Salford Predictive Modeler®

Classification Modeling in CART®

This guide provides a detailed description of classification modeling in CART®.

Minitab

Introduction

The main purpose of this guide is to provide a detailed overview of classification modeling in CART®. We will address the full set of options available during the model setup as well as guide you through all available output reports and displays. A simple dataset coming from the biomedical application field will be used to illustrate all of the key concepts.

Setting up a Classification Model in CART®

Modeling Dataset

We start by walking through a simple classification problem taken from the biomedical literature. The topic is low birth weight of newborns. The task is to understand the primary factors leading to a baby being born significantly underweight. The topic is considered important by public health researchers because low birth weight babies can impose significant burdens and costs on the healthcare system. A cutoff of 2500 grams is typically used to define a low birth weight baby.

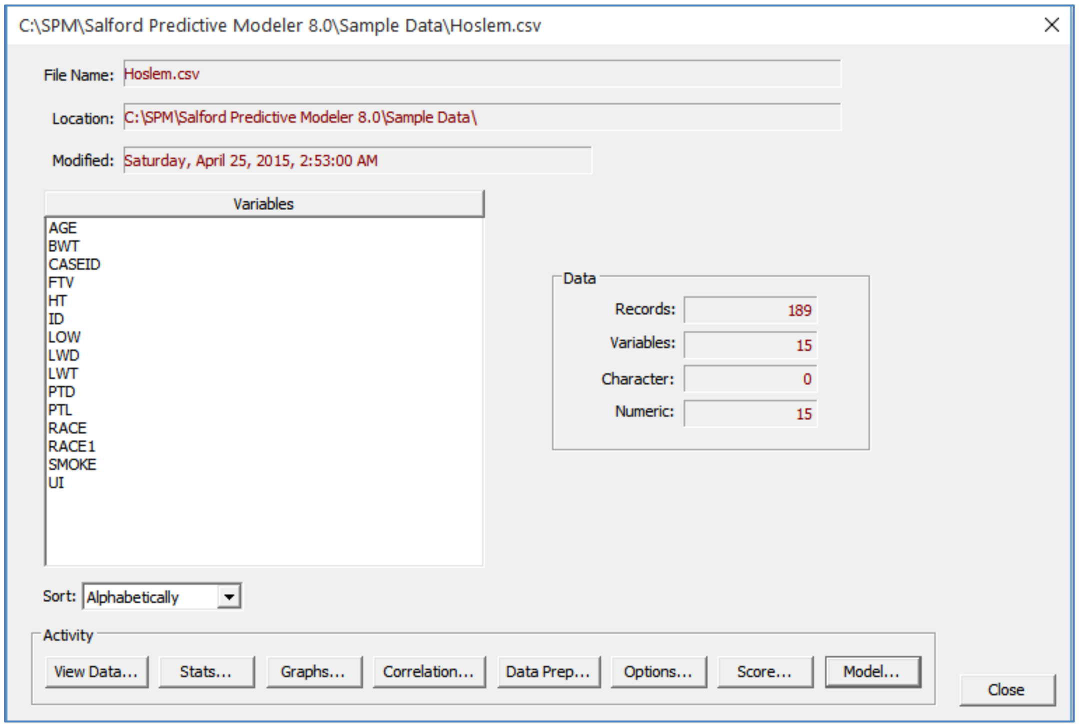
The following variables are available:

♦ **LOW**	- Birth weight less than 2500 grams (coded 1 if <2500, 0 otherwise).
♦ **AGE**	- Mother's age.
♦ **FTV**	- Number of first trimester physician visits.
♦ **HT**	- History of hypertension (coded 1 if present, 0 otherwise).
♦ **LWD**	- Low Mother's weight at last menstrual period (coded 1 if <110 pounds, 0 otherwise).
♦ **PTD**	- Occurrence of pre-term labor (coded 1 if present, 0 otherwise).
♦ **RACE**	- Mother's ethnicity (coded 1, 2 or 3).
♦ **SMOKE**	- Smoking during pregnancy (coded 1 if smoked, 0 otherwise).
♦ **UI**	- Uterine irritability (coded 1 if present, 0 otherwise).

As you might guess we are going to explore the possibility that characteristics of the mother, including demographics, health status, and the mother's behavior, might influence the probability of a low birth weight baby.

Begin by looking for the HOSLEM.CSV data file that should be located in your Sample Data folder. You may consult the generic parts of this manual for a detailed description on how to open datasets for modeling and related simple steps mentioned below:

♦ Open the HOSLEM.CSV dataset located in the Sample Data folder.


C:\SPM\Salford Predictive Modeler 8.0\Sample Data\Hoslem.csv

File Name: Hoslem.csv

Location: C:\SPM\Salford Predictive Modeler 8.0\Sample Data\

Modified: Saturday, April 25, 2015, 2:53:00 AM

Variables: AGE, BWT, CASEID, FTV, HT, ID, LOW, LWD, LWT, PTD, PTL, RACE, RACE1, SMOKE, UI

Data	
Records:	189
Variables:	15
Character:	0
Numeric:	15

Sort: Alphabetically

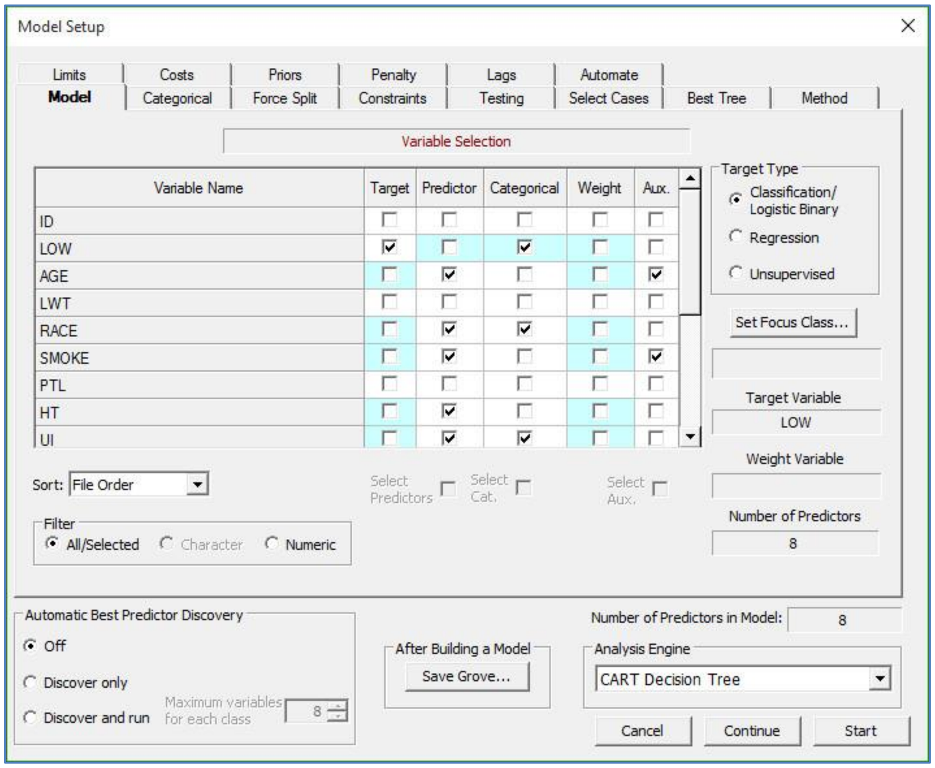
- Press the **[Model…]** button in the activity window (unless the window is suppressed).

You should now have the **Model Setup** window opened. In what follows, we describe the purpose of all individual tabs.

Model Tab

This generic tab is used to set up analysis type, as well as select target and predictors:

- Make sure that the **Analysis Engine** selection box has **CART**.
- Make sure that the **Target Type** is set to **Classification/Logistic Binary**.
- Change the **Sort:** selection box to **File Order**.
- Select LOW as the target variable.
- Select AGE, RACE, SMOKE, HT, UI, FTV, PTD, and LWD as predictors.
- Specify RACE, UI, and FTV as categorical predictors.
- Specify AGE, SMOKE, and BWT as auxiliary variables (see below).



Variable Name	Target	Predictor	Categorical	Weight	Aux.
ID					
LOW	✓		✓		
AGE		✓			✓
LWT					
RACE		✓	✓		
SMOKE		✓			✓
PTL					
HT		✓			
UI		✓	✓		

Sort: File Order

Filter: [✓ All/Selected] Character Numeric

Target Type: [✓ Classification/ Logistic Binary] Regression Unsupervised

Target Variable: LOW

Weight Variable:

Number of Predictors: 8

Automatic Best Predictor Discovery: [✓ Off] Discover only Discover and run

Maximum variables for each class: 8

Number of Predictors in Model: 8

Analysis Engine: CART Decision Tree

Target Column

This is where you specify the target variable for the analysis.

```
MODEL <variable>
Example> MODEL LOW
Model (target and predictors) reset: LOW
```

Predictor Column

This is where you specify the predictors to be used.

```
KEEP <variable>, <variable>, …

Example> KEEP AGE, RACE, SMOKE, HT, UI, FTV, PTD, LWD
```

✓ When the **Target Type** is set to **Classification/Logistic Binary**, the target variable will be automatically defined as categorical and appear with the corresponding checkmark at later invocations of the **Model Setup**. Similarly, the **Regression** radio button will automatically cancel the categorical status of the target variable. In other words, the specified **Target Type** determines whether the target is treated as categorical or continuous.

Categorical Column

This is where you specify which predictors are categorical (nominal or discrete).

```
CATEGORY <variable>, <variable>, …
Example> CATEGORY FTV, LOW, RACE, UI
```

CART supports "high-level categorical variables" through its proprietary algorithms that quickly determine effective splits in spite of the daunting combinatorics of many-valued predictors. This feature is increasingly important in the presence of character predictors, which in "real world" datasets often have hundreds or even thousands of levels. When forming a categorical splitter, traditional CART searches all possible combinations of levels, an approach in which time increases geometrically with the number of levels. In contrast, CART's high-level categorical algorithm increases linearly with time, yet yields the optimal split in most situations.

Character variables are implicitly treated as categorical (discrete), so there is no need to "declare" them categorical. There is no internal limit on the length of character data values (strings). You are limited in this respect only by the data format you choose (e.g., SAS, text, Excel, etc.).

✓ Character variables (marked by “$” at the end of variable name) will always be treated as categorical and cannot be unchecked.

✓ Occasionally columns stored in an Excel spreadsheet will be tagged as “Character” even though the values in the column are intended to be numeric. If this occurs with your data, refer to the READING DATA section to remedy this problem.

Depending whether a variable is declared as continuous or categorical, CART will search for different types of splits. Each takes on a unique form.

Continuous splits will always use the following form.

A case goes left if `[split-variable] <= [split-value]`

A node is partitioned into two children such that the left child receives all the cases with the lower values of the *[split-variable].*

Categorical splits will always use the following form.

A case goes left if *[split-variable]* = *[level_i OR …level_j OR … level_k]*

In other words, we simply list the values of the splitter that go left (and all other values go right).

- One should exercise caution when declaring continuous variables as categorical because a large number of distinct levels may result in significant increases in running times and memory consumption.

- Any categorical predictor with a large number of levels can create problems for the model. While there is no hard and fast rule, once a categorical predictor exceeds about 50 levels there are likely to be compelling reasons to try to combine levels until it meets this limit. We show how CART can conveniently do this for you later in the manual (see Introduction to Data Binning).

Weight Column

In addition to selecting target and predictor variables, the **Model** tab allows you to specify a case-weighting variable.

```
WEIGHT <variable>
```

Case weights, which are stored in a variable on the dataset, typically vary from observation to observation. An observation's case weight can, in some sense, be thought of as a repetition factor. A missing, negative or zero case weight causes the observation to be deleted, just as if the target variable were missing. Case weights may take on fractional values (e.g., 1.5, 27.75, 0.529, 13.001) or whole numbers (e.g., 1, 2, 10, 100).

To select a variable as the case weight, simply put a checkmark against that variable in the **Weight** column.

✓ Case weights do not affect linear combinations in CART Basic, but are otherwise used throughout CART. CART Pro, ProEX, and Ultra include a new linear combination facility that recognizes case weights.

