Deviations from the Mean Coding

```sas
proc freq data=sampsio.hmeq;
   tables job / missing;
   title 'JOB Classification Table';
run;

data hmeq;
   set sampsio.hmeq;
   if job = ' ' then job='Other';
run;

proc transreg data=hmeq design;
   model class (job/deviations);
   id bad loan mortdue value yoj derog clage ninq clno debtinc;
   output;
run;
```
/*
data dumyhm eq;
    set hmeq;
    select (job);
    when ('Mgr')
        do;
            j_mgr=1;
            j_off=0;
            j_other=0;
            j_prof=0;
            j_sales=0;
            j_self=-1;
        end;
    when ('Office')
        do;
            j_mgr=0;
            j_off=1;
            j_other=0;
            j_prof=0;
            j_sales=0;
            j_self=-1;
        end;
    when ('Other')
        do;
            j_mgr=0;
            j_off=0;
            j_other=1;
            j_prof=0;
            j_sales=0;
            j_self=-1;
        end;
    when ('ProfExe')
        do;
            j_mgr=0;
            j_off=0;
            j_other=0;
            j_prof=1;
            j_sales=0;
            j_self=-1;
        end;
    when ('Sales')
        do;
            j_mgr=0;
            j_off=0;
            j_other=0;
            j_prof=0;
            j_sales=1;
            j_self=-1;
        end;
    when ('Self')
        do;
            j_mgr=-1;
            j_off=-1;
            j_other=-1;
            j_prof=-1;
            j_self=-1;
*/
j_sales=-1;
j_self=-1;
end;
otherwise;
end;
run;

* /

proc logistic descending;
    model bad = &_trgind loan mortdue value yoj
derog clage ninq clno debtinc;
    title 'LOGISTIC Home Equity Data: Deviations from the Mean Coding';
run;

proc dmdb batch data=hmeq
    out=dm_data dmdbcat=dm_cat;
    var loan mortdue value yoj derog
clage ninq clno debtinc;
    class bad(desc)
        job(asc);
    target bad;
run;

proc dmreg data=dm_data
    dmdbcat=dm_cat;
    class bad job;
    model bad = job loan mortdue value yoj derog
clage ninq clno debtinc;
    title1 'DMREG Home Equity Data:
        Default Deviations from the Mean Coding';
run;

Output: Deviations from the Mean Coding

FREQ Classification Table for JOB.
The categorical input JOB contains 7 levels. Notice that 279 cases have missing values. Both the DMREG and LOGISTIC procedures omit observations that have missing values from the analysis. For this example, the missing values are imputed using the mode of JOB.

JOB Classification Table
The FREQ Procedure

<table>
<thead>
<tr>
<th>JOB</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mgr</td>
<td>279</td>
<td>4.68</td>
<td>279</td>
<td>4.68</td>
</tr>
<tr>
<td>Office</td>
<td>767</td>
<td>12.87</td>
<td>1046</td>
<td>17.55</td>
</tr>
<tr>
<td>Other</td>
<td>948</td>
<td>15.91</td>
<td>1994</td>
<td>33.46</td>
</tr>
<tr>
<td>ProfEx</td>
<td>2388</td>
<td>40.07</td>
<td>4382</td>
<td>73.52</td>
</tr>
<tr>
<td>Sales</td>
<td>1276</td>
<td>21.41</td>
<td>5658</td>
<td>94.93</td>
</tr>
<tr>
<td>Self</td>
<td>109</td>
<td>1.83</td>
<td>5767</td>
<td>96.76</td>
</tr>
<tr>
<td></td>
<td>193</td>
<td>3.24</td>
<td>5960</td>
<td>100.00</td>
</tr>
</tbody>
</table>
LOGISTIC Output

LOGISTIC Home Equity Data: Deviations from the Mean Coding

The LOGISTIC Procedure

Model Information

Data Set WORK.DATAT
Response Variable BAD
Number of Response Levels 2
Number of Observations 3527
Model binary_logit
Optimization Technique Fisher’s scoring

Response Profile

<table>
<thead>
<tr>
<th>Ordered Value</th>
<th>BAD</th>
<th>Total Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>313</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3214</td>
</tr>
</tbody>
</table>

Probability modeled is BAD=1.

NOTE: 2433 observations were deleted due to missing values for the response or explanatory variables.

Model Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Intercept Only</th>
<th>Intercept and Covariates</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIC</td>
<td>2115.535</td>
<td>1765.011</td>
</tr>
<tr>
<td>SC</td>
<td>2121.703</td>
<td>1857.534</td>
</tr>
<tr>
<td>-2 Log L</td>
<td>2113.535</td>
<td>1795.011</td>
</tr>
</tbody>
</table>

Testing Global Null Hypothesis: BETA=0

<table>
<thead>
<tr>
<th>Test</th>
<th>Chi-Square</th>
<th>DF</th>
<th>Pr &gt; ChiSq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Likelihood Ratio</td>
<td>378.5235</td>
<td>14</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>Score</td>
<td>488.0227</td>
<td>14</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>Wald</td>
<td>250.3468</td>
<td>14</td>
<td>&lt;.0001</td>
</tr>
</tbody>
</table>

Analysis of Maximum Likelihood Estimates

<table>
<thead>
<tr>
<th>Parameter</th>
<th>DF</th>
<th>Estimate</th>
<th>Standard Error</th>
<th>Chi-Square</th>
<th>Wald</th>
<th>Pr &gt; ChiSq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>1</td>
<td>-4.7354</td>
<td>0.4913</td>
<td>120.5087</td>
<td>&lt;.0001</td>
<td></td>
</tr>
<tr>
<td>JOBMgr</td>
<td>1</td>
<td>-0.1672</td>
<td>0.1755</td>
<td>0.3073</td>
<td>0.5408</td>
<td></td>
</tr>
<tr>
<td>JOBOffice</td>
<td>1</td>
<td>-0.5672</td>
<td>0.1745</td>
<td>10.5634</td>
<td>0.0012</td>
<td></td>
</tr>
<tr>
<td>JOBOther</td>
<td>1</td>
<td>-0.2373</td>
<td>0.1337</td>
<td>3.1513</td>
<td>0.0759</td>
<td></td>
</tr>
<tr>
<td>JOBProfExe</td>
<td>1</td>
<td>-0.2414</td>
<td>0.1551</td>
<td>2.4231</td>
<td>0.1196</td>
<td></td>
</tr>
<tr>
<td>JOBSales</td>
<td>1</td>
<td>0.7526</td>
<td>0.3348</td>
<td>5.0514</td>
<td>0.0246</td>
<td></td>
</tr>
<tr>
<td>LOAN</td>
<td>1</td>
<td>-0.00003</td>
<td>7.786E-6</td>
<td>0.4819</td>
<td>0.0012</td>
<td></td>
</tr>
<tr>
<td>MORTDUE</td>
<td>1</td>
<td>-5.98E-6</td>
<td>3.72E-6</td>
<td>2.5780</td>
<td>0.1084</td>
<td></td>
</tr>
<tr>
<td>VALUE</td>
<td>1</td>
<td>4.651E-6</td>
<td>3.184E-6</td>
<td>2.1339</td>
<td>0.1441</td>
<td></td>
</tr>
<tr>
<td>YOJ</td>
<td>1</td>
<td>-0.00818</td>
<td>0.00371</td>
<td>0.7099</td>
<td>0.3995</td>
<td></td>
</tr>
<tr>
<td>DEROG</td>
<td>1</td>
<td>0.8898</td>
<td>0.0984</td>
<td>81.7161</td>
<td>&lt;.0001</td>
<td></td>
</tr>
</tbody>
</table>
DMREG Output

Notice that the DMREG output matches the output generated from the LOGISTIC run.

DMREG Home Equity Data: Default Deviations from the Mean Coding

The DMREG Procedure

Training Data Set: WORK.DM_DATA
DMDB Catalog: WORK.DM_CAT
Target Variable: BAD
Target Measurement Level: Ordinal
Number of Target Categories: 2
Error: MBernoulli
Link Function: Logit
Number of Model Parameters: 15
Number of Observations: 3527

Target Profile

<table>
<thead>
<tr>
<th>Ordered Value</th>
<th>Total BAD</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>313</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3214</td>
</tr>
</tbody>
</table>

Input Class Level Information

Design Variables

<table>
<thead>
<tr>
<th>Class</th>
<th>Value</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB</td>
<td>Mgr</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Office</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ProfExe</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Sales</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Self</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

DMREG Home Equity Data: Default Deviations from the Mean Coding

The DMREG Procedure

Newton-Raphson Ridge Optimization

Without Parameter Scaling

Parameter Estimates 15

Optimization Start

Active Constraints 0
Max Abs Gradient Element 29.970626595

Objective Function 1056.7673865
Optimization Results

<table>
<thead>
<tr>
<th>Iter</th>
<th>Restarts</th>
<th>Function Calls</th>
<th>Active Constraints</th>
<th>Objective Function</th>
<th>Objective Function Change</th>
<th>Max Abs Gradient Element</th>
<th>Ridge</th>
<th>Ratio Between Actual and Predicted Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>912.15841</td>
<td>144.6</td>
<td>116.9</td>
<td>0</td>
<td>0.593</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>868.27555</td>
<td>43.8829</td>
<td>9.4267</td>
<td>0</td>
<td>1.006</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>867.50765</td>
<td>0.7679</td>
<td>0.4844</td>
<td>0</td>
<td>1.033</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>867.50562</td>
<td>0.00203</td>
<td>0.00140</td>
<td>0</td>
<td>1.002</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>867.50562</td>
<td>1.67E-8</td>
<td>1.184E-8</td>
<td>0</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Testing Global Null Hypothesis BETA=0

Criterion Intercept Only Intercept and Covariates Chi-Square for Covariates
-2 LOG L 2113.535 1735.011 378.524 with 14 DF (p < .0001)

Type III Analysis of Effects

<table>
<thead>
<tr>
<th>Effect</th>
<th>DF</th>
<th>Chi-Square</th>
<th>Pr &gt;</th>
<th>Wald</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB</td>
<td>5</td>
<td>14.3409</td>
<td>0.0136</td>
<td></td>
</tr>
<tr>
<td>LOAN</td>
<td>1</td>
<td>10.4820</td>
<td>0.0012</td>
<td></td>
</tr>
<tr>
<td>MORTDUE</td>
<td>1</td>
<td>2.5780</td>
<td>0.1084</td>
<td></td>
</tr>
<tr>
<td>VALUE</td>
<td>1</td>
<td>2.1339</td>
<td>0.1441</td>
<td></td>
</tr>
<tr>
<td>JOY</td>
<td>1</td>
<td>0.7099</td>
<td>0.3995</td>
<td></td>
</tr>
<tr>
<td>DEROG</td>
<td>1</td>
<td>81.7165</td>
<td>&lt;.0001</td>
<td></td>
</tr>
<tr>
<td>CLAGE</td>
<td>1</td>
<td>26.7300</td>
<td>&lt;.0001</td>
<td></td>
</tr>
<tr>
<td>NINQ</td>
<td>1</td>
<td>7.3653</td>
<td>0.0066</td>
<td></td>
</tr>
<tr>
<td>CLNO</td>
<td>1</td>
<td>0.1735</td>
<td>0.6770</td>
<td></td>
</tr>
<tr>
<td>DEBTINC</td>
<td>1</td>
<td>105.6561</td>
<td>&lt;.0001</td>
<td></td>
</tr>
</tbody>
</table>

Analysis of Maximum Likelihood Estimates

<table>
<thead>
<tr>
<th>Parameter</th>
<th>DF</th>
<th>Estimate</th>
<th>Standard Error</th>
<th>Wald</th>
<th>Pr &gt; Chi-Square</th>
<th>Wald</th>
<th>Pr &gt; Chi-Square</th>
<th>Standardized Estimate</th>
<th>exp(Est)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>1</td>
<td>-4.7355</td>
<td>0.4313</td>
<td>120.54</td>
<td>&lt;.0001</td>
<td>0.009</td>
<td>0.009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mgr</td>
<td>1</td>
<td>-0.1672</td>
<td>0.1755</td>
<td>0.91</td>
<td>0.3408</td>
<td>0.846</td>
<td>0.846</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Office</td>
<td>1</td>
<td>-0.5672</td>
<td>0.1745</td>
<td>10.56</td>
<td>0.0012</td>
<td>0.567</td>
<td>0.567</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>1</td>
<td>-0.2373</td>
<td>0.1337</td>
<td>3.15</td>
<td>0.0759</td>
<td>0.789</td>
<td>0.789</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProfExe</td>
<td>1</td>
<td>-0.2415</td>
<td>0.1551</td>
<td>2.42</td>
<td>0.1196</td>
<td>0.785</td>
<td>0.785</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sales</td>
<td>1</td>
<td>0.7526</td>
<td>0.3348</td>
<td>5.05</td>
<td>0.0246</td>
<td>2.122</td>
<td>2.122</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAN</td>
<td>1</td>
<td>-0.00003</td>
<td>7.78E-6</td>
<td>10.48</td>
<td>0.0012</td>
<td>-0.158376</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MORTDUE</td>
<td>1</td>
<td>-5.38E-6</td>
<td>3.72E-6</td>
<td>2.58</td>
<td>0.1084</td>
<td>-0.147394</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VALUE</td>
<td>1</td>
<td>4.651E-6</td>
<td>3.184E-6</td>
<td>2.13</td>
<td>0.1441</td>
<td>0.138844</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JOY</td>
<td>1</td>
<td>-0.00818</td>
<td>0.00971</td>
<td>0.71</td>
<td>0.3995</td>
<td>-0.034557</td>
<td>0.992</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEROG</td>
<td>1</td>
<td>0.8898</td>
<td>0.0984</td>
<td>81.72</td>
<td>&lt;.0001</td>
<td>0.282757</td>
<td>2.435</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLAGE</td>
<td>1</td>
<td>-0.00520</td>
<td>0.00101</td>
<td>26.74</td>
<td>&lt;.0001</td>
<td>-0.235822</td>
<td>0.935</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NINQ</td>
<td>1</td>
<td>0.0070</td>
<td>0.0360</td>
<td>7.37</td>
<td>0.0066</td>
<td>0.002609</td>
<td>1.103</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLNO</td>
<td>1</td>
<td>-0.00314</td>
<td>0.00753</td>
<td>0.17</td>
<td>0.6770</td>
<td>-0.016175</td>
<td>0.997</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEBTINC</td>
<td>1</td>
<td>0.1033</td>
<td>0.0101</td>
<td>105.66</td>
<td>&lt;.0001</td>
<td>0.488171</td>
<td>1.109</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Odds Ratio Estimates
Program: GLM Non-Full Rank (0, 1) Coding

```sas
data dumyhmeq;
  set hmeq;
    j_mgr=(job='Mgr');
    j_off=(job='Office');
    j_other=(job='Other');
    j_prof=(job='ProfExe');
    j_sales=(job='Sales');
    j_self=(job='Self');
run;

proc logistic data=dumyhmeq descending noprint;
  model bad = j_mgr j_off j_other j_prof j_sales j_self
         loan mortdue value yoj derog
         clage ninq clno debtinc;
  output out=logfit(keep=bad p_bad1) p=p_bad1;
  title 'LOGISTIC Home Equity Data: GLM coding';
run;

proc dmdb batch data=hmeq
  out=dm_data dmdbcat=dm_cat;
  var loan mortdue value yoj derog
      clage ninq clno debtinc;
  class bad(desc)
        reason(asc)
        job(asc);

  target bad;
run;

proc dmreg data=dm_data
  dmdbcat=dm_cat
  noprint;
  class bad job;
  model bad = job loan mortdue value yoj derog
           clage ninq clno debtinc / coding=glm;
  score out=dmscore;
  title1 'DMREG Home Equity Data: GLM coding';
run;

proc compare data=dmscore compare=logfit note
  method=absolute
```
criterion=1e-7;
var p_bad1;
run;

Output: GLM Non-Full Rank (0, 1) Coding
PROC COMPARE results.

DMREG Home Equity Data: GLM coding

The COMPARE Procedure
Comparison of WORK.DMSCORE with WORK.LOGFIT
(Method=ABSOLUTE, Criterion=0.0000001)

Data Set Summary

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Created</th>
<th>Modified</th>
<th>NVar</th>
<th>NObs</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK.DMSCORE</td>
<td>01NOV00:14:19:52</td>
<td>01NOV00:14:19:52</td>
<td>20</td>
<td>5960</td>
</tr>
<tr>
<td>WORK.LOGFIT</td>
<td>01NOV00:14:19:50</td>
<td>01NOV00:14:19:50</td>
<td>2</td>
<td>5960</td>
</tr>
</tbody>
</table>

Variables Summary

Number of Variables in Common: 2.
Number of Variables in WORK.DMSCORE but not in WORK.LOGFIT: 18.
Number of Variables with Differing Attributes: 2.
Number of VAR Statement Variables: 1.

Listing of Common Variables with Differing Attributes

<table>
<thead>
<tr>
<th>Variable</th>
<th>Dataset</th>
<th>Type</th>
<th>Length</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_BAD1</td>
<td>WORK.DMSCORE</td>
<td>Num</td>
<td>8</td>
<td>Predicted: BAD=1</td>
</tr>
<tr>
<td></td>
<td>WORK.LOGFIT</td>
<td>Num</td>
<td>8</td>
<td>Estimated Probability</td>
</tr>
</tbody>
</table>

Observation Summary

<table>
<thead>
<tr>
<th>Observation</th>
<th>Base</th>
<th>Compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Obs</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>First Unequal</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Last Unequal</td>
<td>5960</td>
<td>5960</td>
</tr>
<tr>
<td>Last Obs</td>
<td>5960</td>
<td>5960</td>
</tr>
</tbody>
</table>

Number of Observations in Common: 5960.
Total Number of Observations Read from WORK.DMSCORE: 5960.
Total Number of Observations Read from WORK.LOGFIT: 5960.

Number of Observations with Some Compared Variables Unequal: 5822.
Number of Observations with All Compared Variables Equal: 138.

Values Comparison Summary

Number of Variables Compared with All Observations Equal: 0.
Number of Variables Compared with Some Observations Unequal: 1.
Number of Variables with Missing Value Differences: 1.
Total Number of Values which Compare Unequal: 5822.
Total Number of Values not EXACTLY Equal: 5960.
Maximum Difference: 0.00000064507.
PROC FREQ step to create a classification table for the categorical input JOB.

```
proc freq data=sampsio.hmeq;
  tables job / missing;
  title 'JOB Classification Table';
run;
```
SAS DATA step to replace the missing JOB values with the variable's mode. It does not matter whether or not you perform data imputation prior to modeling - DMREG and LOGISTIC will produce the same results if you use the same method to code the class variables. Some of the continuous inputs have missing values. DMREG and LOGISTIC do not use observations that have missing values in the analysis. You can impute the missing values for the continuous inputs by using the STDIZE procedure.

data hmeq;
  set sampsio.hmeq;
  if job = '' then job='Other';
run;
PROC TRANSREG step to create the design matrix for the classification input JOB. The DESIGN option specifies that the goal is design matrix creation, not analysis.

proc transreg data=hmeq design;
The MODEL statement specifies the class variable JOB. The DEVIATIONS (or EFFECTS) t-option requests a deviations from the means coding.

```sas
model class (job/deviations);
```
The ID statement also specifies to output the target and the continuous inputs to the temporary design matrix data set. PROC TRANSREG automatically creates the macro variable &_TRGIND that contains the list of independent variables. This macro variable is used in the MODEL statement in PROC LOGISTIC.

```plaintext
id bad loan mortdue value yoj derog clage ninq clno debtinc;
output;
run;
```
You can also create the design matrix for the classification variable(s) in a SAS DATA step although this task is too time consuming for databases that contain several class variables. The DATA step is commented out, but it does demonstrate how to manually code a categorical variable using the deviations from the MEANS method.

```/*
data dumyhmeq;
  set hmeq;
  select (job);
  when ('Mgr')
    do;
      j_mgr=1;
      j_off=0;
      j_other=0;
      j_prof=0;
      j_sales=0;
      j_self=-1;
    end;

  when ('Office')
    do;
      j_mgr=0;
      j_off=1;
      j_other=0;
      j_prof=0;
      j_sales=0;
      j_self=-1;
    end;

  when ('Other')
    do;
      j_mgr=0;
      j_off=0;
      j_other=1;
      j_prof=0;
      j_sales=0;
      j_self=-1;
    end;

  when ('ProfExe')
    do;
      j_mgr=0;
      j_off=0;
      j_other=0;
      j_prof=1;
      j_sales=0;
      j_self=-1;
    end;
*/
when ('Sales')
  do;
    j mgr=0;
    j off=0;
    j other=0;
    j prof=0;
    j sales=1;
    j self=-1;
  end;

when ('Self')
  do;
    j mgr=-1;
    j off=-1;
    j other=-1;
    j prof=-1;
    j sales=-1;
    j self=-1;
  end;
  otherwise;
end;
run;

*/
PROC LOGISTIC step to model the binary target BAD. The macro variable &\_TRGIND obtains the classification design matrix from the subsequent PROC TRANSREG run. The DESCENDING option causes the procedure to model the probability that BAD = 1 (bad applicants).

```
proc logistic descending;
  model bad = &\_trgind loan mortdue value yoj
             derog clage ninq clno debtinc;
  title 'LOGISTIC Home Equity Data: Deviations from the Mean Coding';
run;
```
PROC DMDB step to create the DMDB data set and catalog that are required as input to DMREG.

```sas
proc dmdb batch data=hmeq
   out=dm_data dmdbcat=dm_cat;
   var loan mortdue value yoj derog
   clage ninq clno debtinc;
   class bad(desc)
       job(asc);
   target bad;
run;
```
Because the order of the target BAD was set to descending in the DMDB data set, DMREG also models the probability that BAD=1 (bad applicants). By default, DMREG using deviation from the means coding to create the design matrix for the class variables.

```
proc dmreg data=dm_data
dmdbcat=dm_cat;
   class bad job;
   model bad = job loan mortdue value yoj derog
            clage ninq clno debtinc;
   title1 'DMREG Home Equity Data:
            Default Deviations from the Mean Coding';
run;
```
DATA step program to code the class variable JOB using GLM non-full rank (0, 1) coding.

data dumyhmeq;
  set hmeq;
  j_mgr=(job='Mgr');
  j_off=(job='Office');
  j_other=(job='Other');
  j_prof=(job='ProfExe');
  j_sales=(job='Sales');
  j_self=(job='Self');
run;
PROC LOGISTIC step to model the binary target BAD.

proc logistic data=dumyehmeq descending noprint;
   model bad = j_mgr j_off j_other j_prof j_sales j_self
              loan mortdue value yoj derog
              clage ning clno debtinc;
   output out=logfit(keep=bad p_bad1) p=p_bad1;
   title 'LOGISTIC Home Equity Data: GLM coding';
run;
The NOPRINT option suppresses the printing of the DMREG output. PROC COMPARE is used to compare the predicted values from the LOGISTIC and DMREG models. The CODING=GLM option creates the design matrix for the class variables using GLM non-full rank coding.

```plaintext
proc dmreg data=dm_data
dmdbcat=dm_cat
noprint;
class bad job;
model bad = job loan mortdue value yoj derog
clage ninq clno debtinc / coding=glm;
score out=dmscore;
title1 'DMREG Home Equity Data: GLM coding';
run;
```
References


The DMSPLIT Procedure

Overview

Procedure Syntax

PROC DMSPLIT Statement
FREQ Statement
TARGET Statement
VARIABLE Statement
WEIGHT Statement

Details

Examples

Example 1: Creating a Decision Tree for a Binary Target with the DMSPLIT Procedure

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DMSPLIT Procedure

Overview

The DMSPLIT procedure performs variable selection using binary variable splits for maximizing the Chi-Square value of a 2 X 2 frequency table. The cutoff threshold is chosen so that the Chi-Square value of the table is maximized.

PROC DMINE and PROC DMSPLIT are underlying procedures for the Variable Selection node.
Procedure Syntax

PROC DMSPLIT <option(s)>;

FREQ variable;

TARGET variable;

VARIABLE variable-list;

 WEIGHT variable;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DMSPLIT Procedure

PROC DMSPLIT Statement

Invokes the DMSPLIT procedure.

PROC DMSPLIT <option(s)>;

Required Arguments

DATA=<libref.>SAS-data-set

Specifies an input data set generated by PROC DMDB. The data set is associated with a valid catalog specified by the DMDBCAT= option. This data set must contain interval scaled variables and CLASS variables in a specific form written by PROC DMDB.

Default: None.

DMDBCAT=<libref.> SAS-catalog

Identifies an input metadata catalog generated by PROC DMDB. The metadata catalog is associated with a valid data set specified by the DATA= option. The catalog contains important information (for example, the range of variables, number of missing values of each variable, moments of variables) that is used by many other Enterprise Miner procedures that require a DMDB data set. The DMDBCAT= catalog and the DATA= data set must be appropriately related to each other in order to obtain proper results.

Default: None.

Options

BINS=integer

Specifies the number of categories in which the range of a numeric (interval) variable is divided for splits.

Range: Integer > 0

Default: 100

CHISQ=number

Specifies a low bound for the Chi-Square value still eligible for variable splits. The value of CHISQ governs the number of splits that are performed: the higher the value of CHISQ, the fewer splits and passes of the input data will be performed.

Range: number is a real number > 0
### OUTVARS=<libref.>SAS-data-set
Specifies an optional output data set containing most of the output table information for the splits.

### PASSES=integer
Specifies an upper bound for the number of passes through the input data set that are used for performing the binary splits.

<table>
<thead>
<tr>
<th>Range</th>
<th>Integer &gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>12</td>
</tr>
</tbody>
</table>

### PRINT | NOPRINT
Specifies whether or not to suppress all output printed in the Output window.

<table>
<thead>
<tr>
<th>Default</th>
<th>NOPRINT</th>
</tr>
</thead>
</table>

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
FREQ Statement

Alias: FREQUENCY

Tip: Specify the FREQ variable in PROC DMDB so that the information is saved in the catalog and so that the variable is automatically used as a FREQ variable in PROC DMSPLIT. This also ensures that the FREQ variable is automatically used by all other Enterprise Miner procedures in the project.

FREQ variable;

Required Argument

variable

Specifies one numeric (interval scaled) FREQUENCY variable.

<table>
<thead>
<tr>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any integer. A rational value is truncated to the next integer.</td>
</tr>
</tbody>
</table>

CAUTION:

If the FREQ variable is specified in PROC DMDB, it must not be specified in PROC DMSPLIT.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The DMSPLIT Procedure

TARGET Statement

Tip: One or more variables may be specified already in PROC DMDB.

TARGET variable;

Required Argument

variable

Specifies the target variable. One variable name can be specified identifying the target (response) variable for the least squares and logistic regressions.

CAUTION:

If a target is specified in PROC DMDB, it must not be specified in PROC DMSPLIT.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
VARIABLE Statement

Alias: VAR

VARIABLE variable-list;

Required Argument

variable-list

Specifies all the variables (numeric and categorical, that is, INTERVAL and CLASS) that can be used for independent variables in the prediction or modeling of the target variable.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
WEIGHT Statement

### Alias:
**WEIGHTS**

### Tip:
Specify the WEIGHT variable in PROC DMDB so that the information is saved in the catalog and so that the variable is used automatically as a WEIGHT variable in PROC DMSPLIT.

```
WEIGHT variable;
```

### Required Argument

**variable**

Specifies one numeric (interval scaled) variable that is used to weight the input variables.

**CAUTION:**
If the WEIGHT variable is specified in PROC DMDB, it must not be specified in PROC DMSPLIT.

---

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Details

Missing Values

For numeric variables, missing values are replaced by the (weighted) mean of the variable. For categorical (CLASS) variables, missing values are treated as an additional category.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Examples

The following examples were executed on the Windows NT operating system; the version of the SAS System was 6.12TS045.

Example 1: Creating a Decision Tree for a Binary Target with the DMSPLIT Procedure

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Example 1: Creating a Decision Tree for a Binary Target with the DMSPLIT Procedure

Features:

- Specifying the target and input variables.
- Setting the number of categories in which the range of each interval variable is divided for splits.
- Setting the number of passes the procedure makes to determine the optimum number of splits.
- Setting the chi-square lower bound for evaluating the splits.
- Importing the DMSPLIT tree to the SPLIT procedure.
- Producing summary statistics for the training data.
- Saving the decision tree from within PROC SPLIT.
- Scoring/validating with a test data set.

As a marketing analyst at a catalog company, you want to determine the inputs that best predict whether or not a customer will make a purchase from your new fall outerwear catalog. The fictitious catalog mailing data set is named SAMSPO.DMEXA1 (stored in the sample library). The data set contains 1,966 customer cases. The binary target (PURCHASE) contains a formatted value of "Yes" if a purchase was made and a formatted value of "No" if a purchase was not made.

Although there are 48 input variables available for predicting the target, only 17 inputs are used to construct the tree. Note that AMOUNT is an interval target and ACCTNUM is an id variable; these variables are not suitable model inputs.

To demonstrate how to score a data set, a sample of customers is selected from the SAMSPO.DMEXA1 training data set.

Program

```plaintext
proc dmdb batch data=sampsio.dmexa1 out=dmbexa1 dmdbcat=catexa1;
  id acctnum;
  var  amount income homeval frequent recency age
       domestic apparel;
  class purchase(desc) marital ntitle gender telind
       origin job statecod numcars edlevel;
run;

proc dmsplit data=dmbexa1 dmdbcat=catexa1
  bins=30
  chisq=2.00
```
passes=20

outvars=vout;

var amount income homeval frequent recency age
domestic apparel marital ntitle gender telind origin
job statecod numcars edlevel;

target purchase;
title 'DMSPLIT: Binary Target';
run;

proc print data=vout(obs=20);
title2 'OUTVARS= Summary Data';
run;

title 'Import and Save Tree from DMSPLT';
proc split dmdbcat=catexa1 indmsplit

    outmatrix=trtree

    outleaf=leafdata

    outtree=savetree;
run;

proc print data=trtree label;
title2 'Training Statistics';
run;

proc print data=leafdata(obs=10) label;
title2 'Leaf Statistics';
run;
data testexa1(drop=ran);
    set sampsio.dmexa1;
    ran=ranuni(3333);
    if ran lt 0.08;
        title 'Create Fictitious Score Data Set';
    run;

proc split intree=savetree;

score data=testexa1 nodmdb
    outfit=tfit out=tout;

    title 'Input Tree and Score Test Data';

proc print data=tfit label;
    title2 'Fit Statistics for the Scored Data Set';
run;

proc freq data=tout;
    tables f_purcha*i_purcha;
    title2 'Scored Data';
    title3 'Misclassification Table';
run;

proc print data=tout(obs=10) label;
    var _node_ a a_yes a_no d_purcha f_purcha
        i_purcha p_puryes p_purno p_pur r_puryes
        r_purno r_pur;
    title2 'Score Summary Data';
run;

---

**PROC DMSPLIT Output**

**Partial Listing of the Splitting Table**

The splitting table contains the following information for each split:

- node number
- parent node
- chi-square value for the split
- splitting variable
- the average of the splitting variable if it is an interval input, or the number of levels if the splitting variable is
non-interval.

### DMSPLIT: Binary Target

<table>
<thead>
<tr>
<th>Node</th>
<th>Parent</th>
<th>ChiSq</th>
<th>Split</th>
<th>Value</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>92.34</td>
<td>FREQUENT</td>
<td>2.36</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>29.48</td>
<td>STATECOD</td>
<td>.</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>30.80</td>
<td>DOMESTIC</td>
<td>3.20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>11.58</td>
<td>JOB</td>
<td>.</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>13.46</td>
<td>STATECOD</td>
<td>.</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>30.88</td>
<td>STATECOD</td>
<td>.</td>
<td>35</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>29.79</td>
<td>STATECOD</td>
<td>.</td>
<td>21</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>7.28</td>
<td>JOB</td>
<td>.</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>9.90</td>
<td>STATECOD</td>
<td>.</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>7.03</td>
<td>HOMEVAL</td>
<td>40000.00</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>6.56</td>
<td>HOMEVAL</td>
<td>240000.00</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>8.29</td>
<td>STATECOD</td>
<td>.</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>8.83</td>
<td>JOB</td>
<td>.</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>7.16</td>
<td>STATECOD</td>
<td>.</td>
<td>17</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>11.85</td>
<td>APPAREL</td>
<td>1.67</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>6.06</td>
<td>HOMEVAL</td>
<td>20000.00</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>9</td>
<td>3.00</td>
<td>AGE</td>
<td>34.60</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>11.12</td>
<td>STATECOD</td>
<td>.</td>
<td>5</td>
</tr>
<tr>
<td>21</td>
<td>10</td>
<td>5.03</td>
<td>JOB</td>
<td>.</td>
<td>10</td>
</tr>
<tr>
<td>22</td>
<td>11</td>
<td>6.42</td>
<td>FREQUENT</td>
<td>2.29</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>12</td>
<td>6.16</td>
<td>JOB</td>
<td>.</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>12</td>
<td>9.15</td>
<td>EDLEVEL</td>
<td>.</td>
<td>2</td>
</tr>
<tr>
<td>27</td>
<td>13</td>
<td>6.58</td>
<td>AMOUNT</td>
<td>1328.50</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>14</td>
<td>7.98</td>
<td>INCOME</td>
<td>14520.00</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>15</td>
<td>10.10</td>
<td>STATECOD</td>
<td>.</td>
<td>13</td>
</tr>
<tr>
<td>31</td>
<td>15</td>
<td>15.13</td>
<td>JOB</td>
<td>.</td>
<td>8</td>
</tr>
<tr>
<td>32</td>
<td>17</td>
<td>11.00</td>
<td>RECENCY</td>
<td>789.37</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>17</td>
<td>4.56</td>
<td>STATECOD</td>
<td>.</td>
<td>4</td>
</tr>
</tbody>
</table>
Effect Summary Table

The Effect Summary table lists the node in which the effect was first split and the total number of times a split occurred for the effect.

<table>
<thead>
<tr>
<th>Effect</th>
<th>Node 1st Split</th>
<th>Total Times Split</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREQUENT</td>
<td>1</td>
<td>47</td>
</tr>
<tr>
<td>STATECOD</td>
<td>2</td>
<td>64</td>
</tr>
<tr>
<td>DOMESTIC</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>JOB</td>
<td>4</td>
<td>35</td>
</tr>
<tr>
<td>HOMEVAL</td>
<td>10</td>
<td>23</td>
</tr>
<tr>
<td>APPAREL</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>AGE</td>
<td>18</td>
<td>44</td>
</tr>
<tr>
<td>EDLEVEL</td>
<td>25</td>
<td>12</td>
</tr>
<tr>
<td>AMOUNT</td>
<td>27</td>
<td>26</td>
</tr>
<tr>
<td>INCOME</td>
<td>29</td>
<td>37</td>
</tr>
<tr>
<td>RECENCY</td>
<td>32</td>
<td>39</td>
</tr>
<tr>
<td>TELIND</td>
<td>38</td>
<td>4</td>
</tr>
<tr>
<td>NTITLE</td>
<td>41</td>
<td>29</td>
</tr>
<tr>
<td>MARITAL</td>
<td>44</td>
<td>12</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>65</td>
<td>19</td>
</tr>
<tr>
<td>NUMCARS</td>
<td>193</td>
<td>10</td>
</tr>
<tr>
<td>GENDER</td>
<td>776</td>
<td>2</td>
</tr>
</tbody>
</table>

Partial Listing of the OUTVARS Data Set

DMSPLIT: Binary Target
OUTVARS= Summary Data
PROC SPLIT Output

PROC PRINT Report of the Training Data Fit Statistics (OUTMATRIX=)

The report consists of the classification counts and proportions for the buyers ("Yes") and the non-buyers ("No"). You can interpret the first two rows of the table as follows:

- 994 of the 999 actual buyers were correctly classified; only 5 buyers were incorrectly classified as non-buyers
- 959 of the 967 non-buyers were correctly classified; only 8 non-buyers were incorrectly classified as buyers.

The values in the STAT column enable you to identify the rows that pertain to counts (N), row and column percentages (Row% and Col%), and overall percentages (%).
Partial PROC PRINT Report of the Leaf Statistics (OUTLEAF=)

The leaf report contains the following information:

- Leaf identification number
- Number of customers in each leaf
- Percentages of the binary target values in each leaf.

Notice the purity of the leaf nodes.

<table>
<thead>
<tr>
<th>LEAF</th>
<th>OBS</th>
<th>ID</th>
<th>N</th>
<th>% Yes</th>
<th>% No</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>16</td>
<td>16</td>
<td>9</td>
<td>100.00</td>
<td>0.00</td>
</tr>
<tr>
<td>12</td>
<td>54</td>
<td>54</td>
<td>10</td>
<td>100.00</td>
<td>0.00</td>
</tr>
<tr>
<td>13</td>
<td>55</td>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>100.00</td>
</tr>
<tr>
<td>14</td>
<td>142</td>
<td>2</td>
<td>2</td>
<td>100.00</td>
<td>0.00</td>
</tr>
<tr>
<td>15</td>
<td>143</td>
<td>2</td>
<td>100.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>89</td>
<td>8</td>
<td>100.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>332</td>
<td>2</td>
<td>100.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>333</td>
<td>8</td>
<td>100.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>416</td>
<td>12</td>
<td>100.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>417</td>
<td>1</td>
<td>0.00</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>

PROC PRINT Report of the Score Fit Statistics (OUTFIT=)

The misclassification rate for the scored data set is almost zero. You can compare the maximum absolute error, sum of squared errors, average squared error, and root average squared error from this tree with other candidate trees (models). Small values for these test statistics are preferred.

<table>
<thead>
<tr>
<th>OBS</th>
<th>Test: Frequency of Classified Cases</th>
<th>Test: Frequency of Unclassified Cases</th>
<th>Test: Misclassification Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>150</td>
<td>300</td>
<td>.0066667</td>
</tr>
<tr>
<td></td>
<td>Test: Sum of Weights Times Freqs</td>
<td>Test: Sum of Classfied Cases</td>
<td>Test: Root Average</td>
</tr>
<tr>
<td>OBS</td>
<td>Absolute Error</td>
<td>of Squared Errors</td>
<td>Squared Error</td>
</tr>
<tr>
<td>-----</td>
<td>----------------</td>
<td>--------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>1</td>
<td>0.72727</td>
<td>1.52883</td>
<td>0.050961</td>
</tr>
</tbody>
</table>

**Misclassification Table for the Scored Data Set (OUT=)**

Only one customer in the test data set was incorrectly classified. Ideally, you should use a mutually exclusive test data set for validating the tree.

```
Input Tree and Score Test Data
Misclassification Table for the Test Data
TABLE OF F_PURCHA BY I_PURCHA

<table>
<thead>
<tr>
<th>F_PURCHA(Formatted Target Value)</th>
<th>I_PURCHA(Predicted Category)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>Percent</td>
</tr>
<tr>
<td>No</td>
<td>64</td>
</tr>
<tr>
<td>Yes</td>
<td>85</td>
</tr>
<tr>
<td>Total</td>
<td>64</td>
</tr>
</tbody>
</table>
```

```
Partial PROC PRINT Report of the Score Summary Data Set
```

```
Score Summary Data

<table>
<thead>
<tr>
<th>OBS</th>
<th>Node Identification</th>
<th>Assesment of Prediction</th>
<th>Assessment: PURCHASE = Yes</th>
<th>Assessment: PURCHASE = No</th>
<th>Decision Assigned to Case</th>
<th>Formatted Target Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>0.58527</td>
<td>0.41473</td>
<td>0.58527</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>0.58527</td>
<td>0.41473</td>
<td>0.58527</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>128</td>
<td>0.89474</td>
<td>0.89474</td>
<td>0.10526</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>0.58527</td>
<td>0.41473</td>
<td>0.58527</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>128</td>
<td>0.89474</td>
<td>0.89474</td>
<td>0.10526</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>OBS</td>
<td>Category</td>
<td>Predicted PURCHASE</td>
<td>Residual PURCHASE</td>
<td>Predicted PURCHASE</td>
<td>Residual PURCHASE</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>----------</td>
<td>--------------------</td>
<td>-------------------</td>
<td>--------------------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>No</td>
<td>0.41473</td>
<td>-0.41473</td>
<td>0.41473</td>
<td>-0.41473</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>0.41473</td>
<td>-0.41473</td>
<td>0.41473</td>
<td>-0.41473</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
<td>0.89474</td>
<td>-0.10526</td>
<td>0.89474</td>
<td>-0.10526</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>0.41473</td>
<td>-0.41473</td>
<td>0.41473</td>
<td>-0.41473</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Yes</td>
<td>0.89474</td>
<td>-0.10526</td>
<td>0.89474</td>
<td>-0.10526</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>0.41473</td>
<td>-0.41473</td>
<td>0.41473</td>
<td>-0.41473</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>0.53237</td>
<td>-0.46763</td>
<td>0.53237</td>
<td>-0.46763</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Yes</td>
<td>1.00000</td>
<td>0.00000</td>
<td>1.00000</td>
<td>0.00000</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Yes</td>
<td>0.53237</td>
<td>-0.46763</td>
<td>0.53237</td>
<td>-0.46763</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td></td>
</tr>
</tbody>
</table>

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Before you analyze the data using the DMSPLIT procedure, you must create the DMDB encoded data set and catalog. For more information about how to do this, see "Example 1: Getting Started with the DMDB Procedure" in the DMDB procedure documentation.

```sas
proc dmdb batch data=sampsio.dmexa1 out=dmbexa1 dmdbcat=catexa1;
   id acctnum;
   var  amount income homeval frequent recency age
do+
   domestic apparel;
   class purchase(desc) marital ntitle gender telind
      origin job statecod numcars edlevel;
run;
```
The PROC DMSPLIT statement invokes the procedure. The DATA= option identifies the DMDB encoded training data set that is used to fit the model. The DMDBCAT= option identifies the DMDB training data catalog.

```
proc dmsplit data=dmbexa1 dmdbcat=catexa1
```
The BINS= option specifies the number of categories in which the range of each interval variable is divided for splits.

bins=30
The CHISQ= option specifies a minimum bound for the Chi-Square value that is still eligible for making a variable split. The value of CHISQ governs the number of splits that are performed. As you increase the CHISQ value, the procedure performs fewer splits and passes through the input data.

chisq=2.00
The PASSES = option specifies an upper bound for the number of passes that are made through the data.

\[ \text{passes}=20 \]
The OUTVARS = option creates a data set containing splitting information.

```
outvars=vout;
```
The VAR statement specifies the numeric and categorical inputs (independent variables).

```
var amount income homeval frequent recency age
domestic apparel marital ntitle gender telind origin
job statecod numcars edlevel;
```
The TARGET statement defines the target (response) variable.

target purchase;
title 'DMSPLIT: Binary Target';
run;
PROC PRINT creates a partial report of the OUTVARS= data set.

```
proc print data=vout(obs=20);
    title2 'OUTVARS= Summary Data';
run;
```
The PROC SPLIT statement invokes the procedure. The INDMSPLT option specifies to read the tree created from PROC DMSPLIT. The DMSPLIT tree information is stored in the DMDB catalog.

title 'Import and Save Tree from DMSPLT';
proc split dmdbcat=catexa1 indmsplit
The OUTTREE= option names the data set that contains tree information.

```
outmatrix=trtree
```
The OUTLEAF= option names the data set that contains statistics for each leaf node.

outleaf=leafdata
The OUTTREE= option specifies the output data set that describes the tree. The OUTTREE data set can be used as input in subsequent executions of PROC SPLIT.

```
outtree=savetree;
run;
```
PROC PRINT creates a report of the training statistics.

proc print data=trtree label;
    title2 'Training Statistics';
run;
PROC PRINT creates a partial report of the leaf statistics for the training data.

proc print data=leafdata(obs=10) label;
   title2 'Leaf Statistics';
run;
The DATA step creates a fictitious score data set.

data testexa1(drop=ran);
    set sampsio.dmexa1;
    ran=ranuni(3333);
    if ran lt 0.08;
    title 'Create Fictitious Score Data Set';
run;
The INTREE = option reads the tree that was saved from the previous PROC SPLIT step.

```
proc split intree=savetree;
```
The SCORE statement scores the DATA= data set. The OUTFIT= option names the output data set containing fit statistics. The OUT= option names the output data set that contains tree statistics for the scored data set. Typically you would want to score a truly mutually exclusive data set that may or may not contain the target values (the WORK.TESTEXA1 data set is a random subset of the SAMPSIO.DMEXA1 training data set).

```plaintext
score data=testexa1 nodmdb
        outfit=tfit out=tout;

title 'Input Tree and Score Test Data';
```
PROC PRINT creates a report of the fit statistics for the scored data set.

proc print data=tfit label;
   title2 'Fit Statistics for the Scored Data Set';
run;
PROC FREQ creates a misclassification table for the scored data set. The F_PURCHA variable is the actual target value for each customer and the I_PURCHA variable is the target value into which the customer is classified.

```sas
proc freq data=tout;
  tables f_purcha*i_purcha;
  title2 'Scored Data';
  title3 'Misclassification Table';
run;
```
PROC PRINT creates a partial report of selected variables from the OUT= score information data set.

```
proc print data=tout(obs=10) label;
  var _node_ a_ a_yes a_no d_purcha f_purcha
     i_purcha p_puryes p_purno p_pur r_puryes
     r_purno r_pur;
  title2 'Score Summary Data';
run;
```
The EMCLUS Procedure

Overview
Procedure Syntax
   PROC EMCLUS Statement
   VAR Statement
   INITCLUS Statement
Output from PROC EMCLUS
EXAMPLES-SECTION
   Example 1: Syntax for PROC FASTCLUS
   Example 2: Use of the EMCLUS Procedure

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The EMCLUS Procedure

Overview

The EMCLUS procedure uses a scalable version of the Expectation-Maximization (EM) algorithm to cluster a data set. You have the option to run the standard EM algorithm, in which the parameters are estimated using the entire data set, or the scaled EM algorithm, in which a portion of the data set is used to estimate the parameters at each iteration of the procedure. The standard EM algorithm is run by not specifying the a value for the option NOBS, and can be run provided the entire data set. The entire data set must fits in memory such that total number of observation can be determined. The scaled EM algorithm can be run by specifying a value for the NOBS option that is less than the total number of observations in the data set. When the scaled EM algorithm is used, it is important that the input data set is randomized beforehand.

The EMCLUS procedure identifies primary clusters, which are the densest regions of data points, and also identifies secondary clusters, which are typically smaller dense clusters. Each primary cluster is modeled with a weighted n-dimensional multivariate normal distribution, where n is the number of variables in the data set. Thus each primary cluster is modeled with the function \( w \cdot f(x | u, V) \), where \( f(x | u, V) \sim MVN(u, V) \) and V is a diagonal matrix. There are four major parts in the EMCLUS procedure:

- Obtain and possibly refine the initial parameter estimates.
- Apply the EM algorithm to update parameter values.
- Summarize data in the primary summarization phase.
- Summarize data in the secondary summarization phase.

The effectiveness of the EMCLUS procedure depends on the initial parameter estimates. Good initial parameter estimates generally heads to faster convergence and better final estimates. The initial parameter estimates can be obtained "randomly", from using PROC FASTCLUS, or from using PROC EMCLUS. See Example 1 for the PROC FASTCLUS syntax. PROC FASTCLUS sometimes returns poor results (clusters corresponding to low frequency counts). In this case, the poor results can be ignored by specifying appropriate values for the INITCLUS option. PROC FASTCLUS can also return clusters that are actually groups of clusters. This can be determined by clusters having a large frequency count and a large root-mean-square standard deviation. In this case, these clusters should not be ignored, and the user should specify a value of INITSTD which is smaller than the root-mean-square standard deviations of the clusters with the large frequency counts and large root-mean-square standard deviations (see Example 2). In the case when PROC FASTCLUS returns poor results, it may be of help to rerun PROC FASTCLUS with a larger number of MAXCLUSTERS, and then choose the best clusters for the initial values for PROC EMCLUS. Initial estimates obtained from PROC EMCLUS can be used to refine the primary clusters or obtain better primary clusters.

The EM algorithm is used to find the primary clusters, and update the model parameters. The EM algorithm terminates when two successive log-likelihood values differ in relative and absolute magnitude by a particular amount or when ITER iterations have been reached.

The primary summarization phase summarizes observations near each of the primary cluster means and
then deletes the summarized observations from memory. For the specified value \( p = p_0 \), all observations falling within the region containing \( 100 \times p_0 \% \) of the volume of the \( \text{MNV}(u,V) \) distribution will be summarized. At then end of the primary summarization phase, the primary clusters are checked to see if any of them contain fewer than \( \text{MIN} \) observations. If a cluster does, then the cluster is declared to be inactive. An inactive cluster is not used in updating the parameter estimates in the EM algorithm. An inactive cluster remains inactive until one of the following two conditions occur:

1. The inactive cluster gets reseeded at a secondary cluster containing at least \( \text{MIN} \) observations.
2. The inactive cluster gets reseeded at point determined by an active cluster having at least one variable with standard deviation greater than \( \text{INITSTD} \).

The secondary summarization phase first uses the k-means clustering algorithm to identify secondary clusters, and then uses a hierarchical agglomerative clustering algorithm to combine similar secondary clusters. At the end of the k-means algorithm, each of the SECCLUS clusters are tested to see if their sample standard deviation for each variable is less than or equal to \( \text{SECSTD} \). If yes, then the cluster becomes a secondary cluster. Setting \( \text{SECCLUS}=0 \) will cause PROC EMCLUS not to perform a secondary summarization phase, which is \textbf{not} recommended. The reason for this is that the secondary summarization phase acts as a backup method for finding the primary clusters when the initial values are poor. If the data set contains many outliers, then setting \( \text{SECCLUS} \) to be larger than the default value will increase the chances of finding clusters. A secondary cluster is disjoint from all other secondary clusters and from all primary clusters.

Although many of the options in PROC EMCLUS are not required to be specified, it is best to specify them if the user has some knowledge of the input data set. Among the most important options is \( \text{NOBS} \). \( \text{NOBS} \) specifies the number of observations that are read in during each iteration, and consequently, \( \text{NOBS} \) determines the number of iterations. For example, if the input data set contains 1,000 observations and \( \text{NOBS} \) is set to 100, then there will be 10 iterations. If \( \text{NOBS} \) is not specified, then it is assumed that you wants to run the standard EM algorithm. Another important option is \( \text{INITSTD} \), the maximum initial standard deviation of each variable in each initial primary cluster. If \( \text{INITSTD} \) is chosen too small, then the EM algorithm may have trouble finding the primary clusters.
PROC EMCLUS <option(s)>;

VAR variable(s);

INITCLUS integer(s);
The EMCLUS Procedure

**PROC EMCLUS Statement**

Invoke the EMCLUS procedure.

```
PROC EMCLUS<option(s)>;
```

**Options**

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

Specifies the data set to be analyzed. All the variables in this data set must be numerical. Observations with missing data are ignored.

**ROLE = TRAIN|SCORE**

Specifies the role of the DATA= data set. Setting ROLE=TRAIN will cause the EMCLUS procedure to cluster the data set. Setting ROLE=SCORE will cause the EMCLUS procedure to compute the probabilities that each observation in the DATA= data set is in each primary cluster. If ROLE=SCORE is specified, then SEED=<file> option must also be used where <file> is the name of the OUTSTAT data set. The default value for ROLE is TRAIN.

**CLUSTERS = positive integer**

Specifies the number of primary clusters.

**SECCLUS = nonnegative integer**

Specifies the number of secondary clusters that the algorithm will search for during the secondary data summarization phase. If SECCLUS=0, then there will not be a secondary data summarization phase. The default value of SECCLUS is twice the number of primary clusters.

**EPS = positive number**

Specifies the stopping tolerance. The default value is $10^{-6}$.

**SECSTD = positive number**

Specifies the maximum allowable sample standard deviation of any variable in a summarized subset of observations to be deemed a secondary cluster. The default value is the smallest positive sample standard deviation obtained from the observations read in during the first iteration.

**P = number**

 Defines a radius around each cluster mean, such that any point which lies inside the radius is summarized in that cluster. The value of P must be between 0 and 1. A value close to 1 defines a larger radius than a value close to 0. The default value is 0.5.

**MAXITER = positive integer**

Specifies the maximum number of iterations of the EMCLUS procedure. The default value is the largest machine integer.

**INITSTD = positive number**

 Specifies the maximum standard deviation of the initial clusters. The default value is determined from a sample of data read in during the first iteration.

**ITER = positive integer**

Specifies the number of iterations in the EM algorithm to update the model parameters. The default value is 50.

**INIT = RANDOM|FASTCLUS|EMCLUS**
Specifies how the initial estimates are obtained. The default value is RANDOM. If INIT=FASTCLUS or INIT=EMCLUS is specified, then the option SEEDS=<libref.SAS-data-set> must also be specified, and the data set must be the OUTSEEDS data set from PROC FASTCLUS or the OUTSTAT data set from PROC EMCLUS, respectively.

MIN = nonnegative integer

Specifies the minimum number of observations in each primary cluster. At any iteration, if the total number of observations summarized in a cluster is less than MIN, then the cluster becomes inactive, and the cluster is reseeded at a more appropriate point, of one exists. The default value is 3.

NOBS = positive integer

Specifies the number of observations to be read in for each iteration. The default value is the number of observations in the data set, provided that this number can be determined. If the number of observations in the data set cannot be determined, the default value is 500.

SEED = libref.SAS-data-set

Specifies the data set that is used for the initial parameter estimates. This option must be used with INIT=FASTCLUS or with INIT=EMCLUS. With PROC FASTCLUS, this specified data set is the resulting SAS data set from the OUTSEEDS option. With PROC EMCLUS, this data set is the resulting SAS data set from the OUTSTAT option.

OUTSTAT = libref.SAS-data-set

Specifies an output data set. This data set has 5+D columns, where D is the number of variables. Column 1 contains the cluster number. Column 2 is the type of cluster (primary or secondary). Column 3 is the cluster frequency. Column 4 is the estimate for the weight parameter. Column 5 is the labelled _TYPE_, where _TYPE_=MEAN or _TYPE_=VAR. Columns 6 through 5+D contain either the estimates for the mean or the variance of each variable. Each variable corresponds to two rows of this data set in the following form:

| CLUSTER_1 | PRIMARY | FREQ_1 | WEIGHT_1 | MEAN | MEAN_1 | MEAN_2 | ..... | MEAN_N |
| CLUSTER_1 | PRIMARY | FREQ_1 | WEIGHT_1 | VAR  | VAR_1  | VAR_2  | ..... | VAR_N |
| CLUSTER_2 | SECONDARY | FREQ_2 | WEIGHT_2 | MEAN | MEAN_1 | MEAN_2 | ..... | MEAN_N |
| CLUSTER_2 | SECONDARY | FREQ_2 | WEIGHT_2 | VAR  | VAR_1  | VAR_2  | ..... | VAR_N |

DIST = nonnegative number

Specifies the minimum distance between initial clusters when INIT=RANDOM is used. It also specifies the minimum distance between initial cluster means in the secondary data summarization phase. The default value is the square root of the average of the sample variances obtained from the observations read in during the first iteration.

PRINT = ALL|LAST|NONE

Specifies how much output are printed. If PRINT=LAST is used, then the initial estimates and the output from the last iteration will be shown. The default is PRINT=LAST.

SECITER = positive integer

Specifies the maximum number of iterations of the k-means algorithm in the secondary data summarization phase. The default value is 1.

OUT = libref.SAS-data-set

Specifies the name of the data set that contains the probabilities PROB_h = P(x is in cluster h), h=1,2,...k. This option must be used with ROLE=SCORE. The resulting data set will also contain the original data.

CLEAR = non-negative integer
Specifies the value of $n$ for which the EMCLUS procedure deletes the observations that are remained in the memory following the secondary summarization phase after every $n$ iterations. The default value is 0, which means that no observation will be deleted from the memory.

OUTLIERS = IGNORE|KEEP

Specifies how the outlier observations are weighted when the scaled EM algorithm is implemented. If OUTLIERS=IGNORE is specified, observation that are not in the 99th percentile of any estimated primary cluster are weighted less. If OUTLIERS=KEEP is specified, these observations are weighted normally as in the standard EM algorithm.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
**VAR Statement**

\[
\text{VAR } \textit{variable(s);} \\
\textit{variable(s)} \\
\text{Specifies which variables are to be used. If this statement is omitted, then all variables from the input data set will be used.}
\]

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
INITCLUS integer(s);

integer(s)

Specifies which clusters from PROC FASTCLUS or PROC EMCLUS are to be used as initial estimates for PROC EMCLUS. For example, when viewing the output from PROC FASTCLUS you see that only clusters 2, 4-8, and 10 have good results, then could use the option INITCLUS 2, 4 TO 8, 10; to only use those cluster estimates as initial estimates in PROC EMCLUS. The number of clusters specified in the INITCLUS statement should not exceed the number of clusters specified with the CLUSTERS= option in PROC EMCLUS statement. The default for INITCLUS with INIT=EMCLUS is that all the clusters will be used as initial estimates. With INIT=FASTCLUS, the default is that the clusters with the highest frequency counts will be used.
Output from PROC EMCLUS

The beginning of the output shows the initial model parameter estimates. Next, the estimated model parameters, sample means, and sample variances for the active primary clusters are displayed. In active clusters are shown with missing values. The sample mean and variance are calculated from the observations that are summarized in the primary clusters.

In the cluster summary table, the following statistics are listed:

**Current Frequency**
the number of observations that are summarized in a cluster during the current iteration.

**Total Frequency**
the cumulative sum of the current frequencies for each cluster.

**Proportion of Data Summarized**
the total frequency divided by the Obs read in.

**Nearest Cluster**
the closest primary cluster to a primary cluster based on the euclidean distance between the estimated mean of the two primary clusters.

**Distance**
the euclidean distance of a primary cluster to its nearest cluster.

The iteration summary table displays:

**Log-likelihood**
the average log-likelihood over all the observations that are read in.

**Obs read in this iteration**
the number of observations that are read in at current iteration.

**Obs read in**
the cumulative sum of observations that are read in.

**Current Summarized**
is the sum of the current frequencies across the primary clusters.

**Total Summarized**
is the sum of the total frequencies across the primary clusters.

**Proportion Summarized**
the Total Summarized divided by the Obs read in.

If there are secondary clusters, the sample mean, sample variance, and the number of observations in secondary clusters are also displayed after the iteration summary table.

**Note:** The estimated variance parameter for each variable is bound from below by the value
(var)*(eps), where var is the sample variance of that variable obtained from the observations read in at the first iteration, and eps is 10^{-6}. Both the standard and scaled EM algorithm sometimes are slow to convergence, however, the scaled EM algorithm generally runs faster than the standard EM algorithm. Convergence may be sped up by increasing p and/or eps, or by using the CLEAR option. Changing these values may alter the parameter estimates.
Example

Example 1: Syntax for PROC FASTCLUS

Example 2: Use of the EMCLUS Procedure
Example 1: Syntax for PROC FASTCLUS

```
PROC EMCLUS <DATA = libref.SAS-data-set>
  OUTSEEDS = libref.SAS-data-set
  MAXCLUSTERS = positive integer;
```

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Example 2: Use of the EMCLUS Procedure

PROC FASTCLUS returns a portion of the total output summarized in the following table:

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Frequency</th>
<th>RMS Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>500</td>
<td>101.34</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1000.34</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>3.79</td>
</tr>
<tr>
<td>4</td>
<td>150</td>
<td>4.05</td>
</tr>
</tbody>
</table>

Clusters 1, 3, and 4 should be used as initial estimates for PROC EMCLUS, because of their high frequency counts. Cluster 1 may actually be a group of clusters because of its high RMS Std. Deviation. Therefore, the syntax when using PROC EMCLUS could look like:

```sas
PROC EMCLUS DATA=<libref.SAS-data-set>
   CLUSTERS = 5
   INIT = FASTCLUS
   SEED = <the data set from OUTSEEDS option in PROC FASTCLUS>
   INITSTD = 50.0
   INITCLUS 1, 3, 4;
run;
```

Note that CLUSTERS is set to 5, but any integer greater than or equal to 3 is appropriate since there are three clusters specified in the INITCLUS option. Also the INITSTD could have been set to any number less than 101.34 and greater than 4.05.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The NEURAL Procedure

Overview

Procedure Syntax
- PROC NEURAL Statement
- ARCHITECTURE Statement
- CODE Statement
- CONNECT Statement
- CUT Statement
- DECISION Statement
- DELETE Statement
- FREEZE Statement
- FREQ Statement
- HIDDEN Statement
- INITIAL Statement
- INPUT Statement
- NETOPTIONS Statement
- NLOPTIONS Statement
- PERTURB Statement
- PRELIM Statement
- QUIT Statement
- RANOPTIONS Statement
- SAVE Statement
- SCORE Statement
- SET Statement
- SHOW Statement
- TARGET Statement
- THAW Statement
- TRAIN Statement
- USE Statement
- ACTIVATION FUNCTIONS
COMBINATION FUNCTIONS

Details

Examples

Example 1: Developing a Simple Multilayer Perceptron (Rings Data)
Example 2: Developing a Neural Network for a Continuous Target
Example 3: Neural Network Hill-and-Plateau Example (Surf Data)

References

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Overview

The NEURAL procedure trains a wide variety of feedforward neural networks using proven statistical methods and numerical algorithms. Please read the chapters "Introduction to Predictive Modeling" and "Neural Network Node: Reference" before reading this chapter.

Terminology and Architectural Description

Each INPUT, HIDDEN and TARGET statement defines a "layer". For the INPUT statement, a layer is a convenient grouping of variables, serving as inputs to the network, having common values for LEVEL and STD. Similarly, for the TARGET statement, a layer is a convenient grouping of variables, serving as outputs of the network, having common values for LEVEL, STD, ACTIVATION, COMBINATION, and other characteristics. Each layer is made up of "units". A unit is synonymous with the term "neuron" in the literature. It is the smallest computational entity in the network.

The INPUT and TARGET statements require a list of variables. If the variables are interval type, there is one unit corresponding to each variable. If the variables are nominal or ordinal, there is a unit for each level of each variable. The HIDDEN statement requires a number which determines the number of units in the associated layer. This layer is a grouping of units having common values for ACTIVATION, COMBINATION, and other characteristics.

Each INPUT statement produces an input layer. Because multiple INPUT statements are allowed, a network can have multiple input layers. However, connections from multiple input layers must be parallel; there cannot be serial connections between input layers. Similarly, multiple TARGET statements generate multiple output layers. The connections to multiple output layers must be in parallel; there cannot be serial connections between output layers. Hidden layers can be connected serially or in parallel.

Running the Neural Procedure

Before running the NEURAL procedure, you must run the DMDB procedure to create a DMDB-encoded training set and a DMDB catalog entry.

A typical application of the NEURAL procedure uses the following statements:

- A PROC NEURAL statement to specify the training set, DMDB catalog, and random number seed. If you specify a 0 or negative random number seed, running the same NEURAL step repeatedly may produce different results because of different random initial weights.
- One or more INPUT statements to specify input variables.
- One or more HIDDEN statements to define hidden layers.
- One or more TARGET statements to specify target variables.
- One or more CONNECT statements to connect layers in the network.
● A PRELIM statement to do preliminary training to avoid bad local optima.
● A TRAIN statement to train the network.
● One or more SCORE statements to create output data sets.

Two kinds of statements are used with the NEURAL procedure. All of the statements in the list above are action statements, which directly affect the network or directly produce output. There are also option statements (NETOPTIONS, NLOPTIONS, and RANOPTIONS) that set options for future use. Options specified in an action statement apply only to that statement and do not affect subsequent statements. For example, the default technique for least-squares training is Levenberg-Marquardt (TECH=LEVMAR). If you execute a TRAIN statement with the option TECH=CONGRA, conjugate gradient training will be used for that particular training run. If you then execute another TRAIN statement without a TECH= option, the technique will revert to the default value of TECH=LEVMAR. But if you submit an NLOPTIONS statement with TECH=CONGRA, conjugate gradient training will be used for all subsequent TRAIN statements until you explicitly specify a different technique.

Each layer in the network has an identifier specified by the ID= option in the INPUT, HIDDEN, or TARGET statements. An identifier can be any SAS name, but to avoid confusion, you should not use the name of a variable in the training set. Layer identifiers are used in various statements, such as CONNECT, to specify previously defined layers.

Each unit in the network has a name. For units corresponding to interval variables in input or output layers, the name of the unit is the same as the name of the variable. For units corresponding to dummy variables for categorical (nominal or ordinal) inputs or targets, the name of each unit is constructed by concatenating the name of the input or target variable with the value of the category, truncating as necessary to make the length of the name eight characters or less. For hidden units, the names are constructed by concatenating the layer ID with an integer.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The NEURAL Procedure

Procedure Syntax

PROC NEURAL <option-list>;

ARCHITECTURE architecture-name
<HIDDEN=n>
<DIRECT>;

CODE FILE=file-name
<FORMAT=format>
<RESIDUAL|NORESIDUAL>
<ERROR|NOERROR>
<GROUP=name>;

CONNECT id-list/
<RANDIST=name>
<RANLOC=number>
<RANSCALE=number>;

CUT id-list| ALL;

DECISION DECDATA=<libref.>SAS-data-set
<DECVARS=decision-variable(s)> <option(s)>;

DELETE id-list| ALL;

FREEZE weight-list /
<VALUE=number>
<EST=<libref.>SAS-data-set>;

FREQ variable(s);

HIDDEN integer /
ID=name
<ACT=keyword>
<BIAS|NOBIAS>
<COMBINE=keyword>;}
INITIAL INEST=<libref.>SAS-data-set
OUTEST=<libref.>SAS-data-set
<BIADJUST=adjustment-value>
<INFAN=number>
<RANDBIAS|NORANDBIAS>
<RANDOUT|NORANDOUT>
<RANDOM=integer>
<RANDSCALE | NORANDSCALE>;

INPUT variable-list /
ID=name
<LEVEL=value>
<STD=method>;

NETOPTIONS network-option(s);

NLOPTIONS <nonlinear-options>;

PERTURB weight-list /
OUTEST=<libref.>SAS-data-set
DF=number
<RANDIST=name>
<RANDOM=integer>
<RANLOC=number>
<RANSCALE=number>;

PRELIM integer
INEST=<libref.>SAS-data-set
OUTEST=<libref.>SAS-data-set
<ACCELERATE=number>
<DECELERATE=number>
<LEARN=number>
<MAXLEARN=number>
<MAX | MAXMOMENTUM=number>
<MINLEARN=number>
<MOM | MOMENTUM=number>
<preiter=integer>
<pretech=name>
<pretime=number>
<RANDBIAS|NORANDBIAS>
<RANDOUT|NORANDOUT>
<RANDOM=integer>;

QUIT;
RANOPTIONS connection-list /
<RANDIST=name>
<RANDOM=integer>
<RANLOC=number>
<RANSCALE=number>;

SAVE OUTEST=<libref.>SAS-data-set
NETWORK=screen-specification;

SCORE DATA=<libref.>SAS-data-set
OUT=<libref.>SAS-data-set
OUTFIT=<libref.>SAS-data-set
<DUMMIES|NODUMMIES>
<ROLE=role-option>;

SET weight-list number;

SHOW weights;

TARGET variable-list /
<ACT=keyword>
<BIAS|NOBIAS>
<COMBINE=keyword>
<ERROR=keyword>
<ID=name>
<LEVEL=value>
<MESTA=number>
<MESTCON=number>
<SIGMA=number>
<STD=method>
<WEIGHT=number>;

THAW weight-list;
TRAIN OUT=<libref.>SAS-data-set
OUTEST=<libref.>SAS-data-set
OUTFIT=<libref.> SAS-data-set
<ACCEL|ACCELERATE=number>
<DECEL|DECELERATE=number>
<DECAY=number>
<DUMMIES|NODUMMIES>
<ESTITER=i>
<LEARN=number>
<MAX|MAXMOMENTUM=number>
<MAXITER=integer>
<MAXLEARN=number>
<MAXTIME =number>
<MINLEARN=number>
<MOM|MOMENTUM=number >
<TECHNIQUE=name>;

USE <libref.>SAS-data-set;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The NEURAL Procedure

PROC NEURAL Statement

Invokes the NEURAL procedure.

```sas
PROC NEURAL <option-list>;
```

Required Arguments

**DATA=**<libref.>SAS-data-set
- Specifies the DMDB-encoded input SAS data set containing the training data.

**DMDBCAT=**<libref.>SAS-catalog
- Specifies the DMDB catalog.

Options

**GRAPH**
- Plots the objective function, validation error, and test error during training.

**NETWORK=screen-specification**
- Constructs a network according to a description that was saved by using a SAVE statement during a previous execution of the NEURAL procedure. `screen-specification` is the catalog entry that was specified in the SAVE statement.

- Default: None

**RANDOM=integer**
- Specifies the random number seed used in network weight initialization.

- Default: 12345

**CAUTION:**
- The weights and predicted outputs from the network cannot be reproduced when a 0 or negative RANDOM= value is specified. When a 0 or negative value is specified, the system clock is used to generate the seed. The actual value of this seed will be unavailable, and you lose control over the initialization of weights. Different initializations will result in different final weights and predicted outputs for repeated runs of the same set of NEURAL statements and same input data sets.

**STOPFILE=’file path name’**
- This option enables you to stop the NEURAL training when you are running a large job. Before
you invoke the NEURAL procedure, specify the file path name in the STOPFILE option. For example, STOPFILE= "c:\mydir\haltneural". Initially, this file should not exist. The NEURAL procedure checks for the existence of this file between iterations in the training process. When you want to stop the job, create the specified file, and the NEURAL procedure will halt the training at the current iteration. The file does not have to contain any contents.

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

Specifies a data set used to compute the test average error during training. At selected iterations (controlled by ESTITER=), each observation in the TESTDATA= data set is read in, scored using the current network weight values, and the error computed. The average test error is then output to the OUTEST= data set.

**Note:** This requires the TESTDATA= data set to contain the inputs and target variables.

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

Specifies a data set used to compute the validation average error during training. At selected iterations (controlled by ESTITER=), each observation in the VALIDATA= data set is read in, scored using the current network weight values, and the error computed. The average validation error is then output to the OUTEST= data set.

**Note:** This requires the VALIDATA= data set to contain the inputs and target variables.
ARCHITECTURE Statement

Constructs a network with 0 or 1 hidden layers, sets the hidden-unit ACT= and COMBINE= options, and sets default values for various other options as described below.

Interaction:

You **cannot** override the hidden-unit ACT= and COMBINE= options implied by the ARCHITECTURE statement, because these are what define the architecture. You **can** override all the other values set by ARCHITECTURE by using an INPUT, a HIDDEN, a TARGET, or a RANOPTIONS statement.

Alias: ARCH

ARCHITECTURE *architecture-name*

<HIDDEN=n>
<DIRECT>;

Required Arguments

*architecture-name*

Names the architecture you want to use to construct the network. Only one *architecture-name* from the following list can be specified:

- **GLIM** Requests a Generalized Linear Model.
- **MLP** Requests a Multilayer Perceptron.
- **ORBFEQ** Requests an Ordinary Radial Basis Function Network with Equal Widths.
- **ORBFUN** Requests an Ordinary Radial Basis Function Network with Unequal Widths.
- **NRBFEQ** Requests a Normalized Radial Basis Function Network with Equal Widths and Heights.
- **NRBFEH** Requests a Normalized Radial Basis Function Network with Equal Heights and Unequal Widths.
- **NRBFEW** Requests a Normalized Radial Basis Function Network with Unequal Heights and Equal Widths.
- **NRBFEV** Requests a Normalized Radial Basis Function Network with Equal Volume.
- **NRBFUN** Requests a Normalized Radial Basis Function Network with Unequal Heights and Unequal Widths.
**Note:** See the following two tables for INPUT, TARGET, and HIDDEN options implied by architecture name and RANOPTIONS implied by architecture name.

**INPUT, TARGET, and HIDDEN Options Implied by Architecture Name**

<table>
<thead>
<tr>
<th>ARCHITECTURE NAME</th>
<th>INPUT Options</th>
<th>TARGET Options</th>
<th>HIDDEN Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLIM</td>
<td>STD=NONE</td>
<td>STD=NONE</td>
<td>No hidden layers</td>
</tr>
<tr>
<td>MLP</td>
<td>ACT=TANH</td>
<td>COMBINE=LINEAR</td>
<td></td>
</tr>
<tr>
<td>ORBFEQ</td>
<td>ACT=EXP</td>
<td>COMBINE=EQRADIAL</td>
<td></td>
</tr>
<tr>
<td>ORBFUN</td>
<td>ACT=EXP</td>
<td>COMBINE=EHRADIAL</td>
<td></td>
</tr>
<tr>
<td>NRBFEQ</td>
<td>NOBIAS</td>
<td>ACT=SOFTMAX</td>
<td>COMBINE=EQRADIAL</td>
</tr>
<tr>
<td>NRBFEH</td>
<td>NOBIAS</td>
<td>ACT=SOFTMAX</td>
<td>COMBINE=EHRADIAL</td>
</tr>
<tr>
<td>NRBFEW</td>
<td>NOBIAS</td>
<td>ACT=SOFTMAX</td>
<td>COMBINE=EWRADIAL</td>
</tr>
<tr>
<td>NRBFEV</td>
<td>NOBIAS</td>
<td>ACT=SOFTMAX</td>
<td>COMBINE=EVRADIAL</td>
</tr>
<tr>
<td>NRBFUN</td>
<td>NOBIAS</td>
<td>ACT=SOFTMAX</td>
<td>COMBINE=XRADIAL</td>
</tr>
</tbody>
</table>

The following definitions apply to the table above:

- **fan_in** specifies the fan_in of a hidden unit, that is, the number of non-bias and non-altitude weights feeding into the unit.
- **n_hidden_units** is the number of hidden units.
- **defloc** is $2 \times \max(0.1, (n\_hidden\_units^{\frac{1}{fan\_in}}))$. 
ranloc is the value of the RANLOC= option.

### RANOPTIONS Implied by Architecture Name

<table>
<thead>
<tr>
<th>ARCHITECTURE NAME</th>
<th>RANOPTIONS for BIAS -&gt; HIDDEN Weights</th>
<th>RANOPTIONS for INPUT -&gt; HIDDEN Weights (not affected by early stopping)</th>
<th>RANOPTIONS for ALTITUDE -&gt; HIDDEN Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLIM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORBFEQ</td>
<td>RANLOC=defloc</td>
<td>RANSCALE=1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RANSCALE=ranloc*.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORBFUN</td>
<td>RANLOC=defloc</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RANSCALE=ranloc*5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NRBFEQ</td>
<td>RANLOC=defloc</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RANSCALE=ranloc*.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NRBFEH</td>
<td>RANLOC=defloc</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RANSCALE=ranloc*.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NRBFEW</td>
<td>RANLOC=defloc</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RANSCALE=ranloc*.1</td>
<td></td>
<td>RANLOC=1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RANSCALE=ranloc*.5</td>
</tr>
<tr>
<td>NRBFEV</td>
<td>RANLOC=.5*defloc</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RANSCALE=ranloc*.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NRBFUN</td>
<td>RANLOC=defloc</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RANSCALE=ranloc*.5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Options

#### HIDDEN= n

Specifies the number of hidden units for all architectures other than GLIM.

**Default:** None
DIRECT
Requests direct connections from inputs to outputs.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
To score a data set based on a previously trained neural network outside of PROC NEURAL, you can specify a CODE statement to write SAS DATA step code to a file or catalog entry. This code can then be included into a DATA step that reads (using a SET statement) the data set to be scored.

**Category**
- Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

**CODE FILE=** file-name
  <ERROR|NOERROR >
  <FORMAT=format>  
  <GROUP=name>
  <RESIDUAL|NORESIDUAL>;

**Required Arguments**

**FILE=** file-name
  Specifies the filename to which the scoring code is written.

**Options**

**ERROR | NOERROR**
  Specifies whether to generate code to compute the error function.
  
  **Default:** NOERROR

**FORMAT=** format
  Specifies the format to use for the weights.
  
  **Default:** BEST12

**RESIDUAL | NORESIDUAL**
  Specifies whether to generate code to compute residuals. If you request code for residuals and then score a data set that does not contain target values, the residuals will have missing values.
  
  **Default:** NORESIDUAL
The NEURAL Procedure

CONNECT Statement

A network can be specified without any CONNECT statements. However, such a network will be connected by default as follows. First all input layers are connected to the first hidden layer, then each hidden layer except the last is connected to the next hidden layer. Finally, the last hidden layer is connected to all output layers. If this particular architecture is not appropriate, use one or more CONNECT statements to explicitly define the network connections.

**Category**
Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

**CONNECT** *id-list*;

**Required Arguments**

*id-list*

Lists the identifiers of two or more layers to connect. The identifiers must have been previously defined by the ID= option in an INPUT, a HIDDEN, or a TARGET statement. Each layer except the last is connected to the next layer in the list. Connections must be feedforward. Loops are not allowed.

For example, the following PROC NEURAL step connects the input layers to the output layer, the input layers to the hidden units, and the hidden units to the output layer.

```plaintext
title 'Fully Connected Network';
proc neural data=mydata dmdbcat=mycat;
   input a b / level= nominal id=nom;
   input x z / level= interval id=int;
   hidden 2 / id= hu;
   target y / level=interval id=tar;
   connect int tar;
   connect nom tar;
   connect int hu;
   connect nom hu;
   connect hu tar;
   train;
run;
```

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The NEURAL Procedure

CUT Statement

If the weights corresponding to the connection between two layers are not contributing the predictive ability of the network, you can remove that connection and the corresponding weights by using a CUT statement.

Category | Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

CUT id-list| ALL;

Options

You must specify either:

id-list

Specifies the identifiers of the layers to disconnect.

ALL

Disconnects all layers.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
DECISION Statement

Specifies information used for decision processing in the DECIDE, DMREG, NEURAL, and SPLIT procedures. This documentation applies to all four procedures.

Tip: The DECISION statement is required for PROC DECIDE. It is optional for DMREG, NEURAL and SPLIT procedures.

DECISION DECDATA=<libref.> SAS-data-set <DECVARS=decision-variable(s)> <option(s)>;

DECDATA= <libref.> SAS-data-set

Specifies the input data set that contains the decision matrix. The DECDATA= data set must contain the target variable.

Note: The DECDATA= data set may also contain decision variables specified by means of the DECVARS= option, and prior probability variable(s) specified by means of the PRIORVAR= option or the OLDPRIORVAR= option, or both.

The target variable is specified by means of the TARGET statement in the DECIDE, NEURAL, and SPLIT procedures or by using the MODEL statement in the DMREG procedure. If the target variable in the DATA= data set is categorical, then the target variable of the DECDATA= data set should contain the category values, and the decision variables will contain the common consequences of making those decisions for the corresponding target level. If the target variable is interval, then each decision variable will contain the value of the consequence for that decision at a point specified in the target variable. The unspecified regions of the decision function are interpolated by a piecewise linear spline.

Tip: The DECDATA= data set may be of TYPE=LOSS, PROFIT, OR REVENUE. If unspecified, TYPE=PROFIT is assumed by default. TYPE= is a data set option that should be specified when the data set is created.

DECVARS=decision-variable(s)

Specifies the decision variables in the DECDATA= data set that contain the target-specific consequences for each decision.

Default: None

COST=cost-option(s)

Specifies numeric constants that give the cost of a decision, or variables in the DATA= data set that contain the case-specific costs, or any combination of constants and variables. There must be the same number of cost constants and variables as there are decision variables in the DECVARS=
option. In the COST= option, you may not use abbreviated variable lists such as D1-D3, ABC--XYZ, or PQR:

| Default       | All costs are assumed to be 0. |

**CAUTION:**

The COST= option may only be specified when the DECDATA= data set is of TYPE=REVENUE.

**PRIORVAR=**\textit{variable}

Specifies the variable in the DECDATA= data set that contains the prior probabilities to use for making decisions.

| Tip                          | In the DECIDE procedure, if PRIORVAR= is specified, OLDPRIORVAR= must also be specified. |

| Default       | None |

**OLDPRIORVAR=**\textit{variable}

Specifies the variable in the DECDATA= data set that contains the prior probabilities that were used when originally fitting the model.

| Tip                          | If OLDPRIORVAR= is specified, PRIORVAR= must also be specified. |

**CAUTION:**

OLDPRIORVAR= is not allowed in PROC SPLIT.

| Default       | None |
DELETE Statement

If an input or hidden layer is not contributing the predictive ability of the network, you can remove that layer with a DELETE statement. The DELETE statement also removes all associated weights.

**Category**

Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

**DELETE** *id-list*;

**Required Argument**

**id-list**

Specifies the identifiers of layers to delete.
FREEZE Statement

Normally during training, all weights are updated. If you freeze one or more weights, those weights will retain their frozen value until a corresponding THAW statement is executed. Freezing weights causes training to proceed faster and require less memory.

**Category**
Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

```
FREEZE weight-list / <VALUE=number>
<EST=<libref.>SAS-data-set>;
```

**Required Argument**

```
weight-list
```

List of weights to freeze.

Weight-list consists of 0 or more repetitions of:

```
wname --> wname-2 where:

wname

  is a unit name, a layer identifier, BIAS, or ALTITUDE

wname-2

  is a unit name or a layer identifier
```

**Options**

You can specify either VALUE= or EST= but not both. If neither option is specified, the weights are frozen to their current values.

```
VALUE=number
```

Specifies the numeric value to which weights are to be frozen.

```
EST=<libref.>SAS-data-set
```

Specifies the SAS data set containing the values to which weights are to be frozen.
FREQ Statement

Specifies the frequency variable for training.

Category Variable Statement - specifies variables.

FREQ variable;

Options

variable

Specifies the frequency variable. (The frequency variable can contain integer and non-integer values.)

Note: The FREQ variable is not required in the DATA= data set. The NEURAL procedure searches for the name of the FREQ variable in the DATA=, VALIDATA=, and TESTDATA= data sets. If the FREQ variable does not appear in any of these data sets, then the procedure issues a warning but continues processing. For any data set that does not contain the FREQ variable, a FREQ value of 1 is used for all observations.

Default: The default value is taken from the DMDB metadata. The FREQ statement overrides whatever is in the DMDB metadata. If a FREQ statement contains no variable name, then no FREQ variable is used.
HIDDEN Statement

You can specify as many HIDDEN statements as you want up to the limits imposed by computer memory, time, and disk space. The hidden layers can be connected in any feedforward pattern using CONNECT statements.

You can specify as many HIDDEN statements as you want up to the limits imposed by computer memory, time, and disk space. The hidden layers can be connected in any feedforward pattern using CONNECT statements.

**Category**

| Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement. |

**HIDDEN integer / ID=name**

<ACT=activation-function>

<BIAS|NOBIAS>

<COMBINE=combination-function>;

**Required Arguments**

integer

Specifies the number of units in the hidden layer.

ID=name

Specifies the identifier for the layer.

**Options**

**ACT=activation-function**

Specifies the activation function. See [ACTIVATION FUNCTIONS](#).

**Default:**

For hidden units, the default activation function depends on the combination function and on the number of units in the layer.

For COMBINE=ADD, the default is ACT=IDENTITY.

For COMBINE=LINEAR or EQSLOPES, the default is ACT=TANH.

For COMBINE=EHRADIAL, EQRADIAL, EVRADIAL, EWRADIAL, or XRADIAL, the default is ACT=EXP if there is only one hidden unit in the layer; otherwise the default is ACT=SOFTMAX.

**BIAS | NOBIAS**

Specifies whether to use bias.
Default: BIAS

COMBINE=combination-function

Specifies the combination function. See COMBINATION FUNCTIONS.
**ACTIVATION FUNCTIONS**

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>RANGE</th>
<th>FUNCTION (OF NET INPUT t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDENTITY</td>
<td>$( -\infty, +\infty )$</td>
<td>$t$</td>
</tr>
<tr>
<td>LINEAR</td>
<td>$( -\infty, +\infty )$</td>
<td>$t$</td>
</tr>
<tr>
<td>EXPONENTIAL</td>
<td>$( 0, \infty )$</td>
<td>$e^t$</td>
</tr>
<tr>
<td>RECIPROCAL</td>
<td>$( 0, \infty )$</td>
<td>$\frac{1}{t}$</td>
</tr>
<tr>
<td>SQUARE</td>
<td>$[ 0, +\infty )$</td>
<td>$t^2$</td>
</tr>
<tr>
<td>LOGISTIC</td>
<td>$( 0, 1 )$</td>
<td>$\frac{1}{1+e^{-t}}$</td>
</tr>
<tr>
<td>MLOGISTIC</td>
<td>$( 0, 1 )$</td>
<td>$\frac{e^t}{\sum \text{exponentials}}$</td>
</tr>
<tr>
<td>SOFTMAX</td>
<td>$( 0, 1 )$</td>
<td>$\frac{e^t}{\sum \text{exponentials}}$</td>
</tr>
<tr>
<td>GAUSS</td>
<td>$( 0, 1 ]$</td>
<td>$e^{-t^2}$</td>
</tr>
<tr>
<td>SINE</td>
<td>$[ 0, 1 ]$</td>
<td>$\sin(t)$</td>
</tr>
<tr>
<td>COSINE</td>
<td>$[ 0, 1 ]$</td>
<td>$\cos(t)$</td>
</tr>
<tr>
<td>ELLIOTT</td>
<td>$( -1, 1 ]$</td>
<td>$\frac{t}{1+</td>
</tr>
<tr>
<td>TANH</td>
<td>$( -1, 1 ]$</td>
<td>$\tanh(t) = 1 - \frac{2}{1+e^{2t}}$</td>
</tr>
<tr>
<td>ARCtan</td>
<td>$(-1, 1)$</td>
<td>$\arctan(t) \times \frac{2}{\pi}$</td>
</tr>
</tbody>
</table>
COMBINATION FUNCTIONS

A combination function combines the values received from preceding nodes into a single number called the net input. Both output and hidden layers are assigned combination functions.

The following combination functions are available.

**Add**

Adds all the incoming values without using any weights or biases.

**Linear**

Is a linear combination of the incoming values and weights.

**EQSlopes**

Is identical to the Linear combination function, except that the same connection weights are used for each unit in the layer, although different units have different biases. EQSlopes is mainly used for ordinal targets.

**EQRadial**

Is a radial basis function with equal heights and widths for all units in the layer.

**EHRadial**

Is a radial basis function with equal heights but unequal widths for all units in the layer.

**EWRadial**

Is a radial basis function with equal widths but unequal heights for all units in the layer.

**EVRadial**

Is a radial basis function with equal volumes for all units in the layer.

**XRadial**

Is a radial basis function with unequal heights and widths for all units in the layer.

The following definitions apply to the Table of Combination Functions:

**All summations**

Are divided by the net inputs indexed by $i$.

$alt_j$

The altitude of the $j$th unit

$bias_j$

The width (bias) of the $j$th unit

$bias$

A common bias shared by all units in the layer
The fan-in of the $j$th unit

The weight connecting the $i$th incoming value to the $j$th unit

The common weight for the $i$th input shared by all units in the layer

The $i$th incoming value

### Combination Functions

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>$\sum_i x_i$</td>
</tr>
<tr>
<td>LINear</td>
<td>$bias_j + \sum_i w_{ij}x_i$</td>
</tr>
<tr>
<td>EQSlopes</td>
<td>$bias_j + \sum_i w_i x_i$</td>
</tr>
<tr>
<td>XRAradial</td>
<td>$f \log (altb_j) - bias_j^2 \sum_i (w_{ij} - x_i)^2$</td>
</tr>
<tr>
<td>EHRadial</td>
<td>$bias_j^2 [ (w_{ij} - x_i)^2 ]$</td>
</tr>
<tr>
<td>EVRadial</td>
<td>$f \log (bias_j) - bias_j^2 [ (w_{ij} - x_i)^2 ]$</td>
</tr>
<tr>
<td>EWRadial</td>
<td>$f \log (altb_j) - bias^2 [ (w_{ij} - x_i)^2 ]$</td>
</tr>
<tr>
<td>EQRadial</td>
<td>$-bias^2 [ (w_{ij} - x_i)^2 ]$</td>
</tr>
<tr>
<td>RADial</td>
<td>defaults to EHRadial</td>
</tr>
</tbody>
</table>

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The NEURAL Procedure

INITIAL Statement

After a network has been defined in terms of input, hidden and output layers, all weights and biases in the network must be given initial values before any training is performed. PROC NEURAL will by default supply appropriate random or computed values for these quantities. If you train a network without supplying an INITIAL or USE statement, the network will be initialized using the default specifications.

**Category**
Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

**INITIAL**

- `<BIADJUST=adjustment-value>`
- `<INEST=]<libref.>SAS-data-set>`
- `<INFAN=number>`
- `<OUTEST=]<libref.>SAS-data-set>`
- `<RANDBIAS|NORANDBIAS>`
- `<RANDOUT|NORANDOUT>`
- `<RANDOM=integer>`
- `<RANDSCALE | NORANDSCALE>;`

**Options**

**BIADJUST= adjustment-value**

Specifies how to adjust the random biases for units with a LINEAR combination function. A random bias is adjusted by multiplying by the function of the weights indicated by `adjustment-value`, and dividing by the scale (RANSCALE=) of the distribution from which the random bias was drawn. `adjustment value` can be one of the following:

- **SUM**
  Adjusts random initial biases for the sum of the absolute connection weights leading into the unit. This value is typically used with STD=MIDRANGE for inputs and RANDIST=UNIFORM.

- **USS**
  Adjusts random initial biases for the square root of the sum of squared connection weights leading into the unit. This value is typically used with STD=STD for inputs and RANDIST=NORMAL.

- **NONE|NO**
  No bias adjustment.
BIADJUST=NONE

`INEST=<libref.>SAS-data-set`

Specifies an input data set that contains some or all of the weights. Any weights in the INEST= data set that have missing values are assigned values according to the RANDOM=, RANDOUT, and RANDBIAS options, as well as the options that pertain to random number distributions that you specify in the Random statements. An INEST= data set will typically have been created by using the OUTEST= option in a SAVE or a TRAIN statement from a previous execution of the NEURAL procedure.

`INFAN=number`

Divide random connection weights by

\[(\text{fan-in of unit})^\text{number}\]

where the "fan-in" of a unit is the number of other units feeding into that unit, not counting the bias or altitude.

**Default:** 0 for radial combination functions, otherwise 0.5

**Range:** between 0 and 1

`OUTEST=<libref.>SAS-data-set`

Specifies the output data set that contains all the initial weights.

`RANDBIAS | NORANDBIAS`

Specifies whether to randomize output biases.

**Note:** NORANDBIAS overrides whatever you specify in the RANOPTIONS statement.

**Default:** NORANDBIAS, which sets bias to the inverse activation function of the target mean.

`RANDOM=integer`

Specifies the random number seed.

**Default:** 0

`RANDOUT | NORANDOUT`

Specifies whether to randomize the output connection weights.

**Note:** NORANDOUT overrides whatever you specify in the RANOPTIONS statement.

**Default:** NORANDOUT, which sets weights to 0.

`RANDSCALE | NORANDSCALE`

Specifies whether to randomize target scale estimates.
| **Default:** | NORANDSCALE, which sets each scale estimate to the standard deviation of the corresponding target variable. |
| **Note:** | NORANDSCALE overrides whatever is specified in the RANOPTIONS statement. |
The INPUT statement allows you to group together input variables having common levels and standardizations. You can specify as many INPUT statements as you want up to the limits imposed by computer memory, time, and disk space. The input layers can be connected to hidden or output layers using CONNECT statements.

**INPUT statement**

```plaintext
INPUT variable-list / ID=name
<LEVEL=value>
<STD=method>;
```

### Required Arguments

**variable-list**

Specifies the input variables.

**ID=name**

Specifies the identifier for the layer.

### Options

**LEVEL=value**

Specifies the measurement level, where `value` can be:

- NOMINAL|NOM
  - Nominal
- ORDINAL|ORD
  - Ordinal
- INTERVAL|INT
  - Interval

**Default:**

Interval (for variables specified by a VAR statement in the DMDB procedure) or nominal (for variables specified by a CLASS statement in the DMDB procedure).

**STD=method**

Specifies the standardization method, where `method` is:
NONE\|NO
   Variables are not altered.

STD
   Variables are rescaled to have a mean of 0 and a standard deviation of 1.

RANGE\|RAN
   Variables are rescaled to have a minimum of 0 and a range of 1. This standardization is not recommended for input variables.

MIDRANGE\|MID
   Variables are rescaled to have a midrange of 0 and a half-range of 1 (that is, a minimum of -1 and a maximum of 1).

Default:  STD
The NEURAL Procedure

NETOPTIONS Statement

Identifies the network options to set.

<table>
<thead>
<tr>
<th>Category</th>
<th>Option Statement - does not directly affect the network, but sets options for use in subsequent action statements. The options persist until reset at a later stage in the processing.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>NETOPTS</td>
</tr>
</tbody>
</table>

NETOPTIONS network-option(s);

Network Options

DECAY=number

Specifies the weight decay.

Range: \( number \geq 0 \)

Default: For the QPROP optimization technique: .0001; for all others: 0

INVALIDTARGET=action

Specifies the action taken during training if an out-of-range target value is found, where \( \text{action} \) can be:

OMITCASE | OMIT If INVALIDTARGET = OMITCASE is specified, and an invalid target value is found in the training data set, a warning is given, the observation is not used, but the training will continue.

STOP If INVALIDTARGET = STOP is specified, an error is issued, and training is terminated.

Example: If ERROR = GAMMA is specified in a target statement, the target values should be positive. If a zero or negative value is found for the variable(s) listed in the target statement, the observation containing that value cannot be used for training. The training either continue with the remaining valid observations or stops depending on the INVALIDTARGET= specification. Note, however, if the \{\text{it mean}\}, over the training data set, of the target variable(s) is zero or negative, an error is issued and the training is stopped, regardless of the INVALIDTARGET= specification.
The default is INVALIDTARGET = STOP. INVALIDETARGET can be abbreviated as INVALIDTARG or INVTARG.

**OBJECT=objective-function**

Specifies the objective function where *objective-function* can be one of the following:

**DEV**

Requests deviance (for ERROR=NORMAL, this is least squares).

**LIKE**

Requests negative log-likelihood.

**MEST**

Requests M estimation.

*Default:* Depends on the error functions that the network uses. To determine the default, examine the table below named Errors by Objective Functions. Scan the table from left to right. The default is the first column that contains a "yes" in every row corresponding to an error function used in the network. If no such column exists in the table, an error message is issued and the network cannot be trained.

### Errors by Objective Functions

<table>
<thead>
<tr>
<th>ERRORS</th>
<th>DEV</th>
<th>LIKE</th>
<th>MEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Cauchy</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Logistic</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Huber</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Biweight</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Wave</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Gamma</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Poisson</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Bernoulli</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Binomial</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Entropy</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Mbernoulli | Yes | Yes | No
---|---|---|---
Multinomial | Yes | Yes | No
Mentropy | Yes | Yes | No

**RANDF=** <i>number</i>

Specifies the degrees of freedom parameter for random numbers. See the following Randomization Options and Default Parameters table.

**RANDIST=** <i>name</i>

Specifies the type of distribution to be used for random initial weights and perturbations. The distributions and default parameter values are as follows:

### Randomization Options and Default Parameters

<table>
<thead>
<tr>
<th>RANDIST</th>
<th>RANLOC</th>
<th>RANSCALE</th>
<th>DF</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL</td>
<td>mean=0</td>
<td>std=1</td>
<td>-</td>
</tr>
<tr>
<td>UNIFORM</td>
<td>mean=0</td>
<td>halfrange=1</td>
<td>-</td>
</tr>
<tr>
<td>CAUCHY</td>
<td>median=0</td>
<td>scale=1</td>
<td>-</td>
</tr>
<tr>
<td>CHIINV</td>
<td>-</td>
<td>scale=1</td>
<td>df=1</td>
</tr>
</tbody>
</table>

**Default:** NORMAL

**RANDOM=** <i>integer</i>

Specifies the random number seed.

**Default:** 0

**RANLOC=** <i>number</i>

Specifies the location parameter for random numbers. See the above Randomization Options and Default Parameters table.

Specifies the scale parameter for random numbers. See the above Randomization Options and Default Parameters table.
The NEURAL Procedure

NLOPTIONS Statement

Identifies the nonlinear optimization options to set.

| Category      | Option Statement - does not directly affect the network, but sets options for use in subsequent action statements. The options persist until reset at a later stage in the processing. |

NLOPTIONS <nonlinear-options>;

Nonlinear Options

**ABSCONV= number**

Specifies an absolute function convergence criterion. ABSCONV= is a function of the log-likelihood for the intercept-only model.

- **Default:** The default value is the negative square root of the largest double precision value.
- **Range:** number > 0

**ABSFCONV= number**

Specifies an absolute function convergence criterion.

- **Default:** 0
- **Range:** number > 0

**ABSGCONV= number**

Specifies the absolute gradient convergence criterion.

- **Default:** 1E-5
- **Range:** number > 0

**ABSXCONV= number**

Specifies the absolute parameter convergence criterion.

- **Default:** 0
- **Range:** number > 0
DAMPSTEP= number
Specifies that the initial step size value for each line search used by the QUANEW, CONGRA, or NEWRAP optimization technique cannot be larger than the product of number and the step size value used in the former iteration.

**Default:** 2
**Range:** number > 0

DIAHES
Forces the optimization algorithm (TRUREG, NEWRAP, or NRRIDG) to take advantage of the diagonality.

FCONV= number
Specifies a function convergence criterion.

**Default:** 1E-4
**Range:** number > 0

FSIZE= number
Specifies the parameter of the relative function and relative gradient termination criteria.

**Default:** Not applicable.
**Range:** number ≥ 0

GCONV= number
Specifies the relative gradient convergence criterion.

**Default:** 1E-8
**Range:** number > 0

HESCAL= 0 | 1 | 2 | 3
Specifies the scaling version of the Hessian or cross-product Jacobian matrix used in NRRIDG, TRUREG, LEVMAR, NEWRAP, or DBLDOG optimization.

**Default:**
1 - for LEVMAR minimization technique
0 - for all others

INHESSIAN= number
Specifies how to define the initial estimate of the approximate Hessian for the quasi-Newton techniques QUANEW and DBLDOG.
The default is to use a Hessian based on the initial weights as the initial estimate of the approximate Hessian. When \( r = 0 \), the initial estimate of the approximate Hessian is computed from the magnitude of the initial gradient.

**Range:** \( number > 0 \)

**INSTEP = number**

Specifies the initial radius of the trust region used in the TRUREG, DBLDOG, and LEVMAR algorithms.

**Default:** 1

**Range:** \( number > 0 \)

**LCEPS | LCEPSILON = number**

Specifies the range for active constraints.

**Range:** \( number > 0 \)

**LCSINGULAR = number**

Specifies the tolerance for dependent constraints

**Range:** \( number > 0 \)

**LINESEARCH = number**

Specifies the line-search method for the CONGRA, QUANEW, and NEWRAP optimization techniques.

**Default:** 2

**Range:** \( 1 \leq number \leq 8 \)

**LSPRECISION = number**

Specifies the degree of accuracy that should be obtained by the second and third line-search algorithms.
### Line-Search Precision Values

<table>
<thead>
<tr>
<th>TECHNIQUE</th>
<th>UPDATE</th>
<th>LSPRECISION VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUANEW</td>
<td>DBFGS, BFGS</td>
<td>0.4</td>
</tr>
<tr>
<td>QUANEW</td>
<td>DDFP, DFP</td>
<td>0.06</td>
</tr>
<tr>
<td>CONGRA</td>
<td>all</td>
<td>0.1</td>
</tr>
<tr>
<td>NEWRAP</td>
<td>no update</td>
<td>0.9</td>
</tr>
</tbody>
</table>

**Range:** \(number \geq 0\)

#### MAXFUNC = number

Specifies the maximum number of function calls in the optimization process.

**Default:** 2147483647 for all techniques

**Range:** \(number \geq 0\)

#### MAXITER = number

Specifies the maximum number of iterations in the optimization process.

**Default:**
- 100 for TRUREG, NRRIDG, NEWRAP, and LEVMAR
- 200 for QUANEW and DBLDOG
- 400 for CONGRA

**Range:** \(number \geq 0\)

#### MAXSTEP = number

Specifies the upper bound for the step length of the line-search algorithms.

**Default:** The largest double precision value

**Range:** \(number \geq 0\)

#### MAXTIME = number

Specifies the upper limit of CPU time for the optimization process. It is measured in seconds.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINITER</td>
<td>7 days, that is, MAXTIME=604800 seconds</td>
<td>number &gt; 0</td>
</tr>
<tr>
<td>NOPRINT</td>
<td>Suppresses all output. Only ERRORs, WARNINGs, and NOTEs are printed on the log file.</td>
<td></td>
</tr>
<tr>
<td>PHISTORY</td>
<td>Prints the optimization history. If PSUMMARY or NOPRINT are not specified, then the PHISTORY option is set automatically and the iteration history is printed by default.</td>
<td></td>
</tr>
<tr>
<td>RESTART</td>
<td>Specifies that the QUANEW or CONGRA algorithm is restarted with a steepest descent/ascent search direction after the number of iterations has been completed.</td>
<td>number ≥ 1</td>
</tr>
<tr>
<td>SINGULAR</td>
<td>Specifies an absolute singularity criterion for the computation of the inertia of Hessian and cross-product Jacobian and their projected forms.</td>
<td>number &gt; 0</td>
</tr>
</tbody>
</table>

### MINITER= number
Specifies the minimum number of iterations in the optimization process.

### Default:
0

### Range:
number ≥ 0

### NOPRINT
Suppresses all output. Only ERRORs, WARNINGs, and NOTEs are printed on the log file.

### PHISTORY
Prints the optimization history. If PSUMMARY or NOPRINT are not specified, then the PHISTORY option is set automatically and the iteration history is printed by default.

### RESTART= number
Specifies that the QUANEW or CONGRA algorithm is restarted with a steepest descent/ascent search direction after the number of iterations has been completed.

### Default:
For TECHNIQUE=CONGRA, and UPDATE= PB, restart is done automatically, so number is not used;

For TECHNIQUE=CONGRA, and UPDATE not = PB, number is the number of parameters.

For TECHNIQUE=QUANEW, number is the largest integer.

### SINGULAR= number
Specifies an absolute singularity criterion for the computation of the inertia of Hessian and cross-product Jacobian and their projected forms.

### Default:
1E-8

### Range:
number > 0
**TECHNIQUE=** *method*

Specifies an optimization technique, where *method* is one of the following:

- **CONGRA**
  - Specifies the Conjugate Gradient optimization technique. This is the default when the number of parameters to be estimated is \( \geq 400 \).

- **DBLDOG**
  - Specifies the Double-Dogleg optimization technique.

- **NEWRAP**
  - Specifies the Newton-Raphson with Line Search optimization technique.

- **NRRIDG**
  - Specifies the Newton-Raphson with Ridging optimization technique. This is the default when the number of parameters to be estimated is \( \leq 40 \).

- **QUANEW**
  - Specifies the quasi-Newton optimization technique. This is the default when the number of parameters to be estimated is in the range \( 40 < n \leq 400 \).

- **TRUREG**
  - Specifies the Trust-Region optimization technique.

**UPDATE=** *update-type*

Specifies an update method, where *update-type* is one of the following:

- **BFGS**
  - For TECHNIQUE= QUANEW, performs the BFGS (Broyden-Fletcher-Goldfarb-Shanno) update of the Cholesky factor of the Hessian matrix.

- **CD**
  - For TECHNIQUE=CONGRA, performs a conjugate descent update of Fletcher.

- **DBFGS**
  - For TECHNIQUE= DBLDOG or QUANEW, performs the dual BFGS (Broyden-Fletcher-Goldfarb-Shanno) update of the Cholesky factor of the Hessian matrix. This is the default for TECHNIQUE=QUANEW and DBLDOG.

- **DDFP**
  - For TECHNIQUE= DBLDOG or QUANEW, performs the dual DFP (Davidson-Fletcher-Powell) update of the Cholesky factor of the Hessian matrix.

- **DFP**
  - For TECHNIQUE= QUANEW, performs the original DFP (Davidson-Fletcher-Powell) update of the inverse Hessian matrix.

- **FR**
  - For TECHNIQUE=CONGRA, performs the Fletcher-Reeves update.
For TECHNIQUE=CONGRA, performs the automatic restart update method of Powell and Beale. This is the default for TECHNIQUE= CONGRA.

PR

For TECHNIQUE=CONGRA, performs the Polak-Ribiere update.

VERSION= 1 2 3

Specifies the version of the hybrid quasi-Newton optimization technique or the version of the quasi-Newton optimization technique with nonlinear constraints.

XCONV= number

Specifies the relative parameter convergence criterion.

XSIZE= number

Specifies the number of successive iterations for which the criterion must be satisfied before the optimization process can be terminated.
PERTURB Statement

Perturbs weights. Perturbing weights can sometimes allow you to escape a local minimum.

**Category**
Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

```plaintext
PERTURB weight-list / <RANDF=number > 
<OUTEST=<libref.>SAS-data-set> 
<RANDIST=name> 
<RANDOM=integer> 
<RANLOC=number> 
<RANSCALE=number> ;
```

**Required Argument**

`weight-list`
List of weights to freeze.

Weight-list consists of 0 or more repetitions of:

`wname --> wname-2` where:

`wname`
  is the unit name, the layer ID, BIAS, or ALTITUDE

`wname-2`
  is the unit name or layer ID

**Options**

RANDF=`number`
Specifies the degrees of freedom parameter for random numbers. See the Randomization Options and Default Parameters table for values.

OUTEST=`<libref.>SAS-data-set`
Specifies the output data set containing all the weights.

Default: none

RANDIST=`name`
Specifies the type of distribution for random numbers. See the Randomization Options and Default Parameters table for values.

**RANDOM=integer**

Specifies the random number seed.

**Default:** 0

**RANLOC=number**

Specifies the location parameter for random numbers. See the Randomization Options and Default Parameters table for values.

**RANSCALE=number**

Specifies the scale parameter for random numbers. See the Randomization Options and Default Parameters table for values.
The NEURAL Procedure

PRELIM Statement

The PRELIM statement performs preliminary training to reduce the risk of bad local optima. The final weights and biases in a trained network depend on the initial values. The PRELIM statement repeatedly trains a network for a small number of iterations (default 20) using different initializations. The final weights of the best trained network are then used to initialize a subsequent TRAIN statement.

Category | Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

PRELIM integer
<ACCELERATE=number>
<DECELERATE=number>
<INEST=<libref.>SAS-data-set>
<LEARN=number>
<MAXLEARN=number>
<MAX | MAXMOMENTUM=number>
<MINLEARN=number>
<MOM | MOMENTUM=number>
<OUTEST=<libref.>SAS-data-set>
<preiter=integer>
<pretech=name>
<pretime=number>
<RANDBIAS|NORANDBIAS>
<RANDOUT|NORANDOUT>
<RANDOM=integer>;

Required Argument

integer

Specifies the number of preliminary optimizations.

Options

ACCEL | ACCELERATE=number

Specifies the rate of increase of learning for the RPROP optimization technique.

Range: number > 1
DECEL | DECELERATE=number
Specifies the rate of decrease of learning for the RPROP optimization technique.

Range: \(0 < \text{number} < 1\)

Default: 0.5

INEST=<libref>.SAS-data-set
Specifies the input data set that contains some or all weights. Any weights in the INEST= data set that have missing values are assigned values according to the RANDOM=, RANDOUT, and RANDBIAS options, as well as the options that pertain to random number distributions that you specify in the Random statements.

LEARN=number
Specifies the learning rate for BPROP or the initial learning rate for QPROP and RPROP.

Range: \(\text{number} > 0\)

Default: 0.1

MAXLEARN=number
Specifies the maximum learning rate for RPROP.

Range: \(\text{number} > 0\)

Default: Reciprocal of the square root of the machine epsilon

MAXMOM | MAXMOMENTUM=number
Specifies the maximum momentum for BPROP.

Range: \(\text{number} > 0\)

Default: 1.75

MINLEARN=number
Specifies the minimum learning rate for RPROP.

Range: \(\text{number} > 0\)

Default: Square root of the machine epsilon

MOM | MOMENTUM=number
Specifies the momentum for BPROP.

Range: \(0 \leq \text{number} < 1\)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OUTEST=</strong>&lt;libref.&gt;<em>SAS-data-set</em>*</td>
<td>Specifies the output data set that contains all the weights.</td>
</tr>
<tr>
<td><strong>PREITER=</strong>&lt;integer&gt;</td>
<td>Specifies the maximum number of iterations in each preliminary optimization. <strong>Default:</strong> 10</td>
</tr>
<tr>
<td>**PRETECH</td>
<td>TECHNIQUE=**&lt;name&gt;</td>
</tr>
<tr>
<td><strong>PRETIME=</strong>&lt;number&gt;</td>
<td>Specifies the amount of time after which training stops.</td>
</tr>
<tr>
<td>**RANDBIAS</td>
<td>NORANDBIAS**</td>
</tr>
<tr>
<td><strong>RANDOM=</strong>&lt;integer&gt;</td>
<td>Specifies the random number seed. <strong>Default:</strong> 0</td>
</tr>
<tr>
<td>**RANDOUT</td>
<td>NORANDOUT**</td>
</tr>
</tbody>
</table>
QUIT Statement

Stops the procedure.

**Category**  Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

QUIT;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The NEURAL Procedure

RANOPTIONS Statement

Specifies distribution of random initial weights.

| Category      | Option Statement - does not directly affect the network, but sets options for use in subsequent action statements. The options persist until reset at a later stage in the processing. |

**RANOPTIONS** connection-list /

<RANDF=number>
<RANDIST=name>
<RANDOM=integer>
<RANLOC=number>
<RANSCALE=number>;

**Required Arguments**

**Note:** When a RANOPTIONS statement is executed, the specified options are stored in all the connections listed before the slash. These options are used whenever an INITIAL or PRELIM statement is executed. If you submit two RANOPTIONS statements for the same connection, the second statement overrides all options in the first. In other words, one RANOPTIONS statement does not remember what options were specified in previous RANOPTIONS statements. To have options persist over multiple statements, use the NETOPTIONS statement.

**connection-list**
- List of connections to randomize.

**connection-list** consists of 0 or more repetitions of:

- **wname --> wname-2** where:
  - **wname**
    - is the layer ID, BIAS, or ALTITUDE
  - **wname-2**
    - is the layer ID

**Options**

**RANDF=number**
- Specifies the degrees of freedom parameter for random numbers. See the Randomization Options.
and Default Parameters table for values.

**Default:** 1

**RANDIST=** `name`

Specifies the type of distribution for random numbers. See the Randomization Options and Default Parameters table for values.

**Default:** NORMAL

**RANDOM=** `integer`

Specifies the random number seed.

**Default:** 0

**RANLOC=** `number`

Specifies the location parameter for random numbers. See the Randomization Options and Default Parameters table for values.

**RANSCALE=** `number`

Specifies the scale parameter for random numbers. See the Randomization Options and Default Parameters table for values.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
SAVE Statement

Writes weights to data set or a description of the network to a catalog entry.

<table>
<thead>
<tr>
<th>Category</th>
<th>Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTE:</td>
<td>At least one option must be specified, but there is no single argument that is required.</td>
</tr>
</tbody>
</table>

```
SAVE OUTEST=<libref.>SAS-data-set
NETWORK=screen-specification;
```

Options

Specify at least one:

**NETWORK=screen-specification**

Saves the definition of the entire network. `screen-specification` is the name of a catalog entry.

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

Saves the network weights in an output data set.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The NEURAL Procedure

SCORE Statement

Creates an output data set containing predicted values and possibly other results such as residuals, classifications, decisions, and assessment values.

<table>
<thead>
<tr>
<th>Category</th>
<th>Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.</th>
</tr>
</thead>
</table>

**Score DATA=**<libref.>SAS-data-set
**OUT=**<libref.>SAS-data-set
**OUTFIT=**<libref.>SAS-data-set
**<DUMMIES | NODUMMIES>**
**<ROLE=role-option>;**

**Required Arguments**

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

- Specifies the output data set that contains the outputs.

**Options**

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

- Specifies the input data to be scored that contains inputs and, optionally, targets.
  - **Default:** Defaults to the training data (DATA= in the PROC statement).

**DUMMIES | NODUMMIES**

- Specifies whether to write dummy variables to the OUT= data set.
  - **Default:** NODUMMIES

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

- Specifies the output data set that contains the fit statistics.

**ROLE=role-option**

- Specifies the role of the DATA= data set. ROLE=role-option primarily affects which fit statistics are computed and what their names and labels are. **Role-option** is one of the following:
  - **TRAIN**
    - Specifies that the DATA= data set is the training set. The data set must contain the target variable.
**VALID | VALIDATION**

Specifies that the DATA= data set is a validation set. The data set must contain the target variable.

**TEST**

Specifies that the DATA= data set is a test set. The data set must contain the target variable.

**SCORE**

Specifies that residuals, error functions, and fit statistics are not produced. The data set does not have to contain the target variable.

<table>
<thead>
<tr>
<th>Default:</th>
<th>TEST, except as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAIN</td>
<td>when the DATA= data set in the PROC statement is the same as the DATA= data set in the SCORE statement. Specifying TRAIN with any data set other than the actual training set is an error.</td>
</tr>
<tr>
<td>VALID</td>
<td>when the DATA= data set in the SCORE statement is the same as the VALIDATA= data set in the PROC statement.</td>
</tr>
</tbody>
</table>

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The NEURAL Procedure

SET Statement

Sets the value of the weight-list to number. The SET statement does not freeze weights, so subsequent training may change the values of the weights specified in a SET statement.

<table>
<thead>
<tr>
<th>Category</th>
<th>Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.</th>
</tr>
</thead>
</table>

**SET weight-list number;**

**Required Arguments**

*weight-list*

   Specifies the list of weights to be affected or changed.

*number*

   Specifies the number to which the weight-list is set.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
SHOW Statement

Prints information about the network.

**Category**
Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

**NOTE:**
At least one option must be specified, but there is no single argument that is required.

SHOWWEIGHTS
STATEMENTS

**Options**

Specify at least one:

**STATEMENTS**
Prints statements that can be used with the NEURAL procedure to reproduce the network.

**WEIGHTS**
Prints the network weights.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
TARGET Statement

Defines an output layer.

<table>
<thead>
<tr>
<th>Category</th>
<th>Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>OUTPUT</td>
</tr>
</tbody>
</table>

```
TARGET | OUTPUT variable-list /
<ACT=activation-function>
<BIAS|NOBIAS >
<COMBINE=combination-function>
<ERROR=keyword>
<ID=name>
<LEVEL=value>
<MESTA=number>
<MESTCON=number>
<SIGMA=number>
<STD=method>;
```

Required Arguments

```
variable-list
    Specifies the target variables.
```

```
ID=name
    Specifies the identifier for the layer.
```

Options

```
ACT=activation-function
    Specifies the activation function. See Activation Functions.
```
**Default:**

Depends on the measurement level, as follows:

If LEVEL=INTERVAL, then the default is IDENTITY.

If LEVEL=ORDINAL then the default is LOGISTIC.

If LEVEL=NOMINAL, then the default is MLOGISTIC
(For Error=MBERNOULLI, MENTROPY, or MULTINOMIAL, the only activation function allowed is MLOGISTIC.)

**BIAS | NOBIAS**

Specifies whether to use bias (or not to use bias).

**Default:**

BIAS

**COMBINE=combination-function**

Specifies the combination function. See [Combination Functions](#).

**ERROR=keyword**

Specifies the Error function. Default is NORMAL for LEVEL=INTERVAL; otherwise, default is MBERNOULLI. For more information, see the Error Functions table that follows.

<p>| ERROR Functions |
|-----------------|-----------------|</p>
<table>
<thead>
<tr>
<th><strong>KEYWORD</strong></th>
<th><strong>TARGET</strong></th>
<th><strong>DESCRIPTION</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions with scale parameters :</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NORmal</td>
<td>any</td>
<td>Normal distribution</td>
</tr>
<tr>
<td>CAUchy</td>
<td>any</td>
<td>Cauchy distribution</td>
</tr>
<tr>
<td>LOGistic</td>
<td>any</td>
<td>Logistic distribution</td>
</tr>
<tr>
<td>HUBer</td>
<td>any</td>
<td>Huber M estimator</td>
</tr>
<tr>
<td>BIWeight</td>
<td>any</td>
<td>Biweight M estimator</td>
</tr>
<tr>
<td>WAVe</td>
<td>any</td>
<td>Wave M estimator</td>
</tr>
<tr>
<td>Function</td>
<td>Domain</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GAMma</td>
<td>$&gt;0$</td>
<td>Gamma distribution</td>
</tr>
<tr>
<td>POIsson</td>
<td>$\geq 0$</td>
<td>Poisson distribution</td>
</tr>
<tr>
<td>Functions with no scale parameter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERnoulli</td>
<td>0,1</td>
<td>Bernoulli distribution (binomial with one trial)</td>
</tr>
<tr>
<td>BINomial</td>
<td>$\geq 0$</td>
<td>Binomial distribution</td>
</tr>
<tr>
<td>ENTropy</td>
<td>0-1</td>
<td>Cross or relative entropy for independent targets</td>
</tr>
<tr>
<td>MBErnoulli</td>
<td>0,1</td>
<td>Multiple Bernoulli (multinomial with one trial)</td>
</tr>
<tr>
<td>MULtinomial</td>
<td>$\geq 0$</td>
<td>Multinomial distribution</td>
</tr>
<tr>
<td>MENtropy</td>
<td>0-1</td>
<td>Cross or relative entropy for targets that sum to 1 (Kullback-Leibler divergence)</td>
</tr>
</tbody>
</table>

**LEVEL=**\textit{value}

Specifies the measurement level, where \textit{value} can be:

- **NOMINAL|NOM**
  Nominal.
- **ORDINAL|ORD**
  Ordinal.
- **INTERVAL|INT**
  Interval.
Default: NOMINAL for character variables. INTERVAL for numeric variables.

**MESTA=number**

Specifies the scale constant for M estimation.

**Default:** Default value is computed from MESTCON to give consistent scale estimates for normal noise.

**MESTCON=number**

Specifies the tuning constant for M estimation.

**Default:**

- Huber: 1.5;
- Biweight: 9;
- Wave: $2.1\pi$

**SIGMA=number**

Specifies the fixed value of the scale parameter.

**Default:** By default, SIGMA is not used; but with OBJECT=LIKE, the scale parameter is estimated.

**STD=method**

Specifies the standardization method, where *method* is

- NONE|NO
  - Variables are not altered.
- STD
  - Variables are rescaled to have a mean of 0 and a standard deviation of 1.
- RANGE|RAN
  - Variables are rescaled to have a minimum of 0 and a range of 1.
- MIDRANGE|MID
  - Variables are rescaled to have a midrange of 0 and a half-range of 1 (that is, a minimum of -1 and a maximum of 1).

**Default:** NO
THAW Statement

Thaws frozen weights.

**Category**  
Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

\[
\text{THAW} \ weight-list;
\]

**Required Argument**

weight-list  
List of weights to thaw.

Weight-list consists of 0 or more repetitions of:

\[
\text{wname} \rightarrow \text{wname-2} \ 	ext{where:}
\]

\[
\text{wname}
\]

is the unit name, the layer ID, BIAS, or ALTITUDE

\[
\text{wname-2}
\]

is the unit name or the layer ID

**Default:** All weights are thawed.
The NEURAL Procedure

TRAIN Statement

Trains the network.

Category
Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

TRAIN OUT=libref:SAS-data-set
OUTEST=libref:SAS-data-set
OUTFIT=libref:SAS-data-set
<ACCEL|ACCELERATE=number>
<DECEL|DECELERATE=number>
<DUMMIES | NODUMMIES>
<ESTITER=i>
<LEARN=number>
<MAX|MAXMOMENTUM=number>
<MAXITER=integer>
<MAXLEARN=number>
<MAXTIME =number>
<MINLEARN=number>
<MOM|MOMENTUM=number>
<TECHNIQUE=name>;

Options

ACCEL | ACCELERATE=number

Specifies the rate of increase of learning for the RPROP optimization technique.

| Range: | number > 1 |
| Default: | 1.2 |

DECEL | DECELERATE=number

Specifies the rate of decrease of learning for the RPROP optimization technique.

| Range: | 0 < number < 1 |
| Default: | 0.5 |

DUMMIES | NODUMMIES
Specifies whether to write dummy variables to the OUT= data set.

**Default:** NODUMMIES

**ESTITER=** \(i\)

\(i = 0\)

Writes only initial and final weights to the OUTEST=data set.

\(i > 0\)

Writes weights after every \(i\) iterations, as well as the final weights, to the OUTEST= data set.

**Default:** 0

**LEARN=** \(number\)

Specifies the learning rate for BPROP or the initial learning rate for QPROP and RPROP.

**Range:** \(number > 0\)

**Default:** 0.1

**MAXITER=** \(integer\)

Maximum number of iterations.

**Default:**

- 100 for TRUREG and LEVMAR
- 200 for QUANEW and DBLDOG
- 400 for CONGRA
- 1000 for BPROP, RPROP, and QPROP

**MAXLEARN=** \(number\)

Specifies the maximum learning rate for RPROP.

**Range:** \(number > 0\)

**Default:** \(1/\text{MACSQRTEPS}\)

**MAXMOM | MAXMOMENTUM=** \(number\)

Specifies the maximum momentum for BPROP.

**Range:** \(number > 0\)

**Default:** 1.75

**MAXTIME=** \(number\)
Specifies the amount of time after which training stops.

**Default:** 7 days, that is, 604800 seconds

---

**MINLEARN=number**

Specifies the minimum learning rate for RPROP.

**Range:** number > 0

**Default:** MACSQRTEPS

---

**MOM | MOMENTUM=number**

Specifies the momentum for BPROP.

**Range:** 0 ≤ number < 1

**Default:**
- For BPROP: 0.9
- For RPROP: 0.1

---

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

Specifies the output data set that contains the outputs.

---

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

Specifies the output data set that contains the network weights.

---

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

Specifies the output data set that contains the fit statistics.

---

**TECHNIQUE=**name

Specifies the optimization technique where name is one of the following:

- **TRUREG**
  Requests Trust region.

- **LEVMAR**
  Requests Levenberg-Marquardt.

- **DBLDOG**
  Requests Double dogleg.

- **QUANEW**
  Requests quasi-Newton.

- **CONGRA**
  Requests Conjugate gradient.

- **BPROP**
  Requests standard backprop (backpropagation), that is, a variation on an algorithm called...
the generalized delta rule. In backpropagation, the difference (delta) between the output value and the target value is the error.

RPROP

Requests the RPROP algorithm.

QPROP

Requests Quickprop.

See the following table for the defaults for weight-based optimization techniques for a given value of the OBJECT= option.

**Defaults for Weight-based Optimization Techniques**

<table>
<thead>
<tr>
<th>OBJECTIVE FUNCTION</th>
<th>OPTIMIZATION TECHNIQUE</th>
<th>NUMBER OF WEIGHTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT=DEV</td>
<td>LEVMAR</td>
<td>0 to 100 weights</td>
</tr>
<tr>
<td>OBJECT=DEV</td>
<td>QUANEW</td>
<td>101 - 501 weights</td>
</tr>
<tr>
<td>OBJECT=DEV</td>
<td>CONGRA</td>
<td>501 or more weights</td>
</tr>
<tr>
<td>(All other objective functions)</td>
<td>QUANEW</td>
<td>up to 500 weights</td>
</tr>
<tr>
<td>(All other objective functions)</td>
<td>CONGRA</td>
<td>501 or more weights</td>
</tr>
</tbody>
</table>

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
USE Statement

Sets all weights to values from a data set.

**Category**  | Action Statement - affects the network or the data sets. Options set in an action statement affect only that statement.

```
USE <libref.>SAS-data-set;
```

**Required Arguments**

`<libref.>SAS-data-set`

Specifies an input data set that contains all the weights. Unlike the INITIAL statement, the USE statement does not generate any random weights, therefore the data set must contain all of the network weights and parameters.
The NEURAL Procedure

Details

For details about neural network architecture and training, see the online Neural Network Node: Reference documentation. For an introduction to predictive modeling, see the online Predictive Modeling document. Both of these documents can be accessed by using the Help pull-down menu to select the "Open the Enterprise Miner Nodes Help" item.

The BPROP, RPROP, and QPROP Algorithms Used in PROC NEURAL

Introduction

While the standard backprop algorithm has been a very popular algorithm for training feedforward networks, performance problems have motivated numerous attempts at finding faster algorithms.

The following discussion of the implementation of the backprop (BPROP), RPROP, and QPROP algorithms in PROC NEURAL relates the details of these algorithms with the printed output resulting from the use of the PDETAIL option. The discussion uses the algorithmic description and notation in Schiffmann, Joost, and Werner (1994) as well as the Neural Net Frequently Asked Questions (FAQ) available as a hypertext document readable by any World-Wide Web browser, such as Mosaic, under the URL: ftp://ftp.sas.com/pub/neural/FAQ.html.

There is an important distinction between "backprop" (or "back propagation of errors") and the "backpropagation algorithm".

The "back propagation of errors" is an efficient computational technique for computing the derivative of the error function with respect to the weights and biases of the network. This derivative, more commonly known as the error gradient, is needed for any first order nonlinear optimization method. The standard backpropagation algorithm is a method for updating with weights based on the gradient. It is a variation of the simple "delta rule". See "What is backprop?" in part 2 of the FAQ for more details and references on standard backprop, RPROP, and Quickprop.

With any of the "prop" algorithms, PROC NEURAL allows detailed printing of the iterations. The PDETAIL option in the TRAIN statement prints, for each iteration, all quantities involved in the algorithm for each weight in the network. This option should be used with caution as it can result in voluminous output. However, by restricting the number of iterations and number of non-frozen weights, a manageable amount of information is produced. The purpose of the PDETAIL option is to allow you to follow the nonlinear optimization of the error function for each of the network weights. For any particular propagation method, as described below, all quantities used to compute an updated weight are printed.

In standard backprop, too low a learning rate makes the network learn very slowly. Too high a learning
rate makes the weights and error function diverge, so there is no learning at all. If the error function is quadratic, as in linear models, good learning rates can be computed from the Hessian matrix. If the error function has many local and global optima, as in typical feedforward neural networks with hidden units, the optimal learning rate often changes dramatically during the training process, because the Hessian also changes dramatically. Trying to train a neural network using a constant learning rate is usually a tedious process requiring much trial and error.

With batch training, there is no need to use a constant learning rate. In fact, there is no reason to use standard backprop at all, because vastly more efficient, reliable, and convenient batch training algorithms exist such as Quickprop and RPROP.

Many other variants of backprop have been invented. Most suffer from the same theoretical flaw as standard backprop: the magnitude of the change in the weights (the step size) should NOT be a function of the magnitude of the gradient. In some regions of the weight space, the gradient is small and you need a large step size; this happens when you initialize a network with small random weights. In other regions of the weight space, the gradient is small and you need a small step size; this happens when you are close to a local minimum. Likewise, a large gradient may call for either a small step or a large step. Many algorithms try to adapt the learning rate, but any algorithm that multiplies the learning rate by the gradient to compute the change in the weights is likely to produce erratic behavior when the gradient changes abruptly. The great advantage of Quickprop and RPROP is that they do not have this excessive dependence on the magnitude of the gradient. Conventional optimization algorithms use not only the gradient but also second-order derivatives or a line search (or some combination thereof) to obtain a good step size.

**Mathematical Notation**

It is helpful to establish notation so that we can relate quantities and describe algorithms.

1. \( \omega_{ij}(n) \) is the weight associated with the connection between the \( i \)th unit in the current layer and the \( j \)th unit from the previous layer. The argument \( n \) refers to iteration.

2. \( \Delta \omega_{ij}(n) \) is the update or change for weight \( \omega_{ij}(n) \). This update results in the \( n+1 \) iteration value for \( \omega_{ij} \).

3. \( \frac{\partial E(n)}{\partial \omega_{ij}} \) is the partial derivative of the error function \( E(\omega) \) with respect to the weight \( \omega_{ij} \) at the \( n \)th iteration.

4. \( y^m_k(x^m; \omega) \) is the \( k \)th component of the output vector for the \( m \)th case as a function of the inputs \( x^m \) and network weights \( \omega \).

5. \( t^m_k(x^m) \) is the \( k \)th component of the target vector for the \( m \)th case as a function of the inputs \( x^m \).

The basic algorithm in all methods is a generic update given by
\[ \omega_{ij} (n + 1) = \omega_{ij} (n) + \Delta \omega_{ij} (n) \]

The BPROP, RPROP, and QPROP algorithms differ in how \( \Delta \omega_{ij} \) is computed.

**Standard Printing for the PROP Algorithms**

When the PDETAIL option is not specified, a standard table is produced displaying the iteration number, the value of the objective function at that iteration, and norm of the gradient vector \( \nabla E(\omega) \).

This table is useful for overall convergence behavior. However, unlike the table produced by the PDETAIL option, no information about individual network weights is given.

In the case of sum of squared error, which results from specifying OBJECTIVE=DEV in the NETOPTS statement and ERROR=NORMAL in the TARGET statement, the error function serves as the objective function, and is given by

\[
E(\omega) = \frac{1}{2} \sum_{m=1}^{M} \sum_{k=1}^{K} (y_k^m(x; \omega) - t_k^m)^2
\]

Candidate network weight values \( \omega^* \) that minimize the objective function \( E(\omega) \) satisfy the first order condition

\[ \nabla E(\omega^*) = 0 \]

Hence, a natural convergence criteria is

\[ \| \nabla E(\omega) \|_{\infty} < \epsilon \]

for some small value \( \epsilon \). This is, in fact, convergence criteria for all prop methods. The value of \( \epsilon \) is set by the ABSGCONV= option in the NLOPTIONS statement, with a default value of \( \epsilon = 10^{-8} \). Note that the \( l_\infty \) norm \( \| z \|_{\infty} \) for some vector \( z \) is simply the maximum of the absolute value of the components of \( z \).

The standard table prints the following quantities:

- Iteration \( n \)
- Objective, $E(\omega)$, using current network weight $\omega$
- Max Abs Gradient Element, $\left\| \nabla E(\omega) \right\|_\infty$

When the PDETAIL option is specified, this standard table is still printed. Each line of the standard tables follows the detail lines for each of the weights at each iteration.

**The Standard Backprop Algorithm**

The standard backprop algorithm is a gradient descent with momentum.

At the $n$th iteration, the update is computed as

$$\Delta \omega_{ij}(n) = -\epsilon \frac{\partial E(n)}{\partial \omega_{ij}} + \alpha \Delta \omega_{ij}(n-1)$$

For TECH=BPROP, the PDETAIL option in the TRAIN statement results in the following quantities being printed:

- $\Delta \omega_{ij}(n-1)$ is labeled "Previous Change"
- $\frac{\partial E(n)}{\partial \omega_{ij}}$ is labeled "Gradient"
- $\Delta \omega_{ij}(n)$ is labeled "Current Change"
- $\omega_{ij}(n)$ is labeled "Previous Weight"
- $\omega_{ij}(n+1)$ is labeled "Current Weight"

The learning rate $\epsilon$ and the momentum $\alpha$ are printed at the beginning of the table. These quantities are set by the LEARN= and MOMENTUM= options respectively, with the default values of $\epsilon = 0.1$ and $\alpha = 0.9$.

**The RPROP Algorithm**

The RPROP algorithm (Riedmiller and Braun 1993) is unusual as a descent algorithm in that it does not use the magnitude of the gradient in calculating the weight update. Instead, the signs of the current and previous gradient are used to determine a step size $\Delta_{ij}(n)$ at each iteration.
To prevent oscillations and underflows, the step size $\Delta_{ij} (n)$ is bounded by

$$\Delta_{\text{min}} \leq \Delta_{ij} (n) \leq \Delta_{\text{max}}$$

The value of $\Delta_{\text{max}}$ is set by the MAXLEARN= option with a default value of $10^7$. The value of $\Delta_{\text{min}}$ is set by the MINLEARN= option with a default value of $10^{-7}$. Note that these values are substantially different from the recommendations given in Schiffmann, Joost and Werner, (1994). These new values improved stability and convergence over a wide range of problems.

For each connection weight, an initial stepsize $\Delta_{ij} (0)$ is given a small value. According to Schiffmann, Joost and Werner, (1994), results are not typically dependent on the exact value given $\Delta_{ij} (0)$. PROC NEURAL uses a default initial step size of 0.1 for all weights and is set by the LEARN= option in the TRAIN statement.

At the $n$th iteration, adjust the step size by

$$\Delta_{ij} (n) = \begin{cases} 
\Delta_{ij} (n - 1) u , & \text{if } \frac{\partial E(n)}{\partial \omega_{ij}} \frac{\partial E(n-1)}{\partial \omega_{ij}} > 0 \\
\Delta_{ij} (n - 1) d , & \text{if } \frac{\partial E(n)}{\partial \omega_{ij}} \frac{\partial E(n-1)}{\partial \omega_{ij}} < 0 \\
\Delta_{\text{max}} , & \text{if } \Delta_{ij} (n) > \Delta_{\text{max}} \\
\Delta_{\text{min}} , & \text{if } \Delta_{ij} (n) < \Delta_{\text{min}} 
\end{cases}$$

The factors $u$ and $d$ are the acceleration and deceleration respectively. The values of these factors are set by the ACCELERATE= and DECELERATE= options in the TRAIN statement. The default value for ACCELERATE= 1.2; for DECELERATE= the default value is 0.5.

For TECH=RPROP, the PDETAIL option in the TRAIN statement results in the following quantities being printed:

- $\Delta_{ij} (n - 1)$ is labeled "Previous Step Size"
- $\frac{\partial E(n-1)}{\partial \omega_{ij}}$ is labeled "Previous Gradient"
- $\frac{\partial E(n)}{\partial \omega_{ij}}$ is labeled "Current Gradient"
The Quickprop Algorithm

The Quickprop algorithm (Fahlman 1989) assumes that the error function behaves locally like a parabola, and uses the method of false position to find the minimum of the approximating quadratic. Variations are required to ensure the change is not uphill and to handle cases where the gradient does not change between iterations (causing the false position method to fail).

The quickprop algorithm uses a modified gradient related to the regular gradient by

$$\frac{\partial E^* (n)}{\partial \omega_{ij}} = \frac{\partial E (n)}{\partial \omega_{ij}} + \text{decay} \times \omega_{ij} (n)$$

At the $n$th iteration, the weight update is given by

$$\Delta \omega_{ij} (n) = -\varepsilon (n) \frac{\partial E^* (n)}{\partial \omega_{ij}} + \alpha_{ij} (n) \Delta \omega_{ij} (n - 1)$$

For initialization at $n=1$, we set $\Delta \omega_{ij} (0) = 0$, so the update step becomes a gradient descent:

$$\Delta \omega_{ij} (1) = -\varepsilon (n) \frac{\partial E^* (1)}{\partial \omega_{ij}}$$

At the second and subsequent iterations, $\varepsilon (n)$ and $\alpha_{ij} (n)$ are computed as follows:
\[ \varepsilon(n) = \begin{cases} 
\varepsilon_0, & \text{if } \frac{\partial E^*(n)}{\partial \omega_{ij}} \Delta \omega_{ij} (n - 1) > 0 \\
\varepsilon_0, & \text{if } \Delta \omega_{ij} (n - 1) = 0 \\
0, & \text{otherwise}
\end{cases} \]

Calculation of \( \alpha_{ij}(n) \) first involves evaluation \( \hat{\alpha}_{ij}(n) \), the numerical estimate of the second derivative:

\[ \hat{\alpha}_{ij}(n) = \frac{\frac{\partial E^*(n)}{\partial \omega_{ij}}}{\frac{\partial E^*(n-1)}{\partial \omega_{ij}} - \frac{\partial E^*(n)}{\partial \omega_{ij}}} \]

This second derivative can become large in absolute value or can signal a move "up" the gradient away from a minimum. The following modifications are applied to account for these situations.

\[ \alpha(n) = \begin{cases} 
\alpha_{max}, & \text{if } |\hat{\alpha}_{ij}(n)| > \alpha_{max} \\
\alpha_{max}, & \text{if } \left( \frac{\partial E^*(n-1)}{\partial \omega_{ij}} - \frac{\partial E^*(n)}{\partial \omega_{ij}} \right) \Delta \omega_{ij} (n - 1) > 0 \\
\hat{\alpha}_{ij}(n), & \text{otherwise}
\end{cases} \]

The value of \( \varepsilon_0 \) is set by the LEARN= option in the TRAIN statement, with a default value of \( \varepsilon_0 = 0.1 \). The bound \( \alpha_{max} \) is set by the MAXMOMENTUM= option in the TRAIN statement, with a default value of \( \alpha_{max} = 1.75 \).

For TECH=QPROP, the PDETAIL option in the TRAIN statement results in the following quantities being printed:

- \( \omega_{ij}(n-1) \) is labeled "Previous Weight"
- \( \frac{\partial E(n)}{\partial \omega_{ij}} \) is labeled "Gradient"
- \( \frac{\partial E^*(n)}{\partial \omega_{ij}} \) is labeled "Modified Gradient"
\[ \frac{\partial E^*}{\partial w_{ij}(n-1)} \] is labeled "Previous Modified Gradient"

\[ \Delta w_{ij}(n-1) \] is labeled "Previous Change"

\[ \alpha(n) \] is labeled "Alpha"

\[ \epsilon(n) \] is labeled "Epsilon"

\[ \Delta w_{ij}(n) \] is labeled "Current Change"

\[ \omega_{ij}(n+1) \] is labeled "Current Weight"

References


Examples

The following examples were executed using the HP-UX version 10.20 operating system and the SAS software release 6.12TS045.

Example 1: Developing a Simple Multilayer Perceptron (Rings Data)

Example 2: Developing a Neural Network for a Continuous Target

Example 3: Neural Network Hill-and-Plateau Example (Surf Data)

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Example 1: Developing a Simple Multilayer Perceptron (Rings Data)

Features

- Specifying Input, Hidden, and Output layers
- Scoring a Test Data Set
- Outputting Fit Statistics
- Creating a Classification Table
- Creating Contour Plots of the Posterior Probabilities

This example demonstrates how to develop a multilayer perceptron network with three hidden units. The example training data set is named SAMPSIO.DMDRING (rings data). It contains a categorical target (C = 0, 1, or 2) plus two continuous inputs (X and Y). There are 180 cases in the data set. The SAMPSIO.DMSRING data set is scored using the scoring formula from the trained models.

Both data sets and the SAMPSIO.DMDRING catalog are stored in the sample library.

Program

```
proc gplot data=sampsio.dmdring;
    plot y*x=c /haxis=axis1 vaxis=axis2;
    symbol c=black i=none v=dot;
    symbol2 c=red i=none v=square;
    symbol3 c=green i=none v=triangle;
    axis1 c=black width=2.5  order=(0 to 30 by 5);
    axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
    title 'Plot of the Rings Training Data';
run;

proc neural data=sampsio.dmdring
dmdbcat=sampsio.dmdring
random=789;
    input x y / level=interval id=i;
    target c / id=o level=nominal;
    hidden 3 / id=h;
    prelim 5;
    train;
    score out=out outfit=fit;
    score data=sampsio.dmsring out=gridout;
    title 'MLP with 3 Hidden Units';
run;
```
proc print data=fit noobs label;
    var _aic_ _ase_ _max_ _rfpe_ _misc_ _wrong_; 
    where _name_ = 'OVERALL';
    title2 'Fits Statistics for the Training Data Set';
run;

proc freq data=out;
    tables f_c*i_c;
    title2 'Misclassification Table';
run;

proc gplot data=out;
    plot y*x=i_c /haxis=axis1 vaxis=axis2;
    symbol c=black i=none v=dot;
    symbol2 c=black i=none v=square;
    symbol3 c=black i=none v=triangle;
    axis1 c=black width=2.5 order=(0 to 30 by 5);
    axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
    title2 'Classification Results';
run;

proc gcontour data=gridout;
    plot y*x=p_c1 / pattern ctext=black coutline=gray;
    plot y*x=p_c2 / pattern ctext=black coutline=gray;
    plot y*x=p_c3 / pattern ctext=black coutline=gray;
    pattern v=msolid;
    legend frame;
    title2 'Posterior Probabilities';
run;

proc gcontour data=gridout;
    plot y*x=h1 / pattern ctext=black coutline=gray;
    plot y*x=h2 / pattern ctext=black coutline=gray;
    plot y*x=h3 / pattern ctext=black coutline=gray;
    pattern v=msolid;
    legend frame;
    title2 'Hidden Unit Values';
run;

Output

PROC GLOT of the Rings Training Data
Notice that the target classes are not linearly separable.

**PROC NEURAL: Preliminary Training Output**

This section lists the objective function for each preliminary iteration. The parameter estimates (weights) from the iteration number that has the smallest objective function are passed as input for final training. Because the target is nominal, the error function is set to multiple Bernoulli. Therefore, the objective function that is being minimized is the negative log-likelihood. Iteration number 4 has the smallest objective function.

The 17 initial parameter estimates are also listed in this section of the output.

MLP with 3 Hidden Units

<table>
<thead>
<tr>
<th>Iteration number</th>
<th>Pseudo-random number seed</th>
<th>Objective function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>789</td>
<td>0.35192</td>
</tr>
<tr>
<td>1</td>
<td>761237432</td>
<td>0.20067</td>
</tr>
<tr>
<td>2</td>
<td>1092694980</td>
<td>0.18602</td>
</tr>
<tr>
<td>3</td>
<td>577625332</td>
<td>0.29195</td>
</tr>
<tr>
<td>4</td>
<td>261548896</td>
<td>0.15312</td>
</tr>
</tbody>
</table>

Optimization Start

Parameter Estimates

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Gradient</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 X_H1</td>
<td>-0.399940</td>
<td>-0.00596</td>
<td>X -&gt; H1</td>
</tr>
<tr>
<td>2 Y_H1</td>
<td>-2.215319</td>
<td>0.01670</td>
<td>Y -&gt; H1</td>
</tr>
<tr>
<td>3 X_H2</td>
<td>2.570511</td>
<td>-0.03857</td>
<td>X -&gt; H2</td>
</tr>
<tr>
<td>4 Y_H2</td>
<td>0.672317</td>
<td>-0.02087</td>
<td>Y -&gt; H2</td>
</tr>
<tr>
<td>5 X_H3</td>
<td>2.589547</td>
<td>0.01907</td>
<td>X -&gt; H3</td>
</tr>
<tr>
<td>6 Y_H3</td>
<td>-1.945493</td>
<td>0.00149</td>
<td>Y -&gt; H3</td>
</tr>
<tr>
<td>7 BIAS_H1</td>
<td>2.153111</td>
<td>0.01586</td>
<td>BIAS -&gt; H1</td>
</tr>
<tr>
<td>8 BIAS_H2</td>
<td>2.276595</td>
<td>0.04635</td>
<td>BIAS -&gt; H2</td>
</tr>
<tr>
<td>9 BIAS_H3</td>
<td>-2.243021</td>
<td>0.00979</td>
<td>BIAS -&gt; H3</td>
</tr>
<tr>
<td>10 H1_C1</td>
<td>5.688000</td>
<td>0.00208</td>
<td>H1 -&gt; C1</td>
</tr>
<tr>
<td>11 H2_C1</td>
<td>5.828867</td>
<td>0.000806</td>
<td>H2 -&gt; C1</td>
</tr>
</tbody>
</table>
Value of Objective Function = 0.1531232148

Levenberg-Marquardt Termination Criteria

This section lists the termination criteria for the Levenberg-Marquardt optimization.

MLP with 3 Hidden Units

Levenberg-Marquardt Optimization

*** Termination Criteria ***
Minimum Iterations ........................................ -1
Maximum Iterations ........................................ 100
Maximum Function Calls ..................................... 2147483647
Maximum CPU Time ........................................... 604800
ABSGCONV Gradient Criterion ............................... 0.0000100
GCONV Gradient Criterion ................................... 1E-8
GCONV2 Gradient Criterion .................................... 0
ABSFCONV Function Criterion ............................... 0
FCONV Function Criterion .................................... 0.0001000
FCONV2 Function Criterion .................................... 0
FSIZE Parameter .............................................. 0
ABSXCONV Parameter Change Criterion .................. 0
XCONV Parameter Change Criterion ....................... 0
XSIZE Parameter ............................................. 0
ABSCONV Function Criterion ................................. -1.341E154

*** Other Control Parameters ***
Trust Region Initial Radius Factor ......................... 1.00000
Singularity Tolerance (SINGULAR) ......................... 1E-8

Levenberg-Marquardt Iteration Log

At the start of optimization, the procedure lists the number of active constraints, the current value of the objective function, the maximum gradient element, and the radius. The iteration history includes the following:

- the iteration number (iter)
- the number of iteration restarts (rest)
- the number of active constraints (act)
- the value of the objective function (optcrit)
- the difference between the adjacent objective functions (difcrit)
- the maximum of the absolute (projected) gradient components (maxgrad)
- the value of lambda
- the value of rho

The optimization results section lists information specific for the optimization technique.

MLP with 3 Hidden Units

Levenberg-Marquardt Optimization
Scaling Update of More (1978)
Number of Parameter Estimates 17

Optimization Start: Active Constraints= 0  Criterion= 0.153
Maximum Gradient Element = 0.046 Radius = 1.000

<table>
<thead>
<tr>
<th>Iter rest</th>
<th>nfun</th>
<th>act</th>
<th>optcrit</th>
<th>dicroit</th>
<th>maxgrad</th>
<th>lambda</th>
<th>rho</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
<td>0.1007</td>
<td>0.0524</td>
<td>0.0708</td>
<td>0.0138</td>
<td>0.582</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>9</td>
<td>0.0883</td>
<td>0.0124</td>
<td>0.0589</td>
<td>0.417</td>
<td>0.330</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>15</td>
<td>0.0546</td>
<td>0.0336</td>
<td>0.0493</td>
<td>0.0093</td>
<td>0.674</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>19</td>
<td>0.0255</td>
<td>0.0291</td>
<td>0.0726</td>
<td>0.0015</td>
<td>0.667</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>21</td>
<td>0.0233</td>
<td>0.00227</td>
<td>0.136</td>
<td>0</td>
<td>0.096</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>23</td>
<td>0.003321</td>
<td>0.0200</td>
<td>0.0154</td>
<td>0</td>
<td>0.897</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>24</td>
<td>0.000742</td>
<td>0.00258</td>
<td>0.00889</td>
<td>5.65E-7</td>
<td>0.816</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>25</td>
<td>0.000283</td>
<td>0.000459</td>
<td>0.00683</td>
<td>0</td>
<td>0.636</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>0.000632</td>
<td>0.0022</td>
<td>0.789</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>27</td>
<td>6.4334E-6</td>
<td>0.000057</td>
<td>0.00018</td>
<td>0</td>
<td>0.908</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>28</td>
<td>1.1769E-6</td>
<td>5.257E-6</td>
<td>0.00006</td>
<td>0</td>
<td>0.823</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>29</td>
<td>2.6787E-7</td>
<td>9.09E-7</td>
<td>0.00002</td>
<td>0</td>
<td>0.776</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>30</td>
<td>1.2592E-7</td>
<td>1.42E-7</td>
<td>0.00002</td>
<td>0</td>
<td>0.532</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>32</td>
<td>1.6879E-8</td>
<td>1.09E-7</td>
<td>2.89E-6</td>
<td>308E-15</td>
<td>0.869</td>
</tr>
</tbody>
</table>

Optimization Results: Iterations = 14 Function Calls = 33 Jacobian Calls = 15
Active Constraints = 0 Criterion = 1.6879056E-8
Maximum Gradient Element = 2.8916E-6 Lambda = 3.08E-13 Rho = 0.8685 Radius = 26.94

NOTE: ABSGCONV convergence criterion satisfied.

Parameter Estimates (MLP Weights)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Gradient</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 X_H1</td>
<td>-0.377915</td>
<td>1.2972E-6</td>
<td>X --&gt; H1</td>
</tr>
<tr>
<td>2 Y_H1</td>
<td>-0.665105</td>
<td>2.8916E-6</td>
<td>Y --&gt; H1</td>
</tr>
<tr>
<td>3 X_H2</td>
<td>1.720212</td>
<td>8.9422E-8</td>
<td>X --&gt; H2</td>
</tr>
<tr>
<td>4 Y_H2</td>
<td>-0.043669</td>
<td>1.63254E-7</td>
<td>Y --&gt; H2</td>
</tr>
<tr>
<td>5 X_H3</td>
<td>0.487788</td>
<td>-3.1918E-7</td>
<td>X --&gt; H3</td>
</tr>
<tr>
<td>6 Y_H3</td>
<td>-0.801489</td>
<td>-5.7188E-7</td>
<td>Y --&gt; H3</td>
</tr>
<tr>
<td>7 BIAS_H1</td>
<td>0.875418</td>
<td>1.87012E-6</td>
<td>BIAS --&gt; H1</td>
</tr>
<tr>
<td>8 BIAS_H2</td>
<td>1.340638</td>
<td>-1.1246E-7</td>
<td>BIAS --&gt; H2</td>
</tr>
<tr>
<td>9 BIAS_H3</td>
<td>-0.649604</td>
<td>-7.5073E-8</td>
<td>BIAS --&gt; H3</td>
</tr>
<tr>
<td>10 H1_C1</td>
<td>388.611382</td>
<td>-1.7823E-9</td>
<td>H1 --&gt; C1</td>
</tr>
<tr>
<td>11 H2_C1</td>
<td>289.584211</td>
<td>-1.6285E-9</td>
<td>H2 --&gt; C1</td>
</tr>
<tr>
<td>12 H3_C1</td>
<td>-309.435371</td>
<td>1.04029E-9</td>
<td>H3 --&gt; C1</td>
</tr>
<tr>
<td>13 H1_C2</td>
<td>177.046552</td>
<td>-1.9849E-9</td>
<td>H1 --&gt; C2</td>
</tr>
<tr>
<td>14 H2_C2</td>
<td>118.043485</td>
<td>1.37679E-8</td>
<td>H2 --&gt; C2</td>
</tr>
<tr>
<td>15 H3_C2</td>
<td>-149.150661</td>
<td>-1.1747E-8</td>
<td>H3 --&gt; C2</td>
</tr>
<tr>
<td>16 BIAS_C1</td>
<td>-580.555136</td>
<td>-2.0973E-9</td>
<td>BIAS --&gt; C1</td>
</tr>
<tr>
<td>17 BIAS_C2</td>
<td>-213.290409</td>
<td>1.39802E-8</td>
<td>BIAS --&gt; C2</td>
</tr>
</tbody>
</table>

Value of Objective Function = 1.6879056E-8

List Report of Selected Variables in the Score OUTFIT= Data Set

The example PROC PRINT report of the OUTFIT = data set lists selected fit statistics for the training data. By default, the OUTFIT data set contains two observations. These observations are distinguished by the value of the _NAME_ variable:

- _NAME_ = ‘Target Variable Name’
- _NAME_ = ‘OVERALL’

Because a WHERE statement was used to select only values of _NAME_ = ‘OVERALL’, this report contains a single observation.
Notice that the MLP network with 3 hidden units correctly classifies all cases in the training data set.

**MLP with 3 Hidden Units**

**Fits Statistics for the Training Data Set**

<table>
<thead>
<tr>
<th></th>
<th>Train: Akaike's Information Criterion</th>
<th>Train: Average Squared Error</th>
<th>Train: Maximum Absolute Error</th>
<th>Train: Final Prediction Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>34.0000</td>
<td>3.7039E-14</td>
<td>.0000030981</td>
<td>.00000020177</td>
</tr>
</tbody>
</table>

**Train: Misclassification Rate.**

<table>
<thead>
<tr>
<th></th>
<th>Train: Number of Wrong Classifications.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

**PROC FREQ Misclassification Table for the Training Data**

**MLP with 3 Hidden Units Misclassification Table**

<table>
<thead>
<tr>
<th>TABLE OF F_C BY I_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_C(From: C)</td>
</tr>
<tr>
<td>Frequency</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Total</td>
</tr>
</tbody>
</table>

**PROC GPLOT Plot of the Classification Results**
PROC GGONTOUR Plots of the Posterior Probabilities

The legend at the bottom of the chart identifies the target level.

PROC GGONTOUR Plots of the Posterior Probabilities

The legend at the bottom of the chart identifies the target level.
PROC GCONTOUR Plots of the Hidden Units

MLP with 3 Hidden Units
Posterior Probabilities

Predicted: C = 2

Predicted: C = 3

PROC GCONTOUR Plots of the Hidden Units
PROC GLOT creates a scatter plot of the Rings training data.

```
proc gplot data=sampsio.dmdring;
  plot y*x=c /haxis=axis1 vaxis=axis2;
  symbol c=black i=none v=dot;
  symbol2 c=red i=none v=square;
  symbol3 c=green i=none v=triangle;
  axis1 c=black width=2.5 order=(0 to 30 by 5);
  axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
  title 'Plot of the Rings Training Data';
run;
```
The PROC NEURAL statement invokes the procedure. The DATA= option identifies the training data set that is used to fit the model. The DMDBCAT= option identifies the DMDB catalog. The RANDOM= option specifies the random number seed.

```plaintext
proc neural data=sampsio.dmdring
    dmdbcat=sampsio.dmdring
    random=789;
```
The INPUT statement specifies an interval input layer. The LEVEL= option specifies the measurement level. The ID= option specifies an identifier for the interval input layer.

    input x y / level=interval id=i;
The TARGET statement defines an output layer. The output layer computes predicted values and compares those predicted values with the value of the target variable. The ID= option specifies an identifier for the output layer. The LEVEL= option specifies the target measurement level. By default, for nominal targets the combination function is set to linear, the activation function is set to mlogistic, and the error function is set to mbernoulli.

target c / id=o level=nominal;
The HIDDEN statement defines the number of hidden units that are used to perform the internal computations. By default, the input units are connected to each hidden unit and each hidden unit is connected to the output unit. The ID= option specifies an identifier for the hidden unit.

hidden 3 / id=h;
The PRELIM statement causes the procedure to search for the best starting weights for subsequent training. The integer value of 5 specifies to use 5 preliminary runs. The weights from the seed with the smallest objective function among all runs is chosen. Preliminary training may help prevent the network from converging in a local minima.

prelim 5;
The TRAIN statement trains the network in order to find the best weights (parameter estimates) that accurately reflect the training data.

train;
The first SCORE statement scores the training data. The OUT= option identifies the output data set that contains outputs. The OUTFIT= option identifies the output data set that contains fit statistics.

```plaintext
score out=out outfit=fit;
```
The second SCORE statement specifies the score data set that you want to score in conjunction with training.

```plaintext
score data=sampsio.dmsring out=gridout;
title 'MLP with 3 Hidden Units';
run;
```
PROC PRINT lists selected training fit statistics.

proc print data=fit noobs label;
    var _aic_ _ase_ _max_ _rfpe_ _misc_ _wrong_;
    where _name_ = 'OVERALL';
    title2 'Fits Statistics for the Training Data Set';
run;
PROC FREQ creates a misclassification table for the training data. The F_C variable is the actual target value for each case and the I_C variable is the target value into which the case is classified.

```sas
proc freq data=out;
  tables f_c*i_c;
  title2 'Misclassification Table';
run;
```
PROC GPLOT plots the classification results for the training data.

proc gplot data=out;
    plot y*x=i_c /haxis=axis1 vaxis=axis2;
    symbol c=black i=none v=dot;
    symbol2 c=black i=none v=square;
    symbol3 c=black i=none v=triangle;
    axis1 c=black width=2.5 order=(0 to 30 by 5);
    axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
    title2 'Classification Results';
run;
PROC GCONTOUR produces a contour plot of the posterior probabilities for the scored data set.

proc gcontour data=gridout;
   plot y*x=p_c1 / pattern ctext=black coutline=gray;
   plot y*x=p_c2 / pattern ctext=black coutline=gray;;
   plot y*x=p_c3 / pattern ctext=black coutline=gray;;
   pattern v=msolid;
   legend frame;
   title2 'Posterior Probabilities';
run;
Example 2: Developing a Neural Network for a Continuous Target

Features

- Specifying Input, Hidden, and Output Layers
- Defining Direct Connections
- Scoring Data with the Score Statement
- Outputting Fit Statistics

This example demonstrates how to develop a neural network model for a continuous target. A simple multilayer perceptron architecture is employed with one hidden unit and direct connections. The example DMDB training data set SAMSIO.DMBASE (baseball data set) contains performance measures and salary levels for regular hitters and leading substitute hitters in major league baseball for the year 1986 (Collier 1987). There is one observation per hitter. The continuous target variable is log of salary (logsalar).

Prior to fitting the neural network model, the number of original model inputs was reduced based on a preliminary stepwise PROC DMREG run. The input set from the model with the smallest SBC (Schwarz's Bayesian Criterion) is used as input to the network. The output from the PROC DMREG analysis can be found in the PROC DMREG chapter, "Example 2. Performing a Stepwise OLS Regression".

The SAMSIO.DMTBASE data set is a test data set that is scored using the scoring formula from the trained model. The SAMSIO.DMBASE and SAMSIO.DMTBASE data sets and the SAMPIO.DMDBASE catalog are stored in the sample library.

Program

```sas
proc dmreg data=sampsio.dmdbase dmdbcat=sampsio.dmdbase
testdata=sampsio.dmtbase outest=regest;
class league division position;
model logsalar = no_atbat no_hits no_home no_runs no_rbi no_bb
    yr_major cr_atbat cr_hits cr_home cr_runs
cr_rbi cr_bb league division position no_outs
    no_assts no_error /
    error=normal selection=stepwise
    slentry=0.25 slstay=0.25 choose=sbc;
    title1 'Preliminary DMDREG Stepwise Selection';
run;
```

```sas
proc neural data=sampsio.dmdbase
dmdbcat=sampsio.dmdbase
random=12345;
```

```sas
input cr_hits no_hits no_outs no_error no_bb
    / level=interval id=int;
input division / level=nominal id=nom;
```

```sas
hidden 1 / id=hu;
```
target logsalar /  
    level=interval  
    id=tar ;

connect int tar;  
connect nom tar;  
connect int hu;  
connect nom hu;  
connect hu tar;

prelim 10;

train;

score data=sampsio.dmtbase outfit=netfit

out=netout(rename=(p_logsal=predict r_logsal=residual));  
title 'NN:1 Hidden Unit, Direct Connections,  
    and Reduced Input Set';
run;

proc print data=netfit noobs label;  
    where _name_ = 'LOGSALAR';  
    var _iter_ _pname_ _tmse_ _trmse_ _tmax_;  
    title 'Partial Listing of the Score OUTFIT= Data Set';
run;

proc gplot data=netout;  
    plot logsalar*predict / haxis=axis1 vaxis=axis2;  
        symbol c=black i=none v=dot h=3 pct;  
        axis1 c=black width=2.5;  
        axis2 c=black width=2.5;  
    title 'Diagnostic Plots for the Scored Test Baseball Data';
    plot residual*predict / haxis=axis1 vaxis=axis2;
run;
quit;

Output
Preliminary Training Output

This section lists the objective function for each preliminary iteration run. The weights from the iteration number that has the smallest objective function are passed as input for final training. Because the target is continuous, the error function is set to normal. Therefore, the objective function that is being minimized is the least squares error. Iteration number 0 has the smallest objective function. The parameter estimates for iteration number 0 are also listed in this section of the output.

### NN:1 Hidden Unit, Direct Connections, and Reduced Input Set

<table>
<thead>
<tr>
<th>Iteration number</th>
<th>Pseudo-random number seed</th>
<th>Objective function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12345</td>
<td>0.14536</td>
</tr>
<tr>
<td>1</td>
<td>845250737</td>
<td>0.14536</td>
</tr>
<tr>
<td>2</td>
<td>111329849</td>
<td>0.28210</td>
</tr>
<tr>
<td>3</td>
<td>1696138964</td>
<td>0.18477</td>
</tr>
<tr>
<td>4</td>
<td>1038363354</td>
<td>0.14536</td>
</tr>
<tr>
<td>5</td>
<td>1071492826</td>
<td>0.18942</td>
</tr>
<tr>
<td>6</td>
<td>117568856</td>
<td>0.14536</td>
</tr>
<tr>
<td>7</td>
<td>1792608669</td>
<td>0.30494</td>
</tr>
<tr>
<td>8</td>
<td>1691324682</td>
<td>0.18331</td>
</tr>
<tr>
<td>9</td>
<td>2114796956</td>
<td>0.28605</td>
</tr>
</tbody>
</table>

### Optimization Start

Parameter Estimates

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Gradient Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR_HITS_</td>
<td>5.240279</td>
<td>-9.8369E-7 CR_HITS -&gt; HU1</td>
</tr>
<tr>
<td>NO_HITS_</td>
<td>-0.575284</td>
<td>0.0000118 NO_HITS -&gt; HU1</td>
</tr>
<tr>
<td>NO_OUTS_</td>
<td>-0.298484</td>
<td>1.91236E-6 NO_OUTS -&gt; HU1</td>
</tr>
<tr>
<td><em>DUP1</em></td>
<td>0.019049</td>
<td>0.0000127 NO_ERROR -&gt; HU1</td>
</tr>
<tr>
<td>NO_BB_HU</td>
<td>-0.097201</td>
<td>8.015E-6 NO_BB -&gt; HU1</td>
</tr>
<tr>
<td><em>DUP2</em></td>
<td>-0.159479</td>
<td>3.44079E-6 DIVISIONEAST -&gt; HU1</td>
</tr>
<tr>
<td>BIAS_HU1</td>
<td>4.099012</td>
<td>1.7976E-6 BIAS -&gt; HU1</td>
</tr>
<tr>
<td><em>DUP3</em></td>
<td>0.114451</td>
<td>-8.5694E-8 CR_HITS -&gt; LOGSALAR</td>
</tr>
<tr>
<td><em>DUP4</em></td>
<td>0.186707</td>
<td>1.18117E-8 NO_OUTS -&gt; LOGSALAR</td>
</tr>
<tr>
<td><em>DUP5</em></td>
<td>0.156401</td>
<td>2.6625E-9 NO_ERRORS -&gt; LOGSALAR</td>
</tr>
<tr>
<td><em>DUP6</em></td>
<td>-0.042491</td>
<td>8.96911E-8 NO_ERROR -&gt; LOGSALAR</td>
</tr>
<tr>
<td>NO_BB_LO</td>
<td>0.151510</td>
<td>4.8488E-9 NO_BB -&gt; LOGSALAR</td>
</tr>
<tr>
<td><em>DUP7</em></td>
<td>0.055166</td>
<td>3.48459E-8 DIVISIONEAST -&gt; LOGSALAR</td>
</tr>
<tr>
<td>HU1_LOGS</td>
<td>0.839297</td>
<td>-9.8363E-8 HU1 -&gt; LOGSALAR</td>
</tr>
<tr>
<td>BIAS_LOG</td>
<td>5.490837</td>
<td>7.82267E-8 BIAS -&gt; LOGSALAR</td>
</tr>
</tbody>
</table>

Value of Objective Function = 0.1453574646

Levenberg-Marquardt Termination Criteria
Levenberg-Marquardt Optimization

*** Termination Criteria ***
Minimum Iterations ........................................ -1
Maximum Iterations ........................................... 100
Maximum Function Calls ................................. 2147483647
Maximum CPU Time ......................................... 604800
ABSGCONV Gradient Criterion .................. 0.00000001
GCONV Gradient Criterion ....................... 1E-8
GCONV2 Gradient Criterion .................. 0
ABSFCONV Function Criterion .................. 0
FCONV Function Criterion ................... 0.0001000
FCONV2 Function Criterion .................. 0
FSIZE Parameter ........................................ 0
ABSXCONV Parameter Change Criterion .......... 0
XCONV Parameter Change Criterion ............ 0
XSIZE Parameter ........................................ 0
ABSCONV Function Criterion ....... -1.341E154

*** Other Control Parameters ***
Trust Region Initial Radius Factor ........ 1.00000
Singularity Tolerance (SINGULAR) ........ 1E-8

Levenberg-Marquardt Iteration Log

NN: 1 Hidden Unit, Direct Connections, and Reduced Input Set

Levenberg-Marquardt Optimization
Scaling Update of More (1978)
Number of Parameter Estimates 15

Optimization Start: Active Constraints= 0 Criterion= 0.145
Maximum Gradient Element= 0.000 Radius= 1.000

Optimization Results: Iterations= 1 Function Calls= 3 Jacobian Calls= 2
Active Constraints= 0 Criterion= 0.14535746
Maximum Gradient Element= 8.81033E-6 Lambda= 0 Rho= 1.694 Radius= 0.002375

NOTE: ABSGCONV convergence criterion satisfied.

Parameter Estimates

Optimization Results
Parameter Estimates

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Gradient</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 CR_HITS_</td>
<td>5.242352</td>
<td>-6.7156E-7</td>
<td>CR_HITS -&gt; HU1</td>
</tr>
<tr>
<td>2 NO_HITS_</td>
<td>-0.575574</td>
<td>8.16507E-6</td>
<td>NO_HITS -&gt; HU1</td>
</tr>
<tr>
<td>3 NO_OUTS_</td>
<td>-0.298618</td>
<td>1.3324E-6</td>
<td>NO_OUTS -&gt; HU1</td>
</tr>
<tr>
<td>4 <em>DUP1</em></td>
<td>0.018999</td>
<td>8.81033E-6</td>
<td>NO_ERROR -&gt; HU1</td>
</tr>
</tbody>
</table>
List Report of Selected Variables in the Score OUTFIT= Data Set

The example PROC PRINT report of the OUTFIT= data set contains selected summary statistics from the scored training data set.

Partial Listing of the Score OUTFIT= Data Set

<table>
<thead>
<tr>
<th><em>ITER</em></th>
<th><em>PNAME</em></th>
<th>Test: Mean</th>
<th>Test: Root Mean</th>
<th>Test: Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P_LOGSAL</td>
<td>0.15595</td>
<td>0.39491</td>
<td>1.60237</td>
</tr>
</tbody>
</table>

Diagnostic Plots for the Scored Test Baseball Data

Plot of the log of salary versus the predicted log of salary.
Plot of the residual values versus the predicted log of salary.
The preliminary PROC DMREG run selects the reduced input set.

```plaintext
proc dmreg data=sampsio.dmdbase dmdbcat=sampsio.dmdbase
testdata=sampsio.dmtbase outest=regest;
class league division position;
model logsalar = no_atbat no_hits no_home no_runs no_rbi no_bb
     yr_major cr_atbat cr_hits cr_home cr_runs
cr_rbi cr_bb league division position no_outs
     no_assts no_error /
     error=normal selection=stepwise
     slentry=0.25 slstay=0.25 choose=sbc;

title1 'Preliminary DMDREG Stepwise Selection';
run;
```
The PROC NEURAL statement invokes the procedure. The DATA= option identifies the training data set that is used to fit the model. The DMBDCAT= option identifies the training catalog. The RANDOM= option specifies the seed that is used to set the random initial weights.

```sas
proc neural data=sampsio.dmdbase
dmdbcat=sampsio.dmdbase
random=12345;
```
The INPUT statements specifies the input layers. There are separate input layers for the interval and nominal inputs. The LEVEL= option specifies the measurement level. The ID= option specifies an identifier for each input layer.

```
input cr_hits no_hits no_outs no_error no_bb
   / level=interval id=int;
input division / level=nominal id=nom;
```
The HIDDEN statement sets the number of hidden units. The ID= option specifies an identifier for the hidden layer. By default, the combination function is set to linear and the activation function is set to hyperbolic tangent.

    hidden 1 / id=hu;
The TARGET statement defines an output layer. The output layer computes predicted values and compares those predicted values with the value of the target variable (LOGSALAR). The LEVEL= option specifies the target measurement level. The ID= option specifies an identifier for the output layer. By default, the combination function is set to linear, the activation function is set to the identity, and the error function is set to normal for continuous targets.

```
target logsalar /
   level=interval
   id=tar ;
```
The CONNECT statements specify how to connect the layers. The id-list specifies the identifier of two or more layers to connect. In this example, each input unit is connected to the hidden unit and to the output unit, and the hidden unit is connected to the output unit.

```
connect int tar;
connect nom tar;
connect int hu;
connect nom hu;
connect hu tar;
```
The PRELIM statement does preliminary training using 10 different sets of initial weights. The weights from the preliminary run with the smallest objective function among all runs are retained for subsequent training when using the TRAIN statement. Preliminary training may help prevent the network from converging to a bad local minima.

prelim 10;
The TRAIN statement trains the network in order to find the best weights (parameter estimates) to fit the training data. By default, the Levenberg-Marquardt optimization technique is used for small least squares networks, such as the one in this example.

train;
The SCORE statement creates output data sets. The DATA= option specifies the data set you want to score. The OUTFIT= option creates a data set containing fit statistics.

score data=sampsio.dmtbase outfit=netfit
The OUT=option identifies the output data for predicted values and residuals. The RENAME= option renames the variables in the OUT= data set containing predicted values and residuals.

```plaintext
out=netout(rename=(p_logsal=predict r_logsal=residual));
title 'NN:1 Hidden Unit, Direct Connections, and Reduced Input Set';
run;
```
PROC PRINT lists selected variables from the OUTFIT= data set.

proc print data=netfit noobs label;
   where _name_ = 'LOGSALAR';
   var _iter_ _pname_ _tmse_ _trmse_ _tmax_;
   title 'Partial Listing of the Score OUTFIT= Data Set';
run;
PROC GPLOT plots diagnostic plots for the scored data set.

proc gplot data=netout;
  plot logsalar*predict / haxis=axis1 vaxis=axis2;
    symbol c=black i=none v=dot h=3 pct;
  axis1 c=black width=2.5;
  axis2 c=black width=2.5;
  title 'Diagnostic Plots for the Scored Test Baseball Data';
  plot residual*predict / haxis=axis1 vaxis=axis2;
run;
quit;
Example 3: Neural Network Hill-and-Plateau Example (Surf Data)

This example demonstrates how to develop a neural network model for a continuous target. A multilayer perceptron architecture is employed with 3 and then 30 hidden units. The example test data set is named SAMPSIO.DMDSURF (Surf Data). It contains the interval target HIPL, and two interval inputs X1 and X2. The data set was artificially generated as a surface containing a hill and a plateau. The hill is easily learned by an RBF architecture. The plateau is easily learned by an MLP architecture.

The SAMPSIO.DMTSURF data set is a test data set that is scored using the scoring formula from the trained model. The SAMPSIO.DMDSURF and SAMPSIO.DMTSURF data sets and the SAMPIO.DMDSURF catalog are stored in the sample library.

Program: 3 Hidden Units

```sas
proc g3d data=sampsio.dmdsurf; plot x2*x1=hipl
   / grid side ctop=blue caxis=green
certext=black zmin=-1.5 zmax=1.5;
title 'Plot of the Surf Training Data';
footnote 'Hill Plateau Response Surface';
run;

title 'Hill & Plateau Data';
%let hidden=3;

proc neural data=sampsio.dmdsurf
   dmdbcat=sampsio.dmdsurf
   random=789;
   input x1 x2 / id=i;
target hipl / id=o;
   hidden &hidden / id=h;
   prelim 10;
   train maxiter=1000 outest=mlpest;
score data=sampsio.dmtsurf out=mlpout outfit=mlpfit;
title2 "MLP with &hidden Hidden Units";
run;

proc print data=mlpfit noobs label;
   var _tase_ _tasel_ _taseu_;
   where _name_ = 'HIPL';
title3 'Fit Statistics for the Test Data';
run;

proc gcontour data=mlpout;
   plot x2*x1=p_hipl / pattern ctext=black coutline=gray;
pattern v=msolid;
legend frame;
title3 'Predicted Values';
footnote;
run;
```
proc g3d data=mlpout;
   plot x2*x1=p_hipl / grid side ctop=blue
caxis=green ctext=black
   zmin=-1.5 zmax=1.5;
run;

Output: 3 Hidden Units

PROC GCONTOUR Plot of the Surf Training Data

Plot of the Surf Training Data

Hill Plateau Response Surface

PROC Neural Output

Hill & Plateau Data
MLP with 3 Hidden Units

<table>
<thead>
<tr>
<th>Iteration number</th>
<th>Pseudo-random number seed</th>
<th>Objective function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>789</td>
<td>0.04023</td>
</tr>
<tr>
<td>1</td>
<td>761237432</td>
<td>0.03673</td>
</tr>
<tr>
<td>2</td>
<td>1092694980</td>
<td>0.04187</td>
</tr>
<tr>
<td>3</td>
<td>577625332</td>
<td>0.05216</td>
</tr>
<tr>
<td>4</td>
<td>261548896</td>
<td>0.03593</td>
</tr>
<tr>
<td>5</td>
<td>616485149</td>
<td>0.04030</td>
</tr>
<tr>
<td>6</td>
<td>692687363</td>
<td>0.03973</td>
</tr>
<tr>
<td>7</td>
<td>1510804008</td>
<td>0.04141</td>
</tr>
<tr>
<td>8</td>
<td>1385020003</td>
<td>0.03583</td>
</tr>
<tr>
<td>9</td>
<td>1070679467</td>
<td>0.04167</td>
</tr>
<tr>
<td>Parameter</td>
<td>Estimate</td>
<td>Gradient</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>1 X1_H1</td>
<td>6.064004</td>
<td>-2.6744E-6</td>
</tr>
<tr>
<td>2 X2_H1</td>
<td>0.880274</td>
<td>5.28411E-6</td>
</tr>
<tr>
<td>3 X1_H2</td>
<td>0.048809</td>
<td>5.37355E-7</td>
</tr>
<tr>
<td>4 X2_H2</td>
<td>-4.988958</td>
<td>6.03873E-7</td>
</tr>
<tr>
<td>5 X1_H3</td>
<td>-5.916343</td>
<td>-6.0486E-6</td>
</tr>
<tr>
<td>6 X2_H3</td>
<td>0.730854</td>
<td>-0.0000207</td>
</tr>
<tr>
<td>7 BIAS_H1</td>
<td>-3.004936</td>
<td>-0.0000105</td>
</tr>
<tr>
<td>8 BIAS_H2</td>
<td>1.791982</td>
<td>2.17127E-6</td>
</tr>
<tr>
<td>9 BIAS_H3</td>
<td>0.864474</td>
<td>-0.0000126</td>
</tr>
<tr>
<td>10 H1_HIPL</td>
<td>-0.261095</td>
<td>0.0000179</td>
</tr>
<tr>
<td>11 H2_HIPL</td>
<td>-0.484358</td>
<td>-0.0000107</td>
</tr>
<tr>
<td>12 H3_HIPL</td>
<td>-0.265490</td>
<td>0.0000123</td>
</tr>
<tr>
<td>13 BIAS_HIP</td>
<td>-0.490112</td>
<td>-0.0000182</td>
</tr>
</tbody>
</table>

Value of Objective Function = 0.0358271766
Optimization Results: Iterations= 10 Function Calls= 12 Jacobian Calls= 11
Active Constraints= 0  Criterion= 0.035825744
Maximum Gradient Element= 0.0000127206 Lambda= 0 Rho= 1.864 Radius= 0.00436

NOTE:  FCONV convergence criterion satisfied.

Optimization Results
Parameter Estimates
------------------------------------------------------------------------------
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Gradient Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 X1_H1</td>
<td>6.020084</td>
<td>-4.8867E-7 X1 -&gt; H1</td>
</tr>
<tr>
<td>2 X2_H1</td>
<td>0.823365</td>
<td>0.0000111 X2 -&gt; H1</td>
</tr>
<tr>
<td>3 X1_H2</td>
<td>0.049663</td>
<td>-3.2277E-8 X1 -&gt; H2</td>
</tr>
<tr>
<td>4 X2_H2</td>
<td>-4.986906</td>
<td>2.1975E-8 X2 -&gt; H2</td>
</tr>
<tr>
<td>5 X1_H3</td>
<td>-5.848915</td>
<td>-1.8967E-6 X1 -&gt; H3</td>
</tr>
<tr>
<td>6 X2_H3</td>
<td>0.767294</td>
<td>-0.0000127 X2 -&gt; H3</td>
</tr>
<tr>
<td>7 BIAS_H1</td>
<td>-3.013469</td>
<td>5.5999E-7 BIAS -&gt; H1</td>
</tr>
<tr>
<td>8 BIAS_H2</td>
<td>1.791192</td>
<td>1.3226E-7 BIAS -&gt; H2</td>
</tr>
<tr>
<td>9 BIAS_H3</td>
<td>0.889276</td>
<td>-3.0043E-6 BIAS -&gt; H3</td>
</tr>
<tr>
<td>10 H1_HIPL</td>
<td>-0.262306</td>
<td>4.5141E-7 H1 -&gt; HIPL</td>
</tr>
<tr>
<td>11 H2_HIPL</td>
<td>-0.484458</td>
<td>1.7993E-8 H2 -&gt; HIPL</td>
</tr>
<tr>
<td>12 H3_HIPL</td>
<td>-0.266660</td>
<td>3.3560E-7 H3 -&gt; HIPL</td>
</tr>
<tr>
<td>13 BIAS_HIP</td>
<td>-0.490183</td>
<td>-3.1205E-7 BIAS -&gt; HIPL</td>
</tr>
</tbody>
</table>

Value of Objective Function = 0.0358257445

PROC PRINT Report of Selected Fit Statistics for the Scored Test Data Set

Hill & Plateau Data
MLP with 3 Hidden Units
Fit Statistics for the Test Data

<table>
<thead>
<tr>
<th>Test: Average Squared Error.</th>
<th>Test: Lower 95% Conf.</th>
<th>Test: Upper 95% Conf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.036830</td>
<td>0.028999</td>
<td>0.045583</td>
</tr>
</tbody>
</table>

GCONTOUR Plot of the Predicted Values
Program: 30 Hidden Units

proc neural data=sampsio.dmdsurf
dmdbcat=sampsio.dmdsurf
random=789;
input x1 x2 / id=i;
target hipl / id=o;

hidden &hidden / id=h;
prelim 10;
train maxiter=1000 outest=mlpept2;
score data=sampsio.dmtsurf out=mlpout2 outfit=mlpfit2;
title2 "MLP with &hidden Hidden Units";
run;

proc print data=mlpfit2 noobs label;
var _tase_ _tasel_ _taseu_; where _name_ ='HIPL'; title3 'Fit Statistics for the Test Data';
run;

proc gcontour data=mlpout2;
plot x2*x1=p_hipl / pattern ctext=black coutline=gray;
pattern v=msolid;
legend frame;
title3 'Predicted Values';
footnote;
run;

proc g3d data=mlpout2;
plot x2*x1=p_hipl / grid side ctop=blue
caxis=green ctext=black
zmin=-1.5 zmax=1.5;
run;

Output: 30 Hidden Units

Preliminary Iteration History

<table>
<thead>
<tr>
<th>Iteration number</th>
<th>Pseudo-random number seed</th>
<th>Objective function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>789</td>
<td>0.03400</td>
</tr>
<tr>
<td>1</td>
<td>1095609817</td>
<td>0.04237</td>
</tr>
<tr>
<td>2</td>
<td>924680074</td>
<td>0.03808</td>
</tr>
<tr>
<td>3</td>
<td>1369944093</td>
<td>0.04887</td>
</tr>
<tr>
<td>4</td>
<td>1527099570</td>
<td>0.04153</td>
</tr>
<tr>
<td>5</td>
<td>430173087</td>
<td>0.03480</td>
</tr>
<tr>
<td>6</td>
<td>739241177</td>
<td>0.03857</td>
</tr>
<tr>
<td>7</td>
<td>321367798</td>
<td>0.02955</td>
</tr>
<tr>
<td>8</td>
<td>58127801</td>
<td>0.04378</td>
</tr>
<tr>
<td>9</td>
<td>1974465768</td>
<td>0.02971</td>
</tr>
</tbody>
</table>

Optimization Start

Parameter Estimates

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Gradient</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1_H1</td>
<td>-0.467800</td>
<td>0.0000599</td>
<td>X1 -&gt; H1</td>
</tr>
<tr>
<td>X2_H1</td>
<td>-0.406536</td>
<td>0.0002672</td>
<td>X2 -&gt; H1</td>
</tr>
<tr>
<td>X1_H2</td>
<td>-1.507694</td>
<td>0.0002234</td>
<td>X1 -&gt; H2</td>
</tr>
</tbody>
</table>
Parameter Estimates

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Gradient</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1_H26</td>
<td>-0.055756</td>
<td>-0.0000101</td>
<td>X1 -&gt; H26</td>
</tr>
<tr>
<td>X2_H26</td>
<td>-0.331826</td>
<td>0.0001698</td>
<td>X2 -&gt; H26</td>
</tr>
<tr>
<td>X1_H27</td>
<td>-0.799268</td>
<td>0.0001618</td>
<td>X1 -&gt; H27</td>
</tr>
<tr>
<td>X2_H27</td>
<td>-0.195293</td>
<td>0.0005865</td>
<td>X2 -&gt; H27</td>
</tr>
<tr>
<td>X1_H28</td>
<td>-1.130014</td>
<td>0.0006003</td>
<td>X1 -&gt; H28</td>
</tr>
<tr>
<td>X2_H28</td>
<td>0.540507</td>
<td>-0.00117</td>
<td>X2 -&gt; H28</td>
</tr>
<tr>
<td>X1_H29</td>
<td>-0.808714</td>
<td>0.0000157</td>
<td>X1 -&gt; H29</td>
</tr>
<tr>
<td>X2_H29</td>
<td>0.442020</td>
<td>-0.0003948</td>
<td>X2 -&gt; H29</td>
</tr>
<tr>
<td>X1_H30</td>
<td>-1.075334</td>
<td>-0.0008229</td>
<td>X1 -&gt; H30</td>
</tr>
<tr>
<td>Parameter</td>
<td>Estimate</td>
<td>Gradient</td>
<td>Label</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>BIAS_H26</td>
<td>-0.241293</td>
<td>0.0000243</td>
<td>BIAS -&gt; H26</td>
</tr>
<tr>
<td>BIAS_H27</td>
<td>0.670665</td>
<td>0.0000766</td>
<td>BIAS -&gt; H27</td>
</tr>
<tr>
<td>BIAS_H28</td>
<td>0.255718</td>
<td>0.0000582</td>
<td>BIAS -&gt; H28</td>
</tr>
<tr>
<td>BIAS_H29</td>
<td>0.195161</td>
<td>0.0000364</td>
<td>BIAS -&gt; H29</td>
</tr>
<tr>
<td>BIAS_H30</td>
<td>1.509699</td>
<td>-0.00366</td>
<td>BIAS -&gt; H30</td>
</tr>
<tr>
<td>H1_HIPL</td>
<td>-0.136499</td>
<td>-0.00160</td>
<td>H1 -&gt; HIPL</td>
</tr>
<tr>
<td>H2_HIPL</td>
<td>-0.091000</td>
<td>-0.00300</td>
<td>H2 -&gt; HIPL</td>
</tr>
<tr>
<td>H3_HIPL</td>
<td>-0.144627</td>
<td>-0.00174</td>
<td>H3 -&gt; HIPL</td>
</tr>
<tr>
<td>H4_HIPL</td>
<td>0.175297</td>
<td>-0.00214</td>
<td>H4 -&gt; HIPL</td>
</tr>
<tr>
<td>H5_HIPL</td>
<td>-0.078282</td>
<td>-0.00190</td>
<td>H5 -&gt; HIPL</td>
</tr>
<tr>
<td>H6_HIPL</td>
<td>0.803260</td>
<td>-0.00174</td>
<td>H6 -&gt; HIPL</td>
</tr>
<tr>
<td>H7_HIPL</td>
<td>-0.132805</td>
<td>0.0004925</td>
<td>H7 -&gt; HIPL</td>
</tr>
<tr>
<td>H8_HIPL</td>
<td>-0.057435</td>
<td>-0.0008912</td>
<td>H8 -&gt; HIPL</td>
</tr>
<tr>
<td>H9_HIPL</td>
<td>-0.333394</td>
<td>0.0005940</td>
<td>H9 -&gt; HIPL</td>
</tr>
<tr>
<td>H10_HIPL</td>
<td>-0.092374</td>
<td>-0.0004988</td>
<td>H10 -&gt; HIPL</td>
</tr>
<tr>
<td>H11_HIPL</td>
<td>-0.152927</td>
<td>0.00433</td>
<td>H11 -&gt; HIPL</td>
</tr>
<tr>
<td>H12_HIPL</td>
<td>0.117821</td>
<td>-0.00294</td>
<td>H12 -&gt; HIPL</td>
</tr>
<tr>
<td>H13_HIPL</td>
<td>0.206879</td>
<td>-0.00262</td>
<td>H13 -&gt; HIPL</td>
</tr>
<tr>
<td>H14_HIPL</td>
<td>-0.302093</td>
<td>0.00236</td>
<td>H14 -&gt; HIPL</td>
</tr>
<tr>
<td>H15_HIPL</td>
<td>-0.221776</td>
<td>0.00168</td>
<td>H15 -&gt; HIPL</td>
</tr>
<tr>
<td>H16_HIPL</td>
<td>-0.366827</td>
<td>-0.00201</td>
<td>H16 -&gt; HIPL</td>
</tr>
<tr>
<td>H17_HIPL</td>
<td>-0.062601</td>
<td>0.00624</td>
<td>H17 -&gt; HIPL</td>
</tr>
<tr>
<td>H18_HIPL</td>
<td>0.693210</td>
<td>0.00477</td>
<td>H18 -&gt; HIPL</td>
</tr>
<tr>
<td>H19_HIPL</td>
<td>-0.438978</td>
<td>-0.00270</td>
<td>H19 -&gt; HIPL</td>
</tr>
<tr>
<td>H20_HIPL</td>
<td>0.004246</td>
<td>-0.00201</td>
<td>H20 -&gt; HIPL</td>
</tr>
<tr>
<td>H21_HIPL</td>
<td>-0.195547</td>
<td>0.00236</td>
<td>H21 -&gt; HIPL</td>
</tr>
<tr>
<td>H22_HIPL</td>
<td>0.004032</td>
<td>0.00203</td>
<td>H22 -&gt; HIPL</td>
</tr>
<tr>
<td>H23_HIPL</td>
<td>0.307420</td>
<td>-0.00475</td>
<td>H23 -&gt; HIPL</td>
</tr>
<tr>
<td>H24_HIPL</td>
<td>0.391365</td>
<td>0.00300</td>
<td>H24 -&gt; HIPL</td>
</tr>
<tr>
<td>H25_HIPL</td>
<td>0.051031</td>
<td>0.00192</td>
<td>H25 -&gt; HIPL</td>
</tr>
<tr>
<td>H26_HIPL</td>
<td>-0.041873</td>
<td>0.00200</td>
<td>H26 -&gt; HIPL</td>
</tr>
</tbody>
</table>
117 H27_HIPL  -0.061313   0.00141 H27 -> HIPL
118 H28_HIPL  -0.285679   -0.00139 H28 -> HIPL
119 H29_HIPL  -0.118849   -0.00205 H29 -> HIPL
120 H30_HIPL  0.374561    -0.00512 H30 -> HIPL
121 BIAS_HIP  -0.623399   -0.00230 BIAS -> HIPL

Hill & Plateau Data
MLP with 30 Hidden Units

Dual Quasi-Newton Optimization

*** Termination Criteria ***
Minimum Iterations . . . . . . . . . . . . . . .         -1
Maximum Iterations . . . . . . . . . . . . . . .       1000
Maximum Function Calls. . . . . . . . . . . . . . 2147483647
Maximum CPU Time . . . . . . . . . . . . . . . .     604800
ABSGCONV Gradient Criterion . . . . . . . . . . .  0.0000100
GCONV Gradient Criterion . . . . . . . . . . . .       1E-8
ABSFCNV Function Criterion . . . . . . . . . . . .  0
FCONV Function Criterion . . . . . . . . . . . . 0.0001000
FCONV2 Function Criterion . . . . . . . . . . . .          0
FSIZE Parameter . . . . . . . . . . . . . . . . .          0
ABSXCONV Parameter Change Criterion . . . . . . .          0
XCONV Parameter Change Criterion . . . . . . . .          0
XSIZE Parameter . . . . . . . . . . . . . . . . .          0
ABSCONV Function Criterion . . . . . . . . . . . -1.341E154

*** Other Control Parameters ***
Line Search Method 2: Starting Alpha . . . . . .    1.00000
Line Search Precision LSPRECISION . . . . . . . 0.40000
DAMPSTEP Parameter for Line Search . . . . . . . .          .
Singularity Tolerance (SINGULAR) . . . . . . . .       1E-8

Hill & Plateau Data
MLP with 30 Hidden Units

Dual Quasi-Newton Optimization
Dual Broyden - Fletcher - Goldfarb - Shanno Update (DBFGS)
Number of Parameter Estimates 121

Optimization Start: Active Constraints= 0  Criterion= 0.030
Maximum Gradient Element= 0.006
Hill & Plateau Data
MLP with 30 Hidden Units

********** Continuation of Iteration History **********

Optimization Results: Iterations= 1000 Function Calls= 1621
Gradient Calls= 1310 Active Constraints= 0 Criterion= 0.00074889869
Maximum Gradient Element= 0.0000589356 Slope= -3.8861E-7

WARNING: QUANEW Optimization cannot be completed.
WARNING: QUANEW needs more than 1000 iterations or 2147483647 function

Hill & Plateau Data
MLP with 30 Hidden Units

Optimization Results
Parameter Estimates

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Gradient Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 X1_H1</td>
<td>-0.532039</td>
<td>2.3205E-6 X1 -&gt; H1</td>
</tr>
<tr>
<td>2 X2_H1</td>
<td>-0.125752</td>
<td>-1.2541E-6 X2 -&gt; H1</td>
</tr>
<tr>
<td>3 X1_H2</td>
<td>-3.790662</td>
<td>-0.0000317 X1 -&gt; H2</td>
</tr>
<tr>
<td>4 X2_H2</td>
<td>1.651981</td>
<td>-0.0000306 X2 -&gt; H2</td>
</tr>
<tr>
<td>5 X1_H3</td>
<td>1.495431</td>
<td>-0.0000166 X1 -&gt; H3</td>
</tr>
<tr>
<td>6 X2_H3</td>
<td>1.222886</td>
<td>-4.1724E-7 X2 -&gt; H3</td>
</tr>
<tr>
<td>7 X1_H4</td>
<td>-3.447110</td>
<td>8.67875E-6 X1 -&gt; H4</td>
</tr>
<tr>
<td>Parameter</td>
<td>Estimate</td>
<td>Gradient</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>X2_H4</td>
<td>-0.166308</td>
<td>0.0000402</td>
</tr>
<tr>
<td>X1_H5</td>
<td>-1.365956</td>
<td>-3.7006E-6</td>
</tr>
<tr>
<td>X2_H5</td>
<td>-0.107199</td>
<td>9.36108E-6</td>
</tr>
<tr>
<td>X1_H6</td>
<td>-0.001925</td>
<td>0.0000471</td>
</tr>
<tr>
<td>X2_H6</td>
<td>2.893215</td>
<td>-0.0000366</td>
</tr>
<tr>
<td>X1_H7</td>
<td>2.017056</td>
<td>2.84015E-6</td>
</tr>
<tr>
<td>X2_H7</td>
<td>0.101604</td>
<td>9.34775E-7</td>
</tr>
<tr>
<td>X1_H8</td>
<td>-0.569829</td>
<td>9.05793E-8</td>
</tr>
<tr>
<td>X2_H8</td>
<td>-0.162453</td>
<td>-1.5378E-6</td>
</tr>
<tr>
<td>X1_H9</td>
<td>-1.840300</td>
<td>0.0000153</td>
</tr>
<tr>
<td>X2_H9</td>
<td>-2.672442</td>
<td>-2.0747E-6</td>
</tr>
<tr>
<td>X1_H10</td>
<td>1.677979</td>
<td>0.0000232</td>
</tr>
<tr>
<td>X2_H10</td>
<td>0.327821</td>
<td>-0.0000115</td>
</tr>
<tr>
<td>X1_H11</td>
<td>-2.323530</td>
<td>0.0000175</td>
</tr>
<tr>
<td>X2_H11</td>
<td>-2.538369</td>
<td>0.0000162</td>
</tr>
<tr>
<td>X1_H12</td>
<td>-2.783121</td>
<td>0.0000251</td>
</tr>
<tr>
<td>X2_H12</td>
<td>0.493515</td>
<td>0.0000307</td>
</tr>
<tr>
<td>X1_H13</td>
<td>-1.702463</td>
<td>0.0000212</td>
</tr>
<tr>
<td>X2_H13</td>
<td>1.200148</td>
<td>0.0000554</td>
</tr>
<tr>
<td>X1_H14</td>
<td>1.327628</td>
<td>-9.5967E-7</td>
</tr>
<tr>
<td>X2_H14</td>
<td>-1.702246</td>
<td>0.0000117</td>
</tr>
<tr>
<td>X1_H15</td>
<td>-1.999156</td>
<td>7.50953E-6</td>
</tr>
<tr>
<td>X2_H15</td>
<td>-2.076552</td>
<td>6.65E-6</td>
</tr>
<tr>
<td>X1_H16</td>
<td>0.026058</td>
<td>1.94184E-7</td>
</tr>
<tr>
<td>X2_H16</td>
<td>0.795836</td>
<td>0.0000344</td>
</tr>
<tr>
<td>X1_H17</td>
<td>0.296149</td>
<td>-0.0000104</td>
</tr>
<tr>
<td>X2_H17</td>
<td>-2.901058</td>
<td>0.0000123</td>
</tr>
<tr>
<td>X1_H18</td>
<td>-1.347832</td>
<td>0.0000242</td>
</tr>
<tr>
<td>X2_H18</td>
<td>-4.329716</td>
<td>8.39321E-6</td>
</tr>
<tr>
<td>X1_H19</td>
<td>2.261145</td>
<td>0.0000117</td>
</tr>
<tr>
<td>X2_H19</td>
<td>2.520761</td>
<td>0.0000350</td>
</tr>
<tr>
<td>X1_H20</td>
<td>-0.829191</td>
<td>2.07643E-6</td>
</tr>
<tr>
<td>X2_H20</td>
<td>1.016295</td>
<td>0.0000459</td>
</tr>
<tr>
<td>X1_H21</td>
<td>1.004049</td>
<td>0.0000209</td>
</tr>
<tr>
<td>X2_H21</td>
<td>-1.005880</td>
<td>5.41321E-6</td>
</tr>
<tr>
<td>X1_H22</td>
<td>0.240370</td>
<td>0.0000339</td>
</tr>
<tr>
<td>X2_H22</td>
<td>-0.516315</td>
<td>0.0000241</td>
</tr>
<tr>
<td>X1_H23</td>
<td>3.033531</td>
<td>2.09368E-6</td>
</tr>
<tr>
<td>X2_H23</td>
<td>0.824219</td>
<td>7.6337E-6</td>
</tr>
<tr>
<td>X1_H24</td>
<td>-3.072703</td>
<td>4.63047E-6</td>
</tr>
<tr>
<td>X2_H24</td>
<td>-1.511832</td>
<td>0.0000404</td>
</tr>
<tr>
<td>X1_H25</td>
<td>1.189984</td>
<td>-3.4502E-6</td>
</tr>
<tr>
<td>X2_H25</td>
<td>-3.514446</td>
<td>0.0000424</td>
</tr>
</tbody>
</table>

Hill & Plateau Data
MLP with 30 Hidden Units

Optimization Results
Parameter Estimates

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1_H26</td>
<td>-0.151007</td>
<td>0.0000270</td>
</tr>
<tr>
<td>X2_H26</td>
<td>-0.521042</td>
<td>0.0000298</td>
</tr>
<tr>
<td>X1_H27</td>
<td>-2.538401</td>
<td>0.0000145</td>
</tr>
<tr>
<td>X2_H27</td>
<td>0.012546</td>
<td>-0.0000355</td>
</tr>
<tr>
<td>X1_H28</td>
<td>-3.430341</td>
<td>0.0000158</td>
</tr>
<tr>
<td>X2_H28</td>
<td>2.727024</td>
<td>-0.0000208</td>
</tr>
<tr>
<td>X1_H29</td>
<td>-0.980648</td>
<td>7.7342E-6</td>
</tr>
<tr>
<td>X2_H29</td>
<td>0.485661</td>
<td>-0.0000146</td>
</tr>
<tr>
<td>X1_H30</td>
<td>-2.265832</td>
<td>9.67792E-6</td>
</tr>
<tr>
<td>X2_H30</td>
<td>3.053364</td>
<td>-1.7164E-6</td>
</tr>
<tr>
<td>Parameter</td>
<td>Estimate</td>
<td>Gradient</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>BIAS_H1</td>
<td>1.621410</td>
<td>3.94302E-7</td>
</tr>
<tr>
<td>BIAS_H2</td>
<td>1.544387</td>
<td>-0.0000555</td>
</tr>
<tr>
<td>BIAS_H3</td>
<td>-0.534732</td>
<td>0.0000123</td>
</tr>
<tr>
<td>BIAS_H4</td>
<td>-0.785714</td>
<td>-0.0000101</td>
</tr>
<tr>
<td>BIAS_H5</td>
<td>0.912585</td>
<td>-0.0000115</td>
</tr>
<tr>
<td>BIAS_H6</td>
<td>-0.768781</td>
<td>-2.2982E-6</td>
</tr>
<tr>
<td>BIAS_H7</td>
<td>1.015622</td>
<td>-9.1444E-6</td>
</tr>
<tr>
<td>BIAS_H8</td>
<td>0.550634</td>
<td>-1.1615E-8</td>
</tr>
<tr>
<td>BIAS_H9</td>
<td>-2.030613</td>
<td>-0.0000519</td>
</tr>
<tr>
<td>BIAS_H10</td>
<td>0.306667</td>
<td>-9.3574E-6</td>
</tr>
<tr>
<td>BIAS_H11</td>
<td>-2.591558</td>
<td>0.0000337</td>
</tr>
<tr>
<td>BIAS_H12</td>
<td>1.854141</td>
<td>0.0000213</td>
</tr>
<tr>
<td>BIAS_H13</td>
<td>-0.378726</td>
<td>-0.0000290</td>
</tr>
<tr>
<td>BIAS_H14</td>
<td>-0.596093</td>
<td>-0.0000340</td>
</tr>
<tr>
<td>BIAS_H15</td>
<td>1.200441</td>
<td>-0.0000417</td>
</tr>
<tr>
<td>BIAS_H16</td>
<td>-1.358588</td>
<td>0.0000232</td>
</tr>
<tr>
<td>BIAS_H17</td>
<td>-2.473932</td>
<td>0.0000370</td>
</tr>
<tr>
<td>BIAS_H18</td>
<td>-1.391321</td>
<td>0.0000284</td>
</tr>
<tr>
<td>BIAS_H19</td>
<td>-1.537930</td>
<td>-0.0000246</td>
</tr>
<tr>
<td>BIAS_H20</td>
<td>0.717883</td>
<td>-0.0000180</td>
</tr>
<tr>
<td>BIAS_H21</td>
<td>0.046762</td>
<td>-0.0000347</td>
</tr>
<tr>
<td>BIAS_H22</td>
<td>-0.062103</td>
<td>-0.0000241</td>
</tr>
<tr>
<td>BIAS_H23</td>
<td>0.090229</td>
<td>0.0000343</td>
</tr>
<tr>
<td>BIAS_H24</td>
<td>0.502362</td>
<td>1.47214E-6</td>
</tr>
<tr>
<td>BIAS_H25</td>
<td>-2.014471</td>
<td>-3.7539E-6</td>
</tr>
<tr>
<td>BIAS_H26</td>
<td>-0.448834</td>
<td>7.76322E-6</td>
</tr>
<tr>
<td>BIAS_H27</td>
<td>2.068868</td>
<td>0.0000197</td>
</tr>
<tr>
<td>BIAS_H28</td>
<td>1.888770</td>
<td>0.0000405</td>
</tr>
<tr>
<td>BIAS_H29</td>
<td>0.249710</td>
<td>-6.3913E-6</td>
</tr>
<tr>
<td>BIAS_H30</td>
<td>3.409735</td>
<td>8.33278E-6</td>
</tr>
<tr>
<td>H1_HIPL</td>
<td>-0.110538</td>
<td>0.0000178</td>
</tr>
<tr>
<td>H2_HIPL</td>
<td>-0.473553</td>
<td>0.0000117</td>
</tr>
<tr>
<td>H3_HIPL</td>
<td>-1.150299</td>
<td>0.0000209</td>
</tr>
<tr>
<td>H4_HIPL</td>
<td>0.477358</td>
<td>4.1644E-6</td>
</tr>
<tr>
<td>H5_HIPL</td>
<td>0.464599</td>
<td>0.0000156</td>
</tr>
<tr>
<td>H6_HIPL</td>
<td>0.868824</td>
<td>0.0000395</td>
</tr>
<tr>
<td>H7_HIPL</td>
<td>-0.305023</td>
<td>2.29377E-6</td>
</tr>
<tr>
<td>H8_HIPL</td>
<td>-0.022398</td>
<td>0.0000134</td>
</tr>
<tr>
<td>H9_HIPL</td>
<td>-0.971155</td>
<td>0.0000579</td>
</tr>
<tr>
<td>H10_HIPL</td>
<td>0.974106</td>
<td>0.0000192</td>
</tr>
</tbody>
</table>

Optimization Results
Parameter Estimates

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Gradient</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>H11_HIPL</td>
<td>0.568802</td>
<td>-0.0000519</td>
<td>H11 -&gt; HIPL</td>
</tr>
<tr>
<td>H12_HIPL</td>
<td>1.227553</td>
<td>0.0000166</td>
<td>H12 -&gt; HIPL</td>
</tr>
<tr>
<td>H13_HIPL</td>
<td>-0.466255</td>
<td>-0.0000231</td>
<td>H13 -&gt; HIPL</td>
</tr>
<tr>
<td>H14_HIPL</td>
<td>-0.894798</td>
<td>-0.0000322</td>
<td>H14 -&gt; HIPL</td>
</tr>
<tr>
<td>H15_HIPL</td>
<td>1.479547</td>
<td>-2.2118E-6</td>
<td>H15 -&gt; HIPL</td>
</tr>
<tr>
<td>H16_HIPL</td>
<td>0.471993</td>
<td>0.0000190</td>
<td>H16 -&gt; HIPL</td>
</tr>
<tr>
<td>H17_HIPL</td>
<td>-0.695108</td>
<td>-0.0000138</td>
<td>H17 -&gt; HIPL</td>
</tr>
<tr>
<td>H18_HIPL</td>
<td>0.411108</td>
<td>-9.7179E-7</td>
<td>H18 -&gt; HIPL</td>
</tr>
<tr>
<td>H19_HIPL</td>
<td>-0.650073</td>
<td>2.01526E-7</td>
<td>H19 -&gt; HIPL</td>
</tr>
<tr>
<td>H20_HIPL</td>
<td>1.728201</td>
<td>0.0000190</td>
<td>H20 -&gt; HIPL</td>
</tr>
<tr>
<td>H21_HIPL</td>
<td>-0.845374</td>
<td>0.0000182</td>
<td>H21 -&gt; HIPL</td>
</tr>
<tr>
<td>H22_HIPL</td>
<td>0.907477</td>
<td>0.0000158</td>
<td>H22 -&gt; HIPL</td>
</tr>
<tr>
<td>H23_HIPL</td>
<td>0.890942</td>
<td>0.0000194</td>
<td>H23 -&gt; HIPL</td>
</tr>
<tr>
<td>H24_HIPL</td>
<td>0.670337</td>
<td>-0.0000501</td>
<td>H24 -&gt; HIPL</td>
</tr>
<tr>
<td>H25_HIPL</td>
<td>0.524875</td>
<td>0.0000589</td>
<td>H25 -&gt; HIPL</td>
</tr>
<tr>
<td>H26_HIPL</td>
<td>0.644267</td>
<td>-0.0000242</td>
<td>H26 -&gt; HIPL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>117</td>
<td>H27_HIPL</td>
<td>-0.968536</td>
<td>-4.3518E-6</td>
</tr>
<tr>
<td>118</td>
<td>H28_HIPL</td>
<td>-0.283694</td>
<td>1.1348E-6</td>
</tr>
<tr>
<td>119</td>
<td>H29_HIPL</td>
<td>0.366848</td>
<td>-2.7894E-6</td>
</tr>
<tr>
<td>120</td>
<td>H30_HIPL</td>
<td>0.731690</td>
<td>0.0000279</td>
</tr>
<tr>
<td>121</td>
<td>BIAS_HIP</td>
<td>-0.008245</td>
<td>0.0000135</td>
</tr>
</tbody>
</table>

Value of Objective Function = 0.0007488987

PROC PRINT Report of the Average Squared Error for the Scored Test Data Set

Hill & Plateau Data
MLP with 30 Hidden Units
Fit Statistics for the Test Data

Test:
Average Squared Error.

.00071717

GCONTOUR Plot of the Predicted Values
Hill & Plateau Data
MLP with 30 Hidden Units
Predicted Values
The %LET statement sets the macro variable HIDDEN to 3.

title 'Hill & Plateau Data';
%let hidden=3;
The MAXITER = option specifies the maximum number of iterations.

```
    hidden &hidden / id=h;
    prelim 10;
    train maxiter=1000 outest=mlpest;
    score data=sampsio.dmtsurf out=mlpout outfit=mlpfit;
    title2 "MLP with &hidden Hidden Units";
run;
```
PROC PRINT creates a report of selected fit statistics.

proc print data=mlpfit noobs label;
  var _tase_ _tasel_ _taseu_;  
  where _name_ = 'HIPL';
  title3 'Fit Statistics for the Test Data';
run;
PROC GCONTOUR creates a plot of the predicted values.

proc gcontour data=mlpout;
   plot x2*x1=p_hi p / pattern ctext=black coutline=gray;
   pattern v=msolid;
   legend frame;
   title3 'Predicted Values';
   footnote;
run;
PROC G3D creates a plot of the predicted values. Note that this network underfits badly.

```plaintext
proc g3d data=mlpout;
  plot x2*x1=p_hi1 / grid side ctop=blue
       caxis=green ctext=black
       zmin=-1.5 zmax=1.5;
run;
```
References


The PMBR Procedure

Overview

Procedure Syntax

PROC PMBR Statement
VAR Statement
TARGET Statement
CLASS Statement

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Overview

The PMBR procedure is used for prediction as an alternative to other predictive modeling techniques in Enterprise Miner, such as the NEURAL, SPLIT, DMSPLIT, DMNEURL, and DMREG procedures. However, the technique in the PMBR procedure is different. Whereas all the other techniques attempt to determine some rules for predicting future examples, the PMBR procedure categorizes an observation in a score data set by retrieving its k closest neighbors from a training data set, and then having each neighbor vote on the target value based on its value for the target variable. These votes then become the posterior probabilities for predicting the target, which are included in an output data set. Training is thus faster than with the alternative techniques, but scoring is generally slower.

The target variable is expected to be either binary, interval, or nominal. Ordinal targets are not specially supported at this time, but could be modeled as interval targets. If the target variable is a class variable in the DMDB, one variable is created on the output data set for each value of the target, representing the appropriate posterior probabilities. Otherwise, one predicted variable is created on the output data set corresponding to the average prediction for the k neighbors.

The neighbors are determined by a simple Euclidean distance between the values on each of the variables in the VAR statement for the probe and target example. Thus, it is assumed that the variables are orthogonal to each other and standardized. If your input data is not in that form, you need precede this procedure with one that will create numeric, orthogonal, and standardized variables -- such as the PRIMCOMP, DMNEURL, PRINQUAL, CORRESP, SPSVD procedures.

The PMBR procedure needs to be run separately and be given the DMDB-name for each of the data sets to be scored, including any training, validation, test, or score data set.

Missing values in either the training or score data set are replaced by the mean of that variable as stored in the DMDB catalog.
The PMBR Procedure

PROC PMBR Statement

Invokes the PMBR procedure.

PROC PMBR <option(s)>; 

Required Arguments

DMDBCAT = <libref.>SAS-catalog
   Specifies the DMDB catalog.

Options

DATA = (or IN =) <libref.>SAS-data-set
   Specifies the DMDB-encoded input SAS data to be trained on. If you omit the DATA= option, the procedure uses the most recently created SAS data set, which must be DMDB-encoded.

SCORE = <libref.>SAS-data-set
   Specifies the data set to be scored. This data set might not have the target variable. It can be the same name as the training data set.

OUT = <libref.>SAS-data-set
   Specifies the name of the output data set. This output data set contains all variable in the score data set and additional variables representing the posterior probabilities. If the target variable is categorical, the names of these variables generally begin with P_, followed by a part of the original variable names and with the values added to the end. These posterior probabilities correspond to the percentages of the k neighbors that have the value as the target. If the target variable is interval, a single posterior variable is produced that averages the target values across the k neighbors. This option is required if the SCORE= option is used.

K = integer
   Specifies the number of nearest neighbors to retrieve.

   Default: 1

PRINT
   Prints out training information and weights (if the WEIGHTED option is specified) to the OUTPUT window.

METHOD = method
   Determines what data representation is used to store the training data set and then to retrieve the nearest neighbors. The following methods are available:
SCAN  Retrieves a nearest neighbor by naively going through every observation in the training data set and calculating its distance to a probe observation.

KDTREE  Uses a KD-Tree to store the observations in the training data set in memory. This enables the nearest neighbors to a point to be found in $o(\log n)$ time, assuming the number of variables is small enough (fewer than ten to twenty). For more information about KD-Tree, see Freidman, Bentley, and Fingel (1977). This method has not been implemented yet.

RDTREE  Uses an RD-Tree to store the observations in the training data set in memory. This is a proprietary representation that, like a KD-Tree, also operates in $o(\log n)$ time, but will generally examine fewer nodes than an RD-Tree to find the neighbors, and can be applied with somewhat greater dimensionality.

**EPSILON = positive number**

Indicates an approximate nearest neighbor search when a non-zero number is specified, where the nearest neighbors determined so far must be at most "epsilon" away from the actual nearest neighbors to terminate the search. For large dimensionality, judicious use of epsilon can result in radically improved performance. This option only applies to the KD-Tree or RD-Tree methods.

Defaults: 0.0

**BUCKET = positive integer**

Indicates the number of buckets to allow a leaf node to grow before splitting into a branch with two new leaves. This value must be greater than or equal to 2. This option only applies to the KD-Tree or RD-Tree methods.

Default: 8

**SHOWNODES**

Includes a variable `_nnodes_` in the output data set that shows the number of point comparisons that had to be done to determine the answer. This is useful as a point of comparison.
VAR Statement

**VAR** `<variable>`;

`variable`

Specifies all numeric variables that you want to treat as dimensions for the nearest neighbor lookup. These should be standardized and orthogonal for the nearest neighbor search to be accurate. If no VAR statement is specified, all numeric variables in the DMDB-encoded data set will be used.
TARGET Statement

Specifies one variable to be used as the target. It can be numeric or character.

TARGET <variable>;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
CLASS Statement

This is currently ignored. The procedure determines whether the target is a class variable based on the contents of the DMDB, and it cannot be changed in this procedure.

CLASS <variable>;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The RULEGEN Procedure

Overview

Procedure Syntax
  PROC RULEGEN Statement

Details

Example
  Example 1: Performing an Association Discovery

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Overview

PROC RULEGEN uses the output data set created by PROC ASSOC and generates association rules and computes statistics, such as confidence and lift, for the rules. PROC ASSOC identifies item sets that are related. The RULEGEN procedure generates the rules governing their association. PROC RULEGEN output is saved as a SAS data set that can be viewed or browsed by SAS procedures that you can create to reflect your own evaluation criteria.
PROC RULEGEN <option(s)>;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
PROC RULEGEN Statement

Invokes the RULEGEN procedure.

PROC RULEGEN<option(s)>;

Required Argument

OUT=<libref.>SAS-data-set

Specifies the output data set to which the rules are written. The output data set has the following variables: CONF, COUNT, EXP_CONF, ITEM1, ITEM2, ..., ITEMn+1, _LHAND, LIFT, _RHAND, RULE, SET_SIZE, SUPPORT.

CONF=COUNT/lhs_count
    Contains the percent of confidence.

**Definition:**

| lhs_count is the number of transactions satisfying the left side of the rule. |

COUNT

Contains the number of transactions meeting the rule.

EXP_CONF=rhs_count/total

Contains the percent of expected confidence.

**Definition:**

| rhs_count is the number of transactions satisfying the right side of the rule. |

ITEM1, ITEM2, ..., ITEMn+1

Contains individual items which make up the rule, including the arrow.

_LHAND

Identifies the left side of the rule, where the rule is expressed: _LHAND ==> _RHAND.

LIFT= CONF/EXP_CONF

Contains the lift ratio.

_RHAND

Identifies the right side of the rule, where the rule is expressed: _LHAND ==> _RHAND.

RULE

Contains the text of the rule, for example, A & B ==> C & D.

SET_SIZE
Contains the number of items in the rule.

**SUPPORT= COUNT/total**

Contains the percent of support, that is, the percent of the total number of transactions that qualify for the rule.

| Definition | total is the total number of transactions in the data set. |

---

### Options

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

Specifies the input data source. The input to PROC RULEGEN is the OUT= data set created in PROC ASSOC.

| Default: | _LAST_ |

**MINCONF=integer**

Specifies the minimum confidence level needed in order to generate a rule. This parameter can be adjusted so that only high confidence rules are retained.

| Default: | 10% |
The output data set created by PROC RULEGEN has the following variables:

SET_SIZE
Contains the number of items in the rule.

RULE
Contains the rule text, for example, A & B ==> C & D.

COUNT
Contains the count of transactions meeting the rule.

CONF
Contains the percent of confidence.

EXP_CONF
Contains the percent of expected confidence.

LIFT
Contains the lift ratio.

SUPPORT
Contains the percent of support.

_LHAND
Contains the left side of the rule.

_RHAND
Contains the right side of the rule.

ITEM1, ITEM2, ..., ITEMn+1
Contains the individual items forming the rule, including the arrow.

Only the rules meeting the minimum confidence value are output. This parameter can be adjusted to retain only the high confidence rules.

The statistical computation is based on Bayes' theorem, stated as probability of event A conditional on event B occurring, and is calculated as the probability of both events A and B occurring divided by the probability of event B.

PROC RULEGEN automatically discovers complex rules with multiple events on either side such as A & B ==> C, implying event C occurred, given that both events A and B occurred.

Consider the rule lhs ==> rhs.
In terms of the output data set variables, the statistics are computed as follows:

- \( \text{CONF}= \frac{\text{COUNT}}{\text{lhs\_count}} \)
- \( \text{EXP\_CONF}=\frac{\text{rhs\_count}}{\text{total}} \)
- \( \text{LIFT}=\frac{\text{CONF}}{\text{EXP\_CONF}} \)
- \( \text{SUPPORT}=\frac{\text{COUNT}}{\text{total}} \)

where \text{total} is the number of transactions in the data set.

As you can see, positioning of items on the left or right side does impact statistical calculations, that is, \( A \Rightarrow B \) and \( B \Rightarrow A \) are entirely different rules.
Example

The following example was executed using the HP-UX version 10.20 operating system and the SAS software release 6.12TS045.

Example 1: Performing an Association Discovery

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Example 1: Performing an Association Discovery

The following example demonstrates how to perform an association discovery using the ASSOCIATION and RULEGEN procedures. The example data set SAMPSIO. ASSOCS (stored in the sample library) contains 7,007 separate customer transactions. The variable CUSTOMER is the ID variable that identifies the customers. The variable PRODUCT is the nominal target variable that identifies the items. As a marketing analyst for a grocery chain, you want to identify the top 10 item sets, where the purchase of one item has a high impact on the purchase of another item(s).

Program

```sas
proc dmdb batch data=sampsio.assocs out=dmassoc dmdbcat=catassoc;
  id customer;
  class product(desc);
run;

proc assoc data=dmassoc dmdbcat=catassoc
  out=datassoc(label='Output from Proc Assoc')
  items=5 support=20;
  cust customer;
  target product;
run;

proc rulegen in=datassoc
  out=datrule(label='Output from Proc Rulegen')
  minconf=75;
run;
```
proc sort data=datrule;
  by descending lift;
run;
proc print data=datrule(obs=5) label;
  var set_size exp_conf conf support lift count
  rule _lhand _rhand;
  title 'Top Ten Rules based on Lift';run;

Output

<table>
<thead>
<tr>
<th>OBS</th>
<th>SET_SIZE</th>
<th>EXP_CONF</th>
<th>CONF</th>
<th>SUPPORT</th>
<th>LIFT</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>7.39</td>
<td>100.00</td>
<td>7.39</td>
<td>13.53</td>
<td>74.00</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>12.59</td>
<td>94.74</td>
<td>8.99</td>
<td>7.53</td>
<td>90.00</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>10.79</td>
<td>78.99</td>
<td>9.39</td>
<td>7.32</td>
<td>94.00</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>11.89</td>
<td>87.04</td>
<td>9.39</td>
<td>7.32</td>
<td>94.00</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>12.69</td>
<td>92.78</td>
<td>8.99</td>
<td>7.31</td>
<td>90.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBS</th>
<th>RULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bordeaux</td>
</tr>
<tr>
<td>2</td>
<td>sardines &amp; baguette &amp; apples ==&gt; peppers &amp; avocado</td>
</tr>
<tr>
<td>3</td>
<td>turkey &amp; coke ==&gt; olives &amp; ice_cream &amp; bourbon</td>
</tr>
<tr>
<td>4</td>
<td>olives &amp; ice_crea &amp; bourbon ==&gt; turkey &amp; coke</td>
</tr>
<tr>
<td>5</td>
<td>peppers &amp; baguette &amp; apples ==&gt; sardines &amp; avocado</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBS</th>
<th>_LHAND</th>
<th>_RHAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>sardines &amp; baguette &amp; apple</td>
<td>peppers &amp; avocado</td>
</tr>
<tr>
<td>3</td>
<td>turkey &amp; coke</td>
<td>olives &amp; ice_crea &amp; bourbon</td>
</tr>
<tr>
<td>4</td>
<td>olives &amp; ice_crea &amp; bourbon</td>
<td>turkey &amp; coke</td>
</tr>
<tr>
<td>5</td>
<td>peppers &amp; baguette &amp; apple</td>
<td>sardines &amp; avocado</td>
</tr>
</tbody>
</table>

Log

1 proc dmdb batch data=sampsio.assocs out=dmassoc dmdbcat=catassoc;
  id customer;
  class product(desc);
run;
Records processed= 7007 Mem used = 511K.
NOTE: The PROCEDURE DMDB used 0:00:01.68 real 0:00:00.83 cpu.

6 proc assoc data=dmassoc dmdbcat=catassoc
  out=datassoc(label='Output from Proc Assoc')
9
10   items=5 support=20;
11
12   cust customer;
13   target product;
14   run;

----- Potential 1 item sets = 20 ----- 
Counting items, records read: 7007
Number of customers: 1001
Support level for item sets: 20
Maximum count for a set: 600
Sets meeting support level: 20
Megas of memory used: 0.51

----- Potential 2 item sets = 190 ----- 
Counting items, records read: 7007
Maximum count for a set: 366
Sets meeting support level: 183
Megas of memory used: 0.51

----- Potential 3 item sets = 1035 ----- 
Counting items, records read: 7007
Maximum count for a set: 234
Sets meeting support level: 615
Megas of memory used: 0.51

----- Potential 4 item sets = 1071 ----- 
Counting items, records read: 7007
Maximum count for a set: 137
Sets meeting support level: 317
Megas of memory used: 0.51

----- Potential 5 item sets = 85 ----- 
Counting items, records read: 7007
Maximum count for a set: 116
Sets meeting support level: 71
Megas of memory used: 0.51
NOTE: The PROCEDURE ASSOC used 0:00:07.86 real 0:00:03.45 cpu.

15
16 proc rulegen in=datassoc
17   out=datrule(label='Output from Proc Rulegen')
18
19   minconf=75;
20 run;
write set 1
write set 2
write set 3
write set 4
write set 5
NOTE: The PROCEDURE RULEGEN used 0:00:06.07 real 0:00:02.69 cpu.
21
22 proc sort data=datrule;
by descending lift;

NOTE: The data set WORK.DATRULE has 939 observations and 15 variables.
NOTE: The PROCEDURE SORT used 0:00:00.92 real 0:00:00.31 cpu.
proc print data=datrule(obs=5) label;
  var set_size exp_conf conf support lift count
    rule _lhand _rhand;
  title 'Top Ten Rules based on Lift';
run;
NOTE: The PROCEDURE PRINT used 0:00:00.18 real 0:00:00.11 cpu.
Before you can run PROC ASSOC, you must create the DMDB data set and the DMDB catalog by using a PROC DMBD step.

```plaintext
proc dmbd batch data=sampsio.assocs out=dmassoc dmdbcat=catassoc;
  id customer;
  class product(desc);
run;
```
The ASSOCIATION procedure determines the products that are related. The DATA= and DMDB= options identify the DMDB data set and catalog, respectively. PROC ASSOC writes the related products to the OUT= data set, which is used as input by the RULEGEN procedure.

```
proc assoc data=dmassoc dmdbcat=catassoc
   out=datassoc(label='Output from Proc Assoc')
```
The ITEMS= option specifies the maximum size of the item set to be considered (default=4). The SUPPORT= option specifies the minimum support level that is required for a rule to be accepted (default =5% of the largest frequency).

```
items=5 support=20;
```
The CUST statement (alias = CUSTOMER) specifies the ID variable. The TARGET statement specifies the nominal target variable.

cust customer;
target product;
run;
The RULEGEN procedure uses the output from PROC ASSOC to generate the rules. The rules are written to the OUT=data set.

```plaintext
proc rulegen in=datassoc
   out=datrule(label='Output from Proc Rulegen')
```
The MINCONF= option specifies the minimum confidence required in order to generate a rule (default =10).

   minconf=75;
   run;
Because neither PROC ASSOC nor RULEGEN generates printed output, the remaining code sorts the data by the LIFT values and then generates a simple list report of the rules that have the top 10 values for LIFT. This is done primarily to limit the amount of output displayed in this example.

```sas
proc sort data=datrule;
   by descending lift;
run;
proc print data=datrule(obs=5) label;
   var set_size exp_conf conf support lift count
   rule _lhand _rhand;
   title 'Top Ten Rules based on Lift';run;
```
The PROC PRINT list report of the top 10 rules based on the LIFT value. The output data set from PROC RULEGEN contains the following variables:

- **SET_SIZE** - contains the number of items in the rule.
- **EXP_CONF** - the expected confidence (right side count/total).
- **CONF** - the confidence (count / left side).
- **SUPPORT** - the support level (count/total).
- **LIFT** - the lift ratio (confidence/expected confidence).
- **COUNT** - number of transactions meeting the rule.
- **RULE** - contains the text rule, for example, Right side ==> Left side.
- **_LHAND** - contains the left side of the rule.
- **_RHAND** - contains the right side of the rule.
- **ITEM1, ITEM2, ..., ITEMn+1** - contains the individual items forming the rule, including the arrow. For this example, the individual items have been omitted from the list report.

<table>
<thead>
<tr>
<th>OBS</th>
<th>SET_SIZE</th>
<th>EXP_CONF</th>
<th>CONF</th>
<th>SUPPORT</th>
<th>LIFT</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>7.39</td>
<td>100.00</td>
<td>7.39</td>
<td>13.53</td>
<td>74.00</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>12.59</td>
<td>94.74</td>
<td>8.99</td>
<td>7.53</td>
<td>90.00</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>10.79</td>
<td>78.99</td>
<td>9.39</td>
<td>7.32</td>
<td>94.00</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>11.89</td>
<td>87.04</td>
<td>9.39</td>
<td>7.32</td>
<td>94.00</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>12.69</td>
<td>92.78</td>
<td>8.99</td>
<td>7.31</td>
<td>90.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBS</th>
<th>RULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bordeaux</td>
</tr>
<tr>
<td>2</td>
<td>sardines &amp; baguette &amp; apples ==&gt; peppers &amp; avocado</td>
</tr>
<tr>
<td>3</td>
<td>turkey &amp; coke ==&gt; olives &amp; ice_cream &amp; bourbon</td>
</tr>
<tr>
<td>4</td>
<td>olives &amp; ice_crea &amp; bourbon ==&gt; turkey &amp; coke</td>
</tr>
<tr>
<td>5</td>
<td>peppers &amp; baguette &amp; apples ==&gt; sardines &amp; avocado</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBS</th>
<th>_LHAND</th>
<th>_RHAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sardines &amp; baguette &amp; apple</td>
<td>peppers &amp; avocado</td>
</tr>
<tr>
<td>2</td>
<td>turkey &amp; coke</td>
<td>olives &amp; ice_crea &amp; bourbon</td>
</tr>
<tr>
<td>3</td>
<td>olives &amp; ice_crea &amp; bourbon</td>
<td>turkey &amp; coke</td>
</tr>
<tr>
<td>4</td>
<td>peppers &amp; baguette &amp; apple</td>
<td>sardines &amp; avocado</td>
</tr>
</tbody>
</table>
The SEQUENCE Procedure

Overview

Procedure Syntax
- PROC SEQUENCE Statement
- CUSTOMER Statement
- TARGET Statement
- VISIT Statement

Details

Examples
- Example 1: Performing a Simple 2-Item Sequence Discovery
- Example 2: Specifying the Maximum Number of Item Events and Setting the Lower Timing Limit

References

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Overview

The SEQUENCE procedure enables you to perform sequence discovery. Sequence discovery goes one step further than association discovery by taking into account the ordering or timing of the relationship among items, for example, "Of those customers who purchase a new computer, 25% of them will purchase a laser printer in the next quarter". To perform a sequence discovery, you must first run the ASSOCIATION procedure to create and output the data set of the assembled items.

PROC SEQ produces rules similar to PROC RULEGEN, however the rules additionally imply an element of timing. A rule A==>B implies that event B occurred 'after' event A occurred. The visit or sequence variable is used for timing comparison. The sequence variable can have any numeric value, including date or time values. Transactions with missing sequence values are ignored entirely during the sequence computation.

In order to determine the timing element, SEQUENCE utilizes a sequence variable or time-stamp that enables you to measure the time span from observation to observation. This procedure is useful for businesses such as banks or mail-order houses.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
PROC SEQUENCE <option(s)>;

CUSTOMER variable(s);

TARGET variable;

VISIT variable /<visit-option(s)>;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
PROC SEQUENCE Statement

Invokes the SEQUENCE procedure.

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

### Required Arguments

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

Specifies the SAS data set that was output from PROC ASSOC and which is also one of the inputs to PROC SEQ.

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

Specifies the input data source in its DMDB form. This data set is read in order to extract the timing information necessary to generate the sequence rules.

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

Identifies the metadata catalog associated with the input DMDB.

### Options

**NITEMS=integer**

Specifies the maximum number of events for which rules, or chains, are generated. If you request more than 2-event chains, \( (integer) - 2 \) additional passes through the input file are required.

| Default: | 2 |

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

Specifies the output data set to which the rules are written. The output data set has the following variables: RULE, COUNT, SUPPORT, CONF, ISET1, ISET2, ..., ISETn.

**RULE**

Contains the rule text, for example, A & B ==> C & D

**COUNT**

Contains the number of the transactions meeting the rule.

**SUPPORT**

Contains the percent of support, that is the percent of the total number of transactions that qualify for the rule.
**Definition:**  
SUPPORT = COUNT/total, where *total* is the total number of transactions in the data set. The support level is an integer that represents how frequently the combination occurs in the database.

**CONF**

Contains the percent of confidence.

**Definition:**  
CONF = COUNT/lhs_count where *lhs_count* is the number of transactions satisfying the left side of the rule.

**ISET1, ISET2,..., ISETn**

Contain, in order, the events that form the event chain. PROC SEQUENCE can detect multiple events occurring at the same time and can report them as rules of the type A & B ==> C & D. This means that events A and B occurred at the same time, followed by C and D, which occurred simultaneously afterwards.

**SUPPORT = integer**

Specifies the minimum number of transactions that must be considered in order for a rule to be accepted. Rules that do not meet the support level are rejected.

**Default:**  
If not specified, SUPPORT is set to a number that is 2% of the total transaction count.
CUSTOMER Statement

Specifies the ID variable that identifies each customer to be analyzed.

Alias: CUST

CUSTOMER variable(s);

Required Argument

variable(s)

Specifies the customer to be analyzed.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
TARGET Statement

Specifies the name of the product to be analyzed.

TARGET variable;

Required Argument

variable(s)

Specifies the name of the product to be analyzed.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
**VISIT Statement**

```plaintext
VISIT variable <visit-option(s)>;
```

**Required Argument**

Specifies the timing variable. See [Details](#) for an example of the SAME and WINDOW options in the VISIT statement.

*variable*

Specifies the time-stamp unit to measure. *Variable* is any numeric value, including date or time values.

**Options**

*visit-option*

SAME and WINDOW specify the upper and lower timing limits of a sequence rule. SAME $\leq$ time difference $\leq$ WINDOW.

*Visit-option* can be as follows:

**SAME=same-number**

Specifies the lower time-limit between the occurrence of two events that you want to associate with each other. If the time difference between the two events is less than or equal to *same-number* (that is, it is 'too soon'), then the two events are treated as occurring in the same visit, and the transaction is not counted.

**Default:** 0

**WINDOW=window-number**

Specifies the maximum time difference between the occurrence of two events that you want to be treated as the same visit. If the time difference is greater than *window-number* (that is, it is 'too late'), then the transaction is treated as falling outside of the timing window, and the transaction is not counted. For *NITEM-long* sequence chain, WINDOW applies to the entire chain.

**Default:** MAX
The SEQUENCE Procedure

Details

SAME and WINDOW Parameters

Two optional parameters, SAME and WINDOW, are available to define what is 'after'. The rule A==B implies SAME < TimeB - TimeA ≤ WINDOW.

Any time difference (TimeB - TimeA) less than or equal to SAME is considered the same time and is consolidated as the same visit and the same transaction. Any time difference exceeding WINDOW falls outside of the timing window and is ignored as well. In other words, SAME lets you define what is 'too soon', that is, event B occurred too soon after event A to qualify for A==B. Likewise, WINDOW defines 'too late', that is, event B occurred too late after event A occurred to be considered for A==B.

Consider the following example:

<table>
<thead>
<tr>
<th>Customer</th>
<th>Visit</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>soda</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>apples</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>juice</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>milk</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>bread</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>soda</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>apples</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>milk</td>
</tr>
</tbody>
</table>

With SAME=1, the visits are consolidated as follows:

<table>
<thead>
<tr>
<th>Customer</th>
<th>Visit</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>soda and apples</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Customer 1 is counted for apples ==> milk, however, Customer 2 is not. Both customers are counted for soda ==> milk.

If WINDOW=3 was also specified, then only Customer 1 would count for soda ==> milk. Using the above criterion, Customer 2 would not qualify.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Examples

The following examples were executed using the HP-UX version 10.20 operating system and the SAS software release 6.12TS045.

**Example 1: Performing a Simple 2-Item Sequence Discovery**

**Example 2: Specifying the Maximum Number of Item Events and Setting the Lower Timing Limit**
Example 1: Performing a Simple 2-Item Sequence Discovery

Features: ASSOCIATION and SEQUENCE Procedures

- Specifying the Maximum Item-Set Size
- Setting the Support Level
- Specifying the Number of Events

The following example demonstrates how to perform a sequence discovery using the ASSOCIATION and SEQUENCE procedures. The example data set SAMPSIO. ASSOCS (stored in the sample library) contains 7,007 separate customer transactions. CUSTOMER is an ID variable that identifies the customers. PRODUCT is the nominal target variable that identifies the items. TIME is the visit variable that measures the time span from observation to observation.

As a marketing analyst for a grocery chain, you want to identify likely 2-item purchase sequences. This information may help you make decisions, such as when to distribute coupons, when to put a product on sale, or how to present items in store displays.

Program

```plaintext
proc print data=sampsio.assocs(obs=10);
   title 'Partial Listing of the ASSOCS Data Set';
run;

proc dmdb batch data=sampsio.assocs out=dmseq dmdbcat=catseq;
   id customer time;
   class product(desc);
run;

proc assoc data=dmseq dmdbcat=catseq
   out=aout(label='Output from Proc Assoc')
   items=5 support=20;
   cust customer;
   target product;
run;
```
proc sequence data=dmseq dmdbcat=catseq
  assoc=aout
  out=sout(label='Output from Proc Sequence')
  nitems=2;

cust customer;
target product;

visit time;
run;

proc sort data=sout;
  by descending support;
run;

proc print data=sout(obs=10);
  var count support conf rule;
  title 'Partial Listing of the 2-Item Sequences';
run;

Output

PROC PRINT Partial Listing of the SAMP5IO.ASSOCS Data Set

<table>
<thead>
<tr>
<th>OBS</th>
<th>CUSTOMER</th>
<th>TIME</th>
<th>PRODUCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>herring</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>corned_beef</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>olives</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>3</td>
<td>ham</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>4</td>
<td>turkey</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>5</td>
<td>bourbon</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>6</td>
<td>ice_cream</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>baguette</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>soda</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>2</td>
<td>herring</td>
</tr>
</tbody>
</table>
Partial Listing of the 2-Item Sequences

The OUT= sequence data set contains the following variables:

- **COUNT** - contains the number of transactions meeting the rule.
- **SUPPORT** - contains the percent of support, that is the percent of the total number of transactions that qualify for the rule.
- **CONF** - contains the percent of confidence.
- **RULE** - contains the text rule.
- **ISET1, ISET, ISETn** - contain, in order, the events that form the event chain. For this example, the ISET variables are not printed.

The first rule, cracker ==> unhidden, indicates that 337 customers bought unhidden after buying crackers. The confidence factor indicates that 69% of the time a customer will buy unhidden after they buy crackers. Thirty-three percent of the customer base supports this rule.

### Partial Listing of the 2-Item Sequences

<table>
<thead>
<tr>
<th>OBS</th>
<th>COUNT</th>
<th>SUPPORT</th>
<th>CONF</th>
<th>RULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>337</td>
<td>33.6663</td>
<td>69.0574</td>
<td>cracker ==&gt; unhidden</td>
</tr>
<tr>
<td>2</td>
<td>235</td>
<td>23.4765</td>
<td>48.3539</td>
<td>herring ==&gt; heineken</td>
</tr>
<tr>
<td>3</td>
<td>233</td>
<td>23.2767</td>
<td>49.2600</td>
<td>olives ==&gt; bourbon</td>
</tr>
<tr>
<td>4</td>
<td>229</td>
<td>22.8771</td>
<td>47.1193</td>
<td>herring ==&gt; corned_beef</td>
</tr>
<tr>
<td>5</td>
<td>226</td>
<td>22.5774</td>
<td>46.5021</td>
<td>herring ==&gt; olives</td>
</tr>
<tr>
<td>6</td>
<td>225</td>
<td>22.4775</td>
<td>57.3980</td>
<td>baguette ==&gt; heineken</td>
</tr>
<tr>
<td>7</td>
<td>220</td>
<td>21.9780</td>
<td>69.1824</td>
<td>soda ==&gt; cracker</td>
</tr>
<tr>
<td>8</td>
<td>220</td>
<td>21.9780</td>
<td>56.1224</td>
<td>baguette ==&gt; herring</td>
</tr>
<tr>
<td>9</td>
<td>220</td>
<td>21.9780</td>
<td>46.5116</td>
<td>olives ==&gt; turkey</td>
</tr>
<tr>
<td>10</td>
<td>218</td>
<td>21.7782</td>
<td>68.5535</td>
<td>soda ==&gt; heineken</td>
</tr>
</tbody>
</table>

### Partial Log Listing

1 proc dmdb batch data=sampsio.assocs out=dmseq dmdbcat=catseq;
2 id customer time;
3 class product(desc);
4 run;
5 Records processed= 7007 Mem used = 511K.
6 NOTE: The PROCEDURE DMDB used 0:00:02.65 real 0:00:00.96 cpu.
7 proc assoc data=dmseq dmdbcat=catseq
8 out=aout(label='Output from Proc Assoc')
9 10 items=5 support=20;
11 cust customer;
12 target product;
13 run;
14
15 ----- Potential 1 item sets = 20 -----
Counting items, records read: 7007
Number of customers: 1001
Support level for item sets: 20
Maximum count for a set: 600
Sets meeting support level: 20
Megs of memory used: 0.51

----- Potential 2 item sets = 190 -----  
Counting items, records read: 7007
Maximum count for a set: 366
Sets meeting support level: 183
Megs of memory used: 0.51

----- Potential 3 item sets = 1035 -----  
Counting items, records read: 7007
Maximum count for a set: 234
Sets meeting support level: 615
Megs of memory used: 0.51

----- Potential 4 item sets = 1071 -----  
Counting items, records read: 7007
Maximum count for a set: 137
Sets meeting support level: 317
Megs of memory used: 0.51

----- Potential 5 item sets = 85 -----  
Counting items, records read: 7007
Maximum count for a set: 116
Sets meeting support level: 71
Megs of memory used: 0.51

NOTE: The PROCEDURE ASSOC used 0:00:06.52 real 0:00:03.16 cpu.

16 17 proc sequence data=dmseq dmdbcat=catseq
18      out=sout(label='Output from Proc Sequence')
19 20
21      nitems=2;
22 23 cust customer;
24      target product;
25 26 visit time;
27 run;
Large itemsets: 1206
Total records read: 7007
Customer count: 1001
Support set to: 20
Total Litem Sequences: 398
Number >= support 291
Memory allocated megs: 2

NOTE: The PROCEDURE SEQUENCE used 0:00:05.74 real 0:00:02.52 cpu.
The PROC PRINT procedure lists the first 10 observations in the SAMPSIO.ASSOCS data set.

``` Sas
proc print data=sampsio.assoc(obs=10);
   title 'Partial Listing of the ASSOCS Data Set';
run;
```
Before you can run the ASSOCIATION and SEQUENCE procedures, you must create the DMDB data set and the DMDB catalog by using a PROC DMBD step.

```sql
proc dmbd batch data=sampsio.assocs out=dmseq dmbcat=catseq;
  id customer time;
  class product(desc);
run;
```
The ASSOCIATION procedure determines the products that are related. The DATA= and DMDB= options identify the DMDB data set and catalog, respectively. PROC ASSOC writes the related products to the OUT= data set; this data set is used as input by the SEQUENCE procedure.

proc assoc data=dmseq dmdbcat=catseq
    out=aout(label='Output from Proc Assoc')
The ITEMS= option specifies the maximum size of the item set to be considered (default=4). The SUPPORT= option specifies the minimum support level that is required for a rule to be accepted (default =5% of the largest frequency).

items=5 support=20;
The CUST statement (alias = CUSTOMER) specifies the ID variable. The TARGET statement specifies the nominal target variable.

cust customer;
target product;
run;
The DATA= and DMDB= options identify the DMDB data set and catalog, respectively. The ASSOC= option identifies the name of the input data set from the previous PROC ASSOC run.

```plaintext
proc sequence data=dmseq dmdbcat=catseq
  assoc=aout
  out=sout(label='Output from Proc Sequence')
```
The NITEMS= option specifies the maximum number of events for which rules, or chains, are generated. By default, the SEQUENCE procedure computes binary sequences (NITEMS=2).

nitems=2;
The CUST statement (alias = CUSTOMER) specifies the ID variable. The TARGET statement specifies the nominal target variable.

cust customer;
target product;
The VISIT statement names the timing or sequence variable.

    visit time;
run;
The SORT procedure sorts the observations in descending order by the values of support.

proc sort data=sout;
    by descending support;
run;
The PRINT procedure lists the first 10 observations in the sorted sequence data set.

```
proc print data=sout(obs=10);
    var count support conf rule;
    title 'Partial Listing of the 2-Item Sequences';
run;
```
Example 2: Specifying the Maximum Number of Item Events and Setting the Lower Timing Limit

This example demonstrates how to specify the maximum number of item events and how to set the lower timing limit of a sequence rule. Before you run the example program, you should submit the PROC DMDB and PROC ASSOC steps from Example 1.

```sas
proc sequence data=dmseq
dmdbcat=catseq
assoc=aout
out=s4out(label = 'Output from Proc Sequence')
   nitems=4;
   cust customer;
target product;
   visit time / same=2;
run;
```

```sas
proc sort data=s4out;
   by descending support;
run;
```

```sas
proc print data=s4out(obs=10);
   var count support conf rule;
   title 'Partial Listing of the 4-Item Sequences';
title2 'Lower Timing Limit Set to 2';
run;
```

Output
Partial PROC PRINT Listing of the 4-Item Sequence Data Set, Lower Time Set to 2

When the lower time limit is set to 2, the rule with the highest support is now a herring purchase followed by a heineken purchase. Twenty-three percent of the customer population supports it, with a 48% confidence.

Partial Listing of the 4-Item Sequences
Lower Timing Limit Set to 2

<table>
<thead>
<tr>
<th>OBS</th>
<th>COUNT</th>
<th>SUPPORT</th>
<th>CONF</th>
<th>RULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>235</td>
<td>23.4765</td>
<td>48.3539</td>
<td>herring ==&gt; heineken</td>
</tr>
<tr>
<td>2</td>
<td>225</td>
<td>22.4775</td>
<td>57.3980</td>
<td>baguette ==&gt; heineken</td>
</tr>
<tr>
<td>3</td>
<td>220</td>
<td>21.9780</td>
<td>69.1824</td>
<td>soda ==&gt; cracker</td>
</tr>
<tr>
<td>4</td>
<td>218</td>
<td>21.7782</td>
<td>68.5535</td>
<td>soda ==&gt; heineken &amp; cracker</td>
</tr>
<tr>
<td>5</td>
<td>218</td>
<td>21.7782</td>
<td>68.5535</td>
<td>soda ==&gt; heineken</td>
</tr>
<tr>
<td>6</td>
<td>215</td>
<td>21.4785</td>
<td>45.4545</td>
<td>olives ==&gt; turkey</td>
</tr>
<tr>
<td>7</td>
<td>213</td>
<td>21.2787</td>
<td>52.8536</td>
<td>bourbon ==&gt; cracker</td>
</tr>
<tr>
<td>8</td>
<td>209</td>
<td>20.8791</td>
<td>100.0000</td>
<td>herring &amp; baguette ==&gt; heineken</td>
</tr>
<tr>
<td>9</td>
<td>201</td>
<td>20.0799</td>
<td>55.3719</td>
<td>avocado ==&gt; heineken</td>
</tr>
<tr>
<td>10</td>
<td>150</td>
<td>14.9850</td>
<td>30.8642</td>
<td>herring ==&gt; cracker</td>
</tr>
</tbody>
</table>

Partial Log Listing

```plaintext
1 proc sequence data=dmseq
dmdbcat=catseq
assoc=aout
   out=s4out(label = 'Output from Proc Sequence')
   nitems=4;
cust customer;
target product;
visit time / same=2;
run;
```

Large itemsets:            1206
Total records read:        7007
Customer count:            1001
Support set to:              20
Total Litem Sequences:     5641
Number >= support           466
   --- Number Items:         3 ---
   Total records read:       7007
   Customer count:           1001
   Total Litem Sequences:    5086
Number >= support           12
   --- Number Items:         4 ---
   Total records read:       7007
   Customer count:           1001
   Total Litem Sequences:    0
Number >= support  0
Memory allocated megs:  2
NOTE: The PROCEDURE SEQUENCE used 0:00:33.42 real 0:00:16.17 cpu.
The NITEMS= option specifies the maximum number of events for which rules, or chains, are generated.

nitems=4;
The SAME= option specifies the lower time-limit between the occurrence of two events that you want to associate with each other (default = 0).

```
visit time / same=2;
run;
```
The SORT procedure sorts the observations in descending order by the values of support.

```
proc sort data=s4out;
  by descending support;
run;
```
The PRINT procedure lists the first 10 observations in the sorted sequence data set.

```plaintext
proc print data=s4out(obs=10);
   var count support conf rule;
   title 'Partial Listing of the 4-Item Sequences';
   title2 'Lower Timing Limit Set to 2';
run;
```

The SPLIT Procedure

Overview

Procedure Syntax

PROC SPLIT Statement
CODE Statement
DECISION Statement
DESCRIBE Statement
FREQ Statement
INPUT Statement
PRIORS Statement
PRUNE Statement
SCORE Statement
TARGET Statement

Details

Examples

Example 1: Creating a Decision Tree with a Categorical Target (Rings Data)
Example 2: Creating a Decision Tree with an Interval Target (Baseball Data)

References

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Overview

An empirical decision tree represents a segmentation of the data created by applying a series of simple rules. Each rule assigns an observation to a segment based on the value of one input. One rule is applied after another, resulting in a hierarchy of segments within segments. The hierarchy is called a tree, and each segment is called a node. The original segment contains the entire data set and is called the root node of the tree. A node with all its successors form a branch of the node that created it. The final nodes are called leaves. For each leaf, a decision is made and applied to all observations in the leaf. The type of decision depends on the context. In predictive modeling, the decision is simply the predicted value.

Besides modeling, decision trees can also select inputs or create dummy variables representing interaction effects for use in a subsequent model, such as regression.

PROC SPLIT creates decision trees to either:

- classify observations based on values of nominal or binary targets,
- predict outcomes for interval targets, or
- predict the appropriate decision when decision alternatives are specified.

PROC SPLIT can save the tree information in a SAS data set, which can be read again into the procedure later.

PROC SPLIT can apply the tree to new data and create an output data set containing the predictions, or the dummy variables for use in subsequent modeling. Alternatively, PROC SPLIT can generate DATA step code for the same purpose.

Tree construction options include the popular features of CHAID (Chi-squared automatic interaction detection) and those described in Classification and Regression Trees (Breiman, et al. 1984).

For example, using chi-square or F-test $p$-values as a splitting criterion, tree construction may stop when the adjusted $p$-value is less significant than a specified threshold level, as in CHAID.

When a tree is created for any splitting criterion, the best sub-tree for each possible number of leaves is automatically found. The sub-tree that works best on validation data may be selected automatically, as in the Classification and Regression Trees method. The notion of "best" is implemented using an assessment function equal to a profit matrix (or function) of target values.

Decision tree models are often easier to interpret than other models because the leaves are described using simple rules. Another advantage of decision trees is in the treatment of missing data. The search for a splitting rule uses the missing values of an input. Surrogate rules are available as backup when missing data prohibit the application of a splitting rule.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Procedure Syntax

PROC SPLIT<option(s)>;

CODE <option(s)>;

DECISION DECDATA=<libref>SAS-data-set <DECVARS=decision-variable(s)> <option(s)>;

DESCRIBE <options>;

FREQ variable;

IN | INPUT variable(s) </option(s)>;

PRIORS probabilities;

PRUNE node-identifier;

SCORE <score-option(s)>;

TARGET variable </LEVEL=value> ;
**PROC SPLIT Statement**

PROC SPLIT <option(s)>;

### Data Set Options

<table>
<thead>
<tr>
<th>OPTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA=</td>
<td>Specifies the data set containing observations used to create the model. Default: none.</td>
</tr>
<tr>
<td>DMDBCAT=</td>
<td>Specifies the DMDB metabase associated with the data. Default: none.</td>
</tr>
<tr>
<td>INDMSPLIT</td>
<td>Requests that the tree created by PROC DMSPLIT be input.</td>
</tr>
<tr>
<td>INTREE=</td>
<td>Specifies the input data set describing a previously created tree.</td>
</tr>
<tr>
<td>OUTAFDS=</td>
<td>Specifies the output data set for the user interface components of SAS/AF. (These components [or widgets] can be scrollbars, pushbuttons, text fields, and so on.)</td>
</tr>
<tr>
<td>OUTIMPORTANCE=</td>
<td>Specifies the output data set with variables importance.</td>
</tr>
<tr>
<td>OUTLEAF=</td>
<td>Names the output data set that is to contain statistics for each leaf node.</td>
</tr>
<tr>
<td>OUTMATRIX=</td>
<td>Names the output data set that is to contain summary statistics.</td>
</tr>
<tr>
<td>OUTSEQ=</td>
<td>Specifies the output data set with sub-tree statistics.</td>
</tr>
<tr>
<td>OUTTREE=</td>
<td>Specifies the output data set describing the tree.</td>
</tr>
<tr>
<td>VALIDATA=</td>
<td>Specifies the validation data set.</td>
</tr>
</tbody>
</table>

*Tree Construction Options*
<table>
<thead>
<tr>
<th>OPTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSESS=</td>
<td>Specifies the model assessment measure.</td>
</tr>
<tr>
<td>COSTSPLIT</td>
<td>Requests that the split search criterion incorporate the decision matrix.</td>
</tr>
<tr>
<td>CRITERION=</td>
<td>Specifies the method of model construction.</td>
</tr>
<tr>
<td>EXCLUDEMISS</td>
<td>Specifies that missing values be excluded during a split search.</td>
</tr>
<tr>
<td>EXHAUSTIVE=n</td>
<td>Specifies the highest number of candidate splits to find in an exhaustive search.</td>
</tr>
<tr>
<td>LEAFSIZE=</td>
<td>Specifies the minimum size of a node.</td>
</tr>
<tr>
<td>LIFTDEPTH=</td>
<td>Specifies the proportion of data to use with the LIFT = ASSESSMENT.</td>
</tr>
<tr>
<td>MAXBRANCH=</td>
<td>Specifies the maximum number of child nodes of a node.</td>
</tr>
<tr>
<td><strong>MAXDEPTH</strong>=</td>
<td>Specifies the limiting depth of tree.</td>
</tr>
<tr>
<td><strong>NODESAMPLE</strong>=</td>
<td>Specifies the size for searches, within the node sample.</td>
</tr>
<tr>
<td><strong>NRULES</strong>=</td>
<td>Specifies the number of rules saved with each node.</td>
</tr>
<tr>
<td><strong>NSURRS</strong>=</td>
<td>Specifies the number of surrogates sought in each non-leaf node.</td>
</tr>
<tr>
<td><strong>PADJUST</strong>=</td>
<td>Specifies the options for adjusting p-values.</td>
</tr>
<tr>
<td><strong>Pvars</strong>=</td>
<td>Specifies the adjusting p-value for the number of variables.</td>
</tr>
<tr>
<td><strong>SPLITSIZE</strong>=</td>
<td>Specifies the minimum size of a node required for split.</td>
</tr>
<tr>
<td><strong>SUBTREE</strong>=</td>
<td>Specifies the method for selecting the sub-tree.</td>
</tr>
<tr>
<td><strong>USEVARONCE</strong></td>
<td>Specifies that no node is split on an input that an ancestor is split on.</td>
</tr>
</tbody>
</table>
**WORTH=**
Specifies worth required of splitting rule.

---

**Required Arguments**

**DATA=<libref.>SAS-data-set**
Names the input training data set if constructing a tree. Variables named in the FREQ, INPUT, and TARGET statements refer to variables in the DATA= SAS data set.

Default: None

**DMDBCAT=<libref.>SAS-catalog**
Names the SAS catalog describing the DMDB metabase. The DMDB metabase contains the formatted values of all NOMINAL variables, and how they are coded in the DATA= SAS data set. Required with the DATA= option.

Default: None

To learn how to create the DMDB encoded data set and catalog, see the PROC DMDB chapter.

---

**Options**

**ASSESS=**
Specifies how to evaluate a tree. The construction of the sequence of sub-trees uses the assessment measure. Possible measures are:

**IMPURITY**
Total leaf impurity (Gini index or Average Squared Error).

**LIFT**
Average assessment in highest ranked observations.

**PROFIT**
Average profit or loss from the decision function.

**STATISTIC**
Nominal Classification Rate or Average Squared Error.
Default: PROFIT

The default PROFIT measure is set to STATISTIC if no DECISION statement is specified.

LIFT restricts the default PROFIT or STATISTIC measure to those observations predicted to have the best assessment. The LIFTDEPTH= option specifies the proportion of observations to use.

If ASSESS=IMPURITY, then the assessment of the tree is measured as the total impurity of all its leaves. For interval targets, this is the same as using Average Squared Error (ASSESS=STATISTIC).

For categorical targets, the impurity of each leaf is evaluated using the Gini index. The impurity measure produces a finer separation of leaves than a classification rate and is, therefore, preferable for lift charts. ASSESS=LIFT generates the sequence of sub-trees using ASSESS=IMPURITY and then prunes using the LIFT measure.

ASSESS=IMPURITY implements class probability trees as described in Brieman et al., section 4.6 (1984).

COSTSPLIT

Requests that the split search criterion incorporate the decision matrix. To use COSTSPLIT, CRITERION must equal ENTROPY or GINI, and the type of the DECDATA data set must be PROFIT or LOSS. For ordinal targets, COSTSPLIT is superfluous because the decision matrix is always incorporated into the criterion.

CRITERION=method

Specifies the method of searching for and evaluating candidate splitting rules. Possible methods depend on the level of measurement appropriate for the target variable, as follows:

BINARY or NOMINAL TARGETS:

Method=CHISQ
Pearson Chi-square statistic for target vs. segments.
Method=PROBCHISQ
p-value of Pearson Chi-square statistic for target vs. segments. Default for NOMINAL.
Method=ENTROPY
Reduction in entropy measure of node impurity.
Method=ERATIO
Reduction in entropy of split.
Method=GINI
Reduction in Gini measure of node impurity.

INTERVAL TARGETS
Method=VARIANCE
    Reduction in squared error from node means.
Method=PROBF
    p-value of F-test associated with node variance. Default for INTERVAL.
Method=F
    F statistic associated with node variance.

EXCLUDEMISS
    Specifies that missing values be excluded during a split search.

EXHAUSTIVE=n
    Specifies the most number of candidate splits to find in an exhaustive search. If more candidates
    would have to be considered, a heuristic search is used instead. The EXHAUSTIVE option applies
    to multi-way splits, and for binary splits on nominal targets with more than 2 values.
    
    Default: The default value is 5000.

INDMSPLIT
    Requests that the tree created by PROC DMSPLIT be input to PROC SPLIT. The tree is expected
    in the DMDBCAT= catalog. The DMDBCAT= option is required, and the INDMTREE and
    INTREE= options are prohibited.

INTREE=SAS-tree-model
    Names a data set created from the PROC SPLIT OUTTREE= option.
    
    Caution: When using the INTREE option, the IN, TARGET, and FREQ statements are
    prohibited, as are the DECISION and PRIORS statements.

LEAFSIZE=n
    Specifies the smallest number of training observations a node can have.

LIFTDEPTH=n
    Specifies the proportion of observations to use with ASSESS=LIFT.

MAXBRANCH=n
    Restricts the number of subsets a splitting rule can produce to n or fewer. A value of 2 results in
    binary trees.
    
    Range: 2 - 100

    Default: 2

MAXDEPTH=depth
    Specifies the maximum number of generations of nodes. The original node, generation 0, is called
    the root node. The children of the root node are the first generation. PROC SPLIT will only
    consider splitting nodes in the nth generation when n is less than the value of depth.
    
    Default: 6
**NODESAMPLE=n**

Specifies the within node sample size used for finding splits. If the number of training observations in a node is larger than \( n \), then the split search for that node is based on a random sample of size \( n \).

<table>
<thead>
<tr>
<th>Default:</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range:</td>
<td>( 1 \leq n \leq 32767 )</td>
</tr>
</tbody>
</table>

**NRULES=n**

Specifies how many splitting rules are saved with each node. The tree only uses one rule. The remaining rules are saved for comparison. Based on the criterion you selected, you can see how well the variable that was used split the data, and how well the next \( n-1 \) would have split the data.

| Default:      | 5                |

**NSURRS=n**

Specifies a number of surrogate rules sought in each non-leaf node. A surrogate rule is a backup to the main splitting rule. When the main splitting rule relies on an input whose value is missing, the first surrogate rule is invoked. For more information, see Missing Values in the Detail section.

**Note:** The option to save surrogate rules in each node is often used by advocates of CART.

| Default:      | 0                |

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

Names the output data set that is to contain a tree description suitable for inputting data into SAS/AF widgets such as ORGCHART and TREERING.

**Definition:** A SAS/AF Widget is a visible part of a window, which can be treated as a separate, isolated entity. For example, a SAS/AF Widget can be a scrollbar, a text field, a pushbutton, and so on. It is an individual component of the user interface.

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

Names the output data set that contains statistics for each leaf node.

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

Names the output data set that contains tree summary statistics. For nominal targets, the summary statistics consist of the counts and proportions of observations correctly classified. For interval targets, the summary statistics include the average squared prediction error and R-squared, which equals
1 - \frac{\text{average squared prediction error}}{\text{sum of squares from the average}}.

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

Names the output data set that contains statistics on each sub-tree in the sub-tree sequence.

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

Names the output data set that contains all the tree information. This data set can then be used on subsequent executions of PROC SPLIT.

**PADJUST=**methods

Names methods of adjusting the $p$-values used with the PROBCHISQ and PROBFTEST criteria. Possible methods are:

- **KASSAFTER**
  
  Bonferroni adjustment applied after split is chosen.

- **KASSBEFORE**
  
  Bonferroni adjustment applied before split is chosen.

- **DEVILLE**
  
  Adjustment independent of number of branches in split.

- **DEPTH**
  
  Adjustment for number of ancestor splits.

- **NOGABRIEL**
  
  Turns off adjustment that sometimes overrides KASS.

- **NONE**
  
  No adjustment is made.

**Caution:** This option is ignored unless CRITERION= PROBCHISQ or PROBFTEST.

**PVARS=n**

Specifies the number of inputs to consider uncorrelated when adjusting $p$-values for the number of inputs.

**SPLITSIZE=n**

Specifies the smallest number of training observations a node must have for PROC SPLIT to consider splitting it.

**Range:** Maximum is 32767 on most machines.

**Default:** The greater of either 50 or the total number of cases in the training data set divided by 100.

**SUBTREE=**method

Specifies how to construct the sub-tree in terms of selection methods. The following methods are
Sub-Tree Selection Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSESSMENT</td>
<td>Best assessment value</td>
</tr>
<tr>
<td>LARGEST</td>
<td>The largest tree in the sequence</td>
</tr>
<tr>
<td>n</td>
<td>Largest sub-tree with no more than n leaves</td>
</tr>
</tbody>
</table>

**Default:** ASSESSMENT

**USEVARONCE**

Specifies that no node is split on an input an ancestor is split on.

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

Names the input SAS data set for validation.

**WORTH=threshold**

Specifies a threshold $p$-value for the worth of a candidate splitting rules. The measure of worth depends on the CRITERION= method.

**Range:** For a method based on $p$-values, the threshold is a maximum acceptable $p$-value; for other criteria, the threshold is the minimal acceptable increase in the measure of worth.

**Default:** For a method based on $p$-values, the default is 0.20; for other criteria, the default is 0.
The SPLIT Procedure

CODE Statement

Generates SAS DATA step code that generally mimics the computations done by the SCORE statement.

```sas
CODE <option(s)>;
```

**Options**

**DUMMY**
Requests creation of a dummy variable for each leaf node. The value of the dummy variable is 1 for observations in the leaf and 0 for all other observations.

**FILE=quoted-filename**
Specifies the file name that contains the code.

Default: LOG

**FORMAT=format**
Specifies the format to be used in the DATA step code for numeric values that don't have a format from the input data set.

**NOLEAFID**
Suppresses the creation of the _NODE_ variable containing a numeric id of the leaf to which the observation is assigned.

**NOPRED**
Suppresses the creation of predicted variables, such as P_*.

**RESIDUAL**
Requests code that assumes the existence of the target variable.

Default: By default, the code contains no reference to the target variable (to avoid confusing notes or warnings). The code computes values that depend on the target variable (such as the R_*, E_*, F_*, CL_*, CP_*, BL_*, BP_*, or ROI_* variables) only if the RESIDUAL option is specified.
DECISION Statement

Specifies information used for decision processing in the DECIDE, DMREG, NEURAL, and SPLIT procedures. *This documentation applies to all four procedures.*

**Tip:** The DECISION statement is required for the DECIDE and NEURAL procedures. It is optional for the DMREG and SPLIT procedures.

```sas
DECISION DECDATA=<libref.> SAS-data-set <DECVARS=decision-variable(s)><option(s)>;
```

**DECDATA=** `<libref.> SAS-data-set`

Specifies the input data set that contains the decision matrix. The DECDATA= data set must contain the target variable.

**Note:** The DECDATA= data set may also contain decision variables specified by means of the DECVARS= option, and prior probability variable(s) specified by means of the PRIORVAR= option or the OLDPRIORVAR= option, or both.

The target variable is specified by means of the TARGET statement in the DECIDE, NEURAL, and SPLIT procedures or the MODEL statement in the DMREG procedure. If the target variable in the DATA= data set is categorical then the target variable of the DECDATA= data set should contain the category values, and the decision variables will contain the common consequences of making those decisions for the corresponding target level. If the target variable is interval, then each decision variable will contain the value of the consequence for that decision at a point specified in the target variable. The unspecified regions of the decision function are interpolated by a piecewise linear spline.

**Tip:** The DECDATA= data set may be of TYPE=LOSS, PROFIT, or REVENUE. If unspecified, TYPE= is assumed to be PROFIT by default. TYPE= is a data set option that should be specified when the data set is created.

**DECVARS=decision-variable(s)**

Specifies the decision variables in the DECDATA= data set that contain the target-specific consequences for each decision.

**Default:** None

**COST=cost-option(s)**

Specifies numeric constants giving the cost of a decision, or variables in the DATA= data set that contain the case-specific costs, or any combination of constants and variables. There must be the same number of cost constants and variables as there are decision variables in the DECVARS=
In the COST= option, you may not use abbreviated variable lists such as D1-D3, ABC--XYZ, or PQR:

| Default:       | All costs are assumed to be 0. |

**CAUTION:**

The COST= option may only be specified when the DECDATA= data set is of TYPE=REVENUE.

**PRIORVAR=** *variable*

Specifies the variable in the DECDATA= data set that contains the prior probabilities to use for making decisions. *In the DECIDE procedure, if PRIORVAR= is specified, OLDPRIORVAR= must also be specified.*

| Default:       | None |

**OLDPRIORVAR=** *variable*

Specifies the variable in the DECDATA= data set that contains the prior probabilities that were used when originally fitting the model. *If OLDPRIORVAR= is specified, PRIORVAR= must also be specified.*

**CAUTION:**

OLDPRIORVAR= is not allowed in PROC SPLIT.

| Default:       | None |
DESCRIBE Statement

Generates the output of a simple description of the rules that define each leaf, along with a few statistics. The description is easier to understand than the equivalent information output using the CODE statement.

DESCRIBE <option(s)>;

Options

FILE=quoted-filename

Specifies the file name that contains the description.

Default: LOG

FORMAT= format

Specifies the format to be used in the DATA step code for numeric values that don't have a format from the input data set.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The SPLIT Procedure

FREQ Statement

Specifies the frequency variable.

```
FREQ variable;
```

Options

```
variable
```
Names a variable that provides frequencies for each observation in the DATA= data set. If \( n \) is the value of the FREQ variable for a given observation, then that observation is used \( n \) times.

Default: If the value of the FREQ variable is missing or less than 0, then the observation is not used in the analysis. The values for FREQ variables are never truncated.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
**INPUT Statement**

Names input variables with common options.

**Tip:** Multiple INPUT statements can be used to specify input variables of a different type and order.

```
INPUT | IN variable-list </option(s)>
```

**Options**

The following options are available:

<table>
<thead>
<tr>
<th>OPTIONS</th>
<th>VALUES</th>
<th>DEFAULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL=</td>
<td>NOMINAL</td>
<td>ORDINAL</td>
</tr>
<tr>
<td>ORDER=</td>
<td>ASCENDING</td>
<td>DESCENDING</td>
</tr>
<tr>
<td></td>
<td>ASCFORMATTED</td>
<td>DESFORMATTED</td>
</tr>
</tbody>
</table>

**Note:** Interval variables have numeric values, so an average of two values is another meaningful value. Values of an ordinal variable represent an ordering, but, unlike interval variables, an average of ordinal values is not meaningful. For example, taking an average of ages 15 and 20 is another meaningful age; but taking an average of "TEENAGER" and "YOUNG ADULT" is not meaningful.

Values of an ordinal variable can be defined either by their formatted values (ORDER= ASCENDING | DESCENDING), or by their unformatted values (ORDER= ASCFORMATTED | DESFORMATTED), or by their order of appearance in the training data (ORDER=DSORDER). The unformatted values can be either numeric or character. When the unformatted value determines the order, the smallest unformatted value for a given formatted value represents that formatted value.
The ORDER= option is only allowed for ordinal variables. Values of a nominal variable have no implicit ordering. Typical nominal inputs are GENDER, GROUP, and JOBCLASS.

A splitting rule based on a nominal input is usually free to assign any subset of categories to any subset of the node. The number of ways to assign the categories becomes very large if there are many categories compared to the number of node subsets. For LEVEL=NOMINAL, values are defined by the formatted value.
PRIORS Statement

Specifies the prior probabilities of the values of a nominal or ordinal target.

**Tip:** A prior probability for a value of a target represents the proportion in which that value appears in the data to which the tree-model is intended to apply.

**Caution:** The PRIORS statement is not allowed if a DECISION statement is used; instead use the PRIORVAR= option to specify prior probabilities. The PRIORS statement is not valid for an interval target and will result in an error if used.

PRIORS probabilities;

### Required Arguments

**Probabilities** can be one of the following:

**PROPORTIONAL | PROP**

Specifies that the proportions are the same as in the training data.

**EQUAL**

Specifies equal proportions.

'value-1'=probability-1 <...'value-n'=probability-n>

Specifies explicit probabilities.

value-1 ... value-n

Specifies each formatted value of the target; each value listed is followed by an equal sign. Formatted values are enclosed in single quotes. All non-missing values of the target should be included.

probability-1 ... probability-n

Specifies the probability that is a numeric constant between 0 and 1.

**Default:** PROPORTIONAL

**Example:** PRIORS '-1'=0.4 '0'=0.2 '1'=0.4;

This example specifies probabilities of 0.4, 0.2, and 0.4 for target values, -1, 0, and 1, respectively. An error occurs if the training data contains other non-missing values of the target. The formatted values depend on the format you choose. If the target uses a format of 5.2, then use: PRIORS '-1.00'=0.4 '0.00'=0.2 '1.00'=0.4;
PRUNE Statement

Deletes all nodes descended from any specified node.

Interaction: The PRUNE statement requires an INTREE= or INDMSPLIT procedure option.

```
PRUNE list-of-node-identifiers;
```

Required Argument

*list-of-node-identifiers*

Specifies the nodes that will have no children.

<table>
<thead>
<tr>
<th>Range:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer &gt; 0</td>
</tr>
</tbody>
</table>

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
**SCORE Statement**

Specifies that the data be scored.

```
SCORE
  DATA= <libref.>SAS-data-set
  OUT= SAS-data-set
  <score-option(s)> </NODES=node-list>;
```

**Required Arguments**

- **DATA= <libref.>SAS-data-set**
  Specifies input data that contains inputs and, optionally, targets.

- **OUT= SAS-data-set**
  Output data set with outputs.

**Options**

- **DUMMY**
  Includes dummy variables for each node. For each observation the value of the dummy variables is 1 if the observation appears in the node and 0 if it does not.

- **NODES=node-list**
  Specifies a list of nodes used to score the observations. If an observation does not fall into any node list, it does not contribute to the statistics and is not output. If an observation occurs in more than one node, it contributes multiple times to the statistics and is output once for each node it occurs in.

**Interaction:**

- The NODES= option requires the INTREE= or INDMSPLIT procedure option.

**Default:**

- The default is the list of leaf nodes. Omitting the NODES= option results in the decisions, utilities, and leaf assignment being output for each observation in the DATA= data set.

- **NOLEAFID**
  Does not include lead identifiers or node numbers.

- **NOPRED**
Does not include predicted values.

**OUTFIT=**\textit{SAS-data-set}

Output data set with fit statistics.

**ROLE=**\textit{role-value}

Specifies the role of the DATA= data set. The ROLE= option primarily affects what fit statistics are computed and what their names and labels are. \textit{Role-value} can be:

- **TRAIN**
  The default when DATA= data set name in the PROC statement is the same as the data set name in the SCORE statement.

- **VALID** | **VALIDATION**
  The default when DATA= data set name in the SCORE statement is the same as DATA= data set name in the VALIDATA= option in the PROC statement.

- **TEST**
  The default when DATA= data set name in the SCORE statement is not the same as the data set name in the DATA= or VALIDATA= option in the PROC statement.

- **SCORE**
  Residuals, computed profit, and fit statistics are not produced.

\textit{Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.}
TARGET Statement

Specifies an output variable.

TARGET variable < / LEVEL=measurement>

Required Argument

variable

Specifies the variable that the model-fitting tries to predict.

Options

LEVEL=measurement

Specifies the measurement level, where measurement can be:

- BINARY
- NOMINAL
- ORDINAL
- INTERVAL

Default: LEVEL=INTERVAL.
Details

Missing Values

Observations in which the target value is missing are ignored when training or validating the tree.

If EXCLUDEMISS is specified, then observations with missing values are excluded during the search for a splitting rule. A search uses only one variable, and so only the observations missing on the single candidate input are excluded. An observation missing input x but not missing input y is used in the search for a split on y but not x. After a split is chosen, the rule is amended to assign missing values to the largest branch.

If EXCLUDEMISS is not specified, the search for a split on an input treats missing values as a special, acceptable value, and includes them in the search. All observations with missing values are assigned to the same branch.

The branch may or may not contain other observations. The branch chosen is the one that maximizes the split worth.

For splits on a categorical variable, this amounts to treating a missing value as a separate category. For numerical variables, it amounts to treating missing values as having the same unknown non-missing value.

One advantage of using missing data during the search is that the worth of split is computed with the same number of observations for each input. Another advantage is that an association of the missing values with the target values can contribute to the predictive ability of the split. One disadvantage is that missing values could unjustifiably dominate the choice of split.

When a split is applied to an observation in which the required input value is missing, surrogate splitting rules are considered before assigning the observation to the branch for missing values.

A surrogate splitting rule is a backup to the main splitting rule. For example, the main splitting rule might use county as input and the surrogate might use region. If the county is unknown and the region is known, the surrogate is used.

If several surrogate rules exist, each surrogate is considered in sequence until one can be applied to the observation. If none can be applied, the main rule assigns the observation to the branch designated for missing values.

The surrogates are considered in the order of their agreement with the main splitting rule. The agreement is measured as the proportion of training observations it and the main rule assign to the same branch. The measure excludes the observations that the main rule cannot be applied to. Among the remaining observations, those on which the surrogate rule cannot be applied count as observations not assigned to the same branch. Thus, an observation with a missing value on the input used in the surrogate rule but
not the input used in the primary rule counts against the surrogate.

The NSURRS= procedure option determines the number of surrogates sought. A surrogate is discarded if its agreement is $\leq 1/B$, where B is the number of branches. As a consequence, a node might have fewer surrogates than the number specified in the NSURRS= option.

**METHOD OF SPLIT SEARCH**

For a specific node and input, PROC SPLIT seeks the split with maximum worth or $-\log(p$-value) subject to the limit on the number of branches and the limit on the minimum number of observations assigned to a branch. The procedure options MAXBRANCH= and LEAFSIZE= set these limits.

The measure of worth depends on the splitting criterion. The ENTROPY, GINI, and VARIANCE reduction criteria measure worth as $I(\text{node}) - \sum_{\text{branches } b} P(b) I(b)$, where $I(\text{node})$ denotes the entropy, gini, or variance measure in the node, and $P(b)$ denotes the proportion of observations in the node assigned to branch $b$. If prior probabilities are specified, then the proportions $P(b)$ are modified accordingly.

The PROBCHISQ and PROBF criteria use the $-\log(p$-value) measure. For these criteria, the best split is the one with the smallest p-value. If the PADJUST=KASSBEFORE procedure option is in effect, as it is by default, then the p-values are first multiplied using the appropriate Bonferroni factor. Adjusting the p-value may cause it to become less significant than an alternative method of computing the p-value, called Gabriel's adjustment. If so, then Gabriel's p-value is used.

For nodes with many observations, the algorithm uses a sample for the split search, for computing the worth, and for observing the limit on the minimum size of a branch. The NODESAMPLE= procedure option specifies the size of the sample and is limited to 32,767 observations (ignoring any frequency variable) on many computer platforms. The samples in different nodes are taken independently.

For nominal targets, the sample is as balanced as possible. Suppose for example that a node contains 100 observations of one value of a binary target, 1,000 observations of the other value, and NODESAMPLE=200 or more. Then all 100 observations of the first target value are in the node sample.

For binary splits on binary or interval targets, the optimal split is always found. For other situations, the data is first consolidated, and then either all possible splits are evaluated or else an heuristic search is used.

The consolidation phase searches for groups of values of the input which seem likely to be assigned to the same branch in the best split. The split search regards observations in the same consolidation group as having the same input value. The split search is faster because fewer candidate splits need evaluating.

If, after consolidation, the number of possible splits is greater than the number specified in the EXHAUSTIVE= procedure option, then a heuristic search is used. The heuristic algorithm alternately merges branches and reassigns consolidated observations to different branches. The process stops when a binary split is reached. Among all candidate splits considered, the one with the best worth is chosen.

The heuristic algorithm initially assigns each consolidated group of observations to a different branch, even if the number of such branches is more than the limit allowed in the final split. At each merge step,
the two branches are merged that degrade the worth of the partition the least. After two branches are merged, the algorithm considers reassigning consolidated groups of observations to different branches. Each consolidated group is considered in turn, and the process stops when no group is reassigned.

When using the PROBCHISQ and PROBF criteria, the \( p \)-value of the selected split on an input is subjected to more adjustments: KASSAFTER, DEPTH, DEVILLE, and INPUTS. If the adjusted \( p \)-value is greater than or equal to the WORTH= procedure option, the split is rejected.

**AUTOMATIC PRUNING AND SUB-TREE SELECTION**

After a node is split, the newly created nodes are considered for splitting. This recursive process ends when no node can be split. The reasons a node will not split are

- The node contains too few observations, as specified in the SPLITSIZE= procedure option.
- The number of nodes in the path between the root node and the given node equals the number specified in the MAXDEPTH= procedure option.
- No split exceeds the threshold worth requirement specified in the WORTH= procedure option.

The last reason is the most informative. Either all the observations in the node contain nearly the same target value, or no input is sufficiently predictive in the node.

A tree adapts itself to the training data and generally does not fit as well when applied to new data. Trees that fit the training data too well often predict too poorly to use on new data.

When SPLITSIZE=, MAXDEPTH, and WORTH= are set to extreme values, and, for PROBCHISQ and PROBF criteria, PADJUST=none, the tree is apt to grow until all observations in a leaf contain the same target value. Such trees typically overfit the training data and will predict new data poorly.

A primary consideration when developing a tree for prediction is deciding how large to grow the tree, or, what comes to the same end, what nodes to prune off the tree.

The CHAID method of tree construction specifies a significance level of a Chi-square test to stop tree growth. The authors of the C4.5 and Classification and Regression Trees methods argue that the right thresholds for stopping tree construction are not knowable in advance, so they recommend growing a tree too large and then pruning nodes off.

PROC SPLIT allows for both methods. The WORTH= option accepts the significance level used in CHAID. After tree construction stops, regardless of why it stops, PROC SPLIT creates a sequence of sub-trees of the original tree, one sub-tree for each possible number of leaves. The sub-tree chosen with three leaves has the best assessment value of all candidate sub-trees with three leaves.

After the sequence of sub-trees is established, PROC SPLIT uses one of four methods to select which sub-tree to use for prediction. The SUBTREE= procedure option specifies the method. If SUBTREE=\( n \), where \( n \) is a positive integer, then PROC SPLIT uses the largest sub-tree with, at most, \( n \) leaves.

If SUBTREE=ASSESSMENT, then PROC SPLIT uses the smallest sub-tree with the best assessment value. The assessment is based on the validation data when available. (This differs from the construction
If SUBTREE=LARGEST, then PROC SPLIT uses the largest sub-tree after pruning nodes that do not increase the assessment based on the training data. For nominal targets, the largest sub-tree in the sequence might be much smaller than the original unpruned tree because a splitting rule may have a good split worth without increasing the number of observations correctly classified.

---

**CHAID**

**Description**

The inputs are either nominal or ordinal. Many software packages accept interval inputs and automatically group the values into ranges before growing the tree.

The splitting criteria is based on p-values from the F-distribution (interval targets) or Chi-square distribution (nominal targets). The p-values are adjusted to accommodate multiple testing.

A missing value may be treated as a separate value. For nominal inputs, a missing value constitutes a new category. For ordinal inputs, a missing value is free of any order restrictions.

The search for a split on an input proceeds stepwise. Initially, a branch is allocated for each value of the input. Branches are alternately merged and split again as seems warranted by the p-values. The original CHAID algorithm by Kass stops when no merge or re-splits creates an adequate p-value. The final split is adopted. A common alternative, sometimes called the exhaustive method, continues merging to a binary split, and then adopts the split with the most favorable p-value among all splits the algorithm considered.

After a split is adopted for an input, its p-value is adjusted, and the input with the best adjusted p-value is selected as the splitting variable. If the adjusted p-value is smaller than a threshold the user specified, then the node is split.

Tree construction ends when all the adjusted p-values of the splitting variables in the unsplit nodes are above the user-specified threshold.

**Relation to PROC SPLIT**

The CHAID algorithm differs from PROC SPLIT in a number of ways: PROC SPLIT seeks the split minimizing the adjusted p-value, the original KASS algorithm does not. The CHAID exhaustive method is similar to the PROC SPLIT heuristic method, except that the exhaustive method "re-splits" and PROC SPLIT "reassigns". Also, CHAID software discretizes interval inputs, while PROC SPLIT sometimes consolidates observations into groups. PROC SPLIT searches on a within node sample, unlike CHAID.
To Approximate CHAID

The interval inputs should be discretized into a few dozen values. Then set the options as follows. For nominal targets:

\[
\text{CRITERION} = \text{PROBCHISQ} \\
\text{SUBTREE} = \text{LARGEST} \text{ (to avoid automatic pruning)}
\]

For interval targets:

\[
\text{CRITERION} = \text{PROBF} \\
\text{SUBTREE} = \text{LARGEST} \text{ (to avoid automatic pruning)}
\]

For any type of target:

\[
\text{EXHAUSTIVE=} \, 0 \text{ (forces a heuristic search)} \\
\text{MAXBRANCH} = \text{maximum number of categorical values in an input} \\
\text{NODESAMPLE= size of data set, up to 32,000} \\
\text{NSURRS} = 0 \\
\text{PADJUST} = \text{KASSAFTER} \\
\text{WORTH} = 0.05 \text{ (or whatever significance level seems appropriate)}
\]

Classification and Regression Trees

Description

Classification and Regression Trees is the name of a book by Breiman, Friedman, Olshen, and Stone (BFOS) in which a tree methodology is described.

The inputs are either nominal or interval. Ordinal inputs are treated as interval.

The available splitting criteria are: reduction in variance, and reduction in least-absolute-deviation for interval targets; gini impurity and twoing for nominal targets, and ordered twoing for ordinal targets.

For binary targets, gini, twoing, and ordered twoing create the same splits. Twoing and ordered twoing were used infrequently.

The BFOS method does an exhaustive search for the best binary split.
Linear combination splits are also available. Using a linear combination split, an observation is assigned to the "left" branch when a linear combination of interval inputs is less than some constant. The coefficients and the constant define the split. (BFOS excludes missing values during a split search.) The BFOS method for searching for linear combination splits is heuristic and may not find the best linear combination.

When creating a split, observations with a missing value in the splitting variable (or variables in the case of linear combination) are omitted. Surrogate splits are also created and used to assign observations to branches when the primary splitting variable is missing. If missing values prevent the use of the primary and surrogate splitting rules, then the observation is assigned to the largest branch (based on the within-node training sample).

When a node has many training observations, a sample is taken and used for the split search. The samples in different nodes are independent.

For nominal targets, prior probabilities and misclassification costs are recognized.

The tree is grown to overfit the data. A sequence of sub-trees is formed by using a cost-complexity measure. The sub-tree with the best fit to validation data is selected. Cross-validation is available when validation data is not.

For nominal targets, class probability trees are an alternative to classification trees. Trees are grown to produce distinct distributions of class probabilities in the leaves. Trees are evaluated in terms of an overall gini index. Neither misclassification costs nor rates are used.

**To Approximate Classification and Regression Trees**

Typically, PROC SPLIT trees should be very similar to ones grown using the BFOS methods. PROC SPLIT does not have linear combination splits, twoing or ordered twoing splitting criterion. PROC SPLIT incorporates a loss matrix into a split search differently than the BFOS method. Therefore, splits in the presence of misclassification costs may differ. PROC SPLIT also handles ordinal targets differently than BFOS.

The BFOS method recommends using validation data unless the data set contains too few observations. PROC SPLIT is intended for large data sets, so first divide the data into training and validation data; then, specify the EXCLUDEMISS option.

For nominal targets:

- **CRITERION = GINI**

For interval targets:

- **CRITERION = VARIANCE**

For any type of target:

- **ASSESS=STATISTIC** (or IMPURITY for CLASS PROBABILITY trees.)

- **EXCLUDEMISS**

- **EXHAUSTIVE= 50000** (or so to enumerate all possible splits)
MAXBRANCH = 2

NODESAMPLE = 1000 (or whatever BFOS recommends)

NSURRS = 5 (or so)

SUBTREE = ASSESSMENT or ASSESS=IMPUNITY for CLASS PROBABILITY.

VALIDATA = validation data set

---

**C4.5 and C5.0**

**Description of C4.5**


The target is nominal. The inputs may be nominal or interval.

The recommended splitting criteria is the Gain Ratio = reduction in entropy / entropy of split.

(Let $P(b)$ denote the proportion of training observations a split assigns to branch $b$, $b=1$ to $B$. The entropy of a split is defined as the entropy function applied to $\{P(b): b = 1 \text{ to } B\}$.)

For interval inputs, C4.5 finds the best binary split. For nominal inputs, a branch is created for every value, and then, optionally, the branches are merged until the splitting measure does not improve. Merging is performed stepwise. At each step, the pair of branches is merged that most improves the splitting measure.

When creating a split, observations with a missing value in the splitting variable are discarded when computing the reduction in entropy, and the entropy of a split is computed as if the split makes an additional branch exclusively for the missing values.

When applying a splitting rule to an observation with a missing value on the splitting variable, the observation is replaced by $B$ observations, one for each branch, and each new observation is assigned a weight equal to the proportion of observations used to create the split sent into that branch. The posterior probabilities of the original observation equal the weighted sum of the probabilities for the split observations.

The tree is grown to overfit the training data. In each node, an upper confidence limit of the number misclassified is estimated assuming a binomial distribution around the observed number misclassified. A sub-tree is sought that minimizes the sum of upper confidences in each leaf.

C4.5 can convert a tree into a "ruleset", which is a set of rules that assigns most observations to the same
class that the tree does. Generally, the ruleset contains fewer rules than needed to describe all root-leaf paths and is consequently more understandable than the tree.

C4.5 can create "fuzzy" splits on interval inputs. The tree is constructed the same as with non-fuzzy splits. If an interval input has a value near the splitting value, then the observation is effectively replaced by two observations, each with some weight related to the proximity of the input value to the splitting value. The posterior probabilities of the original observation equal the weighted sum of probabilities for the two new observations.

Description of C5.0

The Web page http://www.rulequest.com contains some information about C5.0. C5.0 is C4.5 with the following differences:

- The branch-merging option for nominal splits is default.
- The user may specify misclassification costs.
- Boosting and cross-validation are available.

Relation to PROC SPLIT

The algorithm for creating rulesets from trees is much improved.

The tree created with C4.5 will differ from those created with PROC SPLIT for several reasons:

C4.5 creates binary splits on interval inputs and multiway splits on nominal inputs. This favors nominal inputs. PROC SPLIT treats interval and nominal inputs the same in this respect.

C4.5 uses a pruning method designed to avoid using validation data. PROC SPLIT expects validation data to be available and so does not offer the pessimistic pruning method of C4.5.

The option settings most similar to C4.5 are:

- CRITERION = ERATIO
- EXHAUSTIVE= 0 (forces a heuristic search)
- MAXBRANCH = maximum number of nominal values in an input, up to 100
- NODESAMPLE= size of data set, up to 32,000
- NSURRS = 0
- WORTH = 0
- SUBTREE = ASSESSMENT
- VALIDATA = validation data set
Examples

The following examples were executed using the HP-UX version 10.20 operating system; the version of the SAS system was 6.12TS045.

Example 1: Creating a Decision Tree with a Categorical Target (Rings Data)

Example 2: Creating a Decision Tree with an Interval Target (Baseball Data)
Example 1: Creating a Decision Tree with a Categorical Target (Rings Data)

This example demonstrates how to create a decision tree with a categorical target. The ENTROPY splitting criterion is used to search for and evaluate candidate splitting rules.

The example DMDB training data set SAMSIO.DMDRING contains a categorical target with 3 levels (C = 1, 2, or 3) and two interval inputs (X and Y). There are 180 observations in the training data set. The SAMSIO.DMSRING data set is scored using the scoring formula from the trained model. Both data sets and the DMDB training catalog are stored in the sample library.

Program

title 'SPLIT Example: RINGS Data';
title2 'Plot of the Rings Training Data';
goptions gunit=pct ftext=swiss ftitle=swissb htitle=4 htext=3;
proc gplot data=sampsio.dmdring;
 plot y*x=c /haxis=axis1 vaxis=axis2;
 symbol c=black i=none v=dot;
 symbol2 c=red i=none v=square;
 symbol3 c=green i=none v=triangle;
 axis1 c=black width=2.5 order=(0 to 30 by 5);
 axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
run;

title2 'Entropy Criterion';
proc split data=sampsio.dmdring
   dmdbcat=sampsio.dmdring
   criterion=entropy
splitsize=2

maxbranch=3

outtree=tree;

input x y;

target c / level=nominal;

score out=out outfit=fit;
run;

proc print data=fit noobs label;
  title3 'Fit Statistics for the Training Data';
run;

proc freq data=out;
  tables f_c*i_c;
  title3 'Misclassification Table';
run;

proc gplot data=out;
  plot y*x=i_c / haxis=axis1 vaxis=axis2;
  symbol  c=black i=none v=dot;
  symbol2 c=red i=none v=square;
  symbol3 c=green i=none v=triangle;
  axis1 c=black width=2.5 order=(0 to 30 by 5);
  axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
  title3 'Classification Results';
run;

proc split intree=tree;

  score data=sampsio.dmsring nodmdb role=score out=gridout;
run;
proc gcontour data=gridout;
  plot y*x=p_c1 / pattern ctext=black coutline=gray;
  plot y*x=p_c2 / pattern ctext=black coutline=gray;
  plot y*x=p_c3 / pattern ctext=black coutline=gray;
  title2 'Posterior Probabilities';
  pattern v=msolid;
  legend frame;
  title3 'Posterior Probabilities';
run;

proc gplot data=gridout;
  plot y*x=_node_;;
  symbol  c=blue i=none v=dot;
  symbol2 c=red i=none v=square;
  symbol3 c=green i=none v=triangle;
  symbol4 c=black i=none v=star;
  symbol5 c=orange i=none v=plus;
  symbol6 c=brown i=none v=circle;
  symbol7 c=cyan i=none v==;
  symbol8 c=black i=none v=hash;
  symbol9 c=gold i=none v=:
  symbol10 c=yellow i=none v=x;
  title3 'Leaf Nodes';
run;

Output

Scatter Plot of the Rings Training Data
Notice that the target levels are not linearly separable.

**PROC PRINT Report of the Training Data Fit Statistics**

<table>
<thead>
<tr>
<th>Train:</th>
<th>Train: Sum of Case Weights</th>
<th>Train: Frequency of Classified Cases</th>
<th>Train: Frequency of Unclassified Cases</th>
<th>Train: Misclassification Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train:</td>
<td>Sum of Case Freq</td>
<td>180</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Train:</td>
<td>Sum Times Freq</td>
<td>540</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Train:</th>
<th>Train: Root of Squared Error</th>
<th>Train:</th>
<th>Train:</th>
<th>Train:</th>
<th>Train:</th>
<th>Train:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train:</td>
<td>Maximum Absolute Error</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Train:</td>
<td>Train: Sum Errors</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train:</td>
<td>Average Error</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train:</td>
<td>Average Squared Error</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train:</td>
<td>Root Divisor for VASE</td>
<td>540</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train:</td>
<td>Degrees of Freedom</td>
<td>360</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### SPLIT Example: RINGS Data

**Entropy Criterion**

**Misclassification Table**

**TABLE OF F_C BY I_C**

<table>
<thead>
<tr>
<th>F_C(Formatted Target Value)</th>
<th>I_C(Predicted Category)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Frequency</td>
<td>8</td>
</tr>
<tr>
<td>Percent</td>
<td>4.44</td>
</tr>
<tr>
<td>Row Pct</td>
<td>100.00</td>
</tr>
<tr>
<td>Col Pct</td>
<td>100.00</td>
</tr>
<tr>
<td>Total</td>
<td>8</td>
</tr>
<tr>
<td>4.44</td>
<td>34.44</td>
</tr>
</tbody>
</table>

**PROC GPLOT of the Classification Results**
PROC GCONTOUR of the Posterior Probabilities

Note that in each of the contour plots, the contour with the largest posterior probabilities captures the actual distribution of the target level.

SPLIT Example: RINGS Data
Entropy Criterion
Classification Results
GPLOT of the Leaf Nodes
PROC GPLOT creates a scatter plot of the Rings training data.

title 'SPLIT Example: RINGS Data';
title2 'Plot of the Rings Training Data';
goptions gunit=pct ftext=swiss ftitle=swissb htitle=4 htext=3;
proc gplot data=sampsio.dmdring;
  plot y*x=c /haxis=axis1 vaxis=axis2;
  symbol   c=black i=none v=dot;
  symbol2 c=red   i=none v=square;
  symbol3 c=green i=none v=triangle;
  axis1 c=black width=2.5 order=(0 to 30 by 5);
  axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
run;
The SPLIT statement invokes the procedure. The DATA= option names the DMDB encoded training data set. The DMDBCAT= option names the DMDB encoded training catalog.

title2 'Entropy Criterion';
proc split data=sampsio.dmdring
    dmdbcat=sampsio.dmdring
The CRITERION = method specifies to use the ENTROPY method of searching and evaluating candidate splitting rules. The ENTROPY method searches for splits based on a reduction in entropy measure of node impurity. The default CRITERION= method for nominal targets is set to PROBCHISQ.

criterion=entropy
The SPLITSIZE= option specifies the smallest number of training observations a node must have for the procedure to consider splitting it.

splitsize=2
The MAXBRANCH= \textbf{n} option restricts the number of subsets a splitting rule can produce to \textbf{n} or fewer.

\begin{verbatim}
maxbranch=3
\end{verbatim}
The OUTTREE= option names the data set that contains tree information.

outtree=tree;
The INPUT statement specifies the input variables. By default, the measurement level of the inputs is set to INTERVAL.

```
  input x y;
```
The TARGET statement specifies the target variable. The LEVEL= option sets the measurement level to nominal.

```plaintext
target c / level=nominal;
```
Because the DATA= option is not specified, the SCORE statement scores the training data set. The OUT= option names the output data set containing outputs. The OUTFIT= option names the output data set containing fit statistics.

    score out=out outfit=fit;
run;
PROC PRINT creates a report of fit statistics for the training data.

proc print data=fit noobs label;
  title3 'Fit Statistics for the Training Data';
run;
PROC FREQ creates a misclassification table for the training data. The F_C variable is the actual target value for each case, and the I_C variable is the target value into which the case is classified.

```plaintext
proc freq data=out;
  tables f_c*i_c;
  title3 'Misclassification Table';
run;
```
PROC GPLOT produces a plot of the classification results for the training data.

```plaintext
proc gplot data=out;
    plot y*x=i_c / haxis=axis1 vaxis=axis2;
    symbol    c=black i=none v=dot;
    symbol2   c=red   i=none v=square;
    symbol3   c=green i=none v=triangle;
    axis1 c=black width=2.5 order=(0 to 30 by 5);
    axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);
    title3 'Classification Results';
run;
```
The INTREE= option specifies to read the OUTTREE= decision tree data set that was created from the previous run of PROC SPLIT.

```
proc split intree=tree;
```
The SCORE statement scores the DATA= data set and outputs the results to the OUT= data set. The ROLE=SCORE option identifies the data set as a score data set. The ROLE= option primarily affects what fit statistics are computed and what their names and labels are.

```latex
score data=sampsio.dmsring nodmdb role=score out=gridout;
run;
```
The GCONTOUR procedure creates contour plots of the posterior probabilities.

```sas
proc gcontour data=gridout;
    plot y*x=p_c1 / pattern ctext=black coutline=gray;
    plot y*x=p_c2 / pattern ctext=black coutline=gray;
    plot y*x=p_c3 / pattern ctext=black coutline=gray;
    title2 'Posterior Probabilities';
    pattern v=msolid;
    legend frame;
    title3 'Posterior Probabilities';
run;
```
The GPLOT procedure creates a scatter plot of the leaf nodes.

```plaintext
proc gplot data=gridout;
    plot y*x=_node_;;
    symbol  c=blue i=none v=dot;
    symbol2 c=red  i=none v=square;
    symbol3 c=green i=none v=triangle;
    symbol4 c=black i=none v=star;
    symbol5 c=orange i=none v=plus;
    symbol6 c=brown i=none v=circle;
    symbol7 c=cyan i=none v==;
    symbol8 c=black i=none v=hash;
    symbol9 c=gold i=none v=:;
    symbol10 c=yellow i=none v=x;
    title3 'Leaf Nodes';
run;
```
The SPLIT Procedure

Example 2: Creating a Decision Tree with an Interval Target (Baseball Data)

Features

- Specifying the Input Variables and the Target Variable
- Setting the Splitting Criterion
- Setting the P-value Adjustment Method
- Outputting Fit Statistics
- Outputting Leaf Node Statistics
- Outputting Sub-Tree Statistics
- Outputting the Decision Tree Information Data Set
- Scoring Data with the Score Statement
- Creating Diagnostic Scatters Plots

This example demonstrates how to create a decision tree for an interval target. The default PROBF splitting criterion is used to search for and evaluate candidate splitting rules.

The example DMDB training data set SAMSIO.DMBASE contains performance measures and salary levels for regular hitters and leading substitute hitters in major league baseball for the year 1986 (Collier 1987). There is one observation per hitter. The continuous target variable is log of salary (logsalar).

The SAMSIO.DMTBASE data set is a test data set, which is scored using the scoring formula from the trained model. The SAMSIO.DMBASE and SAMSIO.DMTBASE data sets and the SAMSIO.DMDBASE training catalog are stored in the sample library.

Program

```sas
proc split data=sampsio.dmdbase
dmdbcat=sampsio.dmdbase

criterion=probf
padjust=depth

outmatrix=trtree
outtree=treedata
outleaf=leafdata
outseq=subtree;
```
input league division position / level=nominal;
input no_atbat no_hits no_home no_runs no_rbi no_bb
   yr_major cr_atbat cr_hits cr_home cr_runs cr_rbi cr_bb
   no Outs no_assts no_error / level=interval;

target logsalar;

score data=sampsio.dmtbase nodmdb

   outfit=splfit
      out=splout(rename=(p_logsal=predict r_logsal=residual));
title 'Decision Tree: Baseball Data';
run;

proc print data=trtree noobs label;
   title2 'Summary Tree Statistics for the Training Data';
run;

proc print data=leafdata noobs label;
   title2 'Leaf Node Summary Statistics';
run;

proc print data=subtree noobs label;
   title2 'Subtree Summary Statistics';
run;

proc print data=splfit noobs label;
   title2 'Summary Statistics for the Scored Test Data';
run;

proc gplot data=splout;
   plot logsalar*predict / haxis=axis1 vaxis=axis2 frame;
      symbol c=black i=none v=dot h= 3 pct;
      axis1 minor=none color=black width=2.5;
      axis2 minor=none color=black width=2.5;
      title2 'Log of Salary versus the Predicted Log of Salary';

   plot residual*predict / haxis=axis1 vaxis=axis2;
      title2 'Plot of the Residuals versus the Predicted Log of Salary';
run;
quit;

Output
Summary Tree Statistics for the Training Data Set

The OUTMATRIX= data set contains the following summary statistics:

- **N** - the number of observations in the training data set
- **AVERAGE** - the target average
- **AVERAGE SQ ERR** - the average squared prediction error (the sum of squared errors / n)
- **R SQUARED** - the R-square statistic (1 - AVERAGE SQ ERR / sum of squares from the average)

### Decision Tree: Baseball Data
Summary Tree Statistics for the Training Data

<table>
<thead>
<tr>
<th>STATISTIC</th>
<th>==&gt; AVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>163.000</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>5.956</td>
</tr>
<tr>
<td>AVE SQ ERR</td>
<td>0.062</td>
</tr>
<tr>
<td>R SQUARED</td>
<td>0.920</td>
</tr>
</tbody>
</table>

Leaf Node Summary Statistics for the Training Data Set

The OUTLEAF= data set contains the following statistics:

- Leaf ID number
- **N** or number of observations in each leaf node
- The target **AVERAGE** for each leaf node
- The root average squared error (ROOT ASE) for each leaf node

### Decision Tree: Baseball Data
Leaf Node Summary Statistics

<table>
<thead>
<tr>
<th>LEAF ID</th>
<th>N</th>
<th>AVERAGE</th>
<th>ROOT ASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9</td>
<td>4.2885299792</td>
<td>0.0810310161</td>
</tr>
<tr>
<td>16</td>
<td>9</td>
<td>4.581814362</td>
<td>0.086134155</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>5.1647859739</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>14</td>
<td>5.0581554082</td>
<td>0.1033292134</td>
</tr>
<tr>
<td>37</td>
<td>1</td>
<td>4.6539603502</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>4.3820266347</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>5.5274252033</td>
<td>0.0893458944</td>
</tr>
<tr>
<td>20</td>
<td>7</td>
<td>5.5534989096</td>
<td>0.120153823</td>
</tr>
<tr>
<td>21</td>
<td>5</td>
<td>5.200846713</td>
<td>0.1327198925</td>
</tr>
<tr>
<td>38</td>
<td>4</td>
<td>6.0965134867</td>
<td>0.0684629543</td>
</tr>
<tr>
<td>39</td>
<td>7</td>
<td>5.7171876218</td>
<td>0.1860772093</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>5.9148579892</td>
<td>0.1708160326</td>
</tr>
<tr>
<td>41</td>
<td>8</td>
<td>6.3897459513</td>
<td>0.150561041</td>
</tr>
<tr>
<td>23</td>
<td>13</td>
<td>6.4853119091</td>
<td>0.3839130342</td>
</tr>
<tr>
<td>13</td>
<td>16</td>
<td>5.7881752468</td>
<td>0.2598459864</td>
</tr>
<tr>
<td>32</td>
<td>6</td>
<td>5.902511632</td>
<td>0.2696864743</td>
</tr>
<tr>
<td>33</td>
<td>9</td>
<td>6.4895454866</td>
<td>0.1092807452</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>4.4998096703</td>
<td>0</td>
</tr>
<tr>
<td>34</td>
<td>27</td>
<td>6.6125466821</td>
<td>0.3489469023</td>
</tr>
<tr>
<td>35</td>
<td>2</td>
<td>7.6883712553</td>
<td>0.1000475779</td>
</tr>
<tr>
<td>Subtree Summary Statistics for the Training Data Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Decision Tree: Baseball Data</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Subtree Summary Statistics</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Number of Leaves in Sub-tree</strong></td>
<td><strong>N Training Cases Used</strong></td>
<td><strong>Sub-tree Assessment Stats</strong></td>
<td><strong>Sub-tree Average Score.</strong></td>
</tr>
<tr>
<td>1</td>
<td>163</td>
<td>5.95594</td>
<td>0.77002</td>
</tr>
<tr>
<td>2</td>
<td>163</td>
<td>5.95594</td>
<td>0.30330</td>
</tr>
<tr>
<td>3</td>
<td>163</td>
<td>5.95594</td>
<td>0.24138</td>
</tr>
<tr>
<td>4</td>
<td>163</td>
<td>5.95594</td>
<td>0.20403</td>
</tr>
<tr>
<td>5</td>
<td>163</td>
<td>5.95594</td>
<td>0.16712</td>
</tr>
<tr>
<td>6</td>
<td>163</td>
<td>5.95594</td>
<td>0.14940</td>
</tr>
<tr>
<td>7</td>
<td>163</td>
<td>5.95594</td>
<td>0.13476</td>
</tr>
<tr>
<td>8</td>
<td>163</td>
<td>5.95594</td>
<td>0.12154</td>
</tr>
<tr>
<td>9</td>
<td>163</td>
<td>5.95594</td>
<td>0.10956</td>
</tr>
<tr>
<td>10</td>
<td>163</td>
<td>5.95594</td>
<td>0.09996</td>
</tr>
<tr>
<td>11</td>
<td>163</td>
<td>5.95594</td>
<td>0.09235</td>
</tr>
<tr>
<td>12</td>
<td>163</td>
<td>5.95594</td>
<td>0.08675</td>
</tr>
<tr>
<td>13</td>
<td>163</td>
<td>5.95594</td>
<td>0.08176</td>
</tr>
<tr>
<td>14</td>
<td>163</td>
<td>5.95594</td>
<td>0.07807</td>
</tr>
<tr>
<td>15</td>
<td>163</td>
<td>5.95594</td>
<td>0.07448</td>
</tr>
<tr>
<td>16</td>
<td>163</td>
<td>5.95594</td>
<td>0.07134</td>
</tr>
<tr>
<td>17</td>
<td>163</td>
<td>5.95594</td>
<td>0.06891</td>
</tr>
<tr>
<td>18</td>
<td>163</td>
<td>5.95594</td>
<td>0.06667</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Train: Maximum Absolute Error of Squared Errors</th>
<th>Train: Sum of Squared Error Average</th>
<th>Train: Average Squared Error</th>
<th>Train: Root Average Squared Error</th>
<th>Train: Divisor for VASE</th>
<th>Train: Degrees of Freedom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.85198</td>
<td>125.513</td>
<td>0.77002</td>
<td>0.87751</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>1.90402</td>
<td>49.438</td>
<td>0.30330</td>
<td>0.55073</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>2.17653</td>
<td>39.344</td>
<td>0.24138</td>
<td>0.49130</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>1.64524</td>
<td>33.257</td>
<td>0.20403</td>
<td>0.45170</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>1.64524</td>
<td>27.240</td>
<td>0.16712</td>
<td>0.40880</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>1.23531</td>
<td>24.353</td>
<td>0.14940</td>
<td>0.38653</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>1.10168</td>
<td>21.966</td>
<td>0.13476</td>
<td>0.36710</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>1.09120</td>
<td>19.811</td>
<td>0.12154</td>
<td>0.34863</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>0.98673</td>
<td>17.858</td>
<td>0.10956</td>
<td>0.33099</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>0.98673</td>
<td>16.294</td>
<td>0.09996</td>
<td>0.31617</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>0.98673</td>
<td>15.054</td>
<td>0.09235</td>
<td>0.30390</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>0.98673</td>
<td>14.140</td>
<td>0.08675</td>
<td>0.29453</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>0.98673</td>
<td>13.327</td>
<td>0.08176</td>
<td>0.28594</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>0.98673</td>
<td>12.726</td>
<td>0.07807</td>
<td>0.27941</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>0.98673</td>
<td>12.140</td>
<td>0.07448</td>
<td>0.27291</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>0.98673</td>
<td>11.628</td>
<td>0.07134</td>
<td>0.26709</td>
<td>163</td>
<td>163</td>
</tr>
<tr>
<td>0.98673</td>
<td>11.233</td>
<td>0.06891</td>
<td>0.26251</td>
<td>163</td>
<td>163</td>
</tr>
</tbody>
</table>
### Subtree Summary Statistics

**Decision Tree: Baseball Data**

<table>
<thead>
<tr>
<th>Leaves in Sub-tree</th>
<th>Number of Leaves</th>
<th>Cases Used for Sub-tree</th>
<th>Sub-tree Stats</th>
<th>Assessment Score.</th>
<th>Average Squared Error.</th>
<th>Train: Sum of Case Frequencies</th>
<th>Times Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>163</td>
<td>5.95594</td>
<td>0.06444</td>
<td>163</td>
<td>163</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>163</td>
<td>5.95594</td>
<td>0.06256</td>
<td>163</td>
<td>163</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>163</td>
<td>5.95594</td>
<td>0.06163</td>
<td>163</td>
<td>163</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Summary Statistics for the Scored Test Data Set**

**Decision Tree: Baseball Data**

<table>
<thead>
<tr>
<th>Test: Sum of Weights</th>
<th>Test: Sum of Times Freqs</th>
<th>Test: Maximum Absolute Error</th>
<th>Test: Sum of Squared Errors</th>
<th>Test: Average Squared Error</th>
<th>Test: Root Divisor for VASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
<td>2.00738</td>
<td>25.2495</td>
<td>0.25250</td>
<td>0.50249</td>
</tr>
</tbody>
</table>

**GPLOT Diagnostic Plots for the Scored Test Data Set**
Decision Tree: Baseball Data
Log of Salary by the Predicted Log of Salary

Predicted: LOGSALAR
The SPLIT statement invokes the procedure. The DATA=option identifies the training data set that is used to fit the model. The DMDBCAT= option identifies the training data catalog.

```plaintext
proc split data=sampsio.dmdbase
  dmdbcat=sampsio.dmdbase
```
The CRITERION = method specifies the PROBF method of searching and evaluating candidate splitting rules. For interval targets, the default method is PROBF (p-value of F-test associated with node variance).

criterion=probf
The PADJUST=option specifies the DEPTH method for adjusting p-values. DEPTH adjusts for the number of ancestor splits.

padjust=depth
The OUTMATRIX= option names the output data set that contains tree summary statistics for the training data.

```outmatrix=trtree```
The OUTTREE= option names the data set that contains tree information. You can use the INTREE= option to read the OUTTREE= data set in a subsequent execution of PROC SPLIT.

outtree=treedata
The OUTLEAF= option names the data set that contains statistics for each leaf node.

outleaf=leafdata
The OUTSEQ= option names the data set that contains sub-tree statistics.

outseq=subtree;
Each INPUT statement specifies a set of input variables that have the same measurement level. The LEVEL= option identifies the measurement level of each input set.

```
input league division position / level=nominal;
input no_atbat no_hits no_home no_runs no_rbi no_bb
   yr_major cr_atbat cr_hits cr_home cr_runs cr_rbi cr_bb
   no_outs no_assts no_error / level=interval;
```
The TARGET statement specifies the target (response) variable.

target logsalar;
The SCORE statement specifies the data set that you want to score in conjunction with training. The DATA= option identifies the score data set.

```plaintext
score data=sampsio.dmtbase nodmdb
```
The OUTFIT= option names the output data set that contains goodness-of-fit statistics for the scored data set. The OUT= data set contains summary statistics for the scored data set, such as predicted and residual values.

```plaintext
outfit=splfit
out=splout(rename=(p_logsal=predict r_logsal=residual));
title 'Decision Tree: Baseball Data';
run;
```
PROC PRINT lists summary tree statistics for the training data set.

proc print data=trtree noobs label;
   title2 'Summary Tree Statistics for the Training Data';
run;
PROC PRINT lists summary statistics for each leaf node.

proc print data=leafdata noobs label;
   title2 'Leaf Node Summary Statistics';
run;
PROC PRINT lists summary statistics for each subtree in the sub-tree sequence.

proc print data=subtree noobs label;
  title2 'Subtree Summary Statistics';
run;
PROC PRINT lists fit statistics for the scored test data set.

proc print data=splfit noobs label;
    title2 'Summary Statistics for the Scored Test Data';
run;
PROC GPLOT produces diagnostic plots for the scored test data set. The first PLOT statement creates a scatter plot of the target values versus the predicted values of the target. The second PLOT statement creates a scatter plot of the residual values versus the predicted values of the target.

```
proc gplot data=splout;
  plot logsalar*predict / haxis=axis1 vaxis=axis2 frame;
  symbol c=black i=none v=dot h= 3 pct;
  axis1 minor=none color=black width=2.5;
  axis2 minor=none color=black width=2.5;
  title2 'Log of Salary versus the Predicted Log of Salary';
```
References


Hand, D. J. (1987), Construction and Assessment of Classification Rules, New York: John Wiley and Sons, Inc.


Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The SPSVD Procedure

Overview

Procedure Syntax

PROC SPSVD Statement
ROW Statement
COL Statement
ENTRY Statement
OUTPUT Statement

EXAMPLES-SECTION

Example 1: Use the SPSVD procedure for training and validation

References

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The SPSVD Procedure

Overview

The SPSVD procedure has two main functions. The first is to calculate a truncated singular value decomposition of a large sparse matrix. The second is to project the rows or columns of a sparse matrix onto the columns of a dense matrix.

Proc SPSVD takes at least one data set as input. This data set must be a compressed representation of a sparse matrix. The data set must contain at least 3 variables, with one observation for each non-zero entry in the matrix. One variable indicates the row number, another variable indicates the column number and a third indicates the entry. Thus the following matrix

\[
\begin{bmatrix}
1 & 0 & 0 & 2 \\
0 & 3 & 0 & 0 \\
2 & 0 & 1 & 1 \\
\end{bmatrix}
\]

would be represented in compressed form as:

<table>
<thead>
<tr>
<th>Row</th>
<th>Column</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

The last observation in this data set indicates that the number 1 appears in the 3rd row and 4th column of the original matrix. The Text Parsing node in Enterprise Miner produces a data set that is in the correct format. For text mining we generally consider the terms as rows and the documents as columns. The number of times that the term appears in the document is the entry. Thus if term 5 appears in document 10 three times we would say there is a 3 in row 5, column 10. These correspond to the numeric key for the term, the document number, and the number of times that the term appears in the current document. Note that other matrices may be passed to the procedure (via the IN_U option for example) and these matrices are not passed in the sparse matrix format.

PROC SPSVD computes a truncated singular value decomposition of the sparse matrix. Thus, if we call
the sparse matrix $A$, PROC SPSVD will compute the first $k$ ($k$ is chosen by the user) columns of 3 new matrices, $U$, $\Sigma$, and $V$ such that $A = U \Sigma V^T$. Since only the first $k$ columns of these matrices are calculated, this is referred as a truncated SVD. The procedure will calculate the leading columns of these matrices must faster than it can calculate the later columns. Thus smaller values of $k$ result in faster execution times. The algorithm used in this routine is designed to find these leading columns only. The value of $k$ should be less than the minimum of the number of rows and number of columns in the original noncompressed matrix. If a user attempts to calculate too many columns the calculations may not be accurate and calculation times may be slow (in the former case a warning will be written to the log). The matrices $U$ and $V$ are orthonormal (i.e. each column is orthogonal to every other column in the matrix, and the Euclidean norm of each column is 1). $\Sigma$ is diagonal with monotonically decreasing non-negative real entries. These matrices have many interesting and important qualities. For text mining we are interested in using them for dimension reduction.

To see how we accomplish this, we view the columns of the original matrix $A$ as coordinates. Each column then specifies the position of a data point (document) in a high dimensional space. Similarly, the first column of $U$ defines a point in space. If we join this point and the origin, this defines a line. That line is the best least squares fit to the data (distance measured perpendicular to the line). In other words, the first column of $U$, $u_1$, minimize

$$\sum_{i=1}^{n} \left\| a_i - c_i u_1 \right\|$$

where $a_i$ is the $i$th column of $A$, $c_i$ is some constant, $n$ is the number of columns in $A$, and $\| \cdot \|$ is the Euclidean norm. The second column of $U$, $u_2$, also defines a line, and together with $u_1$, it defines a plane. This plane is the best fit 2-dimensional subspace to the data (again in a least squares sense). Generally, the first $k$ columns of $U$ form a best-fit least-square $k$-dimensional subspace. We can therefore project the columns of $A$ onto the first $k$ columns of $U$ to reduce the dimension. The result of projecting a document, $d$, is $k$ real numbers, $p_i$, each of which is the inner-product of $d$ onto a column of $U$. This projection is defined by $p_i = d^T u_i$.

For text mining, PROC SPSVD is generally used to calculate the SVD for the training data set. The training data set is then projected onto the first $k$ columns of $U$ as discussed above. $k$ is chosen by the user. It is difficult to define an optimal value for $k$. Generally, $k$ must be large enough to capture much of the meaning in the document collection, but it should not be so large as to capture the noise. Values between 100 and 200 work well for large document collection that contains thousand of documents. Once the training data set has been projected, the matrices used for the projection are saved, and the PROC is invoked again to project the validation and any test or score data sets onto these same matrices. Many options exist for weighting the input matrix and scaling or normalizing the projected image. These are discussed in the syntax sections.
It should be noted that the first \( k \) columns of \( \mathbf{V} \) form a best fit subspace with respect to the rows of \( \mathbf{A} \). PROC SPSVD allows the user to project the rows onto the first \( k \) columns of \( \mathbf{V} \) for this reason. In the framework of text mining, this is viewed as a representation for the terms in the data set. Possible uses for this include clustering the reduced dimension representation of the terms to find concepts prevalent in the document collection.
Procedure Syntax

PROC SPSVD <option(s)>;

ROW variable;

COL variable;

ENTRY variable;

OUTPUT <option(s)>;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The SPSVD Procedure

PROC SPSVD Statement

Invoke the SPSVD procedure.

PROC SPSVD <option(s)>;

Options

DATA = <libref.>SAS-data-set

Specifies the data set to be analyzed. This data set should be in compressed form as described in the overview. If you omit the DATA= option, the procedure uses the most recently created SAS data set.

IN_GLOBAL = <libref.>SAS-data-set

Specifies the data containing global weights that are calculated previously in order to apply the global weights to the input data set. This option is generally used in conjunction with the GWGT options in the output statement. In a predictive modeling setting, we want to calculate global weights based only on the training data set. These weights are then written to a data set using the GWGT option. We can then apply these same weights to another data set. For example, the weights can be applied to the validation data set by using the IN_GLOBAL option. Note that the IN_GLOBAL option and the GLOBAL option can not both be specified.

IN_U = <libref.>SAS-data-set

Specifies a $U$ matrix to be used for a column projection. If this option is used, the SVD of the input matrix is not calculated. Rather, the specified $U$ matrix is used for projections. For example, if COLPRO is specified, but ROWPRO is not, then only the matrix $U$ is needed as long as SCALECOL or SCALEALL has not been specified. If SCALECOL had been specified, then IN_S would be needed.

IN_S = <libref.>SAS-data-set

Specified a $\Sigma$ matrix to be used for a projection. If this option is used, the SVD of the input matrix is not calculated. Rather, the $U$, $V$, and $\Sigma$ matrices specified are used for projections. Only the matrices needed for the requested projections need to be supplied. Thus, for example, if COLPRO is specified, but ROWPRO is not, then only the matrix $\Sigma$ is needed as long as SCALECOL or SCALEALL has not been specified. If SCALECOL had been specified, then IN_S would be needed.

IN_V = <libref.>SAS-data-set

Specifies a $V$ matrix to be used for a row projection. If this option is used, the SVD of the input
matrix is not calculated. Rather, the $V$ matrix is used for projections. Only the matrices needed for the requested projections need to be supplied. Thus, for example, if ROWPRO is specified, but COLPRO is not, then only the matrix $V$ is needed as long as SCALECOL or SCALEALL has not been specified. If SCALECOL had been specified, then IN_S would be needed.

$k = \text{integer}$

Represents the number of dimensions that the data set will be reduced to, and controls the number of columns of $U$, $\Sigma$, and $V$ to be calculate and used for projections. The procedure will only calculate the number as specified. Therefore, specifying a $k$ larger than is needed will cause the procedure to run for an unnecessary long time. See the Overview section for tips on choosing $k$. If $U$ and $V$ are passed to the procedure via IN_U or IN_V, then the user does not need to specify $k$ as this will be deduced from the number of columns in the passed data sets. If the user would like the SVD calculated, then $k$ must be specified.

$p = \text{integer}$

Specifies the number of iterations, beyond $k$, before the procedure restarts. PROC SPSVD is an iterative procedure. As iterations continue, more and more memory is used and the procedure slows down due to number of calculations required. If the desired quantities have not been calculated to acceptable accuracy within $k+p$ iterations, the procedure will restart, maintaining much of the information learned in the first $k+p$ iterations. Setting the value of $p$ low will cause frequent restarts which will use less memory. However, the restarting takes time so this may slow the procedure. Conversely, if $p$ is too large, the routine will begin to slow due to the calculations required. If this value is not specified, it defaults to $\min\{k,75\}$.

$\text{LOCAL} = \text{BINARY|LOG}$

Specifies a local weight, $L_{ij}$, to be used to weight the entries of the input matrix prior to any calculations. If the WGT = option is specified, the weighted matrix will be written out. Local and global weights are combined so that an entry, $\hat{a}_{ij}$, of the new matrix is calculated from an entry, $a_{ij}$, of the old matrix as $\hat{a}_{ij} = L_{ij}G_{ij}$. If the local weight is not specified, it defaults to $L_{ij} = a_{ij}$. If a global weight is not specified, it defaults to $G_{ij} = 1$.

The following table lists the available local weights:

<table>
<thead>
<tr>
<th>Binary</th>
<th>If the term appears in the document, then $L_{ij} = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Otherwise, $L_{ij} = 0$</td>
</tr>
<tr>
<td>Log</td>
<td>$L_{ij} = \log_2 (a_{ij} + 1)$</td>
</tr>
</tbody>
</table>
GLOBAL = NORMAL|GFIDF|IDF|ENTROPY

Specifies a global weight, $G_i$, to be used to weight the entries of the input matrix prior to any calculations. If the WGT = option is specified, the weighted matrix will be written out. Local and global weights are combined so that an entry, $\hat{a}_{ij}$, of the new matrix is calculated from an entry, $a_{ij}$, of the old matrix as $\hat{a}_{ij} = L_{ij}G_{ij}$. If the local weight is not specified, it defaults to $L_{ij} = a_{ij}$. If a global weight is not specified, it defaults to $G_{ij} = 1$. The GLOBAL option may not be used in conjunction with the IN_GLOBAL option. The GWGT option on the OUTPUT statement enables you to save the calculated global weights so they can be applied to subsequent data sets by using the IN_GLOBAL option.

Global weights are functions of the row entries of the original, noncompressed, sparse matrix. The following table lists the available global weights:

<table>
<thead>
<tr>
<th>Table of Row Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
</tr>
<tr>
<td>Global Frequency divided by Inverse Document Frequency (GFIDF)</td>
</tr>
<tr>
<td>Inverse Document Frequency (IDF)</td>
</tr>
<tr>
<td>Entropy</td>
</tr>
</tbody>
</table>

where $f_{ij}$ is the frequency of term $i$ in document $j$, $d_i$ is the number of documents in which term $i$ appears, $g_i$ is the number of times that term $i$ appears in the whole document collection, $n$ is the number of document in the collection, and $p_{ij} = \frac{f_{ij}}{g_i}$.

TOL = number

Specifies a tolerance for the procedure to stop finding eigenvalues of $A^T A$. The procedure is
actually finding eigenvalues of $A^T A$. Suppose $\theta$ is the eigenvalue estimate and $y$ is the eigenvector estimate, then the procedure terminates when all $k$ sets of values satisfy

$$\left\| A^T A y - y \theta \right\|^2 \leq TOL$$

If TOL is not specified, it defaults to $10^{-6}$, which is more than adequate for most text mining problems.
ROW Statement

Specifies the row variable. This statement is not required if the row variable has a name of ROW.

```
ROW variable;
```

`variable`

Specifies the name of the variable in the input data set that contains the row variable for the compressed matrix format as described in the overview.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
COL Statement

Specifies the row variable. This statement is not required if the column variable has a name of COL.

COL variable;

variable

   Specifies the name of the variable in the input data set that contains the column variable for the compressed matrix format as described in the overview.
ENTRY Statement

Specifies the variable name of the entry values. This statement is not required if the variable has a name of ENTRY.

```
ENTRY variable;
```

variable

Specifies the name of the variable in the input data set that contains the entry values for the compressed matrix format as described in the overview.
OUTPUT Statement

Specifies the data sets to be output.

```
OUTPUT <option(s)>;
```

Options

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

Specifies the name of the data set to store the calculated $\Sigma$ matrix. The matrix is written with rows of the matrix as observations in the SAS data set and columns as variables. The variables are named COL1-COLk. You can not specify S = if the IN_S option has been specified.

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

Specifies the name of the data set to store the calculated $U$ matrix. The matrix is written with rows of the matrix as observations in the SAS data set and columns as variables. The variables are named COL1-COLk. You can not specify U = if the IN_U option has been specified.

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

Specifies the name of the data set to store the calculated $V$ matrix. The matrix is written with rows of the matrix as observations in the SAS data set and columns as variables. The variables are named COL1-COLk. You can not specify V = if the IN_V option has been specified.

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

Specifies the name of the data set that contains the calculated global weights. This data set can be applied to other data sets by using the IN_GLOBAL option. This option must be used in conjunction with the GLOBAL option.

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

Specifies the name of the data set to which the procedure writes the weighted matrix, if the LOCAL, GLOBAL, or both statements are used. If LOCAL and /or GLOBAL is specified, but WGT= is not, then all calculations performed by the procedure are still based on the weighted matrix; but the weighted matrix will not be saved. If WGT= is specified and COLPRO, ROWPRO, U=, S=, and V= are not specified, then the matrix will be weighted and written to disk; no other calculations will be performed.

**COLPRO|DOCPRO = <libref.>SAS-data-set**

Specifies the data set that the projection of the columns of the input matrix onto the columns of the matrix $U$ will be written to. If the IN_U option is specified, the data in the set specified by the IN_U option will be used for the projection. Otherwise U will be calculated from the input data set. If SCALECOL|SCALEDOC or SCALEALL is specified and IN_U is specified, then IN_S must also be specified.
ROWPRO|WORDPRO = <libref.>SAS-data-set

Specifies the data set that the projection of the rows of the input matrix onto the rows of the matrix $V$ will be written to. If the IN_V option is specified, the data in the set specified by the IN_V option will be used for the projection. Otherwise V will be calculated from the input data set. If SCALEROW|SCALEWORD or SCALEALL is specified and IN_V is specified, then IN_S must also be specified.

SCALECOL|SCALEDOC, SCALEROW|SCALEWORD, SCALEALL

Requests that the associated projections (column, row, or all) be scaled by the inverse of the singular values. SCALEALL specifies that both the document (column) and the word (row) projections should be scaled. SCALECOL or SCALEDOC specifies that the document (columns of the input matrix) projections be scaled. SCALEROW or SCALEWORD specifies that the term (rows of the input matrix) projections to be scaled. If $p_{ij}$ is the $i^{th}$ coordinate of the projected image of the $j^{th}$ document, then scaling replaces the formula $p_{ij} = \alpha_j^T u_i$ with $p_{ij} = \alpha_j^T u_i \sigma_i^{-1}$ where $\sigma_i$ is the $i^{th}$ singular value (the $i^{th}$ entry on the diagonal of $\Sigma$).

Scaling has two functions. First, it puts more weight on those themes in a document that are uncommon in the document collection. Second, if either the terms or documents, but not both, are scaled, and both are placed in the same space then the terms and documents that are highly associated are more likely to be near each other.

NORMCOL|NORMDOC, NORROW|NORMWORD, NORMALL

Requests to normalize the Euclidean length of the document (column), word (row) or both projections. For example, if NORMCOL, NORMDOC, or NORMALL is specified, then each observation in the data set specified by the DOCPRO option will have a length of 1. This is useful because it bring documents with similar content but different lengths closer together. For most text mining applications, NORMALL is suggested.
Example

Example 1: Use the SPSVD procedure for training and validation

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Suppose there are two data sets, SASUSER.TRAIN, and SASUSER.VALID produced by the Text Parsing node in Enterprise Miner. You want to use SASUSER.TRAIN for training a predictive model and the SASUSER.VALID for validation.

PROC SPSVD DATA=SASUSER.TRAIN K=200 P=50 LOCAL=LOG GLOBAL=ENTROPY;
   ROW KEY;
   COL DOC;
   ENTRY COUNT;
   OUTPUT U=SASUSER.U V=SASUSER.V S=SASUSER.S NORMALL SCALEALL
   DOCPRO=SASUSER.TRAINDP GWGT=SASUSER.WEIGHTS;
RUN;

PROC SPSVD DATA=SASUSER.VALID IN_U=SASUSER.U IN_S=SASUSER.S
   LOCAL=LOG IN_GLOBAL=SASUSER.WEIGHTS;
   ROW KEY;
   COL DOC;
   ENTRY COUNT;
   OUTPUT NORMALL SCALEALL DOCPRO=SASUSER.VALIDDP;
RUN;

The first PROC statement applies a local log, global entropy weighting scheme to the training data set. The ROW, COL, and ENTRY options specify the names given to these variables by the Text Parsing node in Enterprise Miner. Once the procedure has weighted the matrix, it calculates 200 (specified in the K= option) columns of $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ based on this weighted matrix. The weighted training data set is then projected onto the first 200 columns of $\mathbf{U}$, scaled by the inverse singular values and its length is normalized. The result of the projection is written to SASUSER.TRAINDP. The calculated global weights (entropy in this case) are saved to the data set SASUSER.WEIGHTS.

The second PROC statement is to project the validation data set using the calculations from the training data set. This is done by specifying the $\mathbf{U}$ and $\mathbf{\Sigma}$ matrices calculated in the first PROC step with the IN_U and IN_S options. Notice that you do not need to specify the V data set since you are not projecting the terms. To project the document in the validation data set, specify the same local weighting option for the validation data set and pass the calculated global weights via the IN_GLOBAL option. Then, request that the normalized, scaled projection be written to SASUSER.VALIDDP. This way the validation data set is weighted in exactly the same way as the training data set. Using the GLOBAL option on the validation data set would cause new global weights to be calculated based on the data in this set, which is not appropriate in this example because you want each dimension in the validation data
set to correspond to a dimension in the training data set.
The STDIZE Procedure

Overview

Procedure Syntax

PROC STDIZE Statement
BY Statement
FREQ Statement
LOCATION Statement
SCALE Statement
VAR Statement
WEIGHT Statement

Details

Examples

Example 1: Getting Started with the STDIZE Procedure
Example 2: Unstandardizing a Data Set
Example 3: Replacing Missing Values with Standardizing
Example 4: Replacing Missing Values without Standardizing the Variables

References

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Overview

The STDIZE procedure standardizes one or more numeric variables in a SAS data set by subtracting a location measure and dividing by a scale measure. A variety of location and scale measures are provided, including estimates that are resistant to outliers and clustering (see the METHOD= option). You can also multiply each standardized value by a constant and add a constant. Thus the result is:

\[
\text{result} = \text{adder} + \frac{\text{multiplier} \times (\text{original} - \text{location})}{\text{scale}}
\]

where:
- result is the final output value
- adder is the constant to add (the value specified in the ADD= option)
- multiplier is the constant to multiply by (the value specified in the MULT= option)
- original is the original input value
- location is the location measure
- scale is the scale measure

PROC STDIZE also finds quantiles in one pass of the data. It is especially useful when the data set is very large and PROC UNIVARIATE may either run out of memory or take a long time to compute the quantiles.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
PROC STDIZE <option(s)>;

BY variable-1 < ... variable-n> <NOTSORTED>;

FREQ variable;

LOCATION variable(s);

SCALE variable(s);

VAR variable(s);

WEIGHT variable;

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The STDIZE Procedure

PROC STDIZE Statement

Invokes the STDIZE procedure.

PROC STDIZE <option(s)>;

Options

ADD=number
Specifies the constant to add to each value after standardizing and multiplying by the MULT= number.

Default: 0

DATA=<libref.> SAS-data-set
Specifies the input data source to be standardized.

Default: _LAST_

FUZZ=c
Specifies the relative fuzz factor for writing the output.

Default: 1E-14.

- For OUT= data set: if |result| < scale \times c,
  then result = 0.
- For OUTSTAT= data set: if scale < |location| \times c,
  then SCALE = 0;
  otherwise, if |location| < scale \times c,
  then LOCATION = 0.

INITIAL=method-name
Specifies the method for computing initial estimates for the A estimates: ABW, AWA\^{}VE, and AHUBER. See the Table of Methods for Computing Location and Scale Measures for the list of methods.

CAUTION:

ABW, AWA\^{}VE, AHUBER, and IN are not valid as INITIAL methods.
METHOD=method-name

Specifies the name of the standardization method. See Standardization Methods section for more information on the method-names that are available for computing LOCATION and SCALE measures.

MISSING=method | numeric-value/<missing-option(s)>

Specifies the method or a numeric value for replacing missing values.

- Use the MISSING= option when you want to replace missing values by something other than the location measure associated with the METHOD= option, which is what the REPLACE option uses as the replacement value. The usual methods include MEAN, MEDIAN, and MIDRANGE. Any of the values for the METHOD= option can also be specified for the MISSING= option, and the corresponding location measure will be used to replace missing values. If a numeric value is given, it replaces missing values after standardizing the data. However, the REPONLY option can be used together with the MISSING= option to suppress standardization in case you only want to replace missing values.
- See the Table of Methods for Computing Location and Scale Measures for a list of the values that can be specified for the MISSING= option (with the exception of MISSING=IN).

MULT=c

Specifies the constant to multiply each value by, after standardizing.

Default: 1

NMARKERS=n

Specifies the number of markers for the P2 algorithm (PCTLMTD=P2).

Range: Integer where \( n \geq 5 \).

Default: 101

NOMISS

Omits observations that have missing values in the analyzed variables from computation of the location and scale measures. Otherwise, all nonmissing values are used.

NORM

For METHOD= AGK, IQR, MAD, or SPACING, normalizes the scale estimator to be consistent for the standard deviation of a normal distribution.

OUT=<libref.> SAS-data-set

Specifies the output data set created by PROC STDIZE. The output data set is a copy of the
DATA= data set except that the analyzed variables (those in the VAR statement, or in the absence of a VAR statement, all numeric variables not listed in any other statement) have been standardized.

**Default:** _DATA_. If the OUT= option is omitted, PROC STDIZE creates an output data set and names it according to the DATA_n convention, just as if you had omitted a data set name in a DATA statement.

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

Specifies the output statistics data set that contains the location and scale measures and some other simple statistics. A _TYPE_ variable is also created to help identify the type of statistics for each observation. The value of the _TYPE_ variable can be:

- LOCATION
  - Contains the location measure of each variable.
- SCALE
  - Contains the scale measure of each variable.
- NORM
  - Contains the norm measure of each variable.
- ADD
  - Contains the constant from the ADD= option.
- MULT
  - Contains the constant from the MULT= option.
- N
  - Contains the total number of non-missing positive frequencies of each variable.
- Pn
  - Contains the percentiles of each variable specified through the PCTLPTS= option.

**Range:** $0 \leq n \leq 100$

**PCTLDEF=** value

Specifies one of the five available definitions described in the *Computational Methods* section in the UNIVARIATE procedure that calculates percentiles when PCTLMTD=ORD_STAT is specified.

**Default:** 5.

**Range:** 1, 2, 3, 4, 5

**Tip:** When PCTLMTD=P2, the value of PCTLDEF is always 5.

**PCTLMTD=** method

Specifies the method used to estimate percentiles.

**ORD_STAT**
Uses the order statistics method to compute the confidence limits as described by Hahn and Meeker (1991). See PROC UNIVARIATE for more information.

P2

Uses the P2 algorithm described in Jain and Chlamtac (1985).

**PCTLPTS=percentile(s)**

Creates observations containing percentiles in the OUTSTAT= data set. A percentile requested is identified by a _TYPE_ variable in the OUTSTAT= data set with a value of Pn. For example, if PCTLPTS=10, 30 has been specified, the corresponding observations contain the 10th and 30th percentiles in the OUTSTAT= data set (_TYPE_ =P10 and _TYPE_=30, respectively). See PROC UNIVARIATE for more information.

**Range:** A percentile is any decimal number n where 0 \leq n \leq 100

**PSTAT**

Prints the location and scale measures. (See the Displayed Output section for more information.)

**REPLACE**

Replaces the missing data with 0 in the standardized data (which corresponds to the location measure before standardizing). To replace missing data with something else, see the MISSING= argument.

**CAUTION:**

You may not specify both REPLACE and REPONLY.  ■

**REPONLY**

Replaces the missing data with the location measure and does not standardize the data.

**CAUTION:**

You may not specify both REPLACE and REPONLY.  ■

**SNORM**

Normalizes the scale estimator to have an expectation of approximately 1 for a standard normal distribution when METHOD=SPACING is specified.

**UNSTD | UNSTDIZE**

Unstandardizes variables when METHOD=IN (SAS-data-set) is specified. See Unstandardization section for more information.

**VARDEF=DF | WGT | WEIGHT | N | WDF**

Specifies the divisor to use in the calculation of variances. The following table shows the possible values for the divisor and associated divisors. See PROC UNIVARIATE for more information.

<table>
<thead>
<tr>
<th>Possible Values for VARDEF=</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>DF</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>WDF</td>
</tr>
<tr>
<td>WEIGHT</td>
</tr>
</tbody>
</table>

**Default:** DF

*Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.*
BY Statement

Standardizes observations separately in groups defined by the by-variable(s).

Discussion: When METHOD=IN (SAS-data-set) is specified, you can also specify a BY statement to apply separate groups of standardization measures (location and scale) to part of the DATA= data set and the following rules are applied to BY processing:

- If the IN data set does not contain any of the BY variables, the entire DATA= data set is standardized using location and scale measures that have ADD and MULT constants in the IN data set.

- If the IN data set contains some but not all of the BY variables, or if some BY variables do not have the same type or length in the IN data set as in the DATA= data set, then PROC STDIZE prints an error message and halts processing.

- If all of the BY variables appear in the IN data set with the same type and length as in the DATA= data set, then each BY group in the DATA= data set is standardized using location and scale measures along with ADD and MULT constants from the corresponding BY group in the IN data set. The BY groups in the IN data set must be in the same order as the DATA= data set. All BY groups in the IN data set must appear in the DATA= data set. If you do not specify the NOTSORTED option, some BY groups can appear in the DATA= data set but not in the IN data set; such BY groups are not used in standardizing the data.

**BY** variable-1 <... variable-n> <NOTSORTED>;

Required Argument

**variable-1 ... variable-n**

Specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. These variables are called by-variables.

Options

**NOTSORTED**

Specifies that observations are not necessarily sorted in alphabetic or numeric order. The data is grouped in another way, for example, chronological order. The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option.
**Tip:** If your DATA= data set is not sorted in ascending order, use one of the following alternatives:

- Sort the data by using the SORT procedure with a similar BY statement.
- Specify the BY statement option NOTSORTED or DESCENDING in the BY statement for the STDIZE procedure. The NOTSORTED option does not mean that the data is unsorted but rather that the data is arranged in groups (according to values of the BY variables) and that these groups are not necessarily in alphabetic or increasing numeric order.
- Create an index on the BY variables by using the DATASETS procedure.

For more information about the BY statement, refer to *SAS Language Reference: Concepts*. For more information about the DATASETS procedure, refer to the *SAS Procedures Guide*.
The STDIZE Procedure

FREQ Statement

Specifies a numeric variable whose values represent the frequency of the observation.

Discussion: If one variable in the input data set represents the frequency of occurrence for other values in the observation, specify the variable's name in a FREQ statement. PROC STDIZE then treats the data set as if each observation appeared \( n \) times, where \( n \) is the value of the FREQ variable for the observation.

Alias: FREQUENCY

FREQ variable;

Required Argument

\( \text{variable} \)

Specifies a single numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, the procedure assumes that each observation represents \( n \) observations, where \( n \) is the value of \( \text{variable} \).

Range: If \( \text{variable} \) is not an integer, the SAS System truncates it to the largest integer less than the FREQ value. If \( \text{variable} \) is less than 1 or is missing, the procedure does not use that observation to calculate statistics. The sum of the frequency variable represents the total number of observations and cannot exceed \( 2^{31} \).

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
LOCATION Statement

Specifies the list of numeric variables that contain location measures in the input data set of METHOD=IN.

LOCATION variable(s);

Required Argument

variable(s)

Specifies one or more variables that contain location measures in the input data set identified by the METHOD=IN(SAS-data-set).

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
SCALE Statement

Specifies the list of numeric variables that contain scale measures in the input data set of METHOD=IN.

```
SCALE variable(s);
```

Required Argument

```
variable(s)
```

Identifies one or more variables that contain scale measures in the input data set identified by the METHOD=IN(SAS-data-set).

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
VAR Statement

Specifies a list of variables to be standardized.

<table>
<thead>
<tr>
<th>Alias:</th>
<th>VARIABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>If the VAR statement is omitted, all numeric variables not specified in the BY=, FREQ=, LOCATION=, SCALE=, or WEIGHT= lists are standardized.</td>
</tr>
</tbody>
</table>

```
VAR variable(s);
```

Required Argument

`variable(s)`

Identifies one or more variables to be standardized.

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
WEIGHT Statement

Specifies a single numeric weight in the input SAS data set whose values are used to weight each observation. Only one variable can be specified.

Alias: WGT

WEIGHT variable;

Required Argument

variable

Specifies a numeric variable whose value weights the analysis variables in each observation. The variable does not have to be an integer. If the value of variable is less than 0 or is missing, the procedure uses a value of 0.

Discussion: The sample mean and (uncorrected) sample variances are computed as:

\[
\bar{x}_\omega = \frac{\sum_i w_i x_i}{\sum_i w_i}
\]

\[
u s^2_\omega = \frac{\sum_i w_i x_i^2}{d}
\]

\[
s^2_\omega = \frac{\sum_i w_i (x_i - \bar{x}_\omega)^2}{d}
\]

where \(w_i\) is the weight value of the \(i\)th observation, \(x_i\) is the value of the \(i\)th observation, and \(d\) is the divisor controlled by the VARDEF=option (see VARDEF= option for details).

MEAN

the weighted mean \(\bar{x}_\omega\)
SUM
the weighted sum $\sum \omega_i x_i$

USTD
the weighted uncorrected standard deviation, $\sqrt{\sum \omega

STD
the weighted standard deviation, $\sqrt{\sum \omega^2

EUCLEN
The weighted Euclidean length is computed as the weighted squared root of uncorrected sum of squares:

$$\sqrt{\sum_{i} \omega_i x_i^2}$$

AGK
Refers to the weight statement in PROC ACECLUS for how weight is applied to the AGK estimate (METHOD=COUNT) option.

L
Refers to the weight statement in PROC FASTCLUS for how weight is used to compute weighted cluster means (LEAST= option). Note that the number of clusters is always 1.

Range:
The WEIGHT variable values can be non-integers. An observation is used in the analysis only if the value of the WEIGHT variable is greater than 0.

Tip:
The WEIGHT variable only affects METHOD=MEAN, SUM, EUCLEN, USTD, STD, AGK, and L calculations.
Details

Standardization Methods

The following table lists standardization methods and their corresponding location and scale measures that are available in the METHOD= option.

*Table of Methods for Computing Location and Scale Measures*

<table>
<thead>
<tr>
<th>METHOD</th>
<th>SCALE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABW((c))</td>
<td>Biweight A-estimate</td>
<td>Biweight 1-step M-estimate</td>
</tr>
<tr>
<td>AGK((p))</td>
<td>AGK estimate (ACECLUS)</td>
<td>Mean</td>
</tr>
<tr>
<td>AHUBER ((c))</td>
<td>Huber A-estimate</td>
<td>Huber 1-step M-estimate</td>
</tr>
<tr>
<td>AWAVE ((c))</td>
<td>Wave A-estimate</td>
<td>Wave 1-step M estimate</td>
</tr>
<tr>
<td>EUCLEN</td>
<td>Euclidean length</td>
<td>0</td>
</tr>
<tr>
<td>IN(SAS-data-set)</td>
<td>Read from the data set</td>
<td>Read from the data set</td>
</tr>
<tr>
<td>IQR</td>
<td>Interquartile range</td>
<td>Median</td>
</tr>
<tr>
<td>LEAST</td>
<td>L((p))</td>
<td>L((p))</td>
</tr>
<tr>
<td>MAD</td>
<td>Median absolute deviation from the median</td>
<td>Median</td>
</tr>
<tr>
<td>Variable</td>
<td>Definition</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>MAXABS</td>
<td>Maximum absolute value</td>
<td>0</td>
</tr>
<tr>
<td>MEAN</td>
<td>1</td>
<td>Mean</td>
</tr>
<tr>
<td>MEDIAN</td>
<td>1</td>
<td>Median</td>
</tr>
<tr>
<td>MIDRANGE</td>
<td>Range/2</td>
<td>Midrange</td>
</tr>
<tr>
<td>RANGE</td>
<td>Range</td>
<td>Minimum</td>
</tr>
<tr>
<td>SPACING (p)</td>
<td>Minimum spacing</td>
<td>Mid minimum-spacing</td>
</tr>
<tr>
<td>STD</td>
<td>Standard deviation</td>
<td>Mean</td>
</tr>
<tr>
<td>SUM</td>
<td>Sum</td>
<td>0</td>
</tr>
<tr>
<td>USTD</td>
<td>Standard deviation about the origin</td>
<td>0</td>
</tr>
</tbody>
</table>

- For METHOD=ABW(c), METHOD=AHUBER(c), or METHOD=AWave(c), c is a positive numeric tuning constant (Iglewicz, 1983).
- For METHOD=AGK(p), p is a numeric constant that gives the proportion of pairs to be used with METHOD=COUNT in the ACECLUS procedure (Refer to SAS/STAT Software: Changes and Enhancements for Release 6.12 p. 229).
- For METHOD=SPACING(p), p is a numeric constant that gives the proportion of data to be contained in the spacing.
- For METHOD=L(p), p is a numeric constant greater than or equal to 1 that specifies the power to which differences are to be raised in computing an L(p) or Minkowski metric.
- For METHOD=IN(SAS-data-set), the SAS data set can contain:
  1. a _TYPE_ variable which identifies the observations that contain location and scale measures. For example, PROC STDIZE produces an OUTSTAT= data set that contains LOCATION and SCALE measures and some other statistics. _TYPE_='LOCATION' identifies the observation that contains location measures and _TYPE_='SCALE' identifies the observation that contains scale measures. You can also use the data set created by the OUTSTAT= option from another PROC STDIZE statement as the IN= data set name. See the Output Data Sets section below for the contents of the OUTSTAT data set.
  2. the location and scale variables specified by the LOCATION and SCALE statements.

PROC STDIZE reads in the location and scale variables in the IN=data set according to the following
rules: PROC STDIZE first looks for the _TYPE_ variable in the IN=data set. If it is found, PROC STDIZE continues to search for all variables specified in the VAR statement. If the _TYPE_ variable is not found, PROC STDIZE searches for the location variables specified in the LOCATION statement and the scale variables specified in the SCALE statement.

For robust estimators, see Goodall (1983) and Iglewicz (1983). MAD has the highest breakdown point (50%) but is not very efficient. ABW, AHUBER, and AWAVE provide a good compromise between breakdown and efficiency. L(p) location estimates are increasingly robust as p drops from 2 (least squares, that is, the mean) to 1 (least absolute value, that is, the median), but the L(p) scale estimates are not robust.

Spacing is robust to both outliers and clustering (Jannsen, et al., 1983) and is therefore a good choice for cluster analysis or nonparametric density estimation. The mid minimum spacing estimates the mode for small p. AGK is also robust to clustering and more efficient than SPACING, but it is not as robust to outliers and takes longer to compute. If you expect g clusters, the argument to SPACING or AGK should be 1/g or less. AGK is less biased than SPACING in small samples. It would generally be reasonable to use AGK for samples of size 100 or less and to use SPACING for samples of size 1000 or more, with the treatment of intermediate sample sizes depending on the available computer resources.

### Computation of the Statistics

Formulas for statistics of METHOD= MEAN, MEDIAN, SUM, USTD, STD, RANGE, and IQR are given in Chapter 1, "SAS Elementary Statistics Procedure", in the *SAS Procedures Guide*. Note that the computations of median and upper and lower quartiles depend on the PCTLMTD= option.

The rest of the statistics used in the above Table of Methods for Computing Location and Scale Measures, with the exception of METHOD=IN, are described as follows:

**EUCLEN**

Euclidean length.

\[ \sqrt{\sum_{i=1}^{n} \omega_i x_i^2} \]

where \( x_i \) is the \( i \)th observation and \( n \) is the total number of observations in the sample.

**L(p)**

Minkowski metric. It is documented as the LEAST=\( p \) option in the FASTCLUS procedure (see "The FASTCLUS Procedure" in the *SAS/STAT User's Guide*). Specifying METHOD=L(\( p \)) in the PROC STDIZE statement is almost the same as specifying LEAST=(\( p \)) option with MAXCLUS=1 and using the default values of the MAXITER= option in the PROC FASTCLUS statement. The only difference comes from the fact that the maximum number of iterations is a criterion for convergence on all variables simultaneously in PROC STDIZE while it is a criterion for convergence on a single multivariate statistic in PROC FASTCLUS. The location and scale measures for L(\( p \)) are output to the OUTSEED= data set in PROC FASTCLUS.

**MIDRANGE**
The midrange is defined as $\frac{\max + \min}{2}$.

**ABW(c)**


**AHUBER(c)**


**AWAVE(c)**


**AGK(p)**

This is the non-iterative univariate form of the estimator described by Art, Gnanadesikan, and Kettenring (1982).

The AGK estimate is documented as the METHOD= option in the PROC ACECLUS statement of the ACECLUS procedure. (See "The ACECLUS Procedure" in the SAS/STAT User's Guide). Specifying METHOD= AGK(p) in the PROC STDIZE statement is the same as specifying METHOD=COUNT and P=p in the PROC ACECLUS statement.

**SPACING(p)**

A spacing is the absolute difference between two data values. The minimum spacing for a proportion p is the minimum absolute difference between two data values that contain a proportion p of the data between them. The mid minimum spacing is the mean of these two data values.

### Computing Quantiles

Proc STDIZE offers two methods for computing quantiles:

- the P2 approach
- the order-statistics approach (as in PROC UNIVARIATE)

The P2 approach used in PROC STDIZE modifies the $P^2$ algorithm for histograms proposed by Jain and Chlamtac (1985). The main difference comes from the movement of markers. P2 allows a marker to move to the right (or left) by more than one position (to the largest possible integer) as long as it would not result in two markers being in the same position. This modification is necessary to prorate the FREQ variable.

Using the P2 approach to estimate quantiles beyond the quartiles (<P25 and >P75) will not always produce accurate results and a large sample size (10,000 or more) is required if the tail quantiles (≤P10 and ≥P90) are requested. Also, tail quantiles are not recommended for highly skewed and/or heavy-tailed distributions.
The order statistics approach for estimating quantiles is faster than the P2 method, but it requires that the entire data be stored in memory. The accuracy in estimating the quantiles are comparable for both methods. The default is PCTLMTD=ORD_STAT if enough memory is available; otherwise, PCTLMTD= P2.

**Missing Values**

Missing values can be replaced by the LOCATION measure or by any specified constant (see the REPLACE option and the MISSING= option). You can also suppress standardization if you only want to replace missing values (see the REPONLY option).

If the NOMISS option is used, PROC STDIZE omits observations that have any missing values in the analyzed variables from computation of the location and scale measures. Otherwise, all nonmissing values are used.

**Output Data Sets**

**OUT=Data-Set**

The output data set is a copy of the DATA= data set except that the analyzed variables (those in the VAR statement, or if there is no VAR statement, all numeric variables not listed in any other statement) have been standardized.

**OUTSTAT=Data-Set**

The new data set contains the following variables:

- the BY variables, if any;
- a new variable, _TYPE_, a character variable;
- the variables analyzed, that is, those in the VAR statement, or if there is no VAR statement, all numeric variables not listed in any other statement.

Each observation in the new data set contains some type of statistic as indicated by the _TYPE_ variable. The values of the _TYPE_ variable are as follows:

<table>
<thead>
<tr>
<th><em>TYPE</em></th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCATION</td>
<td>Location measure of each variable.</td>
</tr>
<tr>
<td>SCALE</td>
<td>Scale measure of each variable.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>ADD</td>
<td>Constant from ADD=. This value is the same for each variable.</td>
</tr>
<tr>
<td>MULT</td>
<td>Constant from MULT=. This value is the same for each variable.</td>
</tr>
<tr>
<td>N</td>
<td>Total number of nonmissing positive frequencies of each variable.</td>
</tr>
<tr>
<td>NORM</td>
<td>Norm measure of each variable. This observation is produced only if either the NORM option is specified and METHOD= AGK, IQR, MAD, or SPACING or when the SNORM option is specified and METHOD=SPACING.</td>
</tr>
<tr>
<td>Pn</td>
<td>Percentiles of each variable specified by PCTLPTS= where n is any real number such that 0 ≤ n ≤ 100.</td>
</tr>
</tbody>
</table>

**Displayed Output**

If you specify the PSTAT option, PROC STDIZE displays the following statistics for each variable:

- Name: the name of the variable
- Location: the location estimate
- Scale: the scale estimate
- Norm: the norm estimate
- N: the total non-missing positive frequencies
Unstandardization

The formula for Unstandardization is based upon the location and scale measures and the constants for addition and multiplication. All of these are identified by the _TYPE_ variable in the SAS-data-set.

The SAS-data-set must have a _TYPE_ variable that contains the following observations: a _TYPE_=LOCATION observation and a _TYPE_=SCALE observation. _TYPE_=ADD, and _TYPE_=MULT are optional observations; if they are not found in the SAS-data-set, the constants specified in the ADD= and MULT= options (or their default values) are used for unstandardization. See OUTSTAT= for details about the kind of statistics represented by each value of _TYPE_.

The formula for unstandardization is:

\[
\text{original} = \text{scale} \times \frac{(\text{result} - \text{adder})}{(\text{multiplier} + \text{location})}
\]

where:

result

is the value obtained from the previous standardization

adder

is the constant to add (the value found in the _TYPE_ variable of the SAS-data-set or specified in the ADD= option)

multiplier

is the constant to multiply by (the value found in the _TYPE_ variable or specified in the MULT= option)

original

is the original input value

location

is the location measure

scale

is the scale measure

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Examples

The following examples were executed using the HP-UX version 10.20 operating system and the SAS software release 6.12TS045.

Example 1: Getting Started with the STDIZE Procedure

Example 2: Unstandardizing a Data Set

Example 3: Replacing Missing Values with Standardizing

Example 4: Replacing Missing Values without Standardizing the Variables

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Example 1: Getting Started with the STDIZE Procedure

Features:
- Setting the Method= Standardization Statistic.
- Standardizing Observations using BY-Group Processing
- Outputting the OUT= Standardized data set
- Outputting the OUTSTAT= Summary Statistic data set.

This example demonstrates how to center numeric variables by their medians with the STDIZE procedure. Observations in the input data set are standardized separately in groups for each level of the binary target.

The example uses a fictitious mortgage data set named SAMPSIO.HMEQ, which contains 5,960 cases. It is stored in the sample library. Each case represents an applicant for a home equity loan. All applicants have an existing mortgage. The binary target BAD indicates whether or not an applicant eventually defaulted or was ever seriously delinquent.

Program

```sas
proc sort data=sampsio.hmeq out=hmeq;
   by bad;
run;

proc stdize data=hmeq
   out=stdhmeq
      method=median
      outstat=stdstats;
   var mortdue value yoj derog delinq clage ninq clno debtinc;
   by bad;
   title 'Standardize using METHOD=Median';
   title2 'For Each Level of the Target BAD';
run;
```
proc print data=stdhmeq(obs=10);
  title 'Partial Listing of the OUT= Standardized Data Set';
run;

proc print data=hmeq(obs=10);
  title 'Partial Listing of the Unstandardized Data Set';
run;

proc print data=stdstats;
  title 'Location and Scale Measures Data Set';
run;

Output

PROC PRINT Partial Listing of the OUT= Standardized Data Set and the Unstandardized Input Data Set

Partial Listing of the OUT= Standardized Data Set

| M  | O  | R  | V  | E  | D  | C  | B  | L  | T  | A  | A  | E  | L  | L  | N  | C  | T  | O  | B  | D  | L  | S  | J  | Y  | R  | I  | A  | I  | L  | I  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1  | 0  | 1700 | 30961 | 21341 | HomeImp | Office | -4.0 | 0 | 0 | -87.0825 | -1 | -6 | . |
| 2  | 0  | 2000 | -2303 | -3259 | Mgr | -4.5 | 0 | 0 | -33.2825 | -1 | 4 | . |
| 3  | 0  | 2300 | 35531 | 30294 | HomeImp | Office | -5.0 | 0 | 0 | -89.4233 | -1 | -7 | -2.95317 |
| 4  | 0  | 2400 | 31610 | 26536 | HomeImp | Office | -3.0 | 0 | 0 | -86.6040 | -1 | -7 | -4.85984 |
| 5  | 0  | 2500 | -59610 | -46143 | HomeImp | Self | . | 0 | 0 | 27.5842 | -1 | -8 | . |
| 6  | 0  | 2500 | 4569 | -12059 | HomeImp | ProfExe | 1.0 | 0 | 0 | 75.3175 | -1 | -8 | . |
| 7  | 0  | 2900 | 37110 | 21846 | HomeImp | Office | -6.0 | 0 | 0 | -84.3135 | -1 | -7 | -4.49054 |
| 8  | 0  | 2900 | 37534 | 30043 | HomeImp | Office | -5.0 | 0 | 0 | -78.8755 | -1 | -7 | -4.62581 |
| 9  | 0  | 3000 | 37731 | 31070 | HomeImp | Office | -5.0 | 0 | 0 | -94.5314 | -1 | -6 | -2.48189 |
| 10 | 0  | 3000 | -8839 | -19159 | HomeImp | Mgr | 3.0 | . | 2 | 31.5175 | -1 | 5 | . |

Partial Listing of the Unstandardized Data Set

<table>
<thead>
<tr>
<th>M</th>
<th>O</th>
<th>R</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1700</td>
<td>30961</td>
<td>21341</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2000</td>
<td>-2303</td>
<td>-3259</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2300</td>
<td>35531</td>
<td>30294</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2400</td>
<td>31610</td>
<td>26536</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>2500</td>
<td>-59610</td>
<td>-46143</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>2500</td>
<td>4569</td>
<td>-12059</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>2900</td>
<td>37110</td>
<td>21846</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>2900</td>
<td>37534</td>
<td>30043</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>3000</td>
<td>37731</td>
<td>31070</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>3000</td>
<td>-8839</td>
<td>-19159</td>
</tr>
</tbody>
</table>
PROC PRINT Partial Listing of the OUTSTAT= Location and Scale Measurement Data Set

The OUTSTAT= data set contains a _TYPE_ variable that identifies the following summary statistics for each variable for each value of the BY variable BAD:

- LOCATION - the location value for the METHOD= measure.
- SCALE - the scale value for the METHOD= measure.
- ADD - a constant to add that is specified with the ADD= option.
- MULT - a constant to multiply by that is specified with the MULT= option.
- N - the number of total non-missing positive frequencies.

Location and Scale Measures Data Set

<table>
<thead>
<tr>
<th>OBS</th>
<th>BAD</th>
<th><em>TYPE</em></th>
<th>MORTDUE</th>
<th>VALUE</th>
<th>YOJ</th>
<th>DEROG</th>
<th>DELINQ</th>
<th>CLAGE</th>
<th>NINQ</th>
<th>CLNO</th>
<th>DEBTINC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>LOCATION</td>
<td>66839</td>
<td>90659</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>180.42</td>
<td>1</td>
<td>20</td>
<td>34.54</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>SCALE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>ADD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>MULT</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>N</td>
<td>4359</td>
<td>4764</td>
<td>4321</td>
<td>4150</td>
<td>4263</td>
<td>4541.00</td>
<td>4336</td>
<td>4602</td>
<td>4290.00</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>LOCATION</td>
<td>60279</td>
<td>82000</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>132.87</td>
<td>1</td>
<td>20</td>
<td>38.08</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>SCALE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>ADD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>MULT</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>N</td>
<td>1083</td>
<td>1084</td>
<td>1124</td>
<td>1102</td>
<td>1117</td>
<td>1111.00</td>
<td>1114</td>
<td>1136</td>
<td>403.00</td>
</tr>
</tbody>
</table>
PROC SORT sorts the SAMPSIO.HMEQ data set by the values of the target BAD and outputs the data to the OUT=HMEQ data set.

```plaintext
proc sort data=sampsio.hmeq out=hmeq;
   by bad;
run;
```
The PROC STDIZE statement invokes the procedure. The DATA= option names the input data set. The OUT= option specifies the output data set.

```
proc stdize data=hmeq
   out=stdhmeq
```
The METHOD= option specifies the standardization statistic. By default, PROC STDIZE uses the standard deviation (STD) as the METHOD=location measure.

method=median
The OUTSTAT= option specifies the output statistic data set that contains location and scale measurements plus other simple statistics.

   outstat=stdstats;
The VAR statement specifies a list of the numeric variables to be standardized.

```
    var mortdue value yoj derog delinq
        clage ninq clno debtinc;
```
The BY statement standardizes observations separately in groups for each level of the BY variable.

by bad;
title 'Standardize using METHOD=Median';
title2 'For Each Level of the Target BAD'; run;
PROC PRINT lists the first 10 observations in the standardized data set.

proc print data=stdhmeq(obs=10);
    title 'Partial Listing of the OUT= Standardized Data Set';
run;
PROC PRINT lists the first 10 observations without standardization (for comparison purposes).

proc print data=hmeq(obs=10);
  title 'Partial Listing of the Unstandardized Data Set';
run;
PROC PRINT lists the OUTSTAT= location and scale measurement data set.

proc print data=stdstats;
   title 'Location and Scale Measures Data Set';
run;
Example 2: Unstandardizing a Data Set

Features: ● Using the UNSTDIZE and the METHOD=IN(data set) Options to Unstandardize a Data Set

This example demonstrates how to unstandardize selected variables in the WORK.STDHMEQ standardized data set that was created in Example 1 "Getting Started with the STDIZE Procedure".

Program

```
proc stdize data=stdhmeq
  out=unhmeq
  method=in(stdstats)
  unstdize;

  var mortdue value yoj clno;

  by bad;
  title 'Unstandardizing a Data Set';
  run;

proc print data=unhmeq(obs=10);
  title 'Partial Listing of the Unstandardized Data Set';
  run;
```

Output

PROC PRINT Partial Listing of the Unstandardized Data Set

```
M                                                      D
O             R                   D                    E
R      V      E                 D E     C              B
L     T      A      A                 E L     L     N  C     T
O  B   O     D      L      S    J         Y  R I     A     I  L     I
B  A   A     U      U      O    O         O  O N     G     N N N
```
<table>
<thead>
<tr>
<th>SDNENBJGQEJOQOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  0 1700 97800 112000 HomeImp Office  3.0 0 0 -87.0825 -1 14 .</td>
</tr>
<tr>
<td>2  0 2000 64536 87400 Mgr 2.5 0 0 -33.2825 -1 24 .</td>
</tr>
<tr>
<td>3  0 2300 102370 120953 HomeImp Office 2.0 0 0 -89.4233 -1 13 -2.95317</td>
</tr>
<tr>
<td>4  0 2400 98449 117195 HomeImp Office 4.0 0 0 -86.6040 -1 13 -4.85984</td>
</tr>
<tr>
<td>5  0 2500 7229 44516 HomeImp Self . 0 0 27.5842 -1 12 .</td>
</tr>
<tr>
<td>6  0 2500 71408 78600 HomeImp ProfExe 8.0 0 0 75.3175 -1 12 .</td>
</tr>
<tr>
<td>7  0 2900 103949 112505 HomeImp Office 1.0 0 0 -84.3135 -1 13 -4.49054</td>
</tr>
<tr>
<td>8  0 2900 104373 120702 HomeImp Office 2.0 0 0 -78.8755 -1 13 -4.62581</td>
</tr>
<tr>
<td>9  0 3000 104570 121729 HomeImp Office 2.0 0 0 -94.5314 -1 14 -2.48189</td>
</tr>
<tr>
<td>10  0 3000 58000 71500 HomeImp Mgr 10.0 . 2 31.5175 -1 25 .</td>
</tr>
</tbody>
</table>

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The METHOD=IN(SAS Data Set) option identifies the data set that contains measurement and location statistics for each variable. The WORK.STDSTATS data set was created by using the OUTSTAT= data set option in the PROC STDIZE step of Example 1. The UNSTDIZE option specifies to unstandardize variables when METHOD=IN(SAS data set) is specified.

```plaintext
proc stdize data=stdhmeq
   out=unhmeq
   method=in(stdstats)
   unstdize;
```

The VAR statement specifies a list of numeric variables that you want to unstandardize.

```plaintext
var mortdue value yoj clno;
```
The BY statement specifies to unstandardize the variables using the values of the BY variable. The METHOD=IN(STDSTATS) data set must contain two observations: _TYPE_ =LOCATION and _TYPE_ =SCALE for each value of the BY variable.

```plaintext
by bad;
title 'Unstandardizing a Data Set';
run;
```
PROC PRINT prints the first 10 observations in the unstandardized data set.

proc print data=unhmeq(obs=10);
   title 'Partial Listing of the Unstandardized Data Set';
run;
Example 3: Replacing Missing Values with Standardizing

This example demonstrates how to replace missing values and standardize selected numeric variables in the SAMSPIO.HMEQ (home equity) data set. When you use the REPLACE option in conjunction with the METHOD= location option, the STDIZE procedure replaces and standardizes the numeric variables.

Because the STDIZE procedure only replaces missing numeric values, a predecessor SAS DATA step is used to replace the missing character values with the variable's mode.

Program

```sas
proc freq data=sampsio.hmeq;
  tables reason job;
run;

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

proc stdize data=hmeq
  out=rshmeq
  method=mean
  replace;
  var mortdue value yoj derog delinq
     clage ninq clno debtinc;
  title 'Impute and Standardize';
run;

proc print data=rshmeq(obs=10);
  title 'Partial Listing of the Imputed/Standardized Data Set';
run;

proc print data=hmeq(obs=10);
  title 'Partial Listing of the Input Data Set';
run;
```
### PROC FREQ Frequency Table of the Home Equity Character Variables

#### Partial Listing of the Input Data Set

<table>
<thead>
<tr>
<th>REASON</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>DebtCon</td>
<td>3928</td>
<td>68.8</td>
<td>3928</td>
<td>68.8</td>
</tr>
<tr>
<td>HomeImp</td>
<td>1780</td>
<td>31.2</td>
<td>5708</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Frequency Missing = 252

#### JOB

<table>
<thead>
<tr>
<th>JOB</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mgr</td>
<td>767</td>
<td>13.5</td>
<td>767</td>
<td>13.5</td>
</tr>
<tr>
<td>Office</td>
<td>948</td>
<td>16.7</td>
<td>1715</td>
<td>30.2</td>
</tr>
<tr>
<td>Other</td>
<td>2388</td>
<td>42.0</td>
<td>4103</td>
<td>72.2</td>
</tr>
<tr>
<td>ProfExe</td>
<td>1276</td>
<td>22.5</td>
<td>5379</td>
<td>94.7</td>
</tr>
<tr>
<td>Sales</td>
<td>109</td>
<td>1.9</td>
<td>5488</td>
<td>96.6</td>
</tr>
<tr>
<td>Self</td>
<td>193</td>
<td>3.4</td>
<td>5681</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Frequency Missing = 279

### PROC PRINT Partial Listing of the Imputed/Standardized Data Set and the Input Data Set

#### Partial Listing of the Imputed/Standardized Data Set

<table>
<thead>
<tr>
<th>OBS</th>
<th>BAD</th>
<th>LOAN</th>
<th>MORTDUE</th>
<th>VALUE</th>
<th>REASON</th>
<th>JOB</th>
<th>YOJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1100</td>
<td>-47900.82</td>
<td>-62751.05</td>
<td>HomeImp</td>
<td>Other</td>
<td>1.57773</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1300</td>
<td>-3707.82</td>
<td>-33376.05</td>
<td>HomeImp</td>
<td>Other</td>
<td>-1.92227</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1500</td>
<td>-60260.82</td>
<td>-85076.05</td>
<td>HomeImp</td>
<td>Other</td>
<td>-4.92227</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1500</td>
<td>0.00</td>
<td>0.00</td>
<td>DebtCon</td>
<td>Other</td>
<td>0.00000</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1700</td>
<td>24039.18</td>
<td>10223.95</td>
<td>HomeImp</td>
<td>Office</td>
<td>-5.92227</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1700</td>
<td>-43212.82</td>
<td>-61456.05</td>
<td>HomeImp</td>
<td>Other</td>
<td>0.07773</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1800</td>
<td>-25111.82</td>
<td>-44739.05</td>
<td>HomeImp</td>
<td>Other</td>
<td>-3.92227</td>
</tr>
</tbody>
</table>
### Partial Listing of the Input Data Set

<table>
<thead>
<tr>
<th>OBS</th>
<th>DEROG</th>
<th>DELINQ</th>
<th>CLAGE</th>
<th>NINQ</th>
<th>CLNO</th>
<th>DEBTINC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25457</td>
<td>0.44944</td>
<td>85.400</td>
<td>0.18606</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>2</td>
<td>0.25457</td>
<td>1.55056</td>
<td>57.933</td>
<td>1.18606</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>3</td>
<td>0.25457</td>
<td>0.44944</td>
<td>30.300</td>
<td>0.18606</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>4</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>5</td>
<td>0.25457</td>
<td>0.44944</td>
<td>86.433</td>
<td>1.18606</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>6</td>
<td>0.25457</td>
<td>0.44944</td>
<td>78.300</td>
<td>0.18606</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>7</td>
<td>2.74543</td>
<td>1.55056</td>
<td>102.666</td>
<td>0.18606</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>8</td>
<td>0.25457</td>
<td>0.44944</td>
<td>91.000</td>
<td>1.18606</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>9</td>
<td>0.25457</td>
<td>1.55056</td>
<td>37.167</td>
<td>0.18606</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>10</td>
<td>0.25457</td>
<td>0.44944</td>
<td>63.966</td>
<td>1.18606</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
PROC FREQ creates a frequency table for each character variable. You can use the information in these tables to determine the mode of each character variable in the SAMPSIO.HMEQ.

proc freq data=sampsio.hmeq;
  tables reason job;
run;
The SAS DATA step imputes the missing character values with the variable's mode.

data hmeq;
   set sampsio.hmeq;
   if reason=' ' then reason='DebtCon';
   if job=' ' then job='Other';
run;
The REPLACE option signals the STDIZE procedure to replace the missing values and standardize the numeric variables using the METHOD=MEAN statistic. Therefore, all the missing values should be replaced by 0.

```
proc stdize data=hmeq
   out=rshmeq
   method=mean
   replace;
   var mortdue value yoj derog delinq
       clage ninq clno debtinc;
   title 'Impute and Standardize';
run;
```
PROC PRINT prints the first 10 observations in the OUT= imputed/standardized data set.

proc print data=rshmeq(obs=10);
    title 'Partial Listing of the Imputed/Standardized Data Set';
run;
PROC PRINT prints the first 10 observations in the HMEQ input data set.

proc print data=hmeq(obs=10);
   title 'Partial Listing of the Input Data Set';
run;
Example 4: Replacing Missing Values without Standardizing the Variables

Features: ● Using the REPOONLY option.

This example demonstrates how to replace missing numeric values in the SAMSJO.HMEQ (home equity) data set. When you use the REPOONLY option, the STDIZE procedure does not standardize the numeric variables; it replaces the missing values with the METHOD= location statistic.

Because the STDIZE procedure only accepts numeric variables, a predecessor SAS DATA step is used to replace the missing character values with the variable's mode.

Program

```sas
proc freq data=sampsio.hmeq;
   tables reason job;
run;

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

proc stdize data=hmeq
   out=replhmeq
   method=mean
   reponly;
   var mortdue value yoj derog delinq
       clage ninq clno debtinc;
   title 'Impute Missing Numeric Values';
run;

proc print data=replhmeq(obs=10);
   title 'Partial Listing of the Imputed Data Set';
run;
```
### PROC FREQ Frequency Table of the Home Equity Character Variables

**Partial Listing of the Input Data Set**

<table>
<thead>
<tr>
<th>REASON</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>DebtCon</td>
<td>3928</td>
<td>68.8</td>
<td>3928</td>
<td>68.8</td>
</tr>
<tr>
<td>HomeImp</td>
<td>1780</td>
<td>31.2</td>
<td>5708</td>
<td>100.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Frequency Missing = 252</td>
<td></td>
</tr>
</tbody>
</table>

**PROC FREQ Frequency Table of the Home Equity Character Variables**

**Partial Listing of the Imputed Data Set**

<table>
<thead>
<tr>
<th>JOB</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mgr</td>
<td>767</td>
<td>13.5</td>
<td>767</td>
<td>13.5</td>
</tr>
<tr>
<td>Office</td>
<td>948</td>
<td>16.7</td>
<td>1715</td>
<td>30.2</td>
</tr>
<tr>
<td>Other</td>
<td>2388</td>
<td>42.0</td>
<td>4103</td>
<td>72.2</td>
</tr>
<tr>
<td>ProfExe</td>
<td>1276</td>
<td>22.5</td>
<td>5379</td>
<td>94.7</td>
</tr>
<tr>
<td>Sales</td>
<td>109</td>
<td>1.9</td>
<td>5488</td>
<td>96.6</td>
</tr>
<tr>
<td>Self</td>
<td>193</td>
<td>3.4</td>
<td>5681</td>
<td>100.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Frequency Missing = 279</td>
<td></td>
</tr>
</tbody>
</table>

**PROC PRINT Listing of the Imputed Data Set and the Input Data Set**

**Partial Listing of the Imputed Data Set**

<table>
<thead>
<tr>
<th>OBS</th>
<th>BAD</th>
<th>LOAN</th>
<th>MORTDUE</th>
<th>VALUE</th>
<th>REASON</th>
<th>JOB</th>
<th>YOJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1100</td>
<td>25860.00</td>
<td>39025.00</td>
<td>HomeImp</td>
<td>Other</td>
<td>10.5000</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1300</td>
<td>70053.00</td>
<td>68400.00</td>
<td>HomeImp</td>
<td>Other</td>
<td>7.0000</td>
</tr>
<tr>
<td>OBS</td>
<td>DEROG</td>
<td>DELINQ</td>
<td>CLAGE</td>
<td>NINQ</td>
<td>CLNO</td>
<td>DEBTINC</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>--------</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.0000</td>
<td>0.0000</td>
<td>94.367</td>
<td>1.0000</td>
<td>9.0000</td>
<td>33.7799</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.0000</td>
<td>2.0000</td>
<td>121.833</td>
<td>0.0000</td>
<td>14.0000</td>
<td>33.7799</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.0000</td>
<td>0.0000</td>
<td>149.467</td>
<td>1.0000</td>
<td>10.0000</td>
<td>33.7799</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.25457</td>
<td>0.44944</td>
<td>179.766</td>
<td>1.18606</td>
<td>21.2961</td>
<td>33.7799</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.0000</td>
<td>0.0000</td>
<td>93.333</td>
<td>0.0000</td>
<td>14.0000</td>
<td>33.7799</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.0000</td>
<td>0.0000</td>
<td>101.466</td>
<td>1.0000</td>
<td>8.0000</td>
<td>37.1136</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3.0000</td>
<td>2.0000</td>
<td>77.100</td>
<td>1.0000</td>
<td>17.0000</td>
<td>33.7799</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.0000</td>
<td>0.0000</td>
<td>88.766</td>
<td>0.0000</td>
<td>8.0000</td>
<td>36.8849</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.0000</td>
<td>2.0000</td>
<td>216.933</td>
<td>1.0000</td>
<td>12.0000</td>
<td>33.7799</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.0000</td>
<td>0.0000</td>
<td>115.800</td>
<td>0.0000</td>
<td>13.0000</td>
<td>33.7799</td>
<td></td>
</tr>
</tbody>
</table>

Partial Listing of the Input Data Set

<table>
<thead>
<tr>
<th>M</th>
<th>O</th>
<th>R</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>E</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
<td>L</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>E</td>
<td>L</td>
</tr>
<tr>
<td>N</td>
<td>C</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>B</td>
<td>O</td>
<td>D</td>
</tr>
<tr>
<td>L</td>
<td>S</td>
<td>J</td>
<td>Y</td>
</tr>
<tr>
<td>R</td>
<td>I</td>
<td>A</td>
<td>I</td>
</tr>
<tr>
<td>L</td>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>D</td>
<td>N</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>N</td>
<td>B</td>
<td>J</td>
</tr>
<tr>
<td>G</td>
<td>Q</td>
<td>E</td>
<td>Q</td>
</tr>
<tr>
<td>O</td>
<td>C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Copyright 2000 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
PROC FREQ creates a frequency table for each character variable. You can use this information to determine the mode of each character variable in the SAMPSIO.HMEQ.

proc freq data=sampsio.hmeq;
   tables reason job;
run;
The SAS DATA step imputes the missing character values with the variable's mode.

data hmeq;
  set sampsio.hmeq;
  if reason=' ' then reason='DebtCon';
  if job=' ' then job='Other';
run;
The REPONLY only option signals the STDIZE procedure to replace but not standardize the numeric variables using the location measure of the METHOD=MEAN statistic. Therefore, all the missing values should be replaced by the means of their variables.

```
proc stdize data=hmeq
    out=replhmeq
    method=mean
    reponly;
    var mortdue value yoj derog delinq clage ninq clno debtinc;
    title 'Impute Missing Numeric Values';
run;
```
PROC PRINT prints the first 10 observations in the imputed data set.

proc print data=replhmeq(obs=10);
  title 'Partial Listing of the Imputed Data Set';
run;
PROC PRINT prints the first 10 observations in the HMEQ input data set.

proc print data=hmeq(obs=10);
    title 'Partial Listing of the Input Data Set';
run;
References


The TPARS Procedure

Overview
Procedure Syntax
  PROC TPARS Statement
  COPY Statement
  OUTPUT Statement
Output

Copyright 2001 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The TPARS procedure is used to create a term-by-document frequency table from a collection of
documents. Each document in the collection may be contained in a variable of a SAS data set or on the
file system. If the document is to reside on the file system, a data set variable will hold the path for the
document.

Copyright 2001 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
PROC TPARS <option(s)>;

COPY variables;

OUTPUT option(s);
PROC TPARS Statement

Invoke the TPARS procedure.

PROC TPARS <option(s)>;

Options

DATA = <libref.>SAS-data-set
   Specifies the name of the input data set. This data set has a variable that contains the text to be parsed or a variable that contains the file system path to the text that is to be parsed.

STOPLIST or STOP = <libref.>SAS-data-set
   Specifies the name of the data set that contains the list of words that are not to be indexed by the TPARS procedure. In most situations this list should include articles, prepositions, pronouns, etc. The variable name for the words in the data set must be Term.

VAR or TEXT = variable-name
   Specifies the name of the variable that contains the text to be parsed. This option cannot be used with the FVAR option.

FVAR or FILE = variable-name
   Specifies the name of the variable that contains the text to be parsed. This option cannot be used with the VAR option.

IN_KEY = <libref.>SAS-data-set
   Specifies the name of the SAS data set containing the KEY data. The KEY data is a set that contains index/term pairs. It is used as input after the initial parsing (for training) has been done. Successive runs will take as input the KEY data was output from the training runs. A second use for the IN_KEY option is for term/concept extraction. In this case, the KEY data set is created outside the TPARS procedure. The entries of the data set represent terms that you want to identify as being contained in the data set. For example, if the following data is placed as the IN_KEY data set, then only these words will be indexed.
This enables you to identify which documents contain words about the United States and North Carolina, respectively.

**TOKEN = ALPHANUM|ALPHA|SPACEDELIM**

Specifies what will qualify a term to be indexed. The default value is **ALPHANUM**. The TOKEN option works in conjunction with the ENHANCE option. The following descriptions hold for when the ENHANCE option is not used.

- **ALPHANUM** -- Terms are space and punctuation delimited. Each term will consist only of alpha/numeric characters.
- **ALPHA** -- Terms are space, punctuation, and digit delimited. Each term will consist only of alphabetical characters.
- **SPACEDELIM** -- Terms are space delimited. Terms will contain all characters including punctuation.

**ENHANCE = language**

This option is used to do a limited amount of language specific parsing. Currently only ENGLISH is supported. If parsing any other languages, the ENHANCE option should not be used. The ENHANCE option works in conjunction with the TOKEN option.

The following description hold for when the ENHANCE=ENGLISH is used.

- **ALPHANUM** -- Terms are space delimited. Terms that contain punctuation are omitted unless it is a contraction or a term with a single punctuation at the end. Contractions are kept in their original form. A term with a single punctuation at the end of the term is kept but the punctuation is removed.
- **ALPHA** -- Terms are space delimited. Terms that contain punctuation or digits are omitted unless it is a contraction or a term with a single punctuation at the end. Contractions are kept in their original form. A term with a single punctuation at the end of the term is kept but the punctuation is removed.
- **SPACEDELIM** -- Terms are space delimited. In addition, end of word punctuations are removed.
COPY Statement

COPY variables;

variables

Specifies the variables that you want to keep from the input data set.

Copyright 2001 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
The TPARS Procedure

OUTPUT Statement

```
OUTPUT option(s);
```

Options

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

Specifies the name of the data set that will contain the term-by-document frequency table.

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

Specifies the name of the data set that will contain the index/term pairs, which were indexed in the OUT data set.

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

Specifies the data set that will contain all of the variables listed in the COPY statement along with a new variable called DOC. The DOC variable is an index to the document number. This document number corresponds to the numbers in the _DOCUMENT_ variable of the OUT data set. The MERGE data set is used after a call to the SPSVD procedure and enables you to merge the original data set with the reduced dimension data.

Copyright 2001 by SAS Institute Inc., Cary, NC, USA. All rights reserved.
Output

The TPARS procedure generates two output data sets. One of the output data sets is a table in sparse matrix format that contains the following variables:

- **_TERM_** -- is the parsed text.
- **_TERMNUM_** -- is a unique numerical index associated with each term.
- **_DOCUMENT_** -- is the document number.
- **_COUNT_** -- is the number of times that the term appears in the document.

The table can be interpreted as an encoding of a sparse matrix. The following example represents a collection of four documents.

<table>
<thead>
<tr>
<th>TERM</th>
<th>TERMNUM</th>
<th>DOCUMENT</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>House</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sleep</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Garage</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Sleep</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>House</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Sleep</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

The collection is indexed by only three words. House appears one time in Document 1. House appears two times in Document 4. Garage appears three times in Document 2, etc.

Since the words are encoded into numerical representation, a KEY data set that contains the following variables is also output.

- **TERM** -- is the parsed text.
- **KEY** -- is a unique numerical index associated with each term.
- **FREQ** -- is the total number of times that a term appears in the document collection.
- **NUMDOC** -- is the number of documents in the collection that contain the term.

As an example, the following KEY data set indicates that the terms House, Garage, and Sleep are being identified by 1, 2, and 3, respectively. The term House appears three times in the document collection and two documents in the collection contain the word House.
Note: The values of _TERMNUM_ and KEY are identical. ■