



Salford Predictive Modeler<sup>®</sup>

## **Introducing TreeNet<sup>®</sup> Gradient Boosting Machine**

*This guide describes the TreeNet<sup>®</sup> product and illustrates some practical examples of its basic usage and approach.*

© 2019 Minitab, LLC. All Rights Reserved.

Minitab®, SPM®, SPM Salford Predictive Modeler®, Salford Predictive Modeler®, Random Forests®, CART®, TreeNet®, MARS®, RuleLearner®, and the Minitab logo are registered trademarks of Minitab, LLC. in the United States and other countries. Additional trademarks of Minitab, LLC. can be found at [www.minitab.com](http://www.minitab.com). All other marks referenced remain the property of their respective owners.

## Interactive Table of Contents

<b>INTERACTIVE TABLE OF CONTENTS _TOC488932303</b> .....	<b>3</b>
<b>TREENET® INTRODUCTION</b> .....	<b>6</b>
<b>GRADIENT BOOSTING ALGORITHM</b> .....	<b>6</b>
<b>BASIC PRINCIPLES OF TREENET</b> .....	<b>7</b>
<b>TYPICAL QUESTIONS AND ANSWERS ABOUT TREENET</b> .....	<b>8</b>
Model Tab .....	10
Categorical Tab.....	13
Testing Tab .....	14
Select Cases Tab.....	17
TreeNet Tab.....	18
Criterion Determining Number of Tree Optimal for Logistic Model .....	22
Plots & Options Tab .....	32
Other Plotting Options .....	33
Variable containing Start Values for Model Continuation .....	34
TN Advanced Tab .....	36
TN Interactions Tab.....	40
TN Interactions Tab: Interaction Inhibition via Penalties .....	43
Interaction Statistics .....	44
<b>Class Weights Tab</b> .....	<b>53</b>
<b>Penalty Tab</b> .....	<b>54</b>
Penalties on Variables.....	55
Missing Values Penalty .....	56
High-Level Categorical (HLC) Penalty .....	57
<b>Lags Tab: A Brief Introduction</b> .....	<b>58</b>
<b>Automate Tab: A Brief Introduction</b> .....	<b>60</b>
Using an Automate .....	62
<b>EXAMPLE: BUILDING A REGRESSION MODEL</b> .....	<b>64</b>
<b>Creating a Sample Example Model Data Set</b> .....	<b>64</b>
<b>Reading Data In</b> .....	<b>65</b>
<b>The Activity Window</b> .....	<b>66</b>
<b>Setting up a TreeNet Model for Regression</b> .....	<b>66</b>
<b>Running TreeNet</b> .....	<b>68</b>
<b>TREENET REGRESSION: INTERPRETING RESULTS</b> .....	<b>69</b>
<b>Initial Output</b> .....	<b>69</b>
<b>Error Curves</b> .....	<b>71</b>
<b>Summary Button</b> .....	<b>72</b>
Summary Tab .....	72
Dataset Tab.....	73
Variable Importance Tab .....	74
Resid. Trim % Tab.....	80
Resid. Trim Cnt Tab .....	82
Resid. Outliers CUM % Tab.....	84
Resid. Outliers Counts Tab.....	85
Resid. Box Plot Tab.....	86
Score Distribution Tab.....	87
<b>Display Plots Button</b> .....	<b>88</b>

View All Plots .....	88
View All One (or Two) Variable PDPs Only .....	89
All One Variable PDP Plot Controls .....	90
View an Individual PDP .....	91
<b>Partial Dependency Plots (PDPs) .....</b>	<b>92</b>
Constructing One Variable PDPs .....	92
Constructing Two Variable PDPs .....	93
Interpreting One Variable PDPs .....	94
LAD Loss & Continuous Predictors .....	95
LAD Loss & Categorical Predictors .....	96
<b>Spline Approximations to the PDPs.....</b>	<b>97</b>
<b>Create Plots Button .....</b>	<b>97</b>
<b>EXAMPLE: BUILDING A CLASSIFICATION/LOGISTIC BINARY MODEL</b>	<b>98</b>
<b>Creating a Sample Example Model Data Set.....</b>	<b>99</b>
<b>Reading Data In.....</b>	<b>100</b>
<b>The Activity Window .....</b>	<b>101</b>
<b>Setting up the Classification Model .....</b>	<b>102</b>
Model Tab .....	103
Categorical Tab.....	105
TreeNet Parameters Specific to Classification .....	106
Choosing the Classification Loss Function.....	106
Advanced Options for Classification Models .....	107
Running TreeNet .....	107
<b>CLASSIFICATION/BINARY LOGISTIC: INTERPRETING RESULTS .....</b>	<b>108</b>
<b>Initial Output I .....</b>	<b>108</b>
<b>Error Curves .....</b>	<b>111</b>
<b>Summary Button .....</b>	<b>112</b>
Summary Tab .....	112
Dataset Tab.....	113
Gains/ROC Tab .....	114
Variable Importance Tab .....	117
Misclassification Tab.....	121
Confusion Matrix Tab.....	124
Hosmer-Lemeshow tab.....	130
Odds Graph tab.....	132
Score Distribution Tab.....	133
Display Plots Button: Viewing PDPs .....	134
<b>Partial Dependency Plots (PDPs).....</b>	<b>138</b>
Constructing One Variable PDPs .....	138
Constructing Two Variable PDPs.....	139
<b>Interpreting PDPs: Binary Classification .....</b>	<b>140</b>
Binary Classification & Continuous Predictors .....	140
Binary Classification & Categorical Predictors .....	141
<b>Interpreting PDPs: Multinomial Classification .....</b>	<b>142</b>
Multinomial PDPs & Continuous Predictors .....	142
Multinomial PDPs & Categorical Predictors .....	144
<b>Spline Approximations to the PDPs.....</b>	<b>146</b>
<b>Create Plots.....</b>	<b>146</b>
<b>SPLINE APPROXIMATIONS TO PARTIAL DEPENDENCY PLOTS .....</b>	<b>147</b>
<b>Manual Knot Placement .....</b>	<b>148</b>
<b>Automatic Knot Placement &amp; Fixed Approximations .....</b>	<b>149</b>
<b>Using Splines Approximations in a Model .....</b>	<b>151</b>
<b>Simultaneous Spline Approximations .....</b>	<b>154</b>

<b>CREATE PLOTS BUTTON.....</b>	<b>155</b>
Create Plots Tab.....	155
Pairwise Interactions Tab.....	158
Example: Top Interacting Plots for the 100 Tree Model .....	159
Viewing Results for Non-optimal Trees .....	162
<b>COMMANDS .....</b>	<b>163</b>
<b>TRANSLATING TREENET MODELS.....</b>	<b>164</b>
Example: Translating the Optimal TreeNet into Java Code.....	165
Example: Translating a Non-Optimal Model into C code.....	166
Example: Translating the Rules to a .dat file.....	167
<b>SAVING TREENET MODELS: GROVE FILE .....</b>	<b>168</b>
<b>SCORING TREENET MODELS (GENERATE PREDICTIONS) .....</b>	<b>169</b>
General Tab .....	170
Select Cases Tab .....	173
Generating Predictions for Learn and Test Partitions .....	174
Generating Predictions Using a Saved Model.....	177
<b>RARE TREENET COMMANDS .....</b>	<b>182</b>
<b>USING LOGISTIC CALIBRATION ON TREENET RESULTS .....</b>	<b>183</b>
Viewing the Logistic Calibration Results .....	185
<b>MISSING VALUE HANDLING .....</b>	<b>186</b>

## TreeNet® Introduction

Stochastic gradient boosting, commonly referred to as “gradient boosting”, is a revolutionary advance in machine learning technology. Gradient Boosting was developed by Stanford Statistics Professor Jerome Friedman in 2001. TreeNet® software implements the gradient boosting algorithm and was originally written by Dr. Friedman himself, whose code and proprietary implementation details are exclusively licensed to Salford Systems.

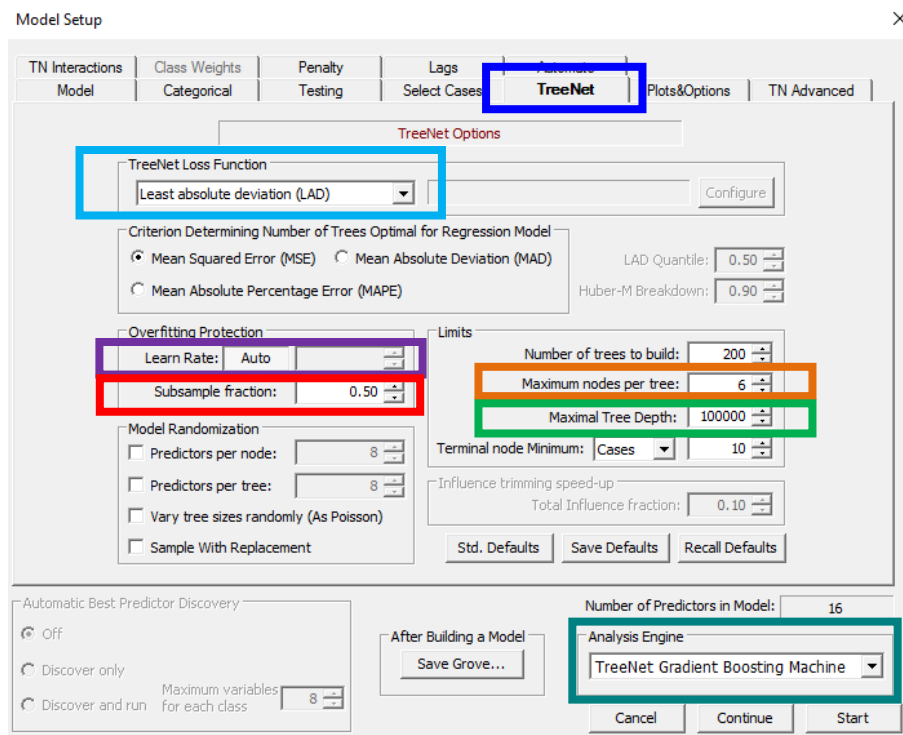
### Gradient Boosting Algorithm

1. For each record in the learn sample, supply an initial value specific to the chosen **loss function**
2. Sample **s percent** of the records in the learn sample randomly
3. Compute the generalized residual for the records in the sample from Step 2
4. For the records sampled in Step 2, fit a CART tree with a **maximum of J terminal nodes** to the generalized residuals computed in Step 3
5. Use the CART tree in Step 3 to update the model (the updates depend on the loss function) but shrink the update by the **learning rate  $\alpha$**

Repeat Steps 2-4 **M times**

The final model is the following:  $TreeNet = initial\ value + \alpha CART_1 + \alpha CART_2 + \dots + \alpha CART_M$

The following is a picture of one of the main model setup menus in TreeNet software. Be sure to set the Analysis Engine to **TreeNet Gradient Boosting Machine** and then navigate to the **TreeNet Tab**.



Each of the **M iterations** corresponds to fitting a CART tree to the generalized residuals so the number of iterations = number of trees built; and thus, we use the terminology “**Number of trees to build.**”

## Basic Principles of TreeNet®

TreeNet is a revolutionary advance in data mining technology developed by Jerome Friedman, one of the world's outstanding data mining researchers. TreeNet offers exceptional accuracy, blazing speed, and a high degree of fault tolerance for dirty and incomplete data. It can handle both classification and regression problems and is remarkably effective in traditional numeric data mining and text mining.

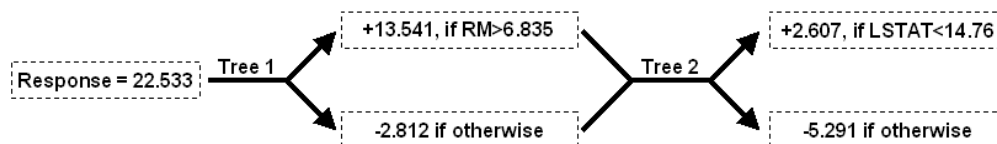
A TreeNet model normally consists of several hundred to several thousand small trees, each typically containing about six terminal nodes. Each tree contributes a small portion to the overall model and the final model prediction is the sum of all the individual tree contributions. You can think of the TreeNet model as a black box that delivers exceptionally accurate models. TreeNet offers detailed self-testing to document its reliability on your own data. In many ways, the TreeNet model is not that mysterious, although it is undeniably complex. You do not need to concern yourself with that complexity because all results, performance measures, and explanatory graphs can be understood by anyone familiar with basic data mining principles. However, if you want to understand the details of TreeNet model construction, refer to the following outline

The model is similar in spirit to a long series expansion, such as a Fourier or Taylor series, which is a sum of factors that becomes progressively more accurate as the expansion continues. The expansion can be written as:

$$F(X) = F_0 + \beta_1 T_1(X) + \beta_2 T_2(X) + \dots + \beta_M T_M(X)$$

In the equation above, each  $T_i$  can be considered a small tree. You should read this as a weighted sum of terms, each of which is obtained from the appropriate terminal node of a small tree.

In this example, the first few terms from a regression model based on a well-known data set: the Boston Housing data extracted from the 1970 US Census. This data set is usually used to build models that predict the median value of houses in a neighborhood based on quality-of-life variables, such as crime rate, school quality, and socioeconomic status of the neighborhood, and a few core housing descriptors, such as typical house age and size.



The model above begins with an estimate of mean home value (in 1970) of \$22,533. We can use this as a baseline from which adjustments will be made to reflect characteristics of the housing and the neighborhood. In the first term, the model states that the mean value would be adjusted upwards by \$13,541 for larger homes, and adjusted upwards again by \$2,607 for neighborhoods with good socioeconomic status indicators. In practice, the adjustments are usually much smaller than shown in this regression example and hundreds of adjustments may be needed. The final model is a collection of weighted and summed trees.

Although this example is a regression, the same scheme is used for classification problems. For binary classification problems, a yes or no response is determined by whether the sign of the predicted outcome is positive or negative. For multi-class problems, a score is developed separately for each class via class-specific expansions and the scores are converted into a set of probabilities of class membership.

The Boston housing regression example above uses the smallest possible two-node tree in each stage. More complicated models tracking complex interactions are possible with three or more nodes at each stage. The TreeNet default uses a six-node tree, but the optimal tree size for any specific modeling task can only be determined through trial and error. We have worked on problems that are handled perfectly well with two-node trees, and one of our award-winning data mining models used nine-node trees. Fortunately, TreeNet models run very quickly, so it is easy to experiment with a few different-sized trees to determine which will work best for your problem.

## Typical Questions and Answers about TreeNet

### What are the advantages of TreeNet?

In our experience TreeNet is the closest tool we have ever encountered to a fully-automated statistician. TreeNet can deal with substantial data quality issues, decide how a model should be constructed, select variables, detect interactions, address missing values, ignore suspicious data values, prevent the model from being dominated by just a few ultra-powerful variables, and resist any overfitting.

Typically, the result is a model that is far more accurate than any model built by data-mining tools and at least as accurate than any model built by experts working for weeks or months. Of course, there is no substitute for a good understanding of the subject matter and problems being addressed and a reasonable familiarity with the data is always required for reliable results. However, given that, TreeNet can give you a substantial advantage in reaching high-performance models quickly.

Below is a summary of TreeNet's key features:

Automatic selection from thousands of candidate predictors.

- No prior variable selection or data reduction is required.

Ability to handle data without preprocessing.

- Data do not need to be rescaled, transformed, or modified in any way.
- Resistance to outliers in predictors or the target variable.
- Automatic handling of missing values.
- General robustness to dirty and partially inaccurate data.

High Speed:

- Trees are grown quickly; small trees are grown extraordinarily quickly.
- TreeNet is able to focus on the data that are not easily predictable as the model evolves. As additional trees are grown, fewer and fewer data need to be processed.
- TreeNet is frequently able to focus much of its training on just 20% of the available data (all accomplished automatically without user intervention).

Resistance to Overtraining.

- When working with large data bases even models with 2,000 trees show little evidence of overtraining.

TreeNet's robustness extends to the most serious of all data errors: when the target variable itself is sometimes incorrect, for example, when a "yes" is incorrectly recorded as a "no," or vice versa. In the



machine learning world, such data are said to be contaminated with erroneous target labels. For example, in medicine, there is some risk that patients labeled as healthy are in fact ill and vice versa. This type of data error can be challenging for conventional data mining methods and can be catastrophic for conventional forms of “boosting.” In contrast, TreeNet is generally immune to such errors as it dynamically rejects training data points too much at variance with the existing model

In addition, TreeNet adds the advantage of a degree of accuracy usually not attainable by a single model or by ensembles such as bagging or conventional boosting. Independent real-world tests in text mining, fraud detection, and credit worthiness have shown TreeNet to be dramatically more accurate on test data than other competing methods.

Of course, no one method can be best for all problems in all contexts. Typically, if TreeNet is not well suited for a problem, it will yield accuracies on par with a single CART® tree.

### **What are the advantages of TreeNet over a neural net?**

Several of our most avid TreeNet users are former neural network advocates who have discovered how much faster and easier it is to get their modeling done with TreeNet while sacrificing nothing in the area of accuracy. In contrast to neural networks, TreeNet is not overly sensitive to data errors and needs no time-consuming data preparation, preprocessing, or imputation of missing values. TreeNet is especially adept at dealing with errors in the target variable, a type of data error that could be catastrophic for a neural net. TreeNet is resistant to overtraining and can be 10 to 100 times faster than a neural net. Finally, TreeNet is not troubled by hundreds or thousands of predictors.

### **What is the technology underlying TreeNet and how does it differ from boosting?**

TreeNet is a proprietary methodology developed in 1997 by Stanford's Jerome Friedman, a co-author of CART®, the author of MARS® and PRIM, and the inventor of Projection Pursuit regression, the methodology known as “stochastic gradient boosting”; the trade secrets of the technology are embedded exclusively in TreeNet. Others may eventually try to develop technology based on the public descriptions offered by Professor Friedman, but hundreds of technical details remain exclusive to Salford Systems.

### **What is the TreeNet track record?**

TreeNet technology has been tested in a broad range of industrial and research settings and has demonstrated considerable benefits. In tests in which TreeNet was pitted against expert modeling teams using a variety of standard data mining tools, TreeNet was able to deliver results within a few hours comparable to or better than results that required months of hands-on development by expert data mining teams.

TreeNet was designated “Most Accurate” in the KDD Cup 2004 data mining competition (sponsored by the Association for Computing Machinery's data mining SIG). TreeNet also won first place in all four competitive categories in the 2002 Duke University/NCR Teradata Churn modeling competition and numerous other competitions since.

### **How does TreeNet fit into the Salford Systems data mining solution?**

The Salford Systems data mining solution rests on two groups of technologies: CART, MARS for accurate easy-to-understand models and TreeNet and Random Forests for ultra-high performing, but potentially very complex, models interpreted via supporting graphical displays. Even in circumstances where interpretability and transparency are mandatory and a model must be expressible in the form of rules, TreeNet can serve a useful function by benchmarking the maximum achievable accuracy against which interpretable models can be compared.

## TreeNet Settings

### Model Tab

Model Setup

Class Weights | Penalty | Lags | Automate | Plots&Options | TN Advanced

Model

Variable Selection

Variable Name	Target	Predictor	Categorical	Weight
Y1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Y2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Y3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
W	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Z1\$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Z2\$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
X1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Sort: File Order

Filter:  All/Selected  Character  Numeric

Automatic Best Predictor Discovery:  Off  Discover only  Discover and run

Maximum variables for each class: 8

After Building a Model: Save Grove...

Analysis Engine: TreeNet Gradient Boosting Machine

Number of Predictors in Model: 12

Target Type:  Classification/Logistic Binary  Regression  Unsupervised

Set Focus Class...

Target Variable: Y1

Weight Variable:

Number of Predictors: 12

Cancel Continue Start

#### Analysis Engine

Specify the type of model to build. To build a TreeNet model, click the dropdown box and select “TreeNet Gradient Boosting Machine.”

#### Target Type

A target type of “Regression” means that the target variable is quantitative, whereas a target type of “Classification” the target variable is a character variable or a numbered target variable where each number corresponds to a class.

#### Target

```
MODEL <depvar name> [= <predictor_list>]
```

Specify the dependent variable. In SPM, the dependent variable is referred to as the “target variable.”

#### Predictor

```
KEEP <predictor_list>
```

Specify the predictor variables to be considered in the model by clicking the desired checkboxes corresponding to the **variable names listed on the left**. To select all variables as predictors, click the “Predictor” label (**top of the purple rectangle above**) and then click the **Select Predictors checkbox** to check all the checkboxes simultaneously. To select a particular subset of variables as predictors, highlight the desired group of variables by clicking and dragging and then click the **Select Predictors checkbox**. In SPM we call the list of predictor variables in the model the “KEEP list.”

**Model Setup**

Class Weights | Penalty | Lags | Automate | Plots&Options | TN Advanced

**Model** | Categorical | Testing | Select Cases | TreeNet

Variable Selection

Variable Name	Target	Predictor	Categorical	Weight
Y1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Y2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Y3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
W	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Z1\$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Z2\$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
X1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Sort: File Order

Filter:  All/Selected  Character  Numeric

Target Type:  Classification/Logistic Binary  Regression  Unsupervised

Set Focus Class...

Target Variable: Y1

Weight Variable:

Number of Predictors: 12

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

After Building a Model: Save Grove...

Analysis Engine: TreeNet Gradient Boosting Machine

Buttons: Cancel, Continue, Start

### Categorical

CATEGORICAL <predictor\_list>

Specify the variables to be treated as categorical in the model by clicking the desired checkboxes corresponding to the **variable names listed on the left**. To select all variables as categorical, click the “Categorical” label (**top of the green rectangle above**) and then click the **Select Cat. Checkbox** to check all the checkboxes simultaneously. To select a particular subset of variables as categorical, highlight the desired group of variables by clicking and dragging and then click the **Select Cat. checkbox**.

### Weight

WEIGHT <weight\_variable\_name>

Specify the variable to use as a case weight. The case weight values must be numeric and may take on fractional, but not negative, values.

### Cancel Button

Exit the model setup without saving the current model settings.

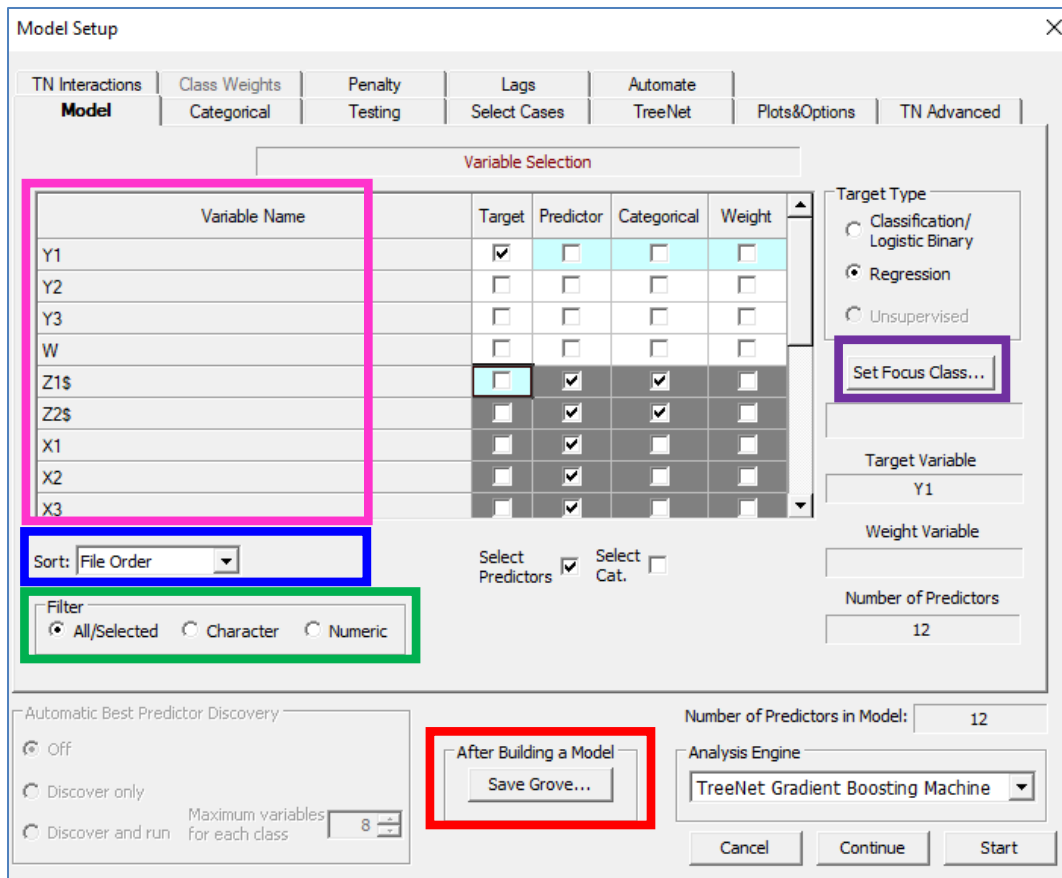
### Continue Button

Exit the model setup while saving the current model settings.

### Start Button

TREENET GO

Build the TreeNet model.



### Sort Control

Sort the **variable names list on the left** by their alphabetical or file order by clicking the arrow and selecting the desired option. The default order is alphabetical and in the picture above it is “File Order.”

### Filter

```
KEEP <_CHARACTER_|_NUMERIC_>
```

You can construct the model using only categorical variables by clicking “Character” or only numeric variables by clicking “Numeric.”

### Set Focus Class... Button

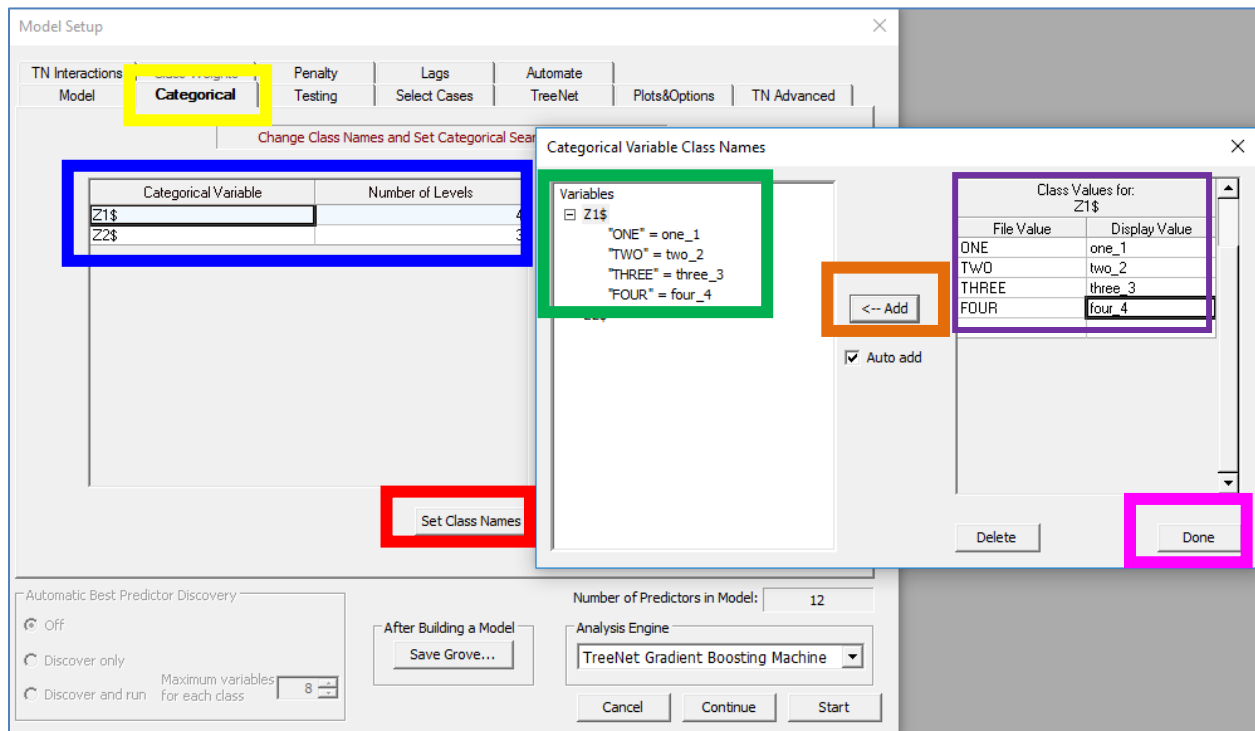
```
FOCUS <variable1> = <class1> [<variable2>=<class2>...]
```

Appropriate for a Target type of “Classification/Logistic Binary” only. This allows you to set the event of interest which is referred to as the “Focus Class” in SPM. For instance, to predict if someone has a disease, then your target variable could have two values: 0 if the patient does not have the disease or 1 if the patient does have the disease. If you want the predicted probabilities to be the probability that a patient has a disease, then set the focus class to 1. See the [Focus Class: Specifying the Event of Interest section](#) for information about the focus class setup.

### Save Grove... Button

Results in SPM are saved in a special file called a *grove*. A grove file stores the model itself as well as useful summary information, SPM commands, and more. Click this button to specify a save location for the model and the model will be saved immediately after the modeling run has been completed.

## Categorical Tab



In the **Categorical Tab** (yellow rectangle above), you will see values for the “Number of Levels” for each categorical variable (**blue rectangle above**) if you compute summary statistics first; otherwise, you will see “Unknown.” Click the summary statistics shortcut button (**light blue rectangle below**) to access the summary statistics option



Click the **Set Class Names button** in the picture above to set labels for categorical variables

### Set Class Names

After you click the **Set Class Names button** in the picture above, you will see the Categorical Variable Class Names menu that is shown on the right in the picture above:

1. “File Value” is the original value of the variable in the dataset whereas “Display Value” is the label that you want to add in place of the original value in the dataset.
  - a. Type both the file values and display values as you want them to appear (**purple rectangle above**).
2. Click the **<-- Add button** to add the label for the variable Z1 and you will see the labels added to the desired variable in the left pane (**green rectangle above**).
3. Click the **Done button** when you are finished adding labels.

## Testing Tab

The screenshot shows the 'Model Setup' dialog box with the 'Testing' tab selected. The 'Testing' tab is highlighted in yellow. The 'Fraction of cases selected at random' is set to 0.20. The 'Fast' radio button is selected and highlighted in blue, and the 'Exact' radio button is highlighted in red. The 'Cross-Validation' section has 'V-fold cross-validation' selected with 'Folds' set to 10. The 'Automatic Best Predictor Discovery' section has 'Off' selected. The 'Number of Predictors in Model' is set to 12. The 'Analysis Engine' is set to 'TreeNet Gradient Boosting Machine'.

### No independent testing – exploratory model

PARTITION NONE

In this mode, no external evaluation is performed. The entire dataset is used for training, and therefore, the final results will include only training data. The best model is almost always the largest.

### Fraction of Cases Selected at Random (FAST)

PARTITION TEST = .2, DRAW = FAST

This results in a model built using approximately 20% of the data for testing. Final results are reported for both the training and testing data, and the percentage can be modified.

### Fraction of Cases Selected at Random (EXACT)

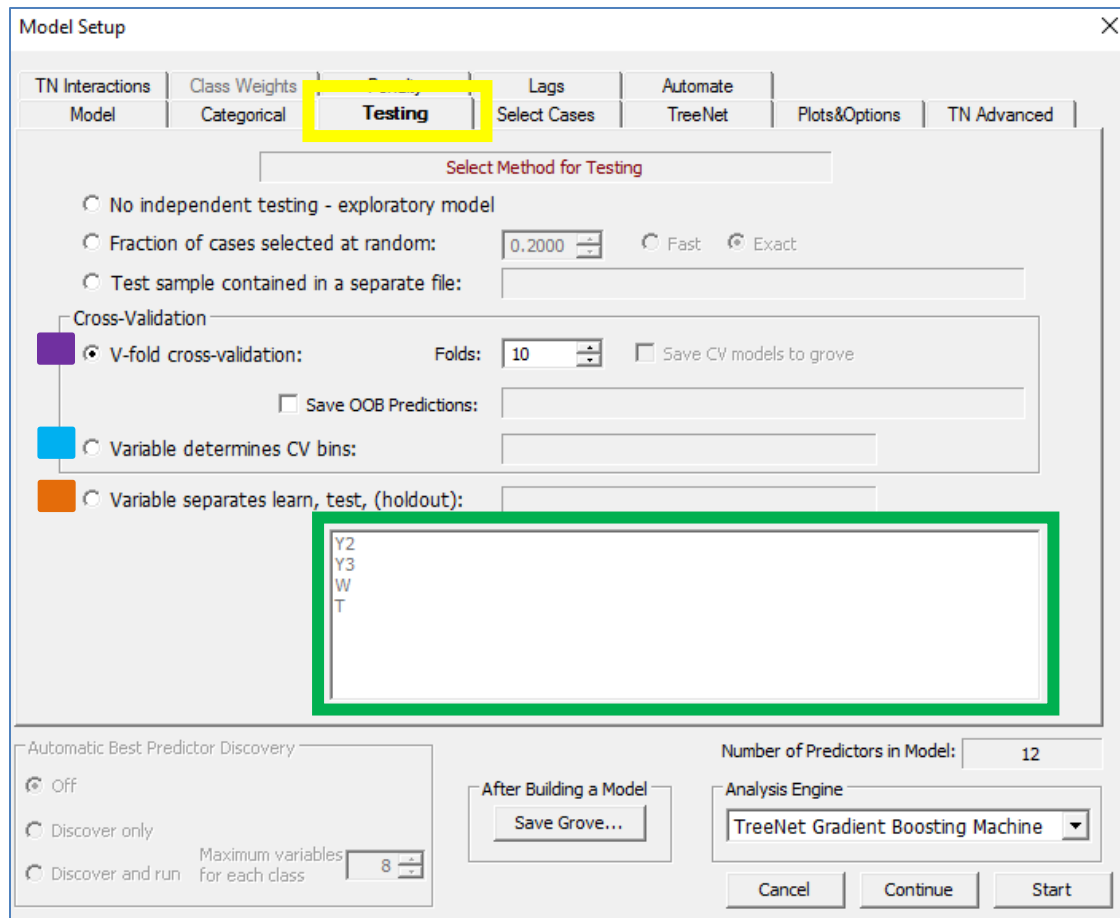
PARTITION TEST = .2, DRAW = EXACT

This results in a model built using exactly 20% (or very close) of the data for testing. Note that there is a small time cost if the EXACT method is used. Final results are reported for both the training and testing data, and the percentage can be modified.

### Test Sample Contained in a Separate File

PARTITION FILE = "C:\Users\..."

This results in a model built using the data in the specified file as the test dataset. Final results are reported for both train and test data.



### V-Fold Cross-Validation

PARTITION CROSS = <V>

Usually used with small datasets when one cannot afford to reserve some data for testing. The entire dataset is used for learning purposes, and then is partitioned into ten bins. At each fold in 10-fold CV, nine bins are used as a training set and the remaining bin is used as a test set. After all 10 folds are completed, the test results from each fold are averaged to get a fair test estimate of the all-data model performance.

### Variable determines CV bins

PARTITION CROSS = VARIABLE\_NAME

Same as the above option, but the user has full control over bin creation by introducing a special variable with integer values 1 through K to mark the bins where K is the desired number of bins. You can opt for a different number of folds. Click the radio button and specify the variable by clicking the variable name (**green rectangle above**).

### Variable Separates Learn, Test, (Holdout):


PARTITION CROSS = VARIABLE\_NAME

This is the most flexible mode. Click the radio button and specify the variable by clicking the desired variable name (**green rectangle above**). When the value for the variable is 0, the corresponding observation is used in the learning data; when the value is 1, the observation is used in the testing data; otherwise observation will not be included in any partition and assumed to be reserved for the holdout sample (i.e. any value other than 0 or 1; the holdout data is data that the model has never seen and can be used as a final assessment of model performance)

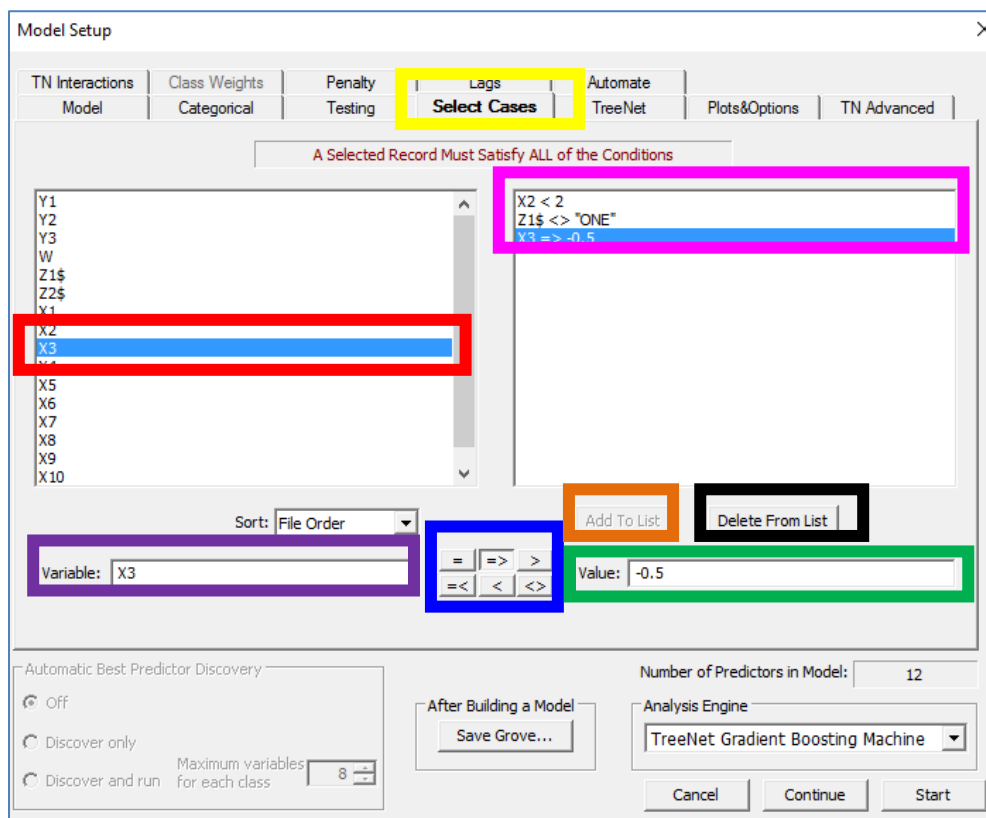
- ✓ Use the indicator variable mode when you want complete control over which part of the data is used for testing. You may also create multiple indicator variables in the same data set to experiment with different random seeds or learn/test ratios.
- ✓ Use “Cross Validation” when the supply of data is limited.
- Use extreme care when creating your own CV bins. Each bin should be of the same size and balanced on the target variable.
- 10-fold cross-validation runs are, on average, 10 times slower than using a test sample. For small data sets, you may not notice the difference but for large data sets using cross validation will substantially increase your run times.



## Select Cases Tab

 SELECT <var\$> <relation> '<string\$>'

The “Select Cases” tab allows you to specify up to ten selection criteria for building a model based on a subset of cases. A selection criterion can be specified in terms of any variable appearing in the data set (i.e. the variable may or may not be involved in the model), and is constructed as follows:



1. Click the desired variable name (**red rectangle above**) which will then appear in the **Variable text box**.
2. Click the desired **operator button** (i.e. =, =>, > =<, <, or <>).
  - a. Note: <> means “not equal”.
3. Enter the desired value for the condition (**green rectangle above**).
4. Click the **Add to List button** to add the condition.

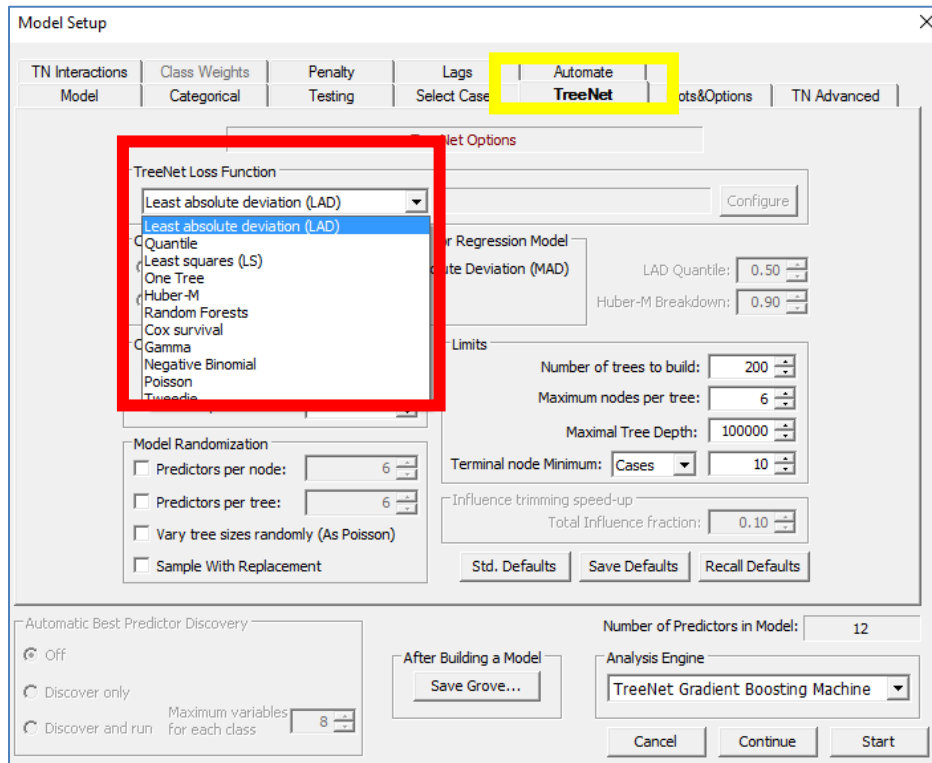
The condition appears in the right pane (**pink rectangle above**). To delete a condition, click the condition in the right pane (**pink rectangle above**), and then click “Delete From List” (**black rectangle above**). Here our filter is defined by three conditions:

1.  $X2 < 2$
2.  $Z1\$ \neq \text{“ONE”}$  (Note: <> means “not equal”)
3.  $X3 \geq -0.5$  (Note: => means “greater than or equal to”)

If the model were built with these three conditions, then the model is built only with observations that satisfy all three of the conditions.

## TreeNet Tab

### Regression Loss Functions



 TREENET LOSS=LAD|LS|HUBER|RF|POISSON|GAMMA|NEGBIN|COX|AUTO|TWEEDIE

Users have the following regression loss function options. Note:  $F_i$  is the prediction for the  $i$ th case  $Y_i$ .

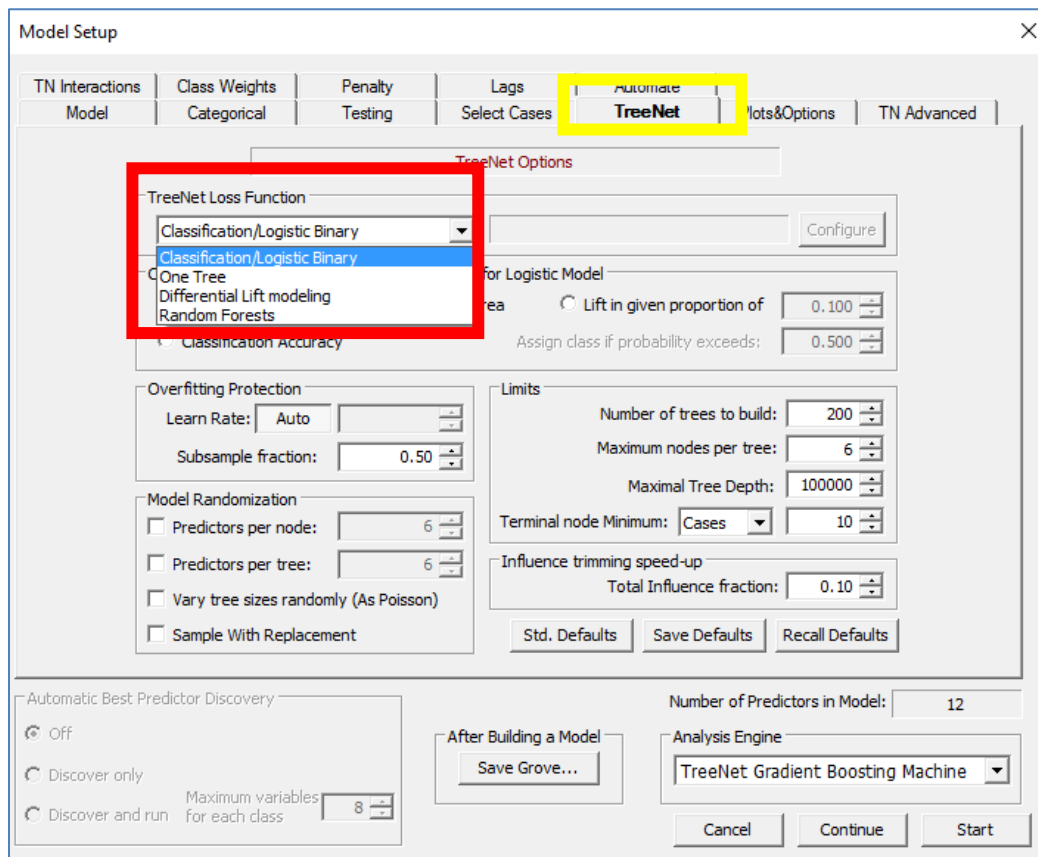
1. **LAD:** Least Absolute Deviation loss function used for quantitative targets
2. **LS:** Least Squares loss function used for quantitative targets
3. **QUANTILE:** model a specific quantile. Example: a quantile value of .5 (specified by the user in the GUI) means that you are modeling the conditional median (this is identical to the LAD loss function).
4. **HUBER:** Huber-M Loss Function that is a hybrid of the least squares loss function and the least absolute deviation loss function
5. **RF:** build a Random Forest using the TreeNet engine. This allows you to see partial dependency plots for the Random Forest model and build RuleLearner and ISLE models for Random Forests.
6. **POISSON:** Poisson loss function designed for the regression modeling of integer COUNT data, typically for small integers such as 0, 1, 2, 3, 4, ....
7. **GAMMA:** designed for the regression modeling of strictly positive target variables
8. **NEGBIN:** Negative Binomial loss function used for counted targets 0, 1, 2, 3, ...
9. **COX:** designed for the regression modeling of non-negative survival time with potentially a CENSOR variable that indicates whether the current record has been censored:

**COX CENSOR Variable:** the censor variable should be coded in the following format: 1 for non-censored and 0 if the case was censored.

10. **TWEEDIE:** Tweedie distribution loss function used for non-negative targets. Here we model the conditional mean under the assumption of constant dispersion, optimizes quasi-deviance using the following option to set the P-parameter POWER = <p> where POWER is in the range [1.0,5.0].
  - ◆ **Special Cases of the POWER = <p> parameter**
    - P = 1 corresponds to the overly dispersed Poisson regression
    - P = 2 corresponds to the Gamma distribution
    - P = 3 corresponds to the inverse Gaussian distribution
  - ◆ **Target Variable Constraints** (the “target variable” is also called the dependent or response variable)
    - For power in [1.0, 2.0], the response may have exact zeroes.
    - For power in [2.0, 5.0], the response must be strictly positive.
11. **One Tree:** builds a TreeNet model having a single tree and presents the single tree in a CART-like way, providing the tree topology as well as pruning and splitting details.

## Classification Loss Functions

Setting the Target Type to be “Classification” in the [Model Tab section](#) changes the types of loss functions **and they are show below**.



**Classification/Logistic Binary:** binary (two classes) or multinomial (more than two classes) classification. SPM will automatically choose between binary or multinomial classification.

**One Tree:** builds a TreeNet model having a single tree and presents the single tree in a CART-like way, providing the tree topology as well as pruning and splitting details. This combines the natural interpretability of CART trees with the extreme speed, loss functions and sampling options of TreeNet.

**Differential Lift modeling-** builds a differential lift model using the TreeNet engine. A binary response variable and a binary 0/1 treatment variable are required.

**Random Forests-** build a Random Forest using the TreeNet engine. This allows you to see partial dependency plots for the Random Forest model and build RuleLearner and ISLE models for Random Forests.

### Criterion Determining Number of Tree Optimal for Regression Model

The screenshot shows the 'Model Setup' dialog box with the 'TreeNet' tab selected. The 'TreeNet Options' section is expanded, and the 'Criterion Determining Number of Trees Optimal for Regression Model' is highlighted with an orange box. This section contains three radio button options: 'Mean Squared Error (MSE)' (selected), 'Mean Absolute Deviation (MAD)', and 'Mean Absolute Percentage Error (MAPE)'. To the right of these options are two numeric input fields: 'LAD Quantile' set to 0.50 and 'Huber-M Breakdown' set to 0.90. Below this section are several other configuration areas: 'Overfitting Protection' with 'Learn Rate' set to 'Auto' and 'Subsample fraction' set to 0.50; 'Model Randomization' with checkboxes for 'Predictors per node' (8), 'Predictors per tree' (8), 'Vary tree sizes randomly (As Poisson)', and 'Sample With Replacement'; 'Limits' with 'Number of trees to build' (200), 'Maximum nodes per tree' (6), 'Maximal Tree Depth' (100000), and 'Terminal node Minimum' (Cases, 10); and 'Influence trimming speed-up' with 'Total Influence Fraction' (0.10). At the bottom, there are buttons for 'Std. Defaults', 'Save Defaults', and 'Recall Defaults'. The 'Automatic Best Predictor Discovery' section is set to 'Off'. The 'After Building a Model' section has a 'Save Grove...' button. The 'Number of Predictors in Model' is set to 16, and the 'Analysis Engine' is set to 'TreeNet Gradient Boosting Machine'. At the very bottom are 'Cancel', 'Continue', and 'Start' buttons.

### Criterion Determining Number of Trees Optimal for Regression Model

 TREENET OPTIMAL= MSE | MAD | MAPE

When the Target Type is set to “Regression” you still have three ways to evaluate model performance: mean squared error (MSE), mean absolute deviation (MAD), and mean absolute percentage error (MAPE).

The sequence of trees generated by TreeNet during the model-building phase is determined by control parameters such as the number of nodes and the learn rate. The model selection criterion determines the size of the model extracted from this sequence.

## Criterion Determining Number of Tree Optimal for Logistic Model

The screenshot shows the 'Model Setup' dialog box with the 'TreeNet' tab selected. The 'TreeNet Options' section is expanded, and the 'Criterion Determining Number of Trees Optimal for Logistic Model' section is highlighted with an orange border. In this section, the 'Cross Entropy (Likelihood)' radio button is selected. Other options include 'ROC area', 'Lift in given proportion of' (set to 0.100), and 'Classification Accuracy' (with a sub-field 'Assign class if probability exceeds' set to 0.500). Below this, there are sections for 'Overfitting Protection' (Learn Rate: Auto, Subsample fraction: 0.50), 'Model Randomization' (Predictors per node: 6, Predictors per tree: 6, Vary tree sizes randomly (As Poisson), Sample With Replacement), and 'Limits' (Number of trees to build: 200, Maximum nodes per tree: 6, Maximal Tree Depth: 100000, Terminal node Minimum: Cases, 10). At the bottom, there are buttons for 'Std. Defaults', 'Save Defaults', and 'Recall Defaults'. The 'Automatic Best Predictor Discovery' section is set to 'Off'. The 'Number of Predictors in Model' is set to 12. The 'After Building a Model' section has a 'Save Grove...' button. The 'Analysis Engine' is set to 'TreeNet Gradient Boosting Machine'. At the bottom right are 'Cancel', 'Continue', and 'Start' buttons.

### Criterion Determining Number of Trees Optimal for Logistic Model Model

```
TREENET OPTIMAL= AVGLL|LIFT|ROC|MISCLASS
```

```
TREENET LTHRESHOLD = .1
```

```
TREENET CTHRESHOLD = .5
```

When the Target Type is set to “Classification” you have four ways to evaluate model performance: negative average log likelihood (Cross Entropy), area under the ROC curve (ROC area), lift (Lift in given proportion of), and the misclassification rate (Classification Accuracy; note that when this option is selected, the user can specify the probability cutoff which is labeled as “Assign class if probability exceeds.”).

The sequence of trees generated by TreeNet during the model-building phase is determined by control parameters such as the number of nodes and the learn rate. The model selection criterion determines the size of the model extracted from this sequence.

## Learn Rate & Subsample Fraction

The screenshot shows the 'Model Setup' dialog box with the 'TreeNet' tab selected. The 'TreeNet Options' section is expanded, showing various settings. The 'Learn Rate' is highlighted in red and set to 'Auto'. The 'Subsample fraction' is highlighted in blue and set to '0.50'. Other settings include 'TreeNet Loss Function' set to 'Huber-M', 'Criterion Determining Number of Trees Optimal for Regression Model' set to 'Mean Squared Error (MSE)', and 'Limits' set to 'Number of trees to build: 200', 'Maximum nodes per tree: 6', and 'Maximal Tree Depth: 100000'.

### Overfitting Protection: Learn Rate

TREENET LEARNRATE=<x | AUTO>

Controls the rate at which the model is updated after each training stage ([see step 4 in the Gradient Boosting Algorithm section](#)). This is also referred to as “shrinkage” in Friedman’s original articles. The default is AUTO, and the allowable range is 0.0001 to 1.0 inclusive. AUTO is calculated as follows:

$$AUTO \text{ value} = \max(0.01, 0.1 * \min(1, nl/10000)) \text{ where } nl = \text{number of LEARN records.}$$

- ☛ This default uses very slow learn rates for small data sets and uses 0.10 for all data sets with more than 10,000 records. “Slow learn rates” are learn rates that have a small value. They are referred to as “slow” because smaller learning rates require more trees for the model to converge which results in longer runtimes (i.e. set the value for the “Number of trees to build” setting to be larger if you set the learning rate to be smaller). We recommend that you experiment with different learn rates, especially rates slower than the default for larger data sets. Values much smaller than .01 significantly slow down the learning process and might be reserved for overnight runs.
- ☛ High learn rates and especially values close to 1.0 typically result in overfitted models with poor performance.

### Overfitting Protection: Subsample Fraction

TREENET SUBSAMPLE=<x>

A new random draw from the learn sample is conducted at each cycle. This setting controls the proportion of data used for learning at each cycle (the sampling is conducted without replacement; [see step 2 in the Gradient Boosting Algorithm section](#)). This not only speeds up the modeling time, but also guards against overfitting and explains occasional minor local increases in the learn error curve as a TreeNet run progresses. The default is 0.5 (half of the learn sample is used at each modeling iteration), and the allowable range is 0.01 to 1.0 inclusive.

- ✓ It may be necessary to use values greater than the .5 default with small training files. Unlike the learn rate, using a sampling rate of 1.0 is not necessarily catastrophic, but values less than 1.0 are still

strongly recommended. You may need to experiment to determine the best rate. Sampling rates that are too small can hurt accuracy substantially while yielding no benefits other than speed.

## Predictors per node & Predictors per tree

### Model Randomization: Predictors per node (RF Style predictor selection)

TREENET PREDS=<n>

In the original TreeNet approach, all predictors would be tried as potential splitters in each node as a tree is being built. This is identical in spirit to the conventional CART process. However, in developing the Random Forests algorithm Leo Breiman demonstrated that additional benefits may be acquired if you use a randomly selected subset of predictors as potential splitters during the tree construction process. A new subset of predictors is randomly selected at each node and then used in searching for the optimal split value. This further reduces the mutual correlation structure of the ensemble and improves its predictive power.

The RF Style predictor control incorporates this approach into the TreeNet algorithm. Now you can set the number of predictors to be sampled at each node during TreeNet model building and explore its impact on the resulting performance.

- ✓ Breiman suggested the square root of the total number of predictors as a good starting point for this control.
- ✓ By default, this control is set to 0 which allows all predictors to be sampled, thus, restoring the original TreeNet approach.

### Model Randomization: Predictors Per Tree

TREENET TPRED=<n>

Specifies the number of predictors that are randomly selected as candidate splitters for each tree as opposed to each node (see the “Predictors Per Node” entry directly above). A set of randomly selected variables is chosen and all splits in that tree are made using only those variables that are randomly selected. For the next tree a different set of randomly selected variables is chosen and this process continues.



## Vary tree sizes randomly & Sample with Replacement

The screenshot shows the 'Model Setup' dialog box with the 'TreeNet' tab selected. The 'TreeNet Options' section includes:

- TreeNet Loss Function:** Huber-M
- Criterion Determining Number of Trees Optimal for Regression Model:** Mean Squared Error (MSE) (selected), Mean Absolute Deviation (MAD), Mean Absolute Percentage Error (MAPE)
- Overfitting Protection:** Learn Rate: Auto, Subsample fraction: 0.50
- Model Randomization:** Predictors per node: 8, Predictors per tree: 8,  Vary tree sizes randomly (As Poisson),  Sample With Replacement
- Limits:** Maximum nodes per tree: 6 (highlighted in purple), Maximal Tree Depth: 100000, Terminal node Minimum: Cases, 10
- Influence trimming speed-up:** Total Influence fraction: 0.10
- Automatic Best Predictor Discovery:** Off (selected), Discover only, Discover and run (Maximum variables for each class: 8)
- After Building a Model:** Save Grove...
- Analysis Engine:** TreeNet Gradient Boosting Machine
- Number of Predictors in Model:** 16

### Model Randomization: Vary tree sizes randomly (As Poisson)

TREENET RNODES=<YES | NO>

Remember that we control the size of each tree (this is the number of terminal nodes in a tree; see [Step 3 in Gradient Boosting Algorithm section](#)). Jerome Friedman recently suggested that fixing the number of nodes for each tree in the TreeNet process may be too rigid in some contexts. For example, the RuleLearner® methodology aimed at extracting useful rule-sets from a TreeNet model may benefit from trees having varying size as the TreeNet model develops.

When the box is checked, TreeNet will make a Poisson draw with the mean value taken from the “Maximum nodes per tree” control from the TreeNet tab (purple rectangle above; in the picture above we would take a random number from a Poisson distribution with a mean of 6). The resulting value will be used as the actual requested size of the next tree to be built. Thus, as the TreeNet process develops, each tree’s size will vary in terms of the number of terminal nodes.

### Model Randomization: Sample With Replacement

TREENET BSAMPLE=<YES | NO>

Remember that each tree is constructed using a random sample (see [Step 2 in the Gradient Boosting Algorithm section](#)). Clicking this checkbox means that a bootstrap sample (i.e. a sample with replacement) is taken at each iteration. The default value is NO (i.e. unchecked which means that the sampling is conducted without replacement).

## Number of Trees & Maximum nodes per tree

### Limits: Number of Trees to Build

TREENET TREES=<n>

Specifies the number of trees (= number of iterations) in the model. The default is 200. This is the “M” that is specified in the [Gradient Boosting Algorithm section](#).

- ✓ You should expect to grow hundreds or even thousands of trees to get a maximum performance model.
- ✓ The default setting for 200 trees is designed to facilitate interactive model selection and provide very rapid feedback. Once you have a good feel for your data and how you want to tackle your modeling effort, be prepared to allow for 500, 1000, 2000 or more trees.
- ✓ Be sure to use a slow learn rate when growing many trees.
- ✓ If the “optimal” model contains close to the maximum number of trees allowed (e.g., more than ¾ of the maximum), consider growing more trees.

### Limits: Maximum Nodes Per Tree

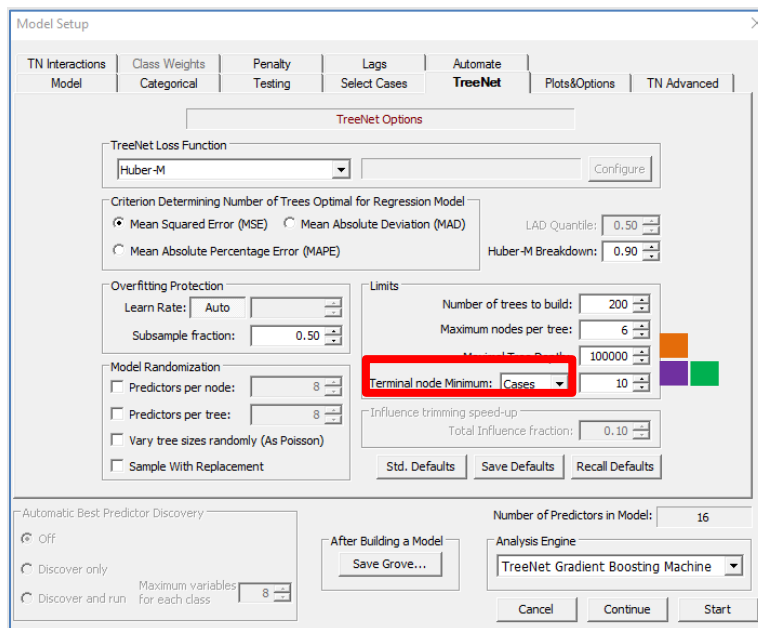
TREENET NODES=<n>

Specifies the maximum number of terminal nodes in each tree. The default is 6. This is the “J” that is specified in [Step 3 in the Gradient Boosting Algorithm section](#). As this number increases, TreeNet’s ability to model interactions increases (more than two nodes are required to detect interactions and the default six-node tree appears to do an excellent job)

- ✓ You can grow TreeNet models made up of two-node trees. These tiny trees (also known as “stumps”) can be surprisingly powerful and are well worth experimenting with.
- ✓ Two-node trees contain only a single variable and a single split. Because only one variable is ever involved in a model update, the models cannot detect interactions and thus can be described as main-effects additive models.
- ☛ In the presence of missing values in the learn sample, TreeNet may choose to build more nodes in a tree to accommodate missing value pre-splits.

- TreeNet may also choose to build smaller trees when the larger trees can't be constructed due to sample size or other limitations.
- Nothing prevents you from requesting trees with quite a few nodes (e.g., more than 12) but doing so is likely to undermine TreeNet's slow-learning strategy. Large trees can also place unnecessary burdens on your computer's memory resources.

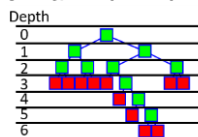
## Max Tree Depth & Terminal Node Minimum



### Limits: Maximum Tree Depth

TREENET DEPTH=<n>

Specifies the maximum depth for each tree. Note that the *Number of Terminal Nodes*  $\geq 2^{\text{Depth}}$ . The default value is 100,000. Note that the setting for the maximum number of terminal nodes (discussed directly above) interferes with this setting: as an example, the default setting for the maximum number of terminal nodes is 6, so with the default settings the tree cannot go beyond 6 terminal nodes. Consider the following CART tree with a maximum depth of 6 (Note: there are 11 terminal nodes so even if the depth were set to 6 this tree would not be possible because the maximum number of terminal nodes is 11).



### Limits: Terminal Node Minimum: Cases

TREENET MINCHILD=<n>

Controls how small individual terminal nodes are allowed to be when the **Terminal node Minimum control** is set to “Cases.” In our example, the value setting of 10 (which is the default value) indicates that only terminal nodes that have more than 10 observations can be in the model. Setting this control to larger values may result to smoother models; however, past a certain point, the model may start experiencing the loss of accuracy. The best values for this control are usually determined experimentally.

- ✓ When working with small training samples it may be vital to lower this setting to five or even three.

### Limits: Terminal Node Minimum: Hessian

TREENET MHES=<v>

When the **Terminal node Minimum control** is set to “Hessian,” this specifies the minimum value for the hessian in each terminal node for NEWTON models (also known as RBOOST models). The default value is 1.0 and works in conjunction with MINCHILD (see Terminal Node Minimum: Cases directly above).

## Influence trimming speed-up

The screenshot shows the 'Model Setup' dialog box with the 'Automate TreeNet' tab selected. The 'TreeNet Options' section is expanded, showing various configuration options. The 'Influence trimming speed-up' section is highlighted with a red box, indicating the 'Total Influence fraction' is set to 0.10. Other visible settings include 'TreeNet Loss Function' set to 'Classification/Logistic Binary', 'Criterion Determining Number of Trees Optimal for Logistic Model' set to 'Cross Entropy (Likelihood)', 'Learn Rate' set to 'Auto', 'Subsample fraction' set to 1.00, 'Number of trees to build' set to 20000, 'Maximum nodes per tree' set to 30, 'Maximal Tree Depth' set to 100000, and 'Terminal node Minimum' set to 'Cases'. The 'Automatic Best Predictor Discovery' section is set to 'Off', and the 'Analysis Engine' is set to 'TreeNet Gradient Boosting Machine'.

## Total Influence Fraction

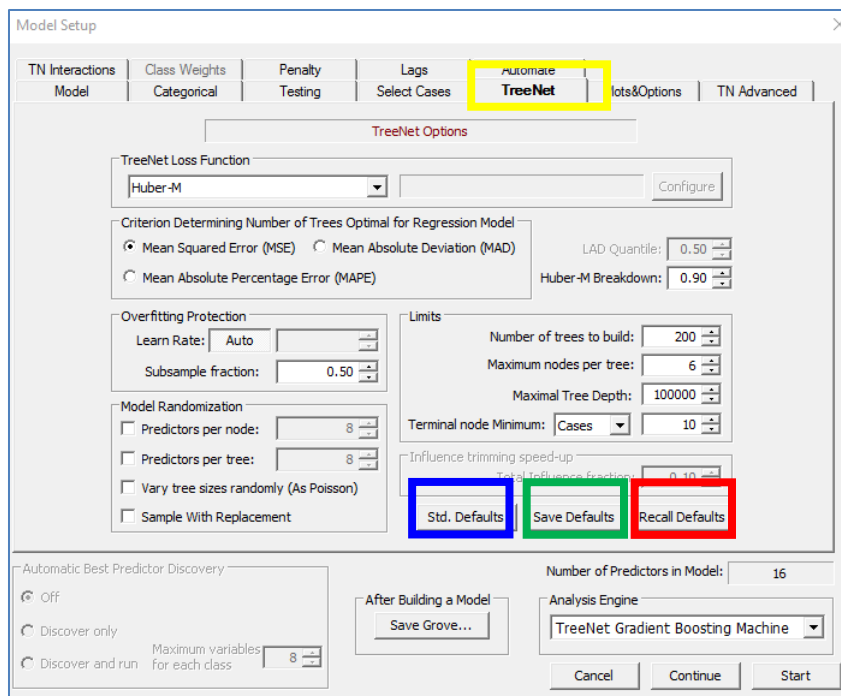
TREENET INFLUENCE=<x>

Classification only. Influence trimming is a form of data selection designed to focus the TreeNet learning process on the most important data records. It is the mechanism by which suspicious data are ignored. There are two types of data records that are dropped due to the influence trimming: those that are far from the decision boundary and classified correctly and those that are far from the decision boundary and misclassified. The former have very small logistic residuals, contribute next to nothing to the model refinements during subsequent iterations and therefore can be safely ignored. The latter have very large logistic residuals, which could be an indication of being outliers or otherwise problematic records with the potential to severely distort model evolution and thus needed to be ignored too.

- ✓ If you have no concerns about the quality of your data you might consider turning this feature off by setting the Total Influence Fraction to 0.
- ✓ It is always worth experimenting with a few values of this factor to see what works best. We recommend settings of 0, 0.1, and even 0.2.

The Influence trimming factor can vary between 0.0 and 0.2, with larger values generally having larger effects.

## Default Buttons



### Defaults buttons

The group of buttons in the lower right corner of the TreeNet tab allows you to specify new defaults for any subsequent TreeNet run. The defaults are saved in the TreeNet initialization file and persist from session to session.

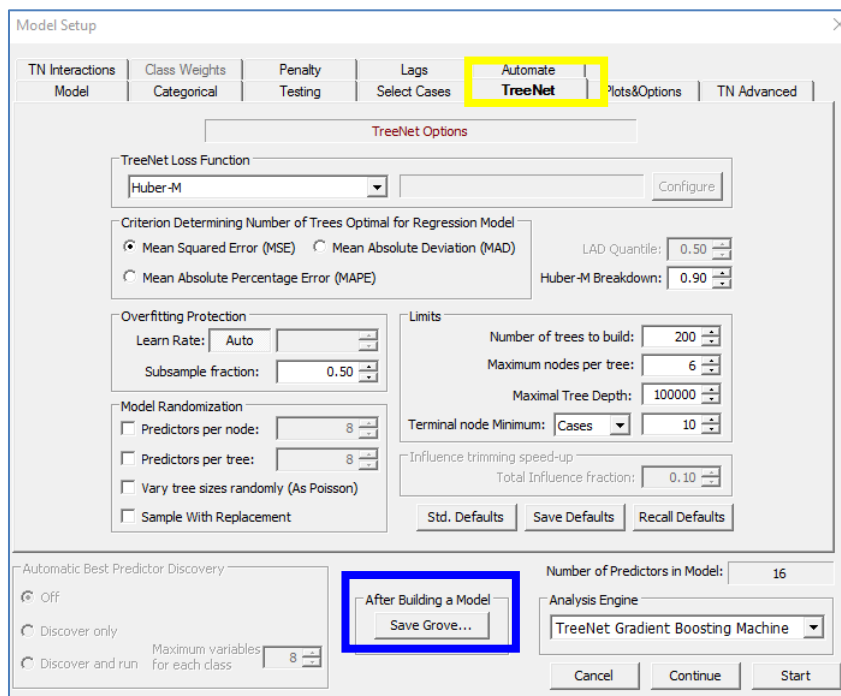
**Std. Defaults** – press this button to restore the original “factory” settings.

**Save Defaults** – press this button to save the current settings as the new defaults.

**Recall Defaults** – press this button to recall previously-saved defaults.

For example, you might decide that your typical TreeNet run should use a 0.01 learn rate, 1000 trees, two nodes (to force additive models), and the least squares loss criterion. Just set these parameters and press **Save Defaults**. In future sessions, TreeNet automatically assumes the new settings unless you manually override them.

## Save Grove... Button



## Save Grove... Button

To save the model after the model has been built, click the Save Grove... button and specify a save location. A *grove* is a special file that stores the model. Groves have a variety of uses: you can view model results without having to rebuild the model (especially useful when the dataset is large), you can send groves to other SPM users so they can open your model in their copy of SPM, and, perhaps most importantly, models that are saved to grove files can be used to generate predictions for datasets (see the Score section for more information).

## Plots & Options Tab

The Plots & Options Tab allows you to set options for creating one and two-variable partial dependency plots (PDPs) that are generated automatically as a part of the TreeNet modeling run (Note: see the [Create Plots...](#) section for details on how to create PDPs after a TreeNet modeling run). PDPs are generated only for variables with a positive variable importance score.

TREENET PLOTS= <YES | NO>, <YES | NO>, <YES | NO>, <YES | NO>

The settings above accomplish the following: Requests **30 one variable partial dependency plots** that are constructed using **500** randomly selected observations), and requests **two variable partial dependency plots** involving the **top 5 variables** that are each constructed using **5000** randomly selected observations. Note: the number of two variable dependency plots created is “5 choose 2” and that the largest number of two variable dependency plots that can be created is set to **500**. The total number of Plots Estimated (**black square above**) is **40** = **30** +  $\binom{5}{2}$

### Warning: Number of PDPs

The user specifies the number of variables involved in creating the partial dependency plots (PDPs) for the one variable PDPs (see **30** above) as well as the number of variables involved in creating the two variable PDPs (see **5** in the picture above). In general, the number of Plots Estimated (**black square above**) is computed using the following formula (refer to the colors in the picture above):

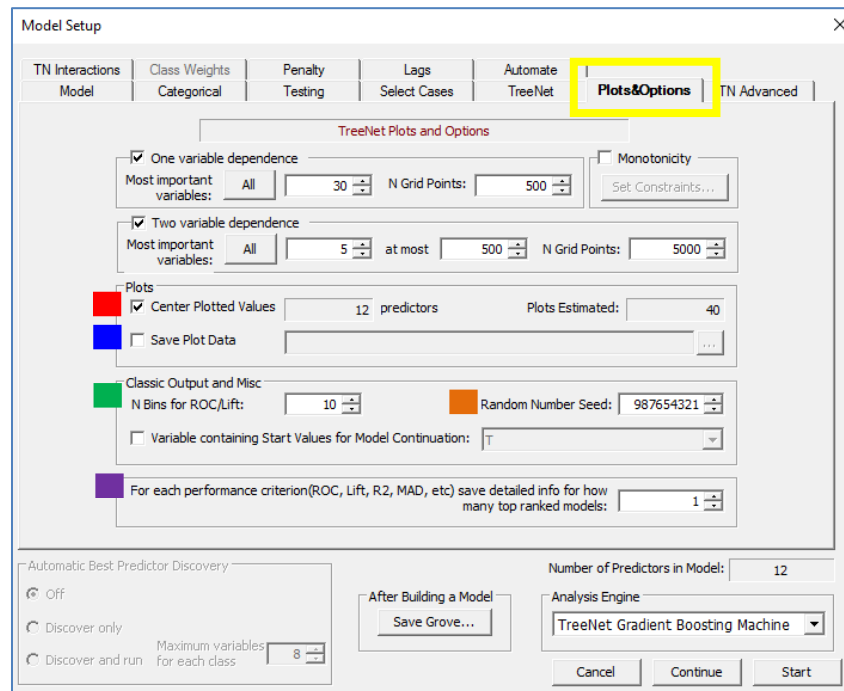
$$\text{Plots Estimated} = \text{Number of Variables for Univariate PDPs} + \binom{\text{Number of Variables For Bivariate PDPs}}{2}$$

The number of plots estimated can become very large when the number of predictor variables is large and the “All” buttons are selected (they are currently not selected; click the button to select). The “**All**” option for the one variable plots requests that a PDP is generated for every variable with a nonzero variable importance score in the model. The “**All**” option for the two variable plots requests that a PDP is generated for every combination of variables with a positive variable importance score. Note that you



can define an **upper limit (in this case the upper limit is 500)** for the number of two variable PDPs generated.

### Other Plotting Options



#### Center Plotted Values

```
TREENET CENTER = <YES|NO>
```

Controls whether the partial dependency plot is centered. The default is YES.

#### Save Plot Data

```
TREENET PF = "C:\Users\Name\plot_data.xml"
```

Specifies an XML-like text file in which TreeNet should store the data upon which the error rate profile, pair plots, and single plots are based.

#### N Bins for ROC/Lift

```
TREENET GB = <n>
```

Specifies the number of bins for ROC and Lift computations. The default value is 10.

#### Random Number Seed

```
TREENET SEED = <n>
```

Specifies the seed for the next TreeNet operation. The default is 987654321.

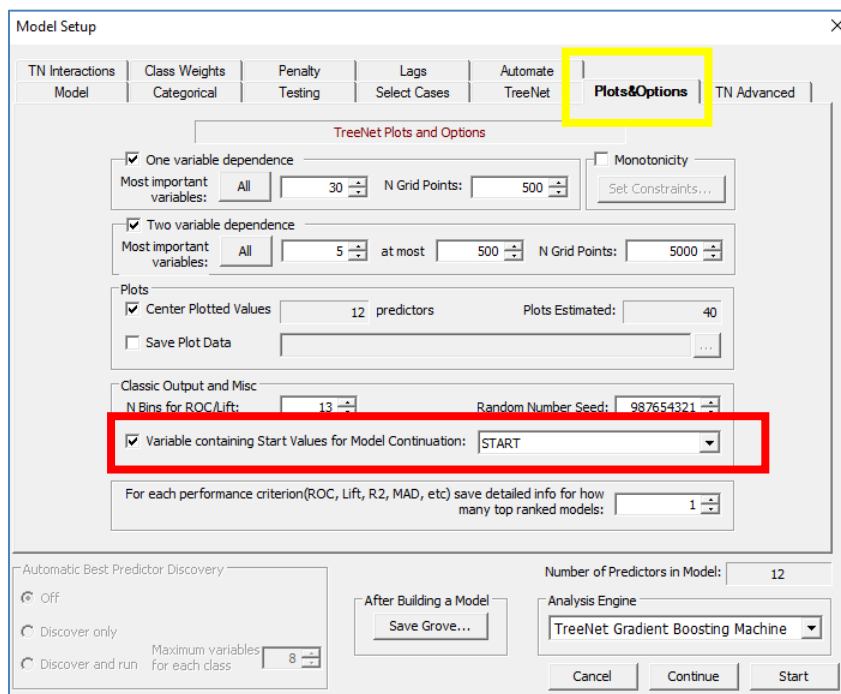
#### For each performance criterion (ROC, Lift, R2, MAD, etc.) save detailed info for how many top ranked models

```
TREENET FR = <n>
```

Controls how many "good models" are identified to generate complete results including confusion matrices, gains, ROC, and threshold tables. The default is 1, which means the best single model for each available optimality criterion is tracked (i.e., 4 for binary classification models and 2 for multinomial classification and regression models). If FR is set to 100 and the model is a binary classification model (with four optimality criteria), then TreeNet will track the best 100 models equally across the four optimality criteria (about 25 models per criterion), accounting for models which are considered good by several criteria. A high value for FR results in a potentially lengthy "reconciliation" after the tree building is done and increases memory requirements of TreeNet as well as the size of the grove file, but it makes full

results available for a greater number of models that are of potential interest to the analyst. We recommend keeping this setting well below 10 for routine modeling.

### Variable containing Start Values for Model Continuation



### Variable Containing Start Values for Model Continuation

`TREENET INIT = <Variable_Name>`

Specifies a variable that contains “start values” for the model. This can be a constant or a record specific adjustment. This is very useful in hybrid modeling approaches where TreeNet model is developed as a follow up model to previously constructed model. For example, a linear regression model (captures main linear trends) may be followed by TreeNet to explore departures from linearity and also interactions.

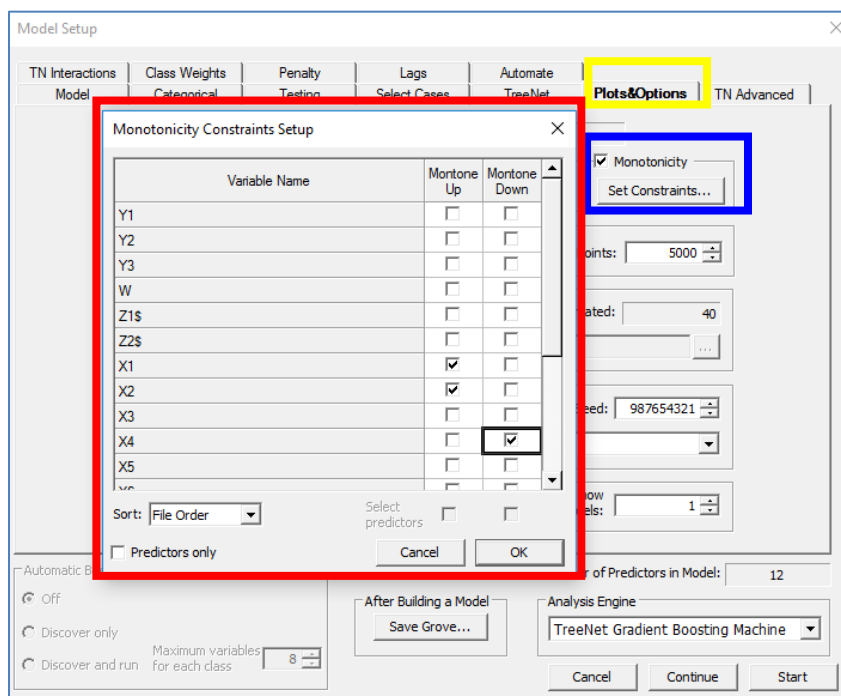
A typical use for the INIT variable is to start a new model from the end-point of a previous TreeNet model. The INIT variable could be derived from a SCORE operation in SPM or predictions saved during the model building (i.e. this is the variable called “RESPONSE” when you write the predictions to a file).

INIT values could also be derived from other models like logistic regression, but the logit score (i.e. the log odds) should be multiplied by 0.5 to conform to the TreeNet scale (gradient boosted trees for binary classification operate on the one-half log-odds scale). INIT values for a logistic loss model must be in the range [-10.0, 10.0].

INIT values for the following losses: POISSON, GAMMA, NEGBIN, TWEEDIE, COX must be in the range [-30.0, 30.0] to reflect the logarithmic scale of the raw internal TN response.

To reset the INIT option so that no start values are used, use INIT = 0.

## Monotonicity



### Monotonicity

MONOTONE UP = <VAR1>, <VAR2>, ...

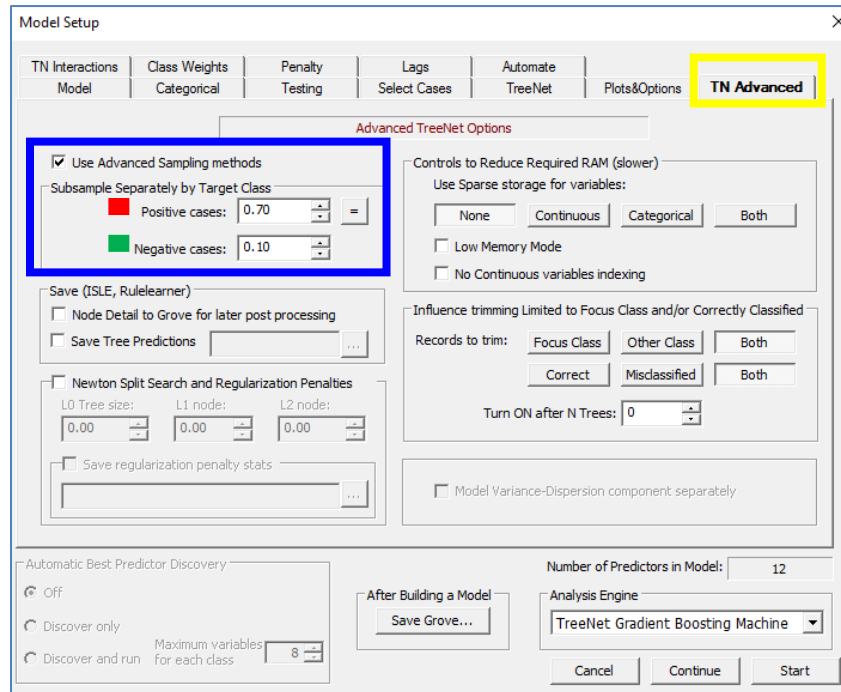
MONOTONE DOWN = <VAR1>, <VAR2>, ...

Constrains the direction of the impact of a predictor in a TreeNet gradient boosting model. For instance, you can constrain a variable to have a monotonically increasing or decreasing relationship with the target variable. You can verify this effect of this constraint by examining the partial dependency plot for the specified variable(s) (these plots will be monotonically increasing or decreasing).

To set the monotonicity control, **click the checkbox next to Monotonicity**, and then specify the desired variables to be “Monotone Up” or “Monotone Down” by clicking the appropriate checkbox (**red rectangle above**). If a variable is not specified to be “Monotone Up” or “Monotone Down,” then that variable will not be constrained.

## TN Advanced Tab

### Subsample Separately by Target Class



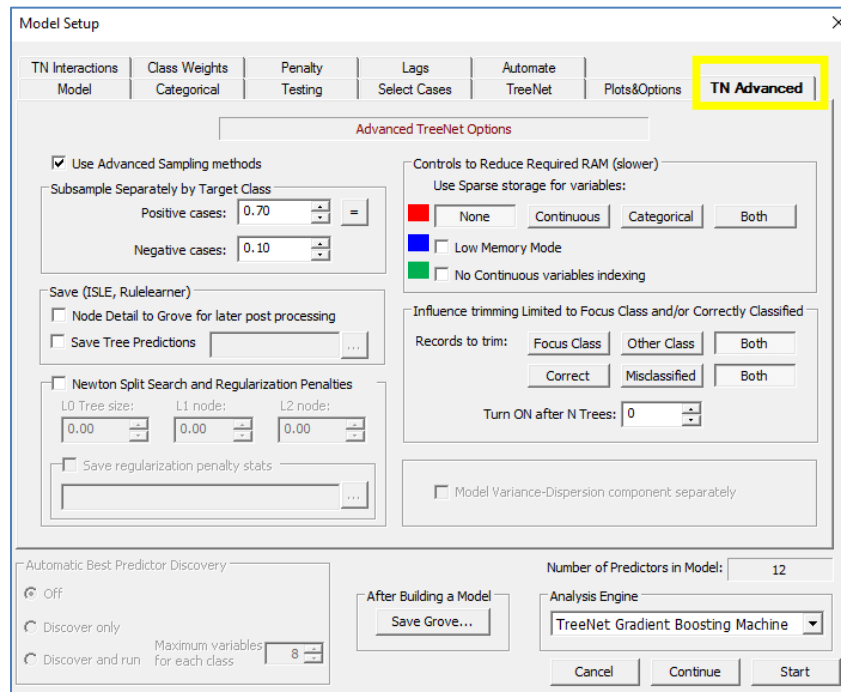
### Use Advanced Sampling Methods

`TREENET SUB1 = <p1>, SUB0 = <p2>`

By default, a subsampling fraction of .5 is used for all records in the dataset (see the “Subsampling Fraction” on the “TreeNet” tab for more information). You can specify separate sampling rates for the target classes in binary classification models only. This is useful if one class is rarer than the other; you may want to subsample 100% of the rare class to ensure representation in the model.

In the **Use Advanced Sampling methods** section in the picture above we set the **sampling rate of the positive cases to be .7** whereas the **sampling rate for the negative cases to be .10**. The picture above corresponds to the following commands: `TREENET SUB1 = .7, SUB0 = .1`

## Controls to Reduce Required RAM (slower)



### Use Sparse Storage for Variables

TREENET SPARSE = <CONT|CAT|BOTH>

Specify sparse storage for continuous variables (CONT), categorical variables (CAT), or both continuous and categorical variables (BOTH).

### Low Memory Mode

TREENET LOWMEMORY = <YES|NO>

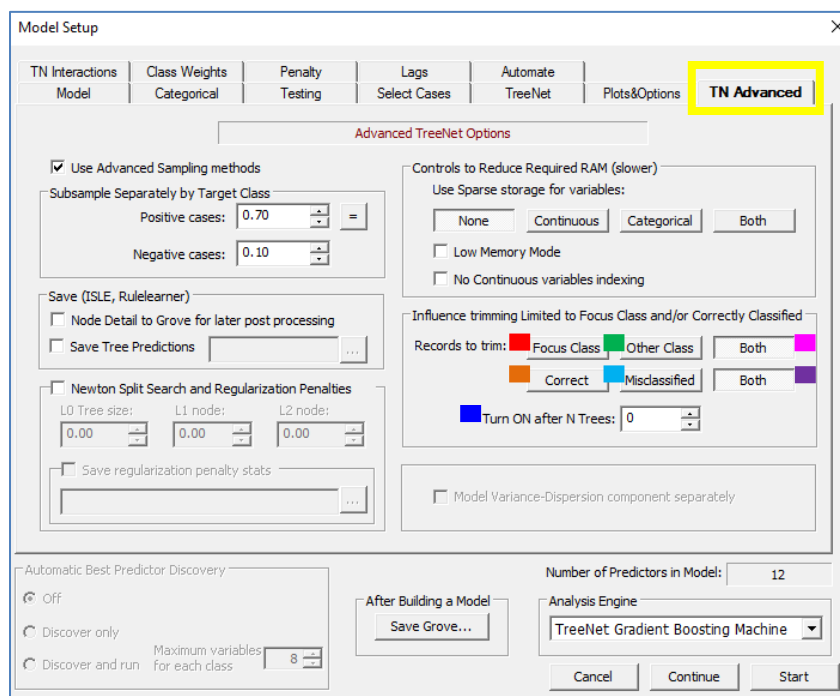
Requests 33% smaller memory footprint during model building, which may result in longer modeling times.

### No Continuous Variables Indexing


TREENET INDEX = <YES|NO>

Controls whether a copy of the continuous variable index is enabled or disabled. The default is YES. When an index is used computations are at their fastest but total memory consumption is 3X (or 2X when LOWMEMORY = YES). Note that categorical predictors never use indexing.

## Influence Trimming Limited to Focus Class and/or Correctly Classified



## Influence Trimming to Focus Class Misclassification and/or Correctly Classified

 TREENET TRIMGOOD=<YES | NO>, TRIMBAD=<YES | NO>, TRIMPOS=<YES | NO>, TRIMNEG=<YES | NO>, TRIMAFTER=<N>

The following options provide more specific controls over when, where, and what to trim:

**TRIMPOS** turns on influence trimming for the FOCUS class records (the default is YES, applies to LOSS=LOGIT only).

**TRIMNEG** turns on influence trimming for the OTHER class records (the default is YES, applies to LOSS=LOGIT only).

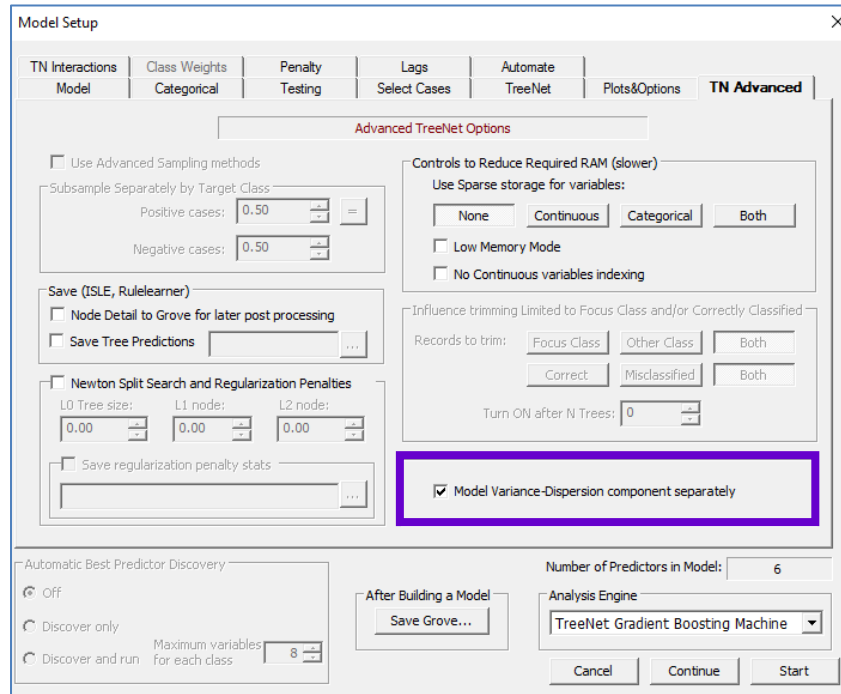
**TRIMGOOD** turns on influence trimming for CORRECTLY classified records (the default is YES).

**TRIMBAD** turns on influence trimming for INCORRECTLY classified records (the default YES).

**TRIMAFTER** turns on influence trimming after <N> trees are grown.

The default setting is that **both** the “Focus” class and the “Other” class are trimmed as well as **both** the “Correct” and “Misclassified”.

## Model Variance-Dispersion Component Separately



### Variance-Dispersion Component

 TN SIGMA=<YES | NO>

Activates modeling of the variance component and the mean component for the key regression losses Least Squares (LS), Least Absolute Deviation (LAD), Huber-M, and the two-parameter losses Gamma and Negative Binomial. SIGMA=YES models both mean/location and variance/dispersion of the response surface, potentially doubling modeling time, but usually producing smoother surfaces.

## TN Interactions Tab

### Interaction Control Language (ICL)

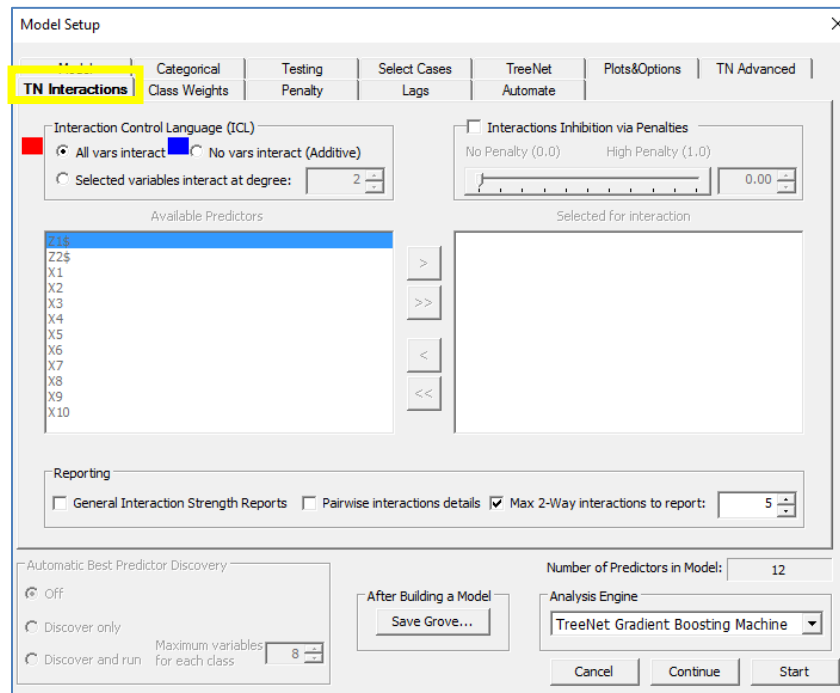
Interaction reporting and enforcing is one of the most recent additions to the TreeNet engine. Most controls and reports are available through command line with some features supported at the GUI level. The reference for the interaction statistics to be described is “Predictive Learning via Rule Ensembles” by Friedman and Popescu (2005). The interaction measures discussed below are sometimes referred to as “H-squared statistics.”

The main machinery behind interaction reporting is based on the comparison between a genuine bivariate plot (where variables are allowed to interact) and an additive combination of the two corresponding univariate plots. By gauging the difference between the two response surfaces you can measure the strength of interaction effect for the given pair of variables. The entire process is automated to quickly identify variables and pairs of variables most suspected to have strong interactions.

The core idea behind interaction enforcing is that variables can only interact within individual trees. Therefore, providing fine controls over how individual variables can enter into a tree is the key to allowing or disallowing different interactions. For example, requesting 2-node trees (one split per tree) effectively disables all interactions because any TreeNet model is always additive across trees. This approach can be further expanded to multiple node trees by controlling which variables are allowed to jointly appear within individual branches.

The screenshot shows the 'Model Setup' dialog box with the 'TN Interactions' tab selected. The 'Interaction Control Language (ICL)' section has 'All vars interact' selected. The 'Interactions Inhibition via Penalties' section has a slider set to 0.00. The 'Reporting' section has 'Max 2-Way interactions to report' set to 5. The 'Automatic Best Predictor Discovery' section has 'Off' selected. The 'Number of Predictors in Model' is set to 12. The 'Analysis Engine' is set to 'TreeNet Gradient Boosting Machine'. The 'Save Grove...' button is visible under 'After Building a Model'.





SPM offers several straightforward ways to control interactions during the model-building process. In the **TN Interactions** tab of the **Model Setup** window one can choose:


#### All variables interact

Click to let all variables interact with each other (i.e. the model is not restricted in any way). This is the default setting.

#### No variables interact (Additive)

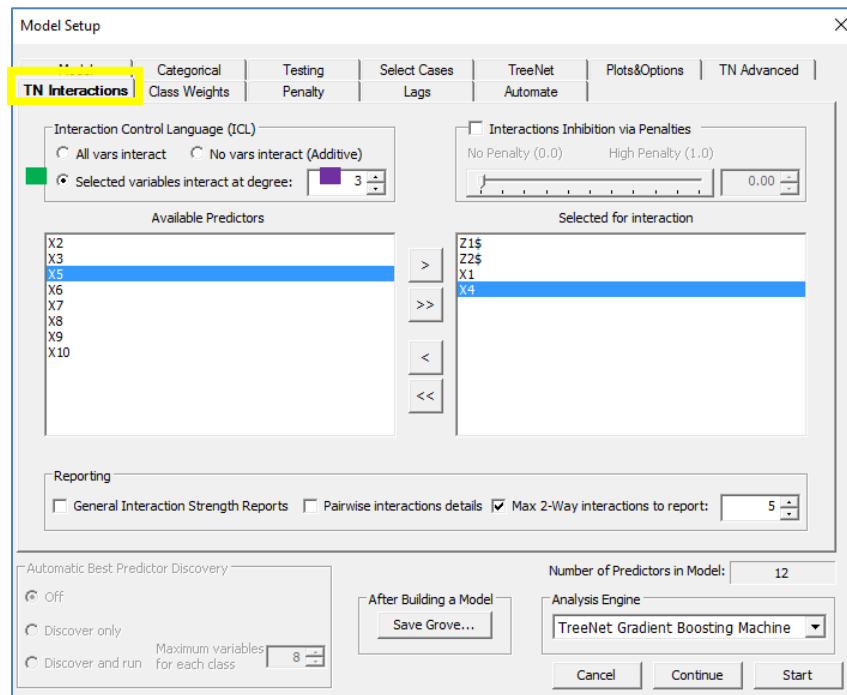
Click to ensure that none of the variables interact. In this case, each tree is built using only one variable. Note that this contrasts with a boosted stumps model: a boosted stump model is a gradient boosted tree model where each of the trees have only one split (i.e. only one variable is used so there are no interactions between variables). Additive trees use only one variable, but, in contrast to boosted stumps, can have more than one split in each of the trees.

### Selected variables interact

 ICL ALLOW <variable list> /degree

The user-selected group of variables can interact among themselves up to the specified degree.

In the following example, we only allowed up to a 3-way interaction between Z1\$, Z2\$, X1, and X4.




This means that in the TreeNet model we can have the following interactions only (the other variables are forced to enter the model additively):

1. Z1\$ and Z2\$
2. Z1\$ and X1
3. Z1\$ and X4
4. Z2\$ and X1
5. Z2\$ and X4
6. X1 and X4
7. Z1\$, Z2\$, and X1
8. Z1\$, Z2\$, and X4
9. Z1\$, X1, and X4
10. Z2\$, X1, X4


The corresponding commands associated with the settings in the picture above are the following:

```
ICL ALLOW = Z1$, Z2$, X1, X4 / 3
```

The command line offers greater control over the specific structure of interactions that are allowed or disallowed in a TreeNet model. Here we provide brief highlights on the subject, the complete description of the command listed below can be found in the Command Reference guide.


 ICL ADDITIVE = <variable list>

This command has the highest priority and prevents any and all interactions in the TreeNet among the variables listed on the command. In the presence of any other ICL command (see below), the remaining variables are allowed to interact as usual.

 ICL ALLOW <variable list> /degree

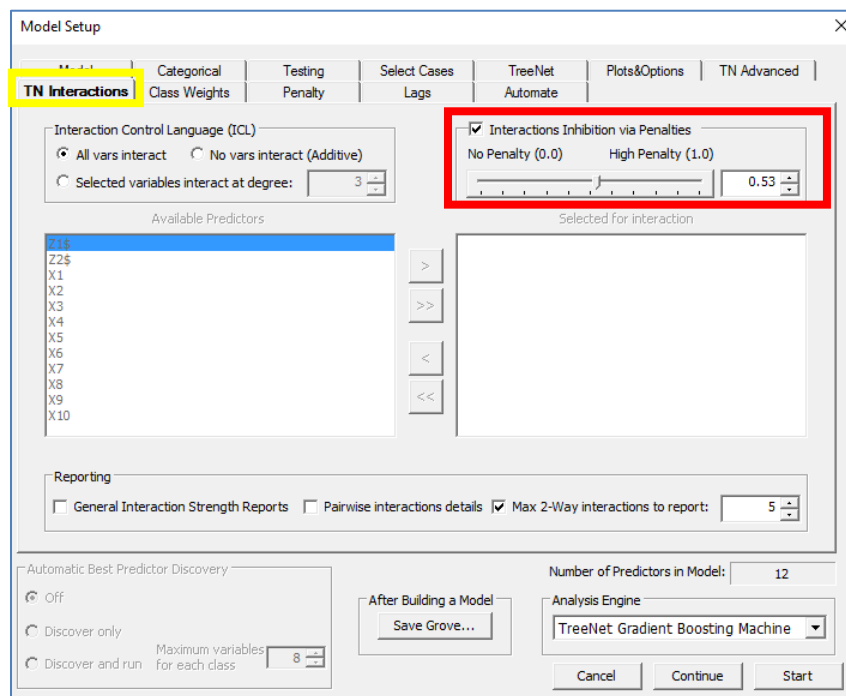
This command allows the listed variables to interact only among themselves, provided that they do not occur on the ADDITIVE list. Multiple ALLOW commands with possibly overlapping lists of variables are admissible. The variables are allowed to interact only if they occur on one or more ALLOW lists and do not occur on the ADDITIVE list,

- ✓ The best practice is either to define an ADDITIVE list or defining a collection of ALLOW lists but never both. The former case is equivalent to a pair of complementing ADDITIVE and ALLOW statements, while the latter is equivalent to having an implied complementing ADDITIVE command.

 ICL DISALLOW <variable list> /degree

This command only works “inside” of an existing ALLOW command. It further refines additional combinations of variables that are specifically not allowed to interact among themselves, even though they are allowed to interact with any other member of the ALLOW group.

## TN Interactions Tab: Interaction Inhibition via Penalties



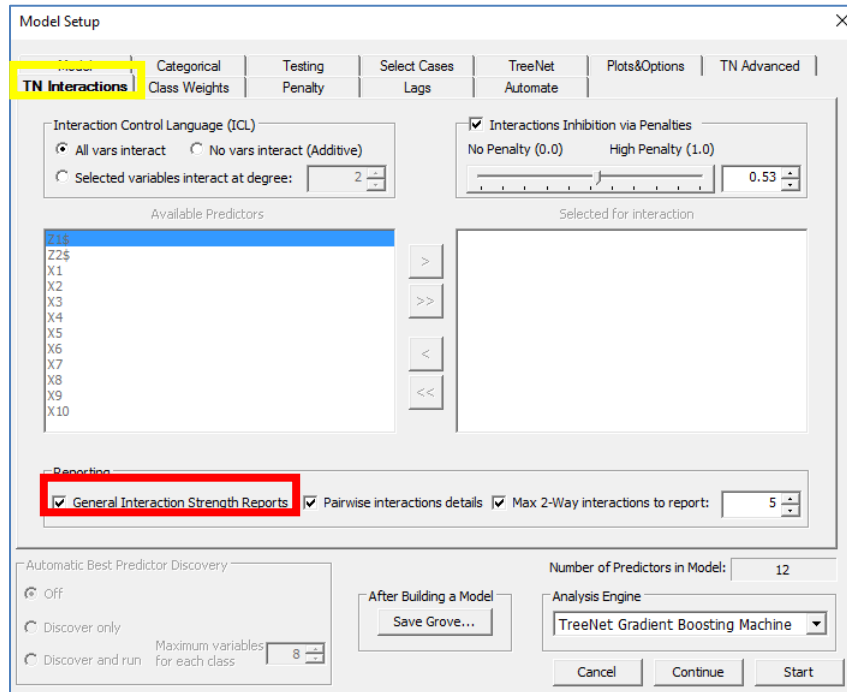
### Interaction Inhibition via Penalties

 ICL PENALTY = <v>

Imposes a penalty on introducing a new variable into a branch of the tree under construction. If X1 and X2 are already on a given branch, then the penalty causes TreeNet to continue using these two variables to

extend the branch. This penalty is intended to inhibit interactions in general. The corresponding commands in the picture above (**red rectangle above**) are `ICL PENALTY = .53`

## Interaction Statistics



### General Interaction Strength Reports

`TREENET INTER = <YES|NO>`

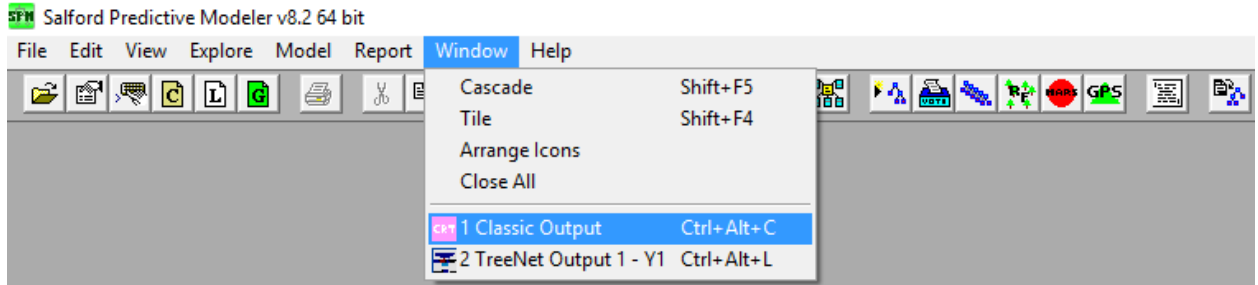
Provides a report concerning the percent of the variance in the predicted response that is accounted for by an interaction between a particular variable and any other variable in the model (this is equation 45 in Section 8.1 in Predictive Learning via Rule Ensembles by Friedman and Popescu, 2005).

- Generating interaction reports consumes approximately same amount of time as generating all partial dependency plots which may be comparable with the model building time and even longer.

The interaction strength report is titled “TreeNet Interactions” and here we will first look at the “Whole Variable Interactions Score” report which is located in the Classic Output Window after running the TreeNet model. Y

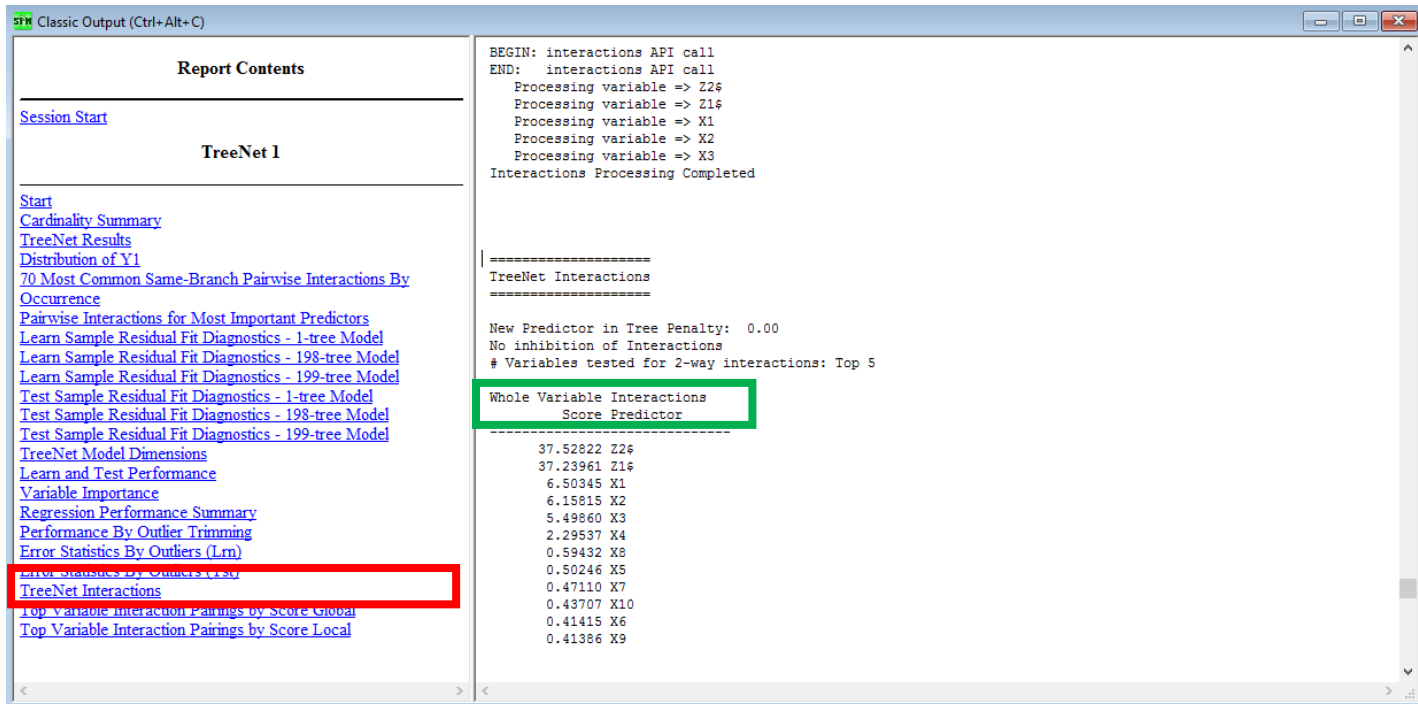
### Viewing the Whole Variable Interaction Score Report:

- After the TreeNet model has finished, click Window > Classic Output



This opens Classic Output Window

- In the Classic Output Window, click the “TreeNet Interactions” link on the left (red rectangle below) to see the “Whole Variable Interactions” report (green rectangle below). See below for information concerning the interpretation.



## Whole Variable Interaction Score Report

```

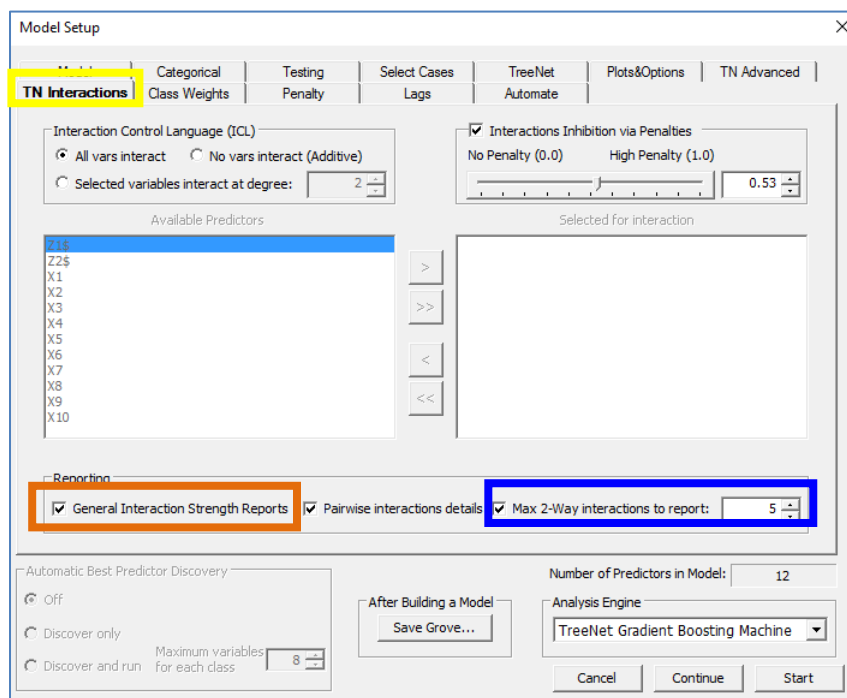
=====
TreeNet Interactions
=====

New Predictor in Tree Penalty: 0.00
No inhibition of Interactions
# Variables tested for 2-way interactions: Top 5

Whole Variable Interactions
Score Predictor
-----
37.52822 Z2$
37.23961 Z1$
6.50245 X1
6.15815 X2
3.49808 X3
2.29537 X4
0.59432 X8
0.50246 X5
0.47110 X7
0.43707 X10
0.41415 X6
0.41386 X9
    
```

This table provides the list of all available predictors sorted according to the suspected overall interaction strength. The interaction strength is on the % scale, which in this example, indicates that 37.52822% of the total variation in the predicted response can be attributed to an interaction of the categorical variable Z2\$ with any other variable in the model. The same interpretation applies to the remaining variables in the list. The report is only approximate and may require further investigation, including trying various interaction enforcing strategies described in the section using the Interaction Control Language (ICL).

### Max 2-Way Interactions to report



### Max 2-Way Interaction to Report

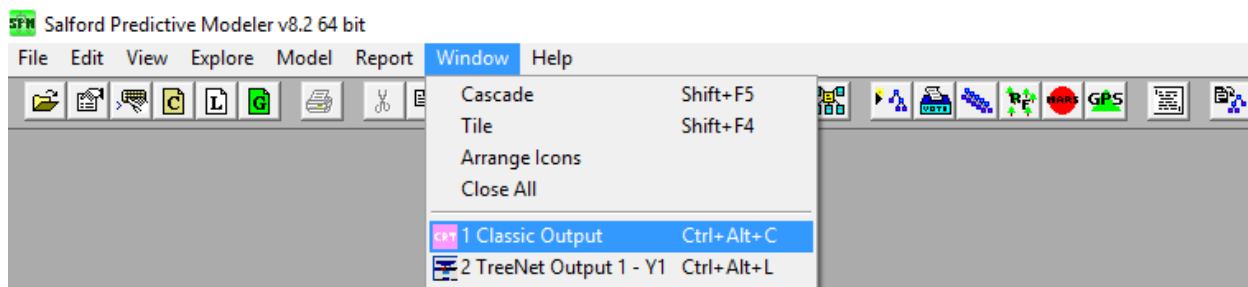
ICL N2WAY = <v>

Specifies a limit on the number of two-way interactions to report after the TreeNet model run finishes and defaults to 5. A value of 5 means that five 2-way interaction reports will be generated. Each interaction report provides an interaction score between the variable of interest and paired with the each of variables in the model (i.e. if X1 is the variable of interest, then there will be a variable interaction score for X1 and X2, X1 and X3, X1 and X4, X1 and Z1\$, and so on; this constitutes 1 of the five two-way interaction reports).

- ✓ Make sure that “General Interaction Strength Reports” (orange rectangle above) is selected in addition to the “Max 2-way interaction to report” option (blue rectangle above)

## Viewing the 2-way Interaction Report

1. After the TreeNet model is finished, click Window > Classic Output



This opens the Classic Output Window.

2. In the Classic Output Window, click the “TreeNet Interactions” link on the left (red rectangle below) and scroll down until you see the “2-way Interaction Stats” report (green rectangle below)

The screenshot shows the 'Classic Output (Ctrl+Alt+C)' window. On the left is a 'Report Contents' sidebar with a tree structure for 'TreeNet 1'. The main area displays 'TreeNet Interactions' and '2-way Interaction Stats'. The '2-way Interaction Stats' section is highlighted with a green box and contains two tables.

**TreeNet Interactions**

New Predictor in Tree Penalty: 0.00  
 No inhibition of Interactions  
 # Variables tested for 2-way interactions: Top 5

Whole Variable Interactions  
 Score Predictor

37.52822	Z2\$
37.23961	Z1\$
6.50345	X1
6.15815	X2
5.49860	X3
2.29537	X4
0.59432	X8
0.50246	X5
0.47110	X7
0.43707	X10
0.41415	X6
0.41386	X9

**2-way Interaction Stats**

Predictor: Z2\$

Score Global	Score Local	Predictor
37.08393	48.75050	Z1\$
3.58071	4.22652	X1
1.33245	2.07347	X3
1.31428	2.05382	X2
1.20089	1.89395	X4
0.11663	0.18888	X10
0.11030	0.17859	X6
0.06480	0.10493	X5
0.03458	0.05599	X9
0.01597	0.02586	X8
0.00000	0.00000	X7

Predictor: Z1\$

Score Global	Score Local	Predictor
37.08393	48.75050	Z2\$
1.24502	1.98278	X1
0.31525	1.04251	X4
0.31161	1.02434	X3
0.26046	0.81912	X2
0.15384	0.59376	X9
0.06906	0.26655	X5
0.05248	0.20266	X8

### Interpreting the 2-way Interaction Report

Here are the top 2 (of 5) requested interaction reports:



```

=====
2-way Interaction Stats
=====

Predictor: Z2$
Score Global Score Local Predictor
-----
37.08393      48.75050 Z1$
3.38071      4.22032 X1
1.33245      2.07347 X3
1.31428      2.05382 X2
1.20089      1.89395 X4
0.11663      0.18888 X10
0.11030      0.17859 X6
0.06480      0.10493 X5
0.03458      0.05599 X9
0.01597      0.02586 X8
0.00000      0.00000 X7

Predictor: Z1$
Score Global Score Local Predictor
-----
37.08393      48.75050 Z2$
1.24502      1.98278 X1
0.31525      1.04251 X4
0.31161      1.02434 X3
0.26046      0.81912 X2
0.15384      0.59376 X9
0.06906      0.26655 X5
0.05248      0.20266 X8
0.04891      0.18884 X10
0.00369      0.01422 X6
0.00001      0.00002 X7

```

- **Score Global** (on the % scale) shows the contribution of the suspected interacting pair normalized to the overall response surface (total variation of the predicted response).
    - A value of zero means that the interaction under consideration has no effect on the overall model.
    - **Example:** 37.08393% of the total variation in the predicted response is accounted for by the interaction between Z2\$ and Z1\$ (see the picture directly above)
    - **Reference:** this metric is described in the last paragraph in Section 8.1 of “Predictive Learning via Rule Ensembles” by Friedman and Popescu. Score Global is equation 44 with the denominator of equation 44 replaced by the denominator used in equation 45 in Section 8.1 in “Predictive Learning via Rule Ensembles” by Friedman and Popescu (2005).
  - **Score Local** (on the % scale) shows the contribution of the suspected interacting pair of variables normalized to the two-variable partial dependency function of the pair of variables under consideration (two-variable partial dependency functions are used to generate the two-variable partial dependency plots)
    - A value of zero means that there is no interaction between the two variables under consideration.
    - **Example:** 48.75050% of the total variation in the joint marginal distribution of Z1\$ and Z2\$ is accounted for by an interaction between Z1\$ and Z2\$ (the rest is accounted for by additive effects for Z1\$ and Z2\$ or random noise)
    - **Reference:** Score Local is equation 44 in Section 8.1 in Predictive Learning via Rule Ensembles by Friedman and Popescu, 2005)
- ✓ Score1 is always smaller than Score2.
- ✓ A small Score1 value combined with large Score2 indicates a strong interaction even though the overall contribution of this pair, compared to the remaining variables, is rather insignificant.

It is worth reminding that all interaction reports are based on the patterns in which variables appear together on the same branch of a tree in the TreeNet model. The reports can be considered as no more than a set of hypotheses because apparent interactions may be due to chance factors or, alternatively, may not be needed to obtain the best possible mode by changing the overall model composition. You can use this report as a starting point to guide the follow up use of the ICL commands (see the next section). One can test whether an interaction is "real" by preventing it from occurring in a new TreeNet model. If imposing such a restriction does not harm model performance, then the interaction is declared to be "spurious." In general, it is best to constrain the models by first restricting the less prominent interactions and gradually imposing more and more restrictions.

### Top Variable Interaction Pairings by Score Global

This report sorts the pairs of interactions using their Score Global value in descending order which is convenient when the number of variables in the model is large.

### Top Variable Interaction Pairings by Score Local

This report sorts the pairs of interactions using their Score Local value in descending order which is convenient when the number of variables in the model is large.

These reports can be accessed by clicking the desired links below (**red rectangle below**) and can be seen in classic output (**green rectangle below** for the Score Global sorted report and the **blue rectangle below** for the Score Local report)

The screenshot shows the 'Classic Output (Ctrl+Alt+C)' window. On the left is a 'Report Contents' sidebar with a list of links. Two links are highlighted with red rectangles: 'Top Variable Interaction Pairings by Score Global' and 'Top Variable Interaction Pairings by Score Local'. The main window displays the output for these reports. The 'Score Global' report is enclosed in a green rectangle, and the 'Score Local' report is enclosed in a blue rectangle. Both reports show a table with columns for Score Global, Score Local, Predictor 1, and Predictor 2. Below each table is a note indicating the sorting criteria.

**Report Contents**

- [Session Start](#)
- TreeNet 1**
- [Start](#)
- [Cardinality Summary](#)
- [TreeNet Results](#)
- [Distribution of Y1](#)
- [70 Most Common Same-Branch Pairwise Interactions By Occurrence](#)
- [Pairwise Interactions for Most Important Predictors](#)
- [Learn Sample Residual Fit Diagnostics - 1-tree Model](#)
- [Learn Sample Residual Fit Diagnostics - 198-tree Model](#)
- [Learn Sample Residual Fit Diagnostics - 199-tree Model](#)
- [Test Sample Residual Fit Diagnostics - 1-tree Model](#)
- [Test Sample Residual Fit Diagnostics - 198-tree Model](#)
- [Test Sample Residual Fit Diagnostics - 199-tree Model](#)
- [TreeNet Model Dimensions](#)
- [Learn and Test Performance](#)
- [Variable Importance](#)
- [Regression Performance Summary](#)
- [Performance By Outlier Trimming](#)
- [Error Statistics By Outliers \(Lrn\)](#)
- [Error Statistics By Outliers \(Tst\)](#)
- [TreeNet Interactions](#)
- Top Variable Interaction Pairings by Score Global**
- Top Variable Interaction Pairings by Score Local**

**Top Variable Interaction Pairings by Score Global**

Score Global	Score Local	Predictor 1	Predictor 2
37.08393	48.75050	Z16	Z26
4.18081	17.13580	X2	X3

Pairs are sorted by Score Global and have either Score Global or Score Local greater than or equal to 10,000.

**Top Variable Interaction Pairings by Score Local**

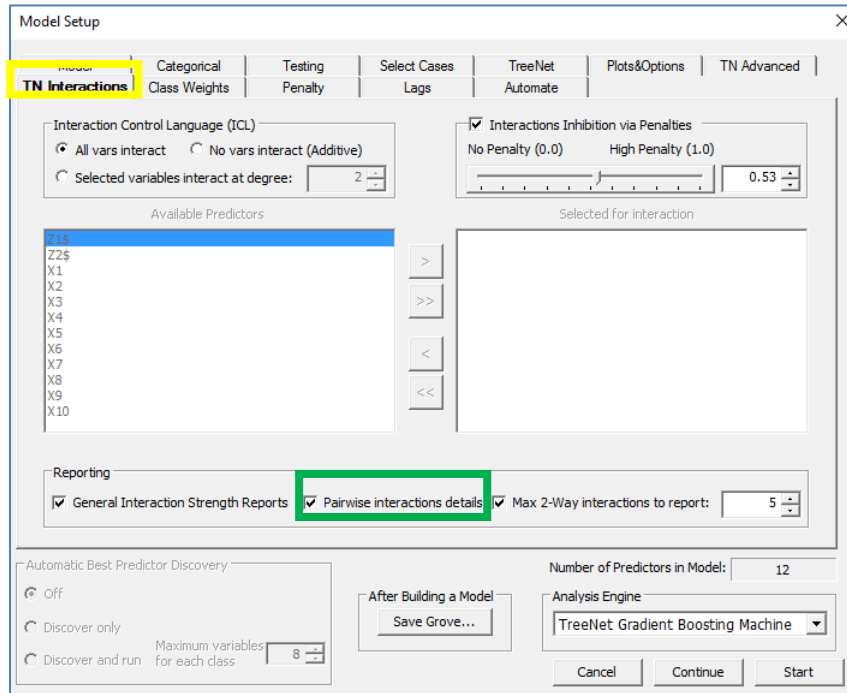
Score Global	Score Local	Predictor 1	Predictor 2
37.08393	48.75050	Z16	Z26
4.18081	17.13580	X2	X3

Pairs are sorted by Score Local and have either Score Global or Score Local greater than or equal to 10,000.

MARTGO: 10.204 sec ( 0.00 hrs)  
 LOADDATA 1: 0.023 sec ( 0.00 hrs, 0.23%)  
 MARTGO: 3.040 sec ( 0.00 hrs, 29.79%)  
 Core model: 0.877 sec ( 0.00 hrs, 28.85%)  
 MARTPRED: 0.136 sec ( 0.00 hrs, 1.34%)  
 PLOTS/INTER: 1.838 sec ( 0.00 hrs, 18.01%)

Reconciling (LEARN): 0.000010 hrs 0.36%  
 Reconciling (TEST ): 0.000004 hrs 0.14%

## Pairwise interactions details



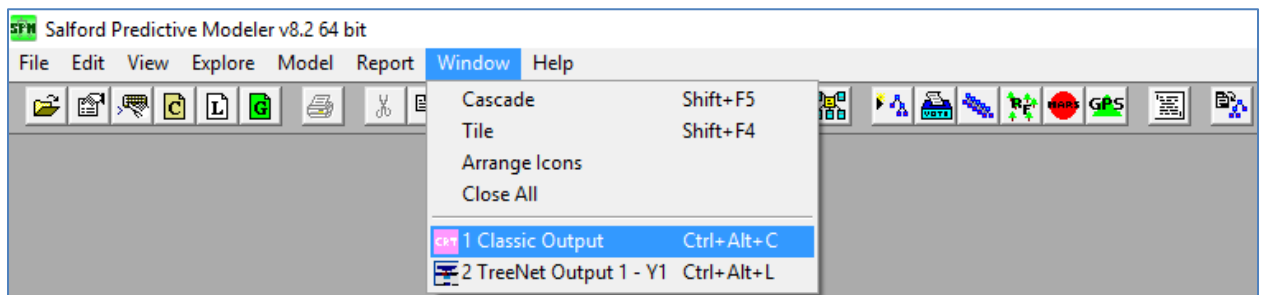
### Pairwise interactions details

 TREENET VPAIRS = <YES|NO>

Requests reports describing pairwise interactions of predictors. In particular, you will see a report with a title similar to “Most Common Same-Branch Pairwise Interaction By Occurrence.”

## Viewing Pairwise Interaction Details

1. After building the TreeNet model, click Window > Classic Output



This opens the Classic Output Window.

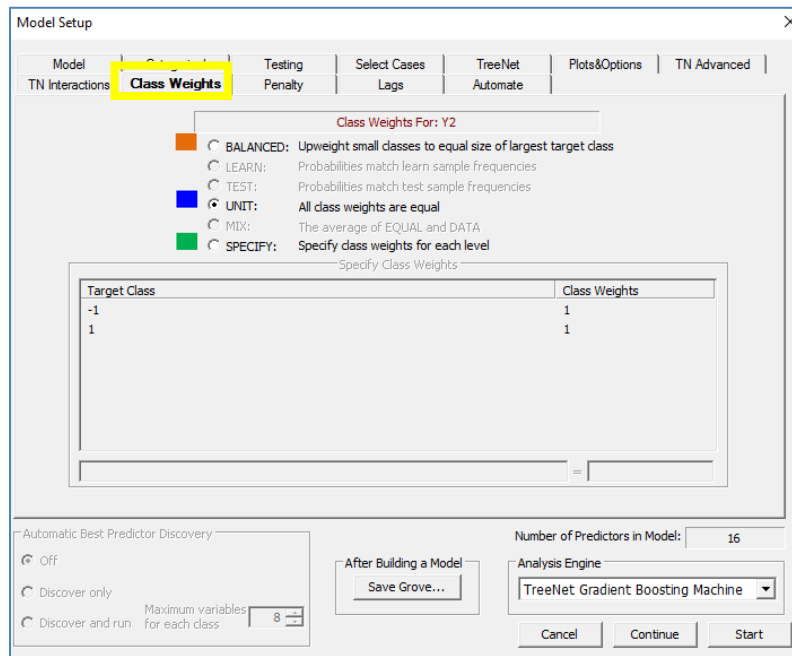
2. In the Classic Output Window, click the “Most Common Same-Branch Pairwise Interactions by Occurrence” link on the left (red rectangle below) and the “Most Common Same-Branch Pairwise Interactions” report appears (black rectangle below).

N	First	Predictor	Predictor
104	1 **	X1	X1
58	19	X2	X1
53	1	Z2\$	X1
51	1	Z1\$	Z2\$
50	29	X3	X2
49	25 **	X2	X2
44	20 **	X3	X3
43	29	X4	X2
38	31	X4	X1
35	29	X4	X3
33	30 **	X4	X4
33	4	Z1\$	X1
29	20	X3	X1
18	1 **	Z1\$	Z1\$
15	19	X2	Z2\$
13	30	X4	Z2\$
12	66	X2	Z1\$
12	92	X4	Z1\$
10	75	X3	Z1\$
9	140	X8	X4
9	125	X5	X2
8	149	X8	X3
8	116	X5	X4
6	130	X8	X1
6	157	X9	X1
6	149	X6	X2
5	142	X5	X3

## Interpreting Pairwise Interaction Details

1. The variable **X1** is used with the variable **X1** on the same branch **104** times and the first time this occurred was in tree **1**.
  - a. Note that the double star **\*\*** denotes a variable being used more than once on the same branch.
2. The variable **Z1\$** (the dollar sign denotes a categorical variable) is used with the variable **X1** are used on the same branch **33** times and the first time this occurred was in tree **4** (i.e. iteration **4**).

## Class Weights Tab



CW BALANCE | UNIT | SPECIFY

The original intent behind the concept of class weights is to allow powerful modeling controls similar to PRIORS controls in the CART engine. Internally, TreeNet trees are optimized for speed and efficiency and do not directly support the prior probabilities concept. This is where class weights become useful.

Class weights allow you to specify weights for each member of a class. For example, an observation from class 1 could count twice as much as an observation from class 0. Note that this is as if one doubled (ignoring some fine differences) the prior probability of class 1. Class weights are distinct from individual case or record weights and both types of weights can be used at the same time.

The following three class weights options for the logistic binary model are available:

**BALANCED** - As the name implies, this setting is intended to rebalance unequal class sizes. Automatic reweighting ensures that the sum of all weights in each class are equal, eliminating any need for manual balancing of classes via record selection or definition of weights.

✓ This is equivalent to the PRIORS EQUAL setting of CART.

**UNIT** - This setting takes the relative sizes of the two classes as given and does not attempt to compensate by adjusting weights for the smaller class. This is the default setting and is recommended for the binary logistic regression.

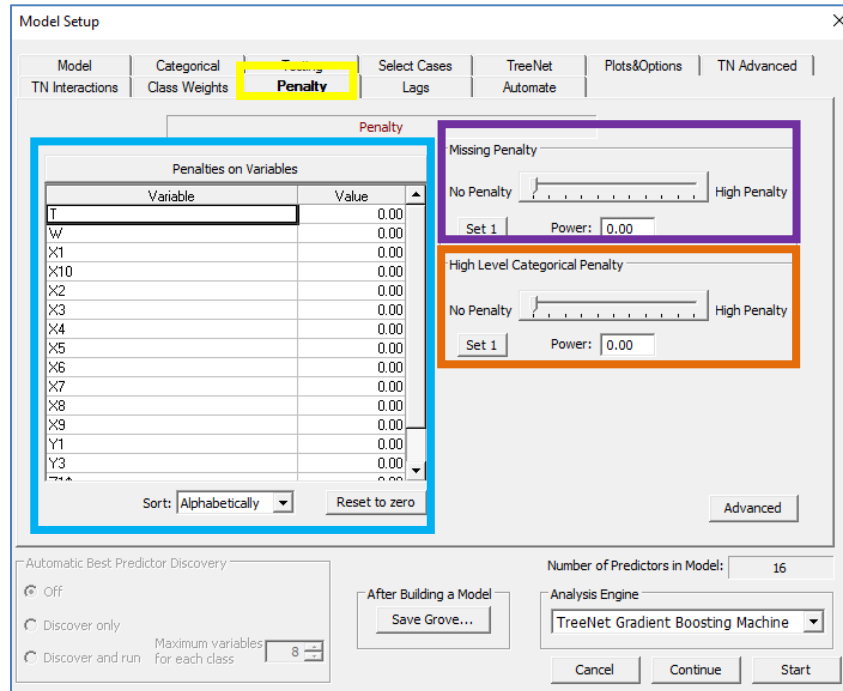
✓ This is equivalent to PRIORS LEARN setting of CART.

**SPECIFY** - Lets the user specify class weights. This is the most flexible option. You can set any positive values for each class.

⚠ While class weights can influence how a TreeNet model evolves and how it performs in prediction, these weights will not influence any reports involving record or class counts. The class weights are “virtual” and are used to influence the model-building process.

In most TreeNet runs, we recommend using UNIT class weights. This simplifies reading the output and usually produces superior results.

## Penalty Tab



The penalties available in the SPM were introduced by Salford Systems starting in 1997 and represent important extensions to machine learning technology. Penalties can be imposed on variables to reflect a reluctance to use a variable as a predictor. Of course, the modeler can always exclude a variable; the penalty offers an opportunity to permit a variable into the model but only under special circumstances. The three categories of penalty are:

**Variable Specific Penalties:** Each predictor can be assigned a custom penalty (**light blue rectangle** above).

```
PENALTY <var>=<penalty> [<var2> = <pen2>, <var3> = <pen3>, ...]
```

**Missing Value Penalty:** Predictors are penalized to reflect how frequently they are missing. The penalty is recalculated for every node in the tree (**purple rectangle** above)

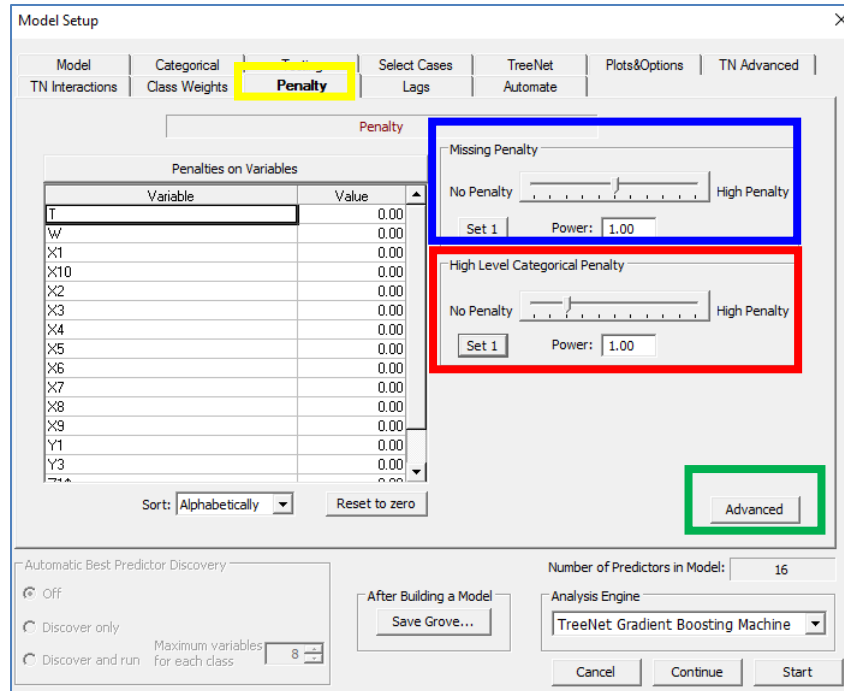
```
PENALTY /MISSING=1,1
```

**High Level Categorical Penalty:** Categorical predictors with many levels can distort a tree due to their explosive splitting power. The HLC penalty levels the playing field. (**orange rectangle** above)

```
HLC=<hlc_val1>,<hlc_val2>
```

A penalty will lower a predictor's improvement score, thus making it less likely to be chosen as the primary splitter. Penalties specific to particular predictors are entered in the left panel next to the predictor name and may range from zero to one inclusive.

Penalties for missing values (for categorical and continuous predictors) and a high number of levels (for categorical predictors only) can range from "No Penalty" to "High Penalty" and are normally set via the slider on the **Penalty** tab, as seen in the following illustration.



In the screenshot above, we have set both the Missing Values and the HLC penalties to the frequently useful values of 1.00 (red rectangle and blue rectangle above, respectively).

### Penalties on Variables

The penalty specified is the amount by which the variable's improvement score is reduced before deciding on the best splitter in a node. Imposing a 0.10 penalty on a variable will reduce its improvement score by 10%. You can think of the penalty as a "handicap": a 0.10 penalty means that the penalized variable must be at least 10% better than any other variable to qualify as the splitter.

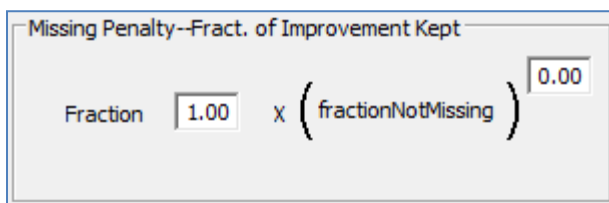
✓ Penalties may be placed to reflect how costly it is to acquire data. For example, in database and targeted marketing, selected data may be available only by purchase from specialized vendors. By penalizing such variables we make it more difficult for such variables to enter the tree, but they will enter when they are considerably better than any alternative predictor.

✓ Predictor-specific penalties have been used effectively in medical diagnosis and triage models. Predictors that are "expensive" because they require costly diagnostics, such as CT scans, or that can only be obtained after a long wait (say 48 hours for the lab results), or that involve procedures that are unpleasant for the patient, can be penalized. If penalizing these variables leads to models that are only slightly less predictive, the penalties help physicians to optimize diagnostic procedures.

☛ Setting the penalty to 1 is equivalent to effectively removing this predictor from the predictor list.

## Missing Values Penalty

In CART, at every node, every predictor competes to be the primary splitter. The predictor with the best improvement score becomes one. Variables with no missing values have their improvement scores computed using all the data in the node, while variables with missing values have their improvement scores calculated using only the subset with complete data. Since it is easier to be a good splitter on a small number of records, this tends to give an advantage to variables with missing values. To level the playing field, variables can be penalized in proportion to the degree to which they are missing. The proportion missing is calculated separately at each node in the tree. For example, a variable with good data for only 30% of the records in a node would receive only 30% of its calculated improvement score. In contrast, a variable with good data for 80% of the records in a node would receive 80% of its improvement score. A more complex formula is available for finer control over the missing value penalty using the Advanced version of the **Penalty** tab (click the button in the **green rectangle above**).



Suppose you want to penalize a variable with 70% missing data very heavily, while barely penalizing a variable with only 10% missing data. The **Advanced** option (click the button in the **green rectangle above**) lets you do this by setting a fractional power on the percent of good data. For example, using the square root of the fraction of good data to calculate the improvement factor would give the first variable (with 70% missing) a .55 factor and the second variable (with 10% missing) a .95 factor.

The expression used to scale improvement scores is:

$$S = a * (\textit{proportion\_not\_missing})^b$$

The default settings of  $a = 1$ ,  $b = 0$  disable the penalty entirely; every variable receives a factor of 1.0. Useful penalty settings set  $a = 1$  with  $b = 1.00$ , or 0.50. The closer  $b$  gets to 0 the smaller the penalty. The fraction of the improvement kept for a variable is illustrated in the following table, where "%good" = the fraction of observations with non-missing data for the predictor.

%good	b=.75	b=.50
0.9	0.92402108	0.948683298
0.8	0.84589701	0.894427191
0.7	0.76528558	0.836660027
0.6	0.68173162	0.774596669
0.5	0.59460355	0.707106781
0.4	0.50297337	0.632455532
0.3	0.40536004	0.547722558
0.2	0.29906975	0.447213595
0.1	0.17782794	0.316227766

In the bottom row of this table, we see that if a variable is only good in 10% of the data, it receives 10% credit if  $b=1$ , 17.78% credit if  $b=.75$ , and 31.62% credit if  $b=.50$ . If  $b=0$ , the variable receives 100% credit because we would be ignoring its degree of missingness.

✓ In most analyses, we find that the overall predictive power of a tree is unaffected by the precise setting of the missing value penalty. However, without any missing value penalty you might find heavily missing



variables appearing high up in the tree. The missing value penalty thus helps generate trees that are more appealing to decision makers.

### High-Level Categorical (HLC) Penalty

Categorical predictors present a special challenge to decision trees. Because a 32-level categorical predictor can split a data set in over two billion ways, even a totally random variable has a high probability of becoming the primary splitter in many nodes. Such spurious splits do not prevent TreeNet from eventually detecting the true data structure in large datasets, but they make the process inefficient. First, spurious splits add unwanted nodes to a tree, and because they promote the fragmentation of the data into added nodes, the reduced sample size as we progress down the tree makes it harder to find the best splits.

To protect against this possibility, the SPM offers a high-level categorical predictor penalty used to reduce the measured splitting power. On the **Basic Penalty** dialog, this is controlled with a simple slider (this is the default view that you see in the picture on the previous page).

The **Advanced Penalty** (click the button in the **green rectangle above**) dialog allows access to the full penalty expression. The improvement factor is expressed as:

$$S = \min \left\{ 1, 1 + c * \left( \left( \frac{\log_2[\text{node\_size}]}{N\_categories - 1} \right)^d - 1 \right) \right\}$$

By default,  $c = 1$  and  $d = 0$ ; these values disable the penalty. We recommend that the categorical variable penalty be set to ( $c = 1$ ,  $d = 1$ ), which ensures that a categorical predictor has no inherent advantage over a continuous variable with unique values for every record.

- ⚙️ The missing value and HLC penalties apply uniformly to all variables. You cannot set different HLC or missing value penalties to different variables. You choose one setting for each penalty and it applies to all variables.
- ✓ You can set variable-specific penalties, general missing value, and HLC penalties. Thus, if categorical variable Z is also sometimes missing all three penalties could apply to this variable at the same time.

## Lags Tab: A Brief Introduction

If you wish to run a time series or panel data (time series cross section) style model, use lagged values of variables as predictors.

Note: if you check any of the variables as predictors then the model will use a variable from the same time period as the target (i.e. the “current” time period). If you don’t want variables from the same time period as the target variable, then keep all predictor boxes unchecked as shown in the **red rectangle** below.

Model Setup ×

TN Interactions | Class Weights | Penalty | Lags | Automate |  
**Model** | Categorical | Testing | Select Cases | TreeNet | Plots&Options | TN Advanced

Variable Selection

Variable Name	Target	Predictor	Categorical	Weight
DIESEL_LA	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
GAS_NY	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
JET_FUEL_GC	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
OK_PRICE	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
PROPANE_TX	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
T_ID	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
WEEK_OF	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Sort:   Select Predictors  Select Cat.

Filter:  All/Selected  Character  Numeric

Target Type:  
 Classification/Logistic Binary  
 Regression  
 Unsupervised

Set Focus Class...

Target Variable: OK\_PRICE

Weight Variable:

Number of Predictors: 6

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: 6

Analysis Engine:

Click the “Lags” tab (yellow rectangle below).

To add lagged variables to the model, enter the number of periods to lag a variable. The lagged versions of the variable are created by SPM automatically and are added to the model. In the cells below for the “DIESEL\_LA” row a value of **3** means that SPM will create a DIESEL\_LA variable that is lagged by **3** periods, a value of **4** means that SPM will create a DIESEL\_LA variables that is lagged by 4 periods, and so on.

Model Setup

Model | Categorical | Testing | **Lags** | TreeNet | Plots&Options | TN Advanced  
 TN Interactions | Class Weights | Penalty | Automate

Variable Name	Block	Enter Lag Values					
T_ID	<input type="checkbox"/>	3					
DIESEL_LA	<input type="checkbox"/>	3	4	5	8	9	10
GAS_NY	<input type="checkbox"/>	3					
JET_FUEL_GC	<input type="checkbox"/>	3					
OK_PRICE	<input type="checkbox"/>	3					
PROPANE_TX	<input type="checkbox"/>	3					
WEEK_OF	<input type="checkbox"/>	3					

Sort: Alphabetically

Clear Value | Clear All | Add Column | Delete Column

Fill the selected column  
 3 | Fill

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

After Building a Model  
 Save Grove...

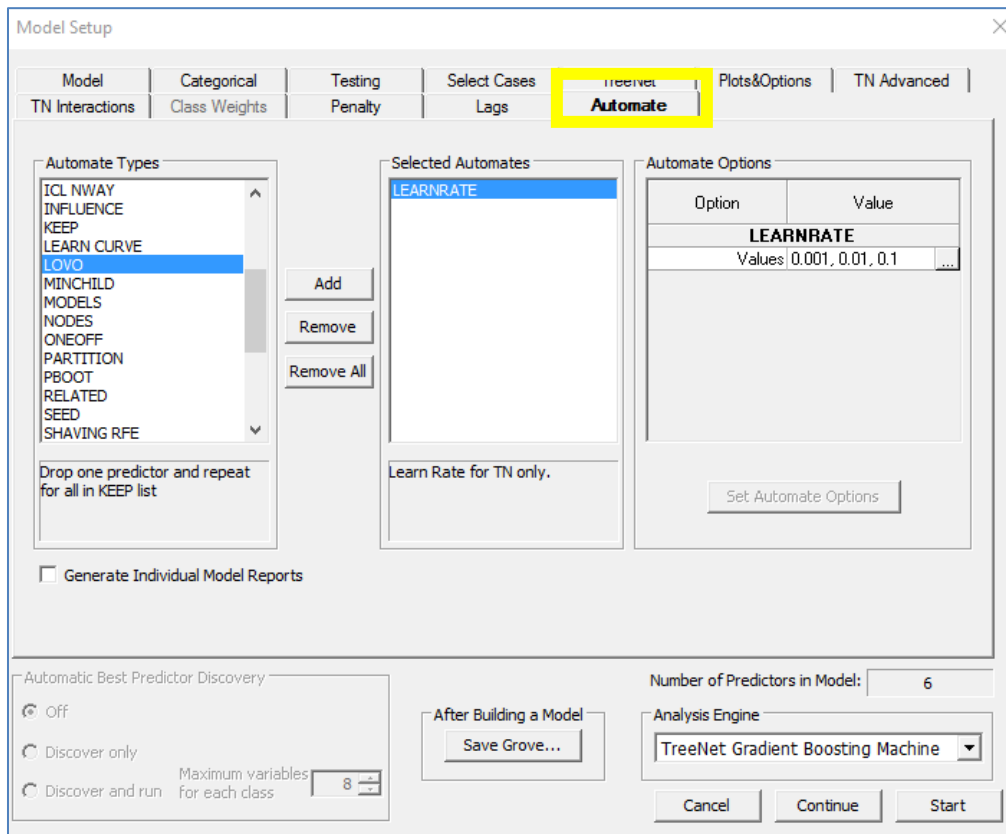
Number of Predictors in Model: 6  
 Analysis Engine: TreeNet Gradient Boosting Machine

Cancel | Continue | Start

- The **Clear Value** button clears the lag value from a desired cell after you click on the cell
- The **Clear All** button clears all lag values
- The **Add Column** button adds a column if you need to add more lags (the default number of columns is 4 and in the above picture there are six columns; the additional two columns were created by clicking the Add Column button twice)
- The **Delete Column** button removes a column if desired
- The **Fill** button in the “Fill the selected column” section adds an entire column of the same lag value which prevents you from having to do this manually

## Automate Tab: A Brief Introduction

An **automate** is a procedure that allows a user to perform experiments and build multiple models simultaneously. Some automates are general in nature and apply to all SPM modeling engines whereas some are specific to individual modeling engines. Some automates are general in nature whereas others are specific to particular modeling engines.



### Examples of general automates

1. **Automate PARTITION** – repeat modeling run with new learn, test, and holdout samples drawn from the original dataset
2. **Automate BOOTSTRAP** – repeat modeling run with new learn samples drawn with replacement (advanced testing options included)
3. **Automate CVREPEATED** – repeat cross validation with different random number seeds
4. **Automate SHAVING RFE** – drop the least important predictor, rerun, and repeat

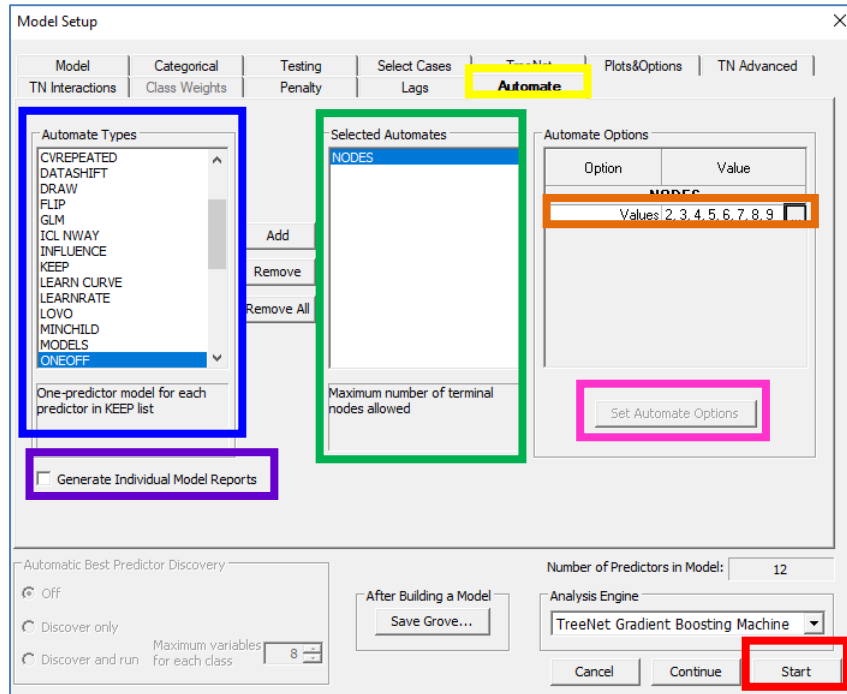
### TreeNet Specific Automates

1. **Automate INFLUENCE**- build multiple TreeNet models, each with a different value for the total influence trimming fraction

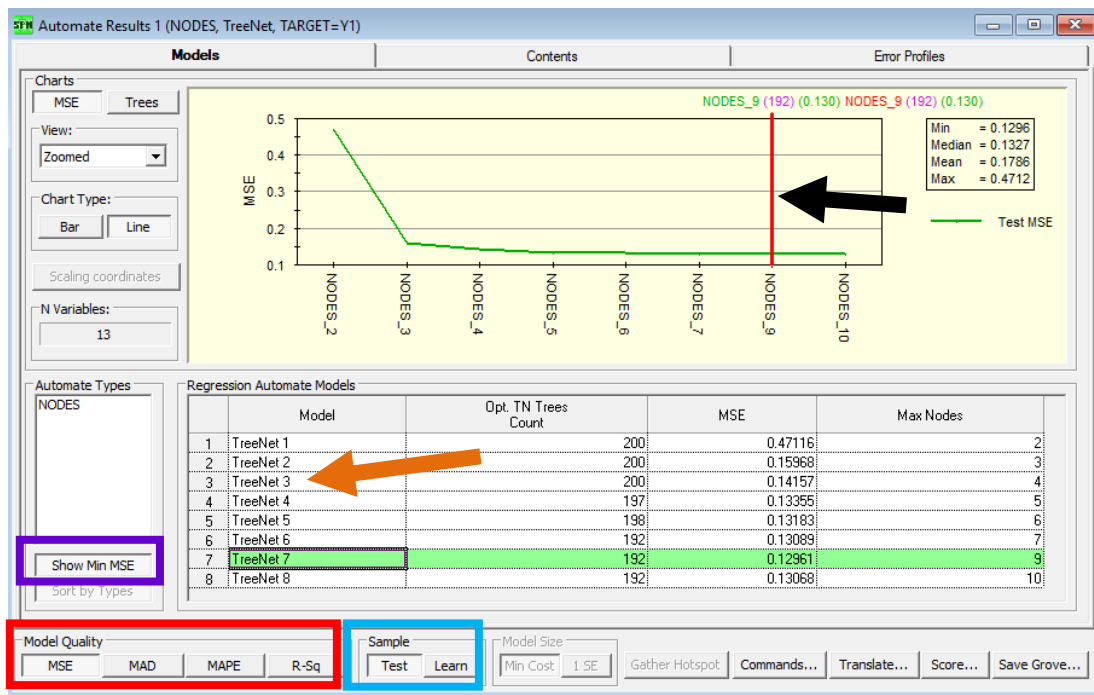
2. **Automate LEARNRATE** – build multiple TreeNet models, each with a different learn rate (the learn rate is the  $\alpha$  in “Step 4” in the “TreeNet Algorithm” section)
3. **Automate NODES** – build multiple TreeNet models, each with a different maximum number of terminal nodes (the maximum number of terminal nodes is the “J” in “Step 3” in the “TreeNet Algorithm section”)
4. **Automate TNCLASSWEIGHTS**- vary class weights between UNIT and BALANCED in N steps for binary TreeNet models
5. **Automate TNOPTIMIZE** – perform a stochastic search for the core TreeNet parameters
6. **Automate TNQUANTILE**- build multiple TreeNet models, each with a different value for the LAD quantile value.
7. **Automate TNRGBOOST**- this will result in N+1 models being built. A first model will be built to determine plausible values for the L0, L1, and L2 parameters and N more models will be built using random combinations of those values. Specifying a different RANDOMSEED value allows you to repeat the experiment with the same number of models but a different group of randomly selected L0, L1, and L2 combinations.
8. **Automate TNSUBSAMPLE** –build multiple TreeNet models, each with a different subsampling fraction (the subsample fraction is the percentage, denoted by  $s$ , mentioned in [Step 2 in the Gradient Boosting Algorithm section](#))

## Using an Automate

To run an automate go to the **Automate** tab (yellow rectangle below) and double click on the desired automate in the **left panel** so that it appears in the **middle panel**, specify your desired values if you want something different than the default values by double clicking in the “**Values**” box or the “**Set Automate Options**” button (if it is not grayed out), and then click the “**Start**”. Click the “**Generate Individual Model Reports**” if you want detailed reports for each model constructed using an automate.

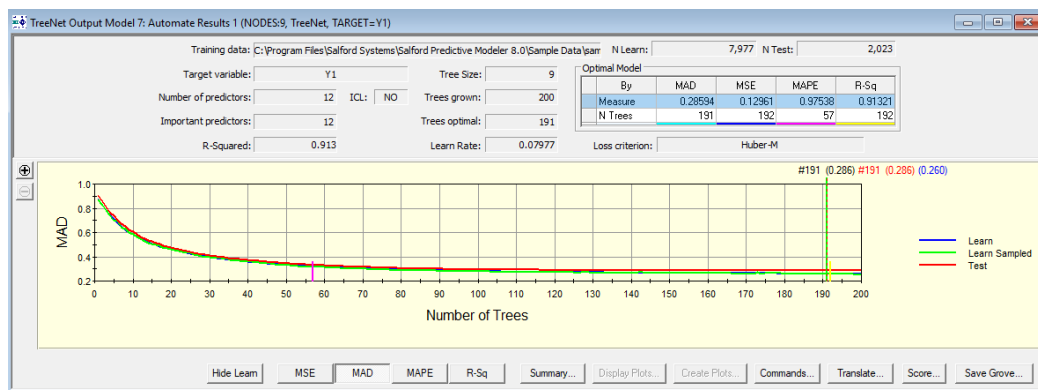


The output from running Automate NODES is the following:



The red beam in the chart above (see the **black arrow in the picture above** pointing to the red beam) indicates that the optimal value for the maximum number of terminal nodes is 9 because the TreeNet model with a maximum number of terminal nodes value of 9 has the smallest Mean Squared Error value on the **test sample**. You can also click the **Show Min MSE button** to highlight the row that corresponds to the optimal value for the maximum number of terminal nodes. Note that you can choose **from four measures of model quality**: Mean Squared Error (MSE; currently the MSE button is selected), Median Absolute Deviation (MAD), Mean Absolute Percentage Error (MAPE), or R-squared (R-Sq).

To view detailed information about any of the models, double click the model ID (i.e. double click "TreeNet1," "TreeNet2," etc.; see the **orange arrow pointing to "TreeNet3"** in the picture above). Double clicking will open up a TreeNet output window similar to this:



You can now view detailed model information as discussed in the two interpretation sections of this manual: [TreeNet Regression: Viewing and Interpreting Results](#) or [Classification/Binary Logistic: Viewing and Interpreting Results](#).

## Example: Building a Regression Model

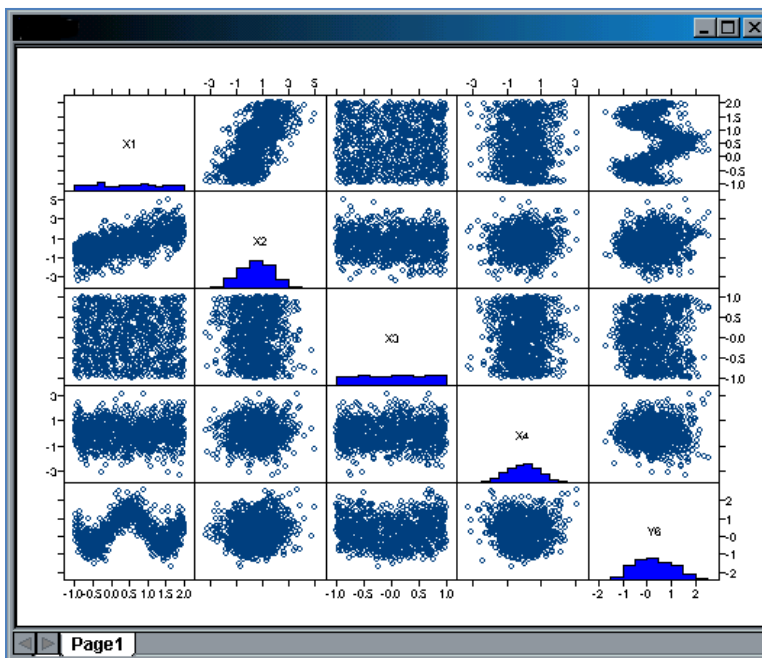
### Creating a Sample Example Model Data Set

A sample data set SAMPLE.CSV is supplied as part of the TreeNet installation. It is located in the *Sample Data* folder.

The following table lists the major groups of variables in the data set.

Variable Names	Description
X1, ..., X10	Continuous predictors
Z1\$, Z2\$	Categorical predictors (character)
Y1	Continuous target
Y2	Binary target coded +1 and -1
Y3	Categorical target with 4 levels: 1, 2, 3, and 4
W	Weight variable
T	Learn/Test dummy indicator