✓ If you are using a test sample contained in a separate dataset, the case weight variable must exist and have the same name in that dataset as in your main (learn sample) dataset.

Aux. Column

This is where you can mark Auxiliary variables.

```
AUXILIARY <variable>, <variable>, …
Example> AUXILIARY AGE, SMOKE, BWT
```

Auxiliary variables are variables that are tracked throughout the CART tree but are not necessarily used as predictors. By marking a variable as Auxiliary, you indicate that you want to be able to retrieve basic summary statistics for such variables in any node in the CART tree. In our modeling run based on the HOSLEM.CSV data, we mark AGE, SMOKE and BWT as auxiliary.

Later in this guide, we discuss how to view auxiliary variable distributions on a node-by-node basis.

Setting Focus Class

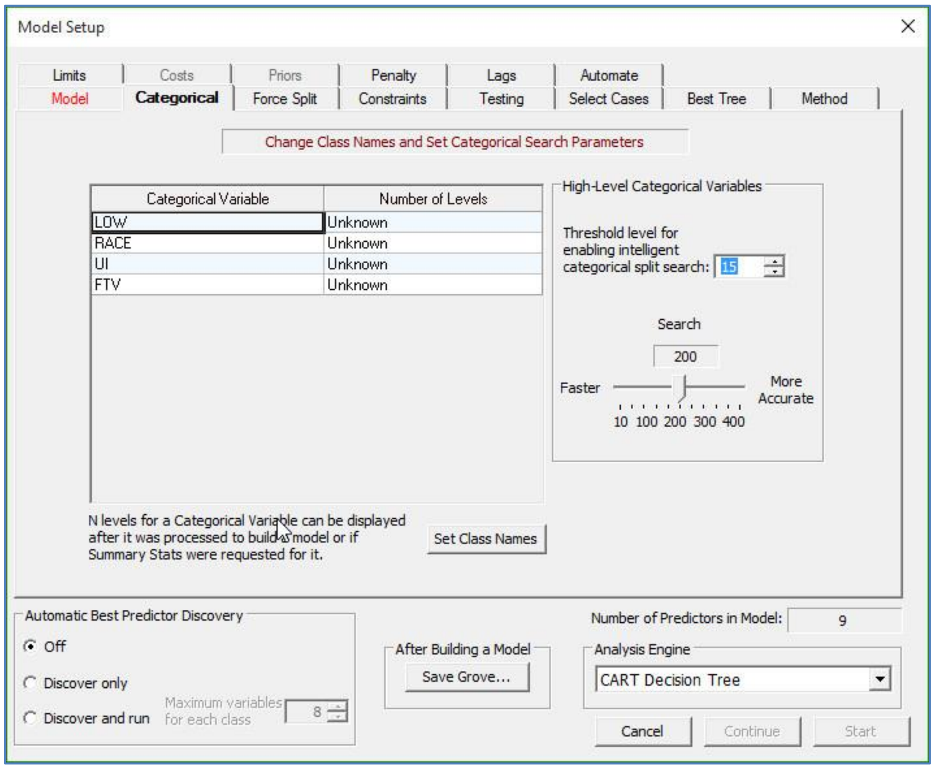
In classification runs some of the reports generated by CART (gains, prediction success, color-coding, etc.) have one target class in focus. By default, CART will put the **first** class it finds in the dataset *in focus*. A user can overwrite this by pressing the **[Set Focus Class…]** button and selecting the desired class.

Sorting Variable List

The variable list can be sorted either in physical order or alphabetically by changing the **Sort:** control box. Depending on the dataset, one of those modes will be preferable, which is usually helpful when dealing with large variable lists.

Categorical Tab

The **Categorical** tab allows you to manage text labels for categorical predictors and it also offers controls related to how we search for splitters on high-level categorical predictors. The splitter controls are discussed later as this is a rather technical topic and the defaults work well.



Categorical Variable	Number of Levels
LOW	Unknown
RACE	Unknown
UI	Unknown
FTV	Unknown

Threshold level for enabling intelligent categorical split search: 15

Search: 200

N levels for a Categorical Variable can be displayed after it was processed to build a model or if Summary Stats were requested for it.

Automatic Best Predictor Discovery: [✓ Off] Discover only / Discover and run — Maximum variables for each class: 8

Number of Predictors in Model: 9

Analysis Engine: CART Decision Tree

Setting Class Names

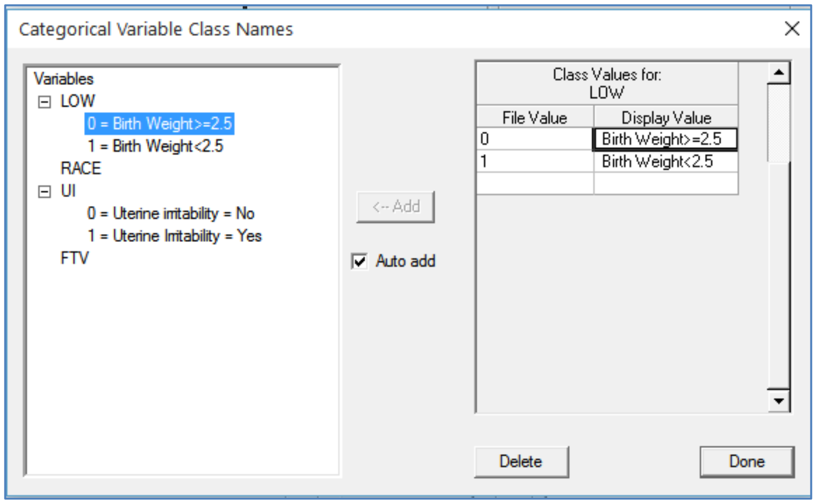
```
CLASS <variable> <value1> = "<label1>", <value2> = "<label2>", …
```

Class names are defined in the **Categorical** tab. Press **[Set Class Names]** to get started. In the left panel, select a variable for which labels are to be defined. If any class labels are currently defined for this variable, they will appear in the left panel and, if the variable is selected, in the right panel as well (where they may be altered or deleted). To enter a new class name in the right panel for the selected variable, define a numeric value (one that will appear in your data) in the "**Level**" column and its corresponding text label in the "**Class Values for:**" column. Repeat for as many class names as necessary for the selected variable.

You need not define labels for all levels of a categorical variable. A numeric level, which does not have a class name, will appear in the CART output as it always has, as a number. Also, it is acceptable to define labels for levels that do not occur in your data. This allows you to define a broad range of class names for a variable, all of which will be stored in a command script (.CMD file), but only those actually appearing in the data you are using will be used.

In a classification tree, class names have the greatest use for categorical numeric target variables (i.e., in a classification tree). For example, for a four-level target variable PARTY, classes such as "Independent," "Liberal," "Conservative," and "Green" could appear in CART reports and the navigator rather than levels "1", "2", "3", and "4." In general, only the first 32 characters of a class name are used, and some text reports use fewer due to space limitations.

In our example we specify the following class names for the target variable LOW and predictor UI. These labels then will appear in the tree diagrams, the CART text output, and most displays. The setup dialog appears as follows.


Categorical Variable Class Names
Variables
LOW
0 = Birth Weight>=2.5
1 = Birth Weight<2.5
RACE
UI
0 = Uterine irritability = No
1 = Uterine Irritability = Yes
FTV
<-- Add
Auto add
Delete
Done

Class Values for: LOW

File Value	Display Value
0	Birth Weight>=2.5
1	Birth Weight<2.5

GUI CART users who use class names extensively should consider defining them with commands in a command file and submitting the command file from the CART notepad once the dataset has been opened. The **CLASS** commands must be given before the model is built.

- ✓ If you use the GUI to define class names and wish to reuse the class names in a future session, save the command log before exiting CART. Cut and paste the **CLASS** commands appearing in the command log into a new command file.
- ✓ You can add labels to the target variable AFTER a tree is grown, but these will appear only in the navigator window (not in the text reports). Activate a navigator window, pull down the **View** menu and select the **Assign Class Names…** menu item.

High-Level Categorical Predictors

```
BOPTIONS NCLASSES = <n>
BOPTIONS HLC = <n>, <n>
```

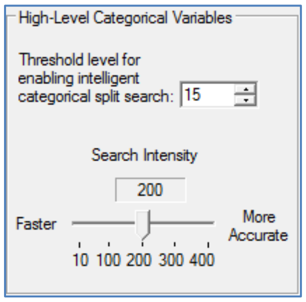
We take great pride in noting that CART is capable of handling categorical predictors with thousands of levels (given sufficient RAM workspace). However, using such predictors in their raw form is generally not a good idea. Rather, it is usually advisable to reduce the number of levels by grouping or aggregating levels, as this will likely yield more reliable predictive models. It is also advisable to impose the HLC penalty on such variables (from the **Model Setup—Penalty** tab). These topics are discussed at greater length later in the manual. In this section we discuss the simple mechanics for handling any HLC predictors you have decided to use.

For the binary target, high-level categorical predictors pose no special computational problem as exact short cut solutions are available and the processing time is minimal no matter how many levels there are.

For the multi-class target variable (more than two classes), we know of no similar exact short cut methods, although research has led to substantial acceleration. HLCs present a computational challenge because of the sheer number of possible ways to split the data in a node. The number of distinct splits that can be generated using a categorical predictor with K levels is 2^{K-1} -1. If K=4, for example, the number of candidate splits is 7; if K=11, the total is 1,023; if K=21, the number is over one million; and if K=35, the number of splits is more than 34 billion! Naïve processing of such problems could take days, weeks, months, or even years to complete!

To deal more efficiently with high-level categorical (HLC) predictors, CART has an intelligent search procedure that efficiently approximates the exhaustive split search procedure normally used. The HLC procedure can radically reduce the number of splits actually tested and still find a near optimal split for a high-level categorical.

The control option for high-level categorical predictors appears in the **Categorical** tab as follows.


High-Level Categorical Variables

Threshold level for enabling intelligent categorical split search: 15

Search Intensity

200

The settings above indicate that for categorical predictors with 15 or fewer levels we search all possible splits and are guaranteed to find the overall best partition. For predictors with more than 15 levels we use intelligent shortcuts that will find very good partitions but may not find the absolute overall best. The threshold level of 15 for enabling the short-cut intelligent categorical split searches can be increased or decreased in the Categorical dialog. In the short cut method we conduct "local" searches that are fast but explore only a limited range of possible splits. The default setting for the number of local splits to search is around 200. To change this default and thus search more or less intensively, increase or decrease the search intensity gauge. Our experiments suggest that 200 is a good number to use and that little can be gained by pushing this above 400. As indicated in the Categorical dialog, a higher number leads to more intensive and longer searching whereas a lower number leads to faster, less thorough searching. If you insist on more aggressive searching you should go to the command line.

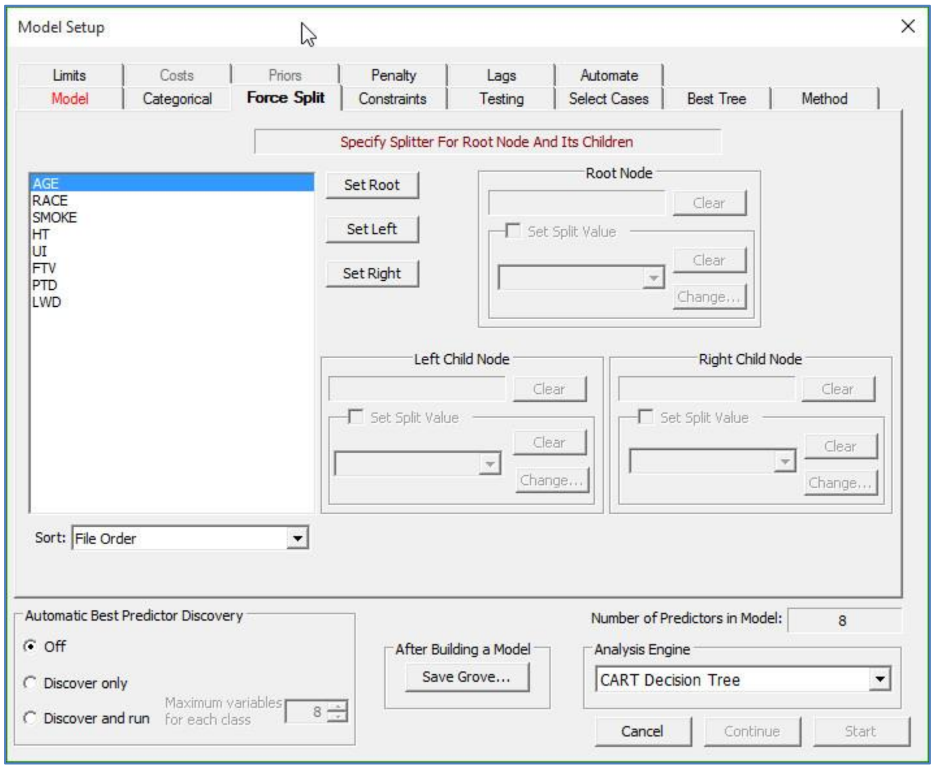
✓ These controls are only relevant if your target variable has more than two levels. For the two-level binary target (the YES/NO problem), CART has special shortcuts that always work.

💣 There are actually disadvantages to searching too aggressively for the best HLC splitter, as such searches increase the likelihood of overfitting the model to the training data.

Force Split tab

```
FORCE ROOT ON <categorical variable> AT <value1>, <value2>, …
FORCE LEFT ON <categorical variable> AT <value1>, <value2>, …
FORCE RIGHT ON <continuous variable> AT <value>
```

Sometimes, there is a need to override splits that were automatically found by CART, or simply force alternative splits into a tree. The Force Split tab allows you to define your own split variable and/or split value at the root node and also any of the child nodes of the root node during the model setup.


Model Setup

Specify Splitter For Root Node And Its Children

AGE
RACE
SMOKE
HT
UI
FTV
PTD
LWD

Sort: File Order

Automatic Best Predictor Discovery: [✓ Off] / Discover only / Discover and run

Maximum variables for each class: 8

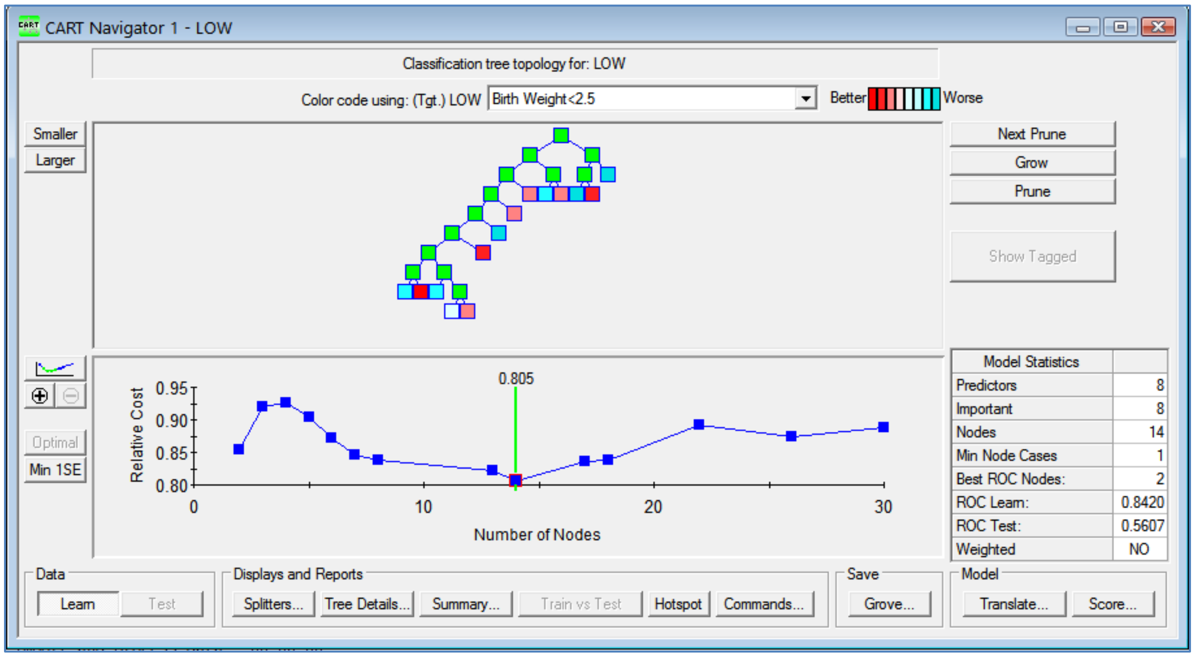
Number of Predictors in Model: 8

Analysis Engine: CART Decision Tree

✓ Command-line offer more flexible FORCE facility which, in conjunction with the **Force Commands** pop-up menu available in the **Navigator** window, allows you to change any split anywhere within the current tree. This is described later in this manual.

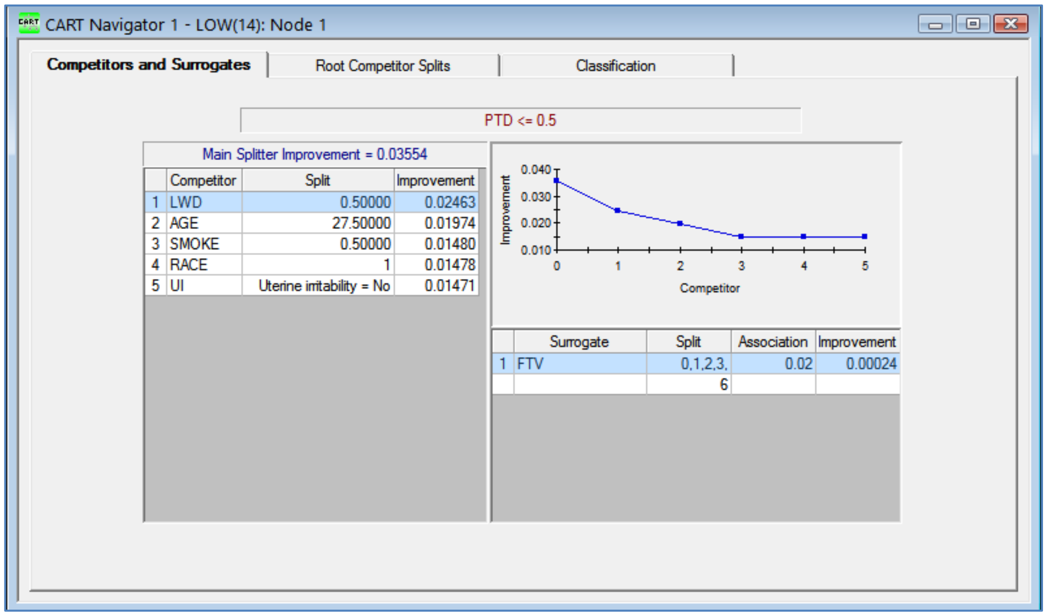
Specifying the Root Node Splitter

Press the **[Start]** button in the **Model Setup** window to build a CART tree based on the current settings.



Model Statistics	
Predictors	8
Important	8
Nodes	14
Min Node Cases	1
Best ROC Nodes:	2
ROC Learn:	0.8420
ROC Test:	0.5607
Weighted	NO

Now click on the root node to see detailed split information



Main Splitter Improvement = 0.03554

	Competitor	Split	Improvement
1	LWD	0.50000	0.02463
2	AGE	27.50000	0.01974
3	SMOKE	0.50000	0.01480
4	RACE	1	0.01478
5	UI	Uterine irritability = No	0.01471

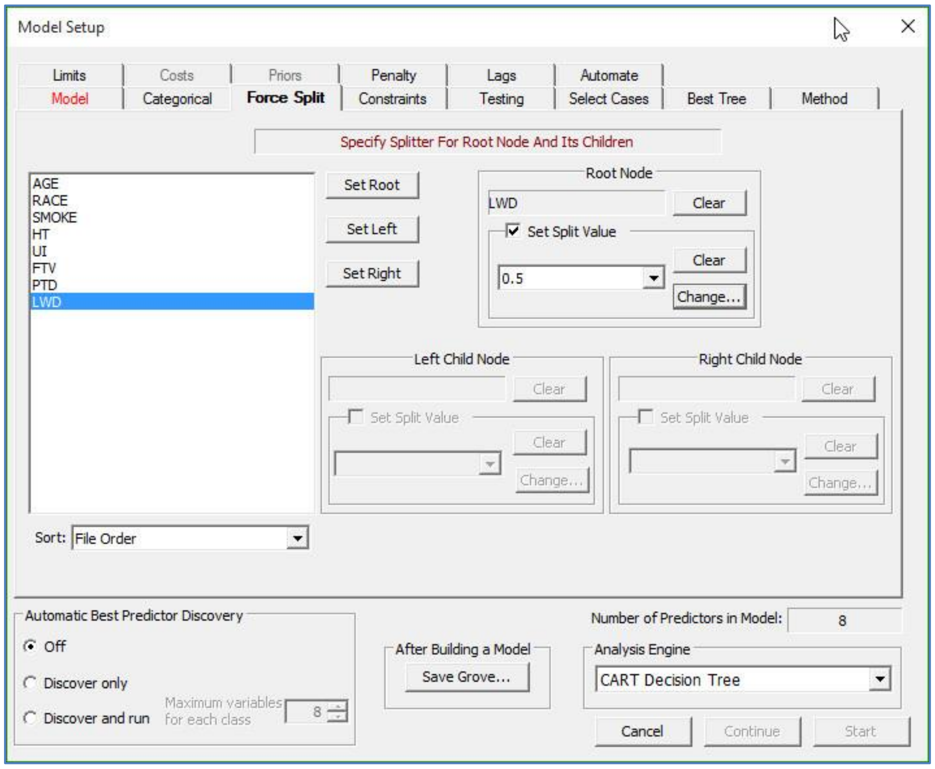
	Surrogate	Split	Association	Improvement
1	FTV	0,1,2,3, 6	0.02	0.00024

CART decided to use PTD as the main splitter; however, LWD while having somewhat smaller improvement is a good competitor. Suppose you would like to override CART's decision and force LWD as the root node splitter.

Here is how you can accomplish this:

- Go back to the **Model Setup** window and click on the **Force Split** tab.
- Select the LWD variable in the predictor list on the left.
- Press the **[Set Root]** button; this will place this variable in the **Root Node** group.
- Check the **Set Split Value** checkbox, and then press the **[Change…]** button.
- In the resulting dialog window, enter the continuous split value of 0.5 and click **[OK]**.

The Force Split tab now should look like this:



Specify Splitter For Root Node And Its Children

AGE
RACE
SMOKE
HT
UI
FTV
PTD
LWD

Sort: File Order

Root Node: LWD
[✓] Set Split Value: 0.5

Left Child Node
[ ] Set Split Value

Right Child Node
[ ] Set Split Value

Automatic Best Predictor Discovery: [✓ Off] Discover only Discover and run
Maximum variables for each class: 8

After Building a Model

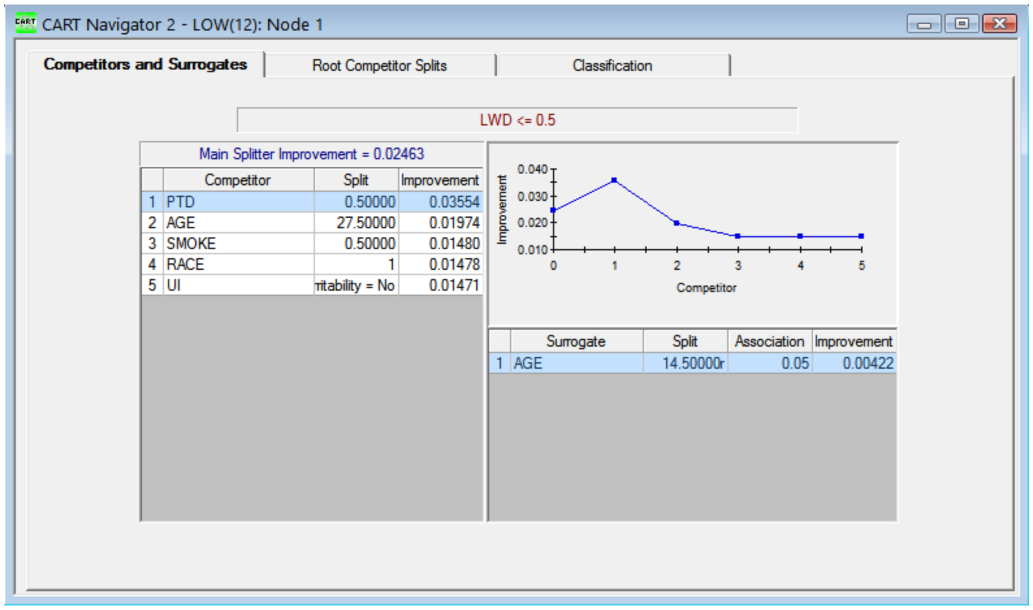
Number of Predictors in Model: 8

Analysis Engine: CART Decision Tree

- Be careful when you set the split value, trying to set a wrong value may result to a degenerate split (all cases go on one side of the split) and will produce a warning.

✓ The **Set Split Value** step above can be skipped; this will only force the splitter variable while letting CART automatically find the split value for you.

Press the **[Start]** button and confirm that the root node uses the enforced split by clicking on the root node in the new **Navigator** window.



Main Splitter Improvement = 0.02463

	Competitor	Split	Improvement
1	PTD	0.50000	0.03554
2	AGE	27.50000	0.01974
3	SMOKE	0.50000	0.01480
4	RACE	1	0.01478
5	UI	ritability = No	0.01471

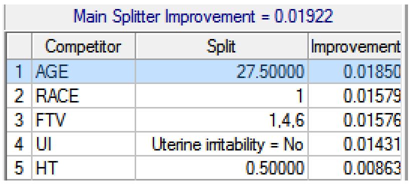
	Surrogate	Split	Association	Improvement
1	AGE	14.50000r	0.05	0.00422

✓ The PTD now moved to the top competitor position, observe that in terms of improvement it is still superior to the enforced split.

Specifying the Left/Right Child Node Splitter

Using the same root node force split variable and value we now demonstrate how to specify the right/left child node splits. Like the root node split, the user can specify not only the variable, but also a split value.

First, click on the left child of the root node and observe the table of competitors CART Navigator 1:

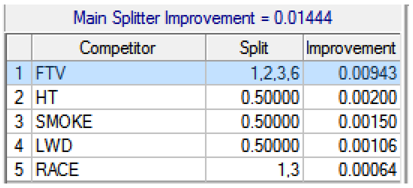


Main Splitter Improvement = 0.01922

	Competitor	Split	Improvement
1	AGE	27.50000	0.01850
2	RACE	1	0.01579
3	FTV	1,4,6	0.01576
4	UI	Uterine irritability = No	0.01431
5	HT	0.50000	0.00863

We will try to force the split based on the AGE variable.

Now click on the right child and observe the table of competitors in CART Navigator 1:



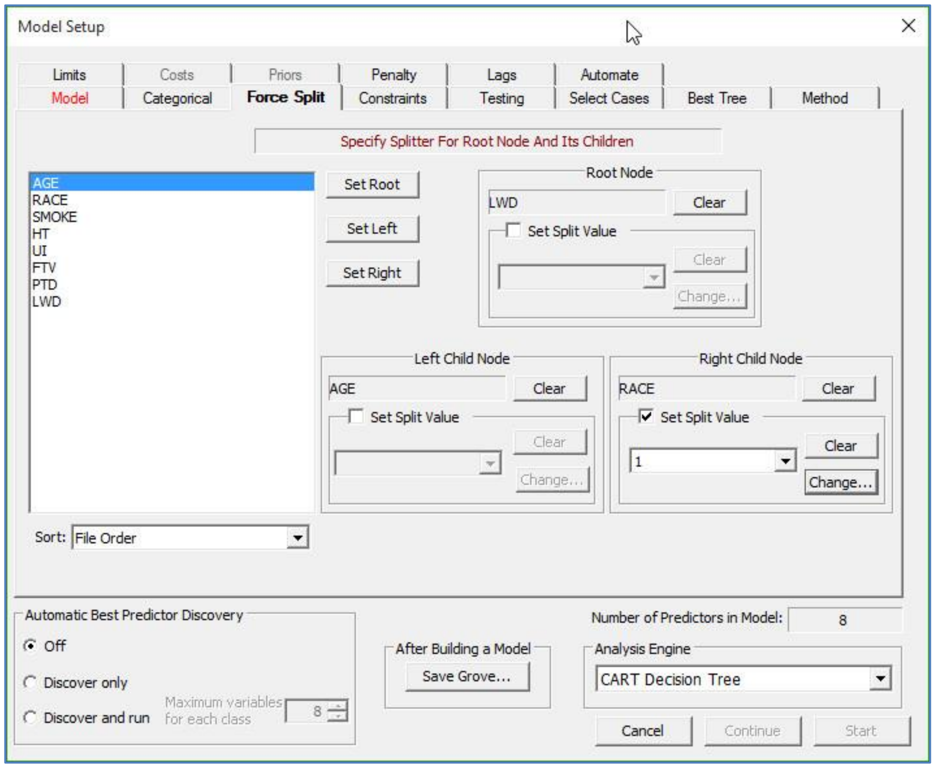
Main Splitter Improvement = 0.01444

	Competitor	Split	Improvement
1	FTV	1,2,3,6	0.00943
2	HT	0.50000	0.00200
3	SMOKE	0.50000	0.00150
4	LWD	0.50000	0.00106
5	RACE	1,3	0.00064

Here, we will enforce categorical split based on RACE values 1 and 2 (in place of the suggested 1 and 3).

Below is the sequence of steps to force the above conditions:

- Go back to the **Model Setup** window and click on the **Force Split** tab.
- Select the LWD variable in the predictor list on the left.
- Press the **[Set Root]** button; this will place LWD in the **Root Node** group.
- Select the AGE variable in the predictor list on the left.
- Press the **[Set Left]** button; this will place AGE in the **Left Child Node** group.
- Select the RACE variable in the predictor list on the left.
- Press the **[Set Right]** button; this will place RACE in the Right **Child Node** group.
- Check the **Set Split Value** checkbox, and then press the **[Change…]** button.
- In the resulting dialog window use the control buttons to have values 1 and 2 go to the left side and value 3 go to the right side, click **[OK]**.



Specify Splitter For Root Node And Its Children

AGE
RACE
SMOKE
HT
UI
FTV
PTD
LWD

Root Node: LWD
☐ Set Split Value

Left Child Node: AGE
☐ Set Split Value

Right Child Node: RACE
☑ Set Split Value: 1

Sort: File Order

Automatic Best Predictor Discovery: [✓ Off] Discover only / Discover and run
Maximum variables for each class: 8

After Building a Model

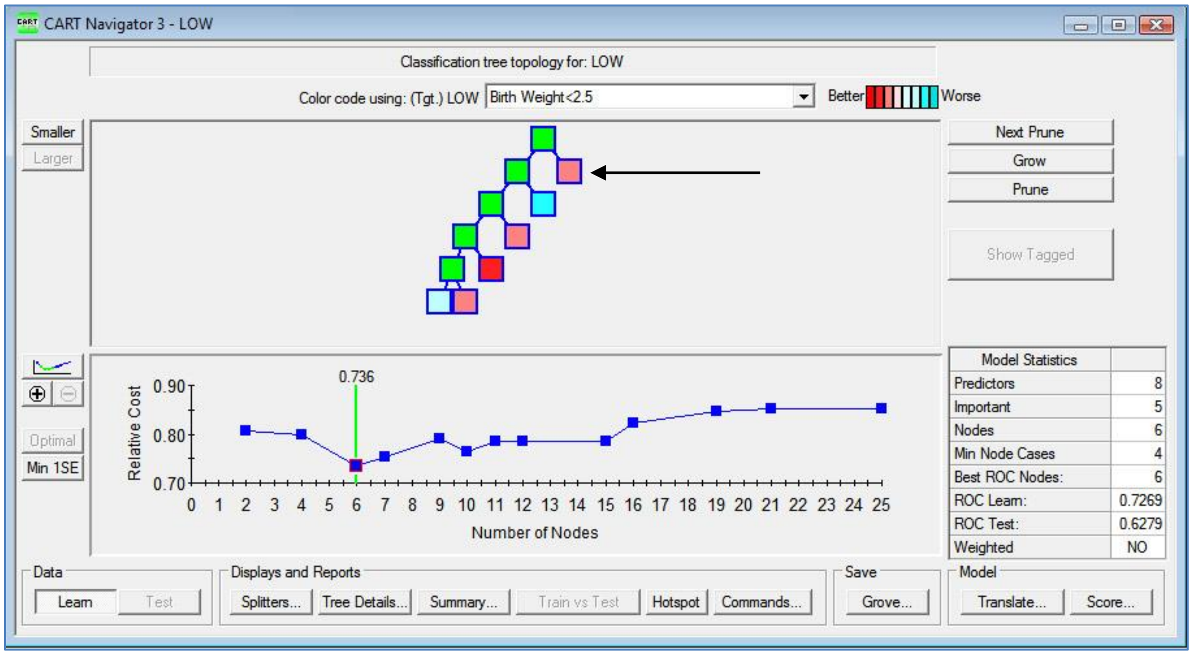
Number of Predictors in Model: 8

Analysis Engine: CART Decision Tree

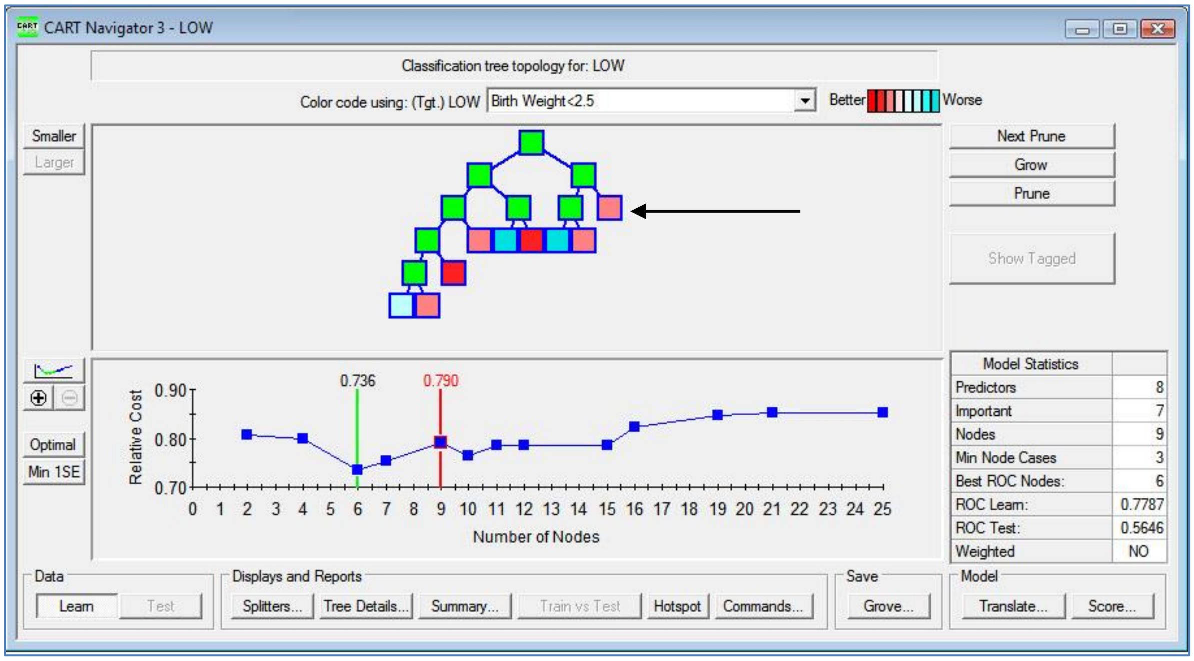
✓ We had to force the root node split again.

✓ We did not specify split values for the LWD and AGE to let CART do the search for us.

In the resulting Navigator file, CART declared the right child node as terminal, this may happen when your choice of the split produces inferior data partition which fails to validate on the test sample. However, you may still see the split by picking a larger tree, in our case 9-node tree:

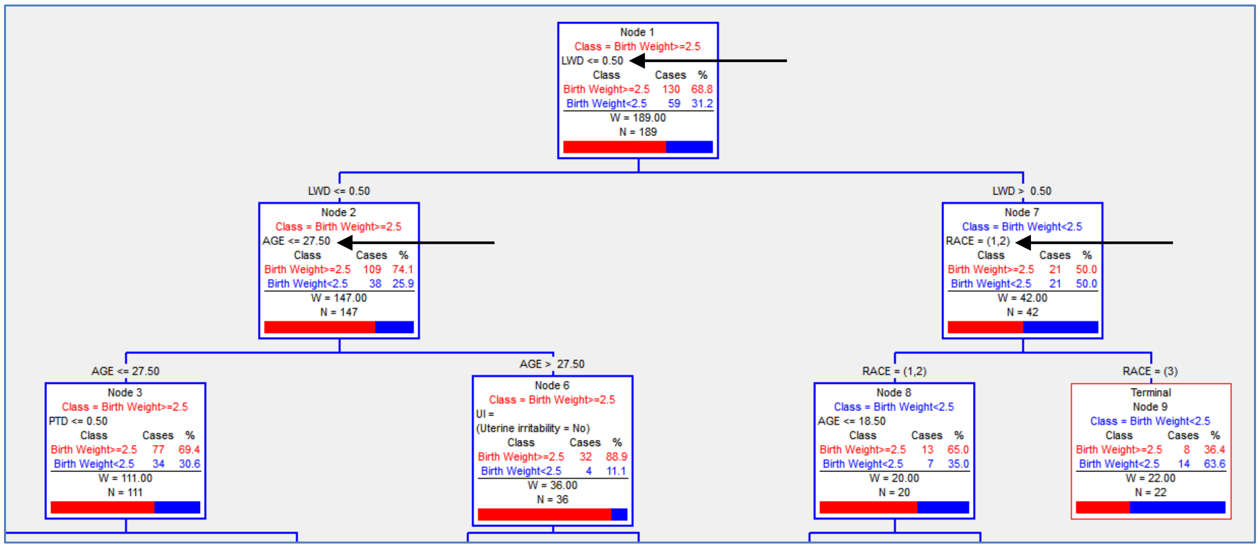


Model Statistics	
Predictors	8
Important	5
Nodes	6
Min Node Cases	4
Best ROC Nodes:	6
ROC Learn:	0.7269
ROC Test:	0.6279
Weighted	NO



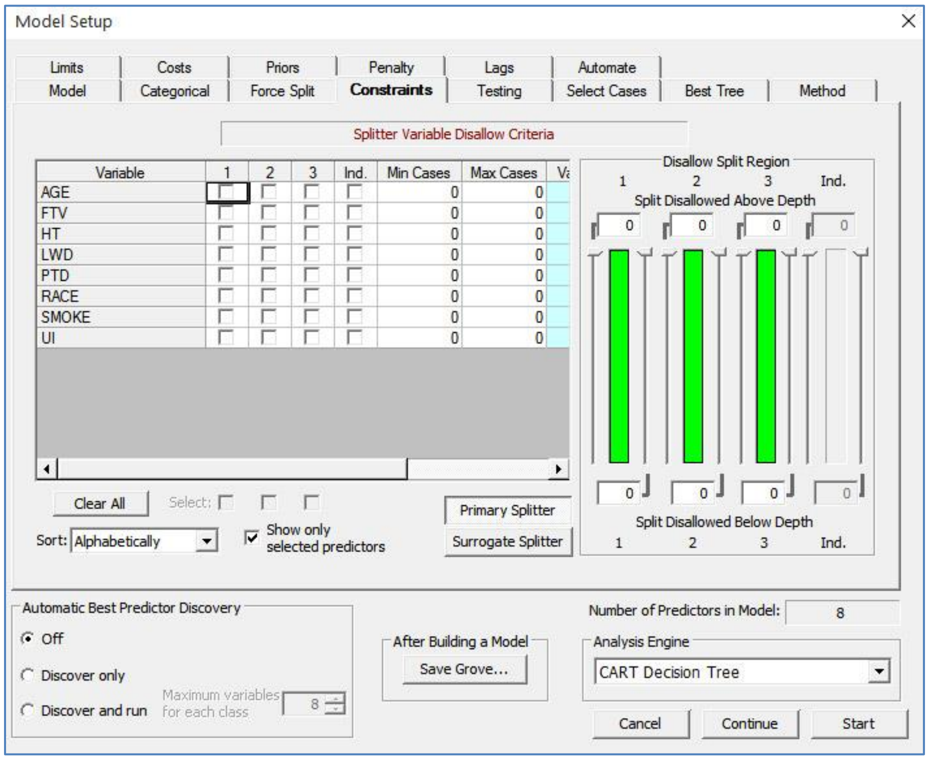
Model Statistics	
Predictors	8
Important	7
Nodes	9
Min Node Cases	3
Best ROC Nodes:	6
ROC Learn:	0.7787
ROC Test:	0.5646
Weighted	NO

Press the **[Tree Details]** button to confirm that all three splits have been enforced:



Constraints tab

This tab allows enforcing additional structure on trees. It is described later in this guide.


Model Setup

Tabs: Limits | Costs | Priors | Penalty | Lags | Automate | Model | Categorical | Force Split | **Constraints** | Testing | Select Cases | Best Tree | Method

Splitter Variable Disallow Criteria

Variable	1	2	3	Ind.	Min Cases	Max Cases
AGE	☐	☐	☐	☐	0	0
FTV	☐	☐	☐	☐	0	0
HT	☐	☐	☐	☐	0	0
LWD	☐	☐	☐	☐	0	0
PTD	☐	☐	☐	☐	0	0
RACE	☐	☐	☐	☐	0	0
SMOKE	☐	☐	☐	☐	0	0
UI	☐	☐	☐	☐	0	0

Clear All   Select: ☐ ☐ ☐

Sort: Alphabetically   ☑ Show only selected predictors

Primary Splitter   Surrogate Splitter

Disallow Split Region

	1	2	3	Ind.
Split Disallowed Above Depth	0	0	0	0
Split Disallowed Below Depth	0	0	0	0

Automatic Best Predictor Discovery
(● Off) ○ Discover only ○ Discover and run
Maximum variables for each class: 8

After Building a Model: Save Grove...

Number of Predictors in Model: 8

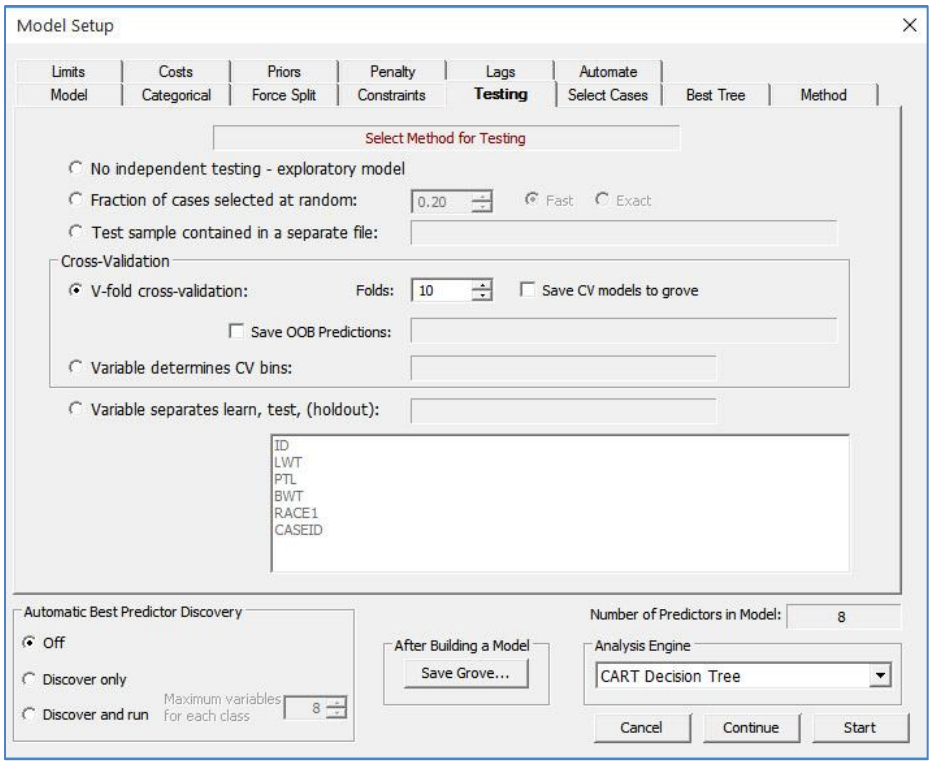
Analysis Engine: CART Decision Tree

Cancel   Continue   Start

Testing tab

PARTITION

This generic tab offers a number of options to partition your data into learn, test, and validate samples. It also includes cross-validation as an alternative. The tab is described in detail in the SPM® Infrastructure guide.


Model Setup

Tabs: Limits | Costs | Priors | Penalty | Lags | Automate | Model | Categorical | Force Split | Constraints | Testing | Select Cases | Best Tree | Method

Select Method for Testing

- No independent testing - exploratory model
- Fraction of cases selected at random: 0.20 [✓ Fast] Exact
- Test sample contained in a separate file:

Cross-Validation

- [✓ V-fold cross-validation:] Folds: 10 ☐ Save CV models to grove
- ☐ Save OOB Predictions:
- Variable determines CV bins:

- Variable separates learn, test, (holdout):

ID
LWT
PTL
BWT
RACE1
CASEID

Automatic Best Predictor Discovery: [✓ Off] Discover only, Discover and run; Maximum variables for each class: 8

After Building a Model: Save Grove...

Number of Predictors in Model: 8

Analysis Engine: CART Decision Tree

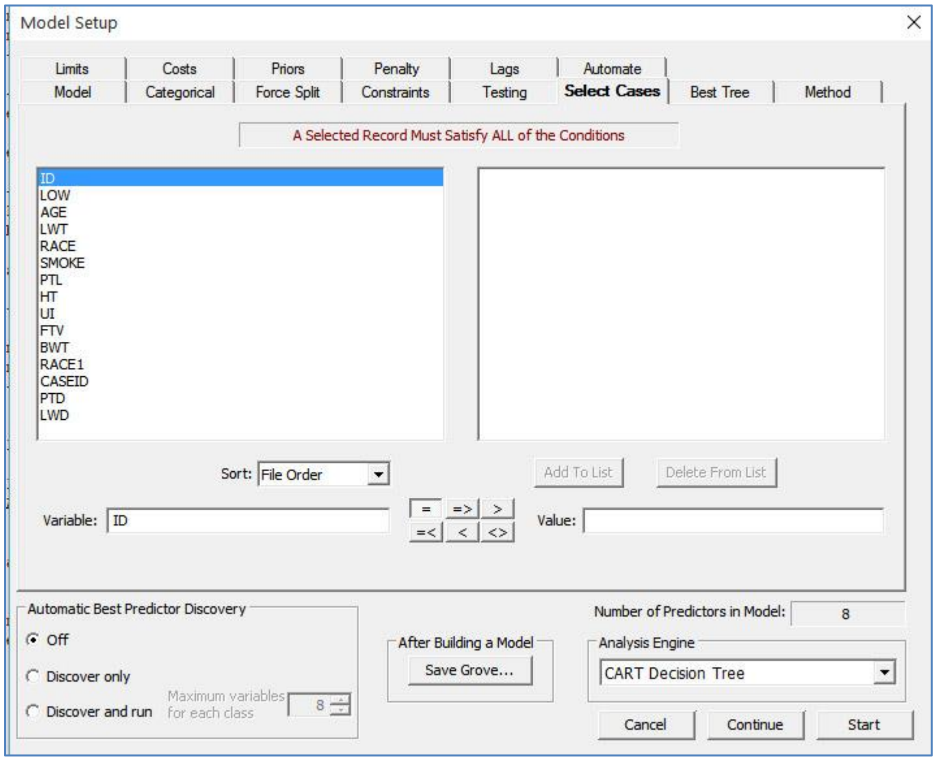
There are a number of different test method available. They include the include the following. These test methods are described in detail in the SPM Infrastructure guide.

- **No independent testing** – Results in an exploratory model with no independent testing. Resubstitution is used instead to approximate a test sample.
- **Fraction of cases selected at random.** Specifies a fraction of cases selected at random for testing and, optionally, validation.
- **Test sample contained in a separate file.** The test sample is contained in a separate dataset.
- **Cross Validation—v-fold cross validation.** Request number of folds, for example, 2 or 5 or 20
- **Cross Validation—Variable determines CV bins.** Specify variable which assigns records to cross-validation bins.
- **Variable separates learn, test, (holdout).** Specify variable which separates learn and test samples.

Select Cases tab

SELECT

This generic tab allows you to specify various selection criteria for building a tree based on a subset of cases. Details of this tab are described in the generic SPM Infrastructure guide.



A Selected Record Must Satisfy ALL of the Conditions

ID
LOW
AGE
LWT
RACE
SMOKE
PTL
HT
UI
FTV
BWT
RACE1
CASEID
PTD
LWD

Sort: File Order

Variable: ID

Value:

Automatic Best Predictor Discovery
[✓ Off]
Discover only
Discover and run
Maximum variables for each class: 8

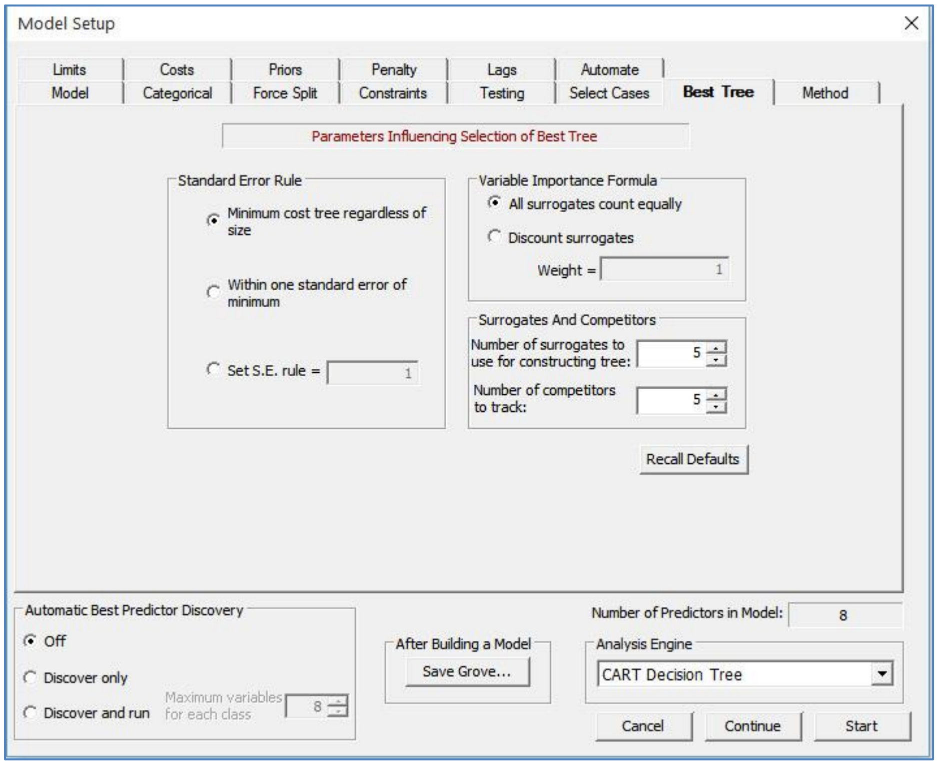
After Building a Model

Number of Predictors in Model: 8

Analysis Engine: CART Decision Tree

Best Tree tab

The **Best Tree** tab is largely of historical interest as it dates to a time when CART would produce a single tree in any run. Specifying how you wanted that single tree to be selected was an important part of the model setup procedure. In today's CART you have full access to every tree in the pruned tree sequence and you can readily select trees of a size different than considered optimal. Nonetheless, when a tree is saved to a grove, CART always marks one of the pruned sub-trees as optimal. This tree will be selected by default for scoring. When you are working with many trees in a batch-scoring mode it will be most convenient if they are all marked with your preferred method for optimal tree selection.


Model Setup

Parameters Influencing Selection of Best Tree

Standard Error Rule: [✓ Minimum cost tree regardless of size] Within one standard error of minimum / Set S.E. rule = 1

Variable Importance Formula: [✓ All surrogates count equally] Discount surrogates, Weight = 1

Surrogates And Competitors: Number of surrogates to use for constructing tree: 5; Number of competitors to track: 5

Automatic Best Predictor Discovery: [✓ Off] Discover only / Discover and run; Maximum variables for each class 8

Number of Predictors in Model: 8

Analysis Engine: CART Decision Tree

Standard Error Rule

```
BOPTIONS SERULE = <value>
```

The standard error rule is the parameter influencing how CART selects the optimal tree following testing. The default setting is the minimum cost tree regardless of size, that is, the tree that is most accurate given the specified testing method. In certain situations, you may wish to trade a more accurate tree for a smaller tree by selecting the smallest tree within one standard error of the minimum cost tree or by setting the standard error parameter equal to any nonnegative value.

The primary use of the standard error rule is for processing many models in batch mode, or when you do not expect to be able to inspect each model individually. In such circumstances you will want to give some thought to specifying how the best model should be selected automatically. If you are examining each model visually on screen, then the best tree definition is not that important as you can readily select another tree interactively on screen.

Variable Importance Formula

```
BOPTIONS IMPORTANCE = <weight>
```

In the Best Tree dialog, you can also specify how variable importance scores are calculated and how many surrogates are used to construct the tree. Rather than counting all surrogates equally, the default calculation, you can fine-tune the variable importance calculation by specifying a weight to be used to discount the surrogates. Click on the **Discount surrogates** radio button and enter a value between 0 and 1 in the **Weight** text box.

Number of Surrogates

```
BOPTIONS SURROGATES = <n>
```

After CART has found the best splitter (primary splitter) for any node it proceeds to look for surrogate splitters: splitters that are similar to the primary splitter and can be used when the primary split variable is missing. You have control over the number of surrogates CART will search for; the default value is five. When there are many predictors with similar missing value patterns you might want to increase the default value.

You can increase or decrease the number of surrogates that CART searches for and saves by entering a value in the **Number of surrogates to use for constructing tree** box.

- ✓ The number of surrogates that can be found will depend on the specific circumstances of each node. In some cases there are no surrogates at all. Your specification sets limits on how many will be searched for but does not guarantee that this is the number that will actually be found.
- ✓ If all surrogates at a given node are missing or no surrogates were found for that particular node, a case that has a missing value for the primary splitter will be moved to the left or right child node according to a default rule discussed later.
- ✓ Because the number of surrogates you request *can* affect the details of the tree grown we have placed this control on the **Best Tree** tab. Usually the impact of this setting on a tree will be small, and it will only affect trees grown on data with missing values.

Method Tab

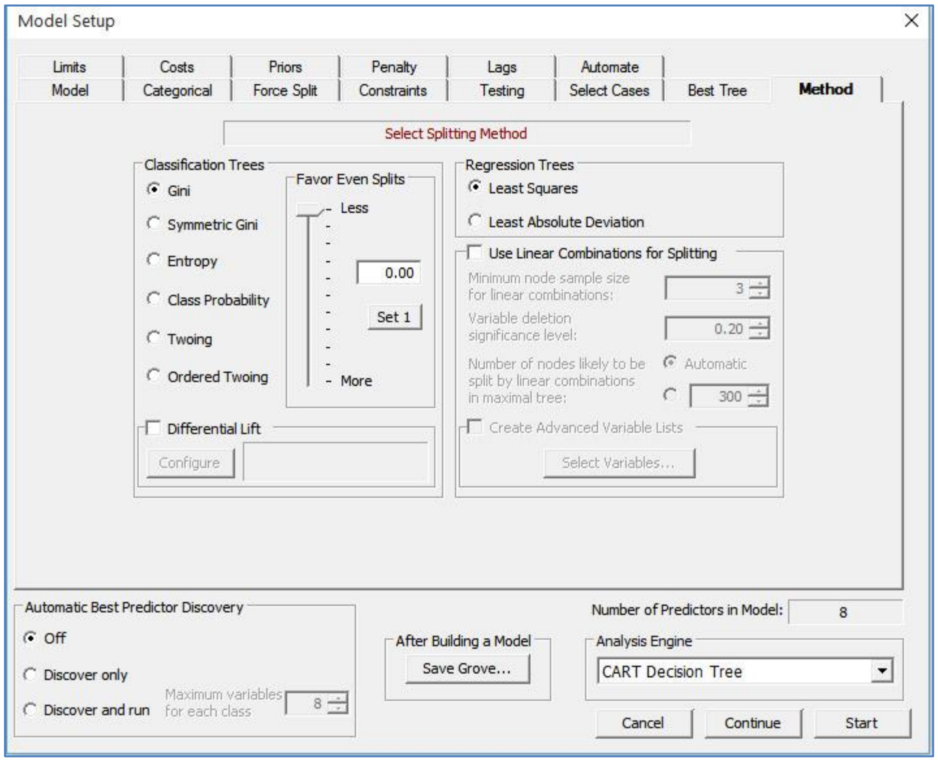
The **Method** tab allows you to specify the splitting rule used to construct the classification or regression tree and to turn on the linear combinations option.

Splitting Rules

```
METHOD [ GINI | SYMGINI | TWOING | ORDERED | PROB | ENTROPY ]
```

A splitting rule is a method and strategy for growing a tree. A good splitting rule is one that yields accurate trees! Since we often do not know which rule is best for a specific problem it is a good practice to experiment. For classification trees the default rule is the Gini. This rule was introduced in the CART monograph and was selected as the default because it generally works quite well. We have to agree with the original CART authors: working with many hundreds of data sets in widely different subject matters we have still seen the Gini rule to be an excellent choice. Further, there is often only a small difference in performance among the rules.

However, there will be circumstances in which the performance between, say, the Gini and Entropy is quite substantial, and we have worked on problems where using the Twoing rule has been the only way to obtain satisfactory results. Accuracy is not the only consideration people weigh when deciding on which model to use. Simplicity and comprehensibility can also be important. While the Gini might give you the most accurate tree, the Twoing rule might tell a more persuasive story or yield a smaller although slightly less accurate tree. Our advice is to not be shy about trying out the different rules and settings available on the **Method** tab.



Classification Trees: [✓ Gini] Symmetric Gini, Entropy, Class Probability, Twoing, Ordered Twoing

Favor Even Splits: Less – More: 0.00 Set 1

Differential Lift: ☐

Regression Trees: [✓ Least Squares] Least Absolute Deviation

☐ Use Linear Combinations for Splitting

Minimum node sample size for linear combinations: 3

Variable deletion significance level: 0.20

Number of nodes likely to be split by linear combinations in maximal tree: [✓ Automatic] 300

☐ Create Advanced Variable Lists

Automatic Best Predictor Discovery: [✓ Off] Discover only, Discover and run

Maximum variables for each class: 8

After Building a Model

Number of Predictors in Model: 8

Analysis Engine: CART Decision Tree

Here are some brief remarks on different splitting rules:

- **Gini** - This default rule often works well across a broad range of problems. Gini has a tendency to generate trees that include some rather small nodes highly concentrated with the class of interest. If you prefer more balanced trees you may prefer the results of the Twoing rule.
- **Symmetric Gini** - This is a special variant of the Gini rule designed specifically to work with a cost matrix. If you are not specifying different costs for different classification errors, the Gini and the Symmetric Gini are identical. See the discussions on cost matrices for more information.
- **Entropy** - The Entropy rule is one of the oldest decision tree splitting rules and has been very popular among computer scientists. Although it was the rule first used by CART authors Breiman, Friedman, Olshen, and Stone, they devote a section in the CART monograph to explaining why they switched to Gini. The simple answer is that the Entropy rule tends to produce even smaller terminal nodes (“end cut splits”) and is usually less accurate than Gini. In our experience about one problem in twenty is best handled by the Entropy rule.
- **Class Probability** - The probability tree is a form of the Gini tree that deserves much more attention than it has received. Probability trees tend to be larger than Gini trees and the predictions made in individual terminal nodes tend to be less reliable, but the details of the data structure that they reveal can be very valuable. When you are primarily interested in the performance of the top few nodes of a tree you should be looking at probability trees.
- **Twoing** - The major difference between the Twoing and other splitting rules is that Twoing tends to produce more balanced splits (in size). Twoing has a built-in penalty that makes it avoid unequal splits whereas other rules do not take split balance into account when searching for the best split. A Gini or Entropy tree could easily produce 90/10 splits whereas Twoing will tend to produce 50/50 splits. The differences between the Twoing and other rules become more evident when modeling multi-class targets with more than two levels. For example, if you were modeling segment membership for an eight-way segmentation, the Twoing and Gini rules would probably yield very different trees and performances.

- ♦ **Ordered Twoing** - The Ordered Twoing rule is useful when your target levels are ordered classes. For example, you might have customer satisfaction scores ranging from 1 to 5 and in your analysis you want to think of each score as a separate class rather than a simple score to be predicted by a regression. If you were to use the Gini rule CART would think of the numbers 1,2,3,4, and 5 as arbitrary labels without having any numeric significance. When you request Ordered Twoing you are telling CART that a "4" is more similar to a "5" than it is to a "1." You can think of Ordered Twoing as developing a model that is somewhere between a classification and a regression.

✓ Ordered Twoing works by making splits that tend to keep the different levels of the target together in a natural way. Thus, we would favor a split that put the "1" and "2" levels together on one side of the tree and we would want to avoid splits that placed the "1" and "5" levels together. Remember that the other splitting rules would not care at all which levels were grouped together because they ignore the numeric significance of the class label.

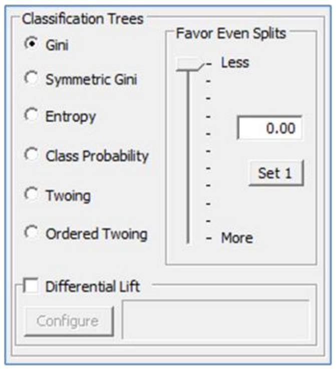
As always, you can never be sure which method will work best. We have seen naturally ordered targets that were better modeled with the Gini method. You will need to experiment.

✓ Ordered Twoing works best with targets with numeric levels. When a target is a character variable, the ordering conducted by CART might not be to your liking. See the command reference manual section on the DISCRETE command for more useful information.

Favor Even Splits

```
METHOD POWER=<x>
```

The "favor even splits" control is also on the Method tab and offers an important way to modify the action of the splitting rules. By default, the setting is 0, which indicates no bias in favor of even or uneven splits. In the display below we have set the splitting rule to Twoing and the "favor even splits" setting to 1.00.



The GUI limits your POWER setting to a maximum value of 2.00. This is to protect users from setting outlandish values. There are situations, however, in which a higher setting might be useful, and if so you will need to enter a command with a POWER setting of your choice. Using values greater than 5.00 is probably not helpful.

✓ On binary targets when both "Favor Even Splits" and the unit cost matrix are set to 0, Gini, Symmetric Gini, Twoing, and Ordered Twoing will produce near identical results.

Although we make recommendations below as to which splitting rule is best suited to which type of problem, it is good practice to always use several splitting rules and compare the results. You should experiment with several different splitting rules and should expect different results from each. As you work with different types of data and problems, you will begin to learn which splitting rules typically work best for specific problem types. Nevertheless, you should never rely on a single rule alone; experimentation is always wise.

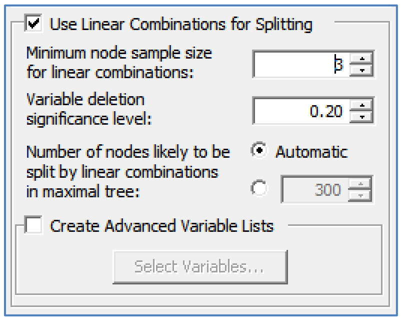
The following rules of thumb are based on our experience in the telecommunications, banking, and market research arenas, and may not apply to other subject areas. Nevertheless, they represent such a consistent set of empirical findings that we expect them to continue to hold in other domains and data sets more often than not.

- For a two-level dependent variable that can be predicted with a relative error of less than 0.50, the Gini splitting rule is typically best.
- For a two-level dependent variable that can be predicted with a relative error of only 0.80 or higher, Power-Modified Twoing tends to perform best.
- For target variables with four to nine levels, Twoing has a good chance of being the best splitting rule.
- For higher-level categorical dependent variables with 10 or more levels, either Twoing or Power-Modified Twoing is often considerably more accurate than Gini.

Linear Combination Splits

```
LINEAR N=<min_cases>, DELETE=<signif_level>, SPLITS=<max_splits>
```

To deal more effectively with linear structure, CART has an option that allows node splits to be made on linear combinations of non-categorical variables. This option is implemented by clicking on the **Use Linear Combinations for Splitting** check box on the **Method** tab as seen below.



[✓] Use Linear Combinations for Splitting
Minimum node sample size for linear combinations: 3
Variable deletion significance level: 0.20
Number of nodes likely to be split by linear combinations in maximal tree: [✓ Automatic] 300
[ ] Create Advanced Variable Lists

The **Minimum node sample size for linear combinations**, which can be changed from the default of three by clicking the up or down arrows, specifies the minimum number of cases required in a node for linear combination splits to be considered. Nodes smaller than the specified size will be split on single variables only.

✓ The default value is far too small for most practical applications. We would recommend using values such as 20, 50, 100 or more.

The **Variable deletion significance level**, set by default at 0.20, governs the backwards deletion of variables in the linear combination stepwise algorithm. Using a larger setting will typically select linear combinations involving fewer variables. We often raise this threshold to 0.40 for this purpose.

By default, CART automatically estimates the maximum number of linear combination splits in the maximal tree. The automatic estimate may be overridden to allocate more linear combination workspace. To do so, click on the **Number of nodes likely to be split by linear combinations in maximal tree** radio button and enter a positive value.

✓ CART will terminate the model-building process prematurely if it finds that it needs more linear combination splits than were actually reserved.

✓ Linear combination splits will be automatically turned off for all nodes that have any constant predictors (all values the same for all records). Thus, having a constant predictor in the training data will effectively turn off linear combinations for the entire tree.

LC Lists: Create Advanced Variable Lists

```
LCLIST <variable>, <variable>, …
```

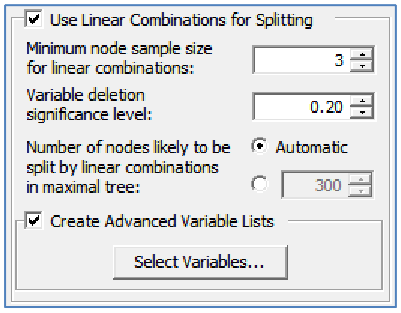
LC lists are a new addition to CART and can radically improve the predictive power and intuitive usefulness of your trees. In legacy CART if you request a search for linear combination splitters ALL the numeric variables in your predictor (KEEP) list are eligible to enter the linear combination (LC). In every node with a large enough sample size CART will look for the best possible LC regardless of which variables combine to produce that LC.

We have found it helpful to impose some structure on this process by allowing you to organize variables into groups from which LCs can be constructed. If you create such groups, then any LC must be constructed entirely from variables found in a single group. In a biomedical study you might consider grouping variables into demographics such as AGE and RACE, lifestyle or behavioral variables such as SMOKE and FTV, and medical history and medical condition variables such as UI, PTD, and LWT. Specifying LCLISTS in this way will limit any LCs constructed to those that can be created from the variables in a *single* list.

✓ Time series analysts can create one LCLIST for each predictor and its lagged values. LCs constructed from such a list can be thought of as distributed lag predictors.

✓ A variable can appear on more than one LCLIST, meaning that LC lists can overlap. You can even create an LCLIST with all numeric variables on it if you wish.

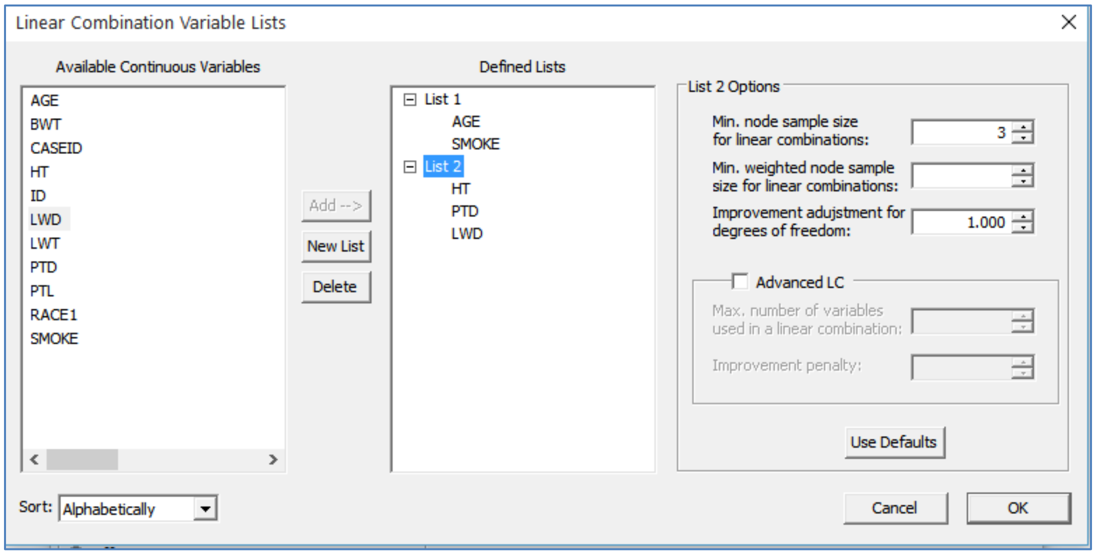
Below we have checked the box that activates LC lists for our example:


[✓] Use Linear Combinations for Splitting
Minimum node sample size for linear combinations: 3
Variable deletion significance level: 0.20
Number of nodes likely to be split by linear combinations in maximal tree: [● Automatic] ○ 300
[✓] Create Advanced Variable Lists

Press the [**Select Variables]** button to bring up a window in which you may create your LC Advanced Variables lists.

✓ Only numeric variables will be displayed in this window. Categorical variables will not be considered for incorporation into an LC even if they are simple 0/1 indicators. This is one good reason to treat your 0/1 indicators as numeric rather than categorical predictors.

Press the **[New List]** button to get started and then select the variables you want to include in the first list. We will select AGE and SMOKE. Add them and then click again on **New List** to start a second list. Now Add HT, PTD, LWD and click **OK** to complete the **LCLIST** setup. Click **Start** to begin the run.



Available Continuous Variables: AGE, BWT, CASEID, HT, ID, LWD, LWT, PTD, PTL, RACE1, SMOKE

Defined Lists:
- List 1
  - AGE
  - SMOKE
- List 2
  - HT
  - PTD
  - LWD

List 2 Options

Min. node sample size for linear combinations: 3

Min. weighted node sample size for linear combinations:

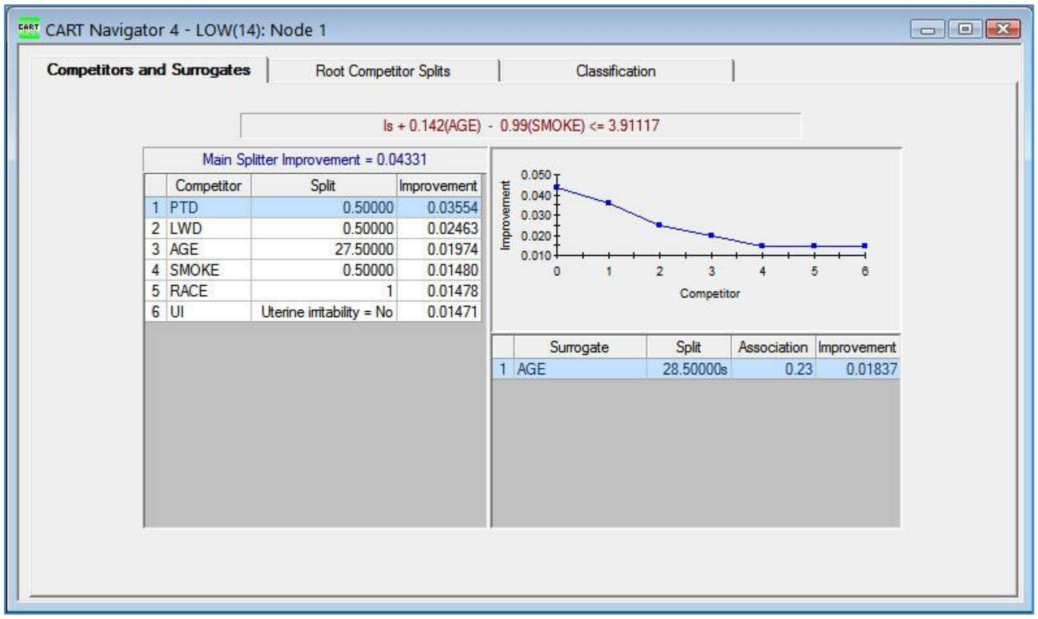
Improvement adjustment for degrees of freedom: 1.000

Max. number of variables used in a linear combination:

Improvement penalty:

Sort: Alphabetically

Hovering your mouse over the nodes of the new tree will allow you to quickly spot where linear combination splits have been found. Here we click on the root node of the navigator to bring up this display.


CART Navigator 4 - LOW(14): Node 1
Competitors and Surrogates
Root Competitor Splits
Classification
Improvement
Competitor

ls + 0.142(AGE) - 0.99(SMOKE) <= 3.91117

Main Splitter Improvement = 0.04331

	Competitor	Split	Improvement
1	PTD	0.50000	0.03554
2	LWD	0.50000	0.02463
3	AGE	27.50000	0.01974
4	SMOKE	0.50000	0.01480
5	RACE	1	0.01478
6	UI	Uterine irritability = No	0.01471

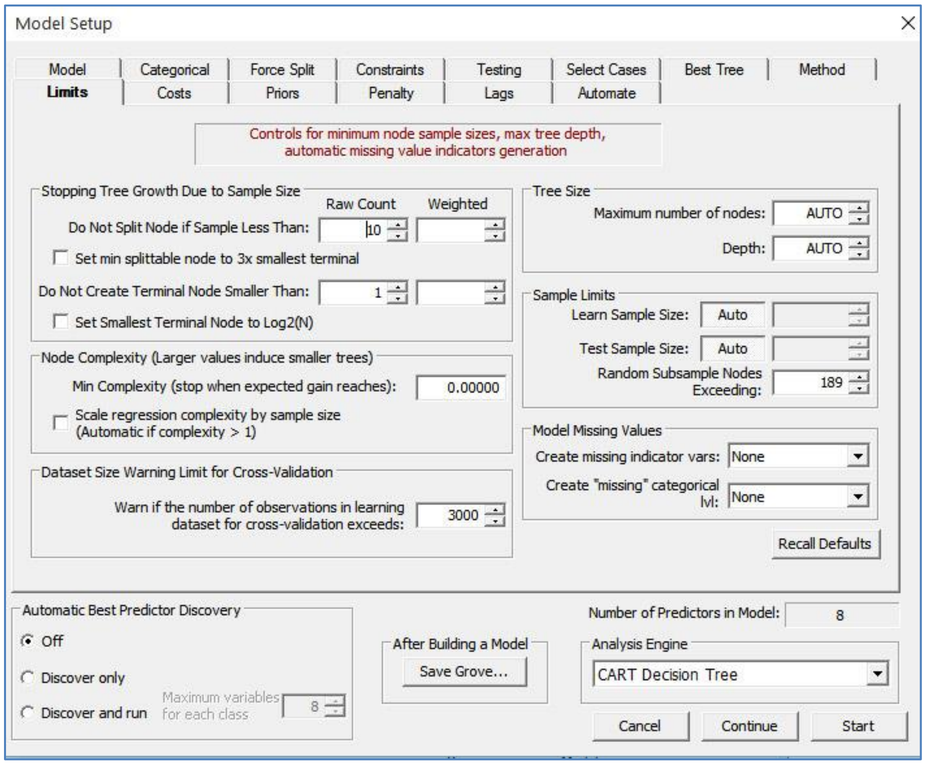
	Surrogate	Split	Association	Improvement
1	AGE	28.50000s	0.23	0.01837

Observe that the node is split on a linear combination of the two variables AGE and SMOKE with the splitter displayed near the top of the window. The improvement score of this LC is .0433, which is about 20% better than the best single-variable splitter PTD, which has an improvement score of .0355.

If you do not restrict the LCs with LCLISTs and instead run a legacy CART with linear combinations, you won't find any LCs reported. This is not a surprise; we have found it many times. Limiting LCs to a few choice variables is likely to yield better results than allowing CART to search over all available variables, a reflection of the fact that the LC search procedure cannot guarantee a global maximum.

Limits Tab

The **Limits** tab allows you to specify additional tree-building control options and settings. You should not hesitate to learn the meaning and use of these controls, as they can be the key to getting the best results.


Model Setup
Model | Categorical | Force Split | Constraints | Testing | Select Cases | Best Tree | Method
Limits | Costs | Priors | Penalty | Lags | Automate
Recall Defaults
Save Grove...
Cancel
Continue
Start

Controls for minimum node sample sizes, max tree depth, automatic missing value indicators generation

Stopping Tree Growth Due to Sample Size

	Raw Count	Weighted
Do Not Split Node if Sample Less Than:	10	
Do Not Create Terminal Node Smaller Than:	1	

☐ Set min splittable node to 3x smallest terminal

☐ Set Smallest Terminal Node to Log2(N)

Node Complexity (Larger values induce smaller trees)

Min Complexity (stop when expected gain reaches): 0.00000

☐ Scale regression complexity by sample size (Automatic if complexity > 1)

Dataset Size Warning Limit for Cross-Validation

Warn if the number of observations in learning dataset for cross-validation exceeds: 3000

Tree Size

Maximum number of nodes: AUTO

Depth: AUTO

Sample Limits

Learn Sample Size: Auto

Test Sample Size: Auto

Random Subsample Nodes Exceeding: 189

Model Missing Values

Create missing indicator vars: None

Create "missing" categorical lvl: None

Automatic Best Predictor Discovery

[✓ Off]

Discover only

Discover and run

Maximum variables for each class 8

Number of Predictors in Model: 8

After Building a Model

Analysis Engine: CART Decision Tree

Parent Node Minimum Cases (Do Not Split Node if Sample Less Than)

```
LIMIT ATOM = <N>
```

When do we admit that we do not have enough data to continue? Theoretically, we can continue splitting nodes until we run out of data, for example, when there is only one record left in a node. In practice it makes sense to stop tree growing when the sample size is so small that no one would take the split results seriously. The default setting for the smallest node we consider splitting is 10, but we frequently set the minimum to 20, 50, 100 or even 200 in very large samples.

Terminal Node Minimum Cases (Do Not Create Terminal Node Smaller Than)

```
LIMIT MINCHILD = <N>
```

This control specifies the smallest number of observations that may be separated into a child node. A large node might theoretically be split by placing one record in one child node and all other records into the other node. However, such a split would be rightfully regarded as unsatisfactory in most instances. The MINCHILD control allows you to specify a smallest child node, below which no nodes can be constructed. Naturally, if you set the value too high you will prevent the construction of any useful tree.

✓ Increasing allowable parent and child node sizes enables you to both control tree growth and to potentially fit larger problems into limited workspace (RAM).

✓ You will certainly want to override the default settings when dealing with large datasets.

✓ The parent node limit (ATOM) must be at least twice the terminal node (MINCHILD) limit and otherwise will be adjusted by CART to comply with the parent limit setting.

✓ We recommend that ATOM be set to at least three times MINCHILD to allow CART to consider a reasonable number of alternative splitters near the bottom of the tree. If ATOM is only twice MINCHILD then a node that is just barely large enough to be split can be split only into two equal-sized children.

Minimum Complexity

```
BOPTIONS COMPLEXITY = <x> [,SCALED]
```

This is a truly advanced setting with no good short explanation for what it means, but you can quickly learn how to use it to best limit the growth of potentially large trees. The default setting of zero allows the tree-growing process to proceed until the “bitter end”. Setting complexity to a value greater than zero places a penalty on larger trees, and causes CART to stop its tree-growing process before reaching the largest possible tree size. When CART reaches a tree size with a complexity parameter equal to or smaller than your pre-specified value, it stops the tree-splitting process on that branch. If the complexity parameter is judiciously selected, you can save computer time and fit larger problems into your available workspace. (See the main reference manual for guidance on selecting a suitable complexity parameter.)

✓ Check the **Complexity Parameter** column in the **TREE SEQUENCE** section of the **Classic Output** to get the initial feel for which complexity values are applicable for your problem.

The **Scale Regression** check box specifies that, for a regression problem, the complexity parameter should be scaled up by the learn-sample size.

Dataset Size Warning Limit for Cross-Validation

```
BOPTIONS CVLEARN = <N>
```

By default, 3,000 is the maximum number of cases allowed in the learning sample before cross validation is disallowed and a test sample is required. To use cross validation on a file containing more than 3,000 records, increase the value in this box to at least the number of records in your data file.

Maximum number of nodes

```
LIMIT NODES = <N>
```

Allows you to specify a maximum allowable number of nodes in the largest tree grown. If you do not specify a limit CART may allow as many as one terminal node per data record. When a limit on NODES is specified the tree generation process will stop when the maximum allowable number of nodes (internal plus terminal) is reached. This is a crude but effective way to limit tree size.

Depth

```
LIMIT DEPTH = <N>
```

This setting limits the tree growing to a maximum depth. The root node corresponds to the depth of one. Limiting a tree in this way is likely to yield an almost perfectly balanced tree with every branch reaching the same depth. While this may appeal to your aesthetic sensibility it is unlikely to be the best tree for predictive purposes.

By default CART sets the maximum DEPTH value so large that it will never be reached.

- ✓ Unlike complexity, these NODES and DEPTH controls may handicap the tree and result in inferior performance.
- ✓ Some decision tree vendors set depth values to small limits such as five or eight. These limits are generally set very low to create the illusion of fast data processing. If you want to be sure to get the best tree you need to allow for somewhat deeper trees.

Learn Sample Size (LEARN)

```
LIMIT LEARN = <N>
```

The **LEARN** setting limits CART to processing only the first part of the data available and simply ignoring any data that comes after the allowed records. This is useful when you have very large files and want to explore models based on a small portion of the initial data. The control allows for faster processing of the data because the entire data file is never read.

Test Sample Size

```
LIMIT TEST = <N>
```

The **TEST** setting is similar to LEARN: it limits the test sample to no more than the specified number of records for testing. The test records are taken on a first-come-first served basis from the beginning of the file. Once the TEST limit is reached no additional test data are processed.

Random Subsample Nodes Exceeding

```
LIMIT SUBSAMPLE = <N>
```

Node sub-sampling is an interesting approach to handling very large data sets and also serves as a vehicle for exploring model sensitivity to sampling variation. Although node sub-sampling was introduced in the first release of the CART mainframe software in 1987, we have not found any discussion of the topic in the scientific literature. We offer a brief discussion here.

Node sub-sampling is a special form of sampling that is triggered for special purposes during the construction of the tree. In node sub-sampling the analysis data are *not* sampled. Instead we work with the complete analysis data set. When node sub-sampling is turned on we conduct the process of searching for a best splitter for a node on a subsample of the data in the node. For example, suppose our analysis data set contained 100,000 records and our node sub-sampling parameter was set to 5,000. In the root node we would take our 100,000 records and extract a random sample of 5,000. The search for the best splitter would be conducted on the 5,000 random record extract. Once found, the splitter would be applied to the full analysis data set. Suppose this splitter divided the 100,000 root node into 55,000 records on the left and 45,000 records on the right. We would then repeat the process of selecting 5,000 records at random in each of these child nodes to find their best splitters.

As you can see, the tree generation process continues to work with the complete data set in all respects except for the split search procedure. By electing to use node sub-sampling we create a shortcut for split finding that can materially speed up the tree-growing process.

But is node sub-sampling a good idea? That will depend in part on how rare the target class of interest is. If the 100,000 record data set contains only 1,000 YES records and 99,000 NO records, then any form of sub-sampling is probably not helpful. In a more balanced data set the cost of an abbreviated split search might be minimal and it is even possible that the final tree will perform better. Since we cannot tell without trial and error we would recommend that you explore the impact of node sub-sampling if you are inclined to consider this approach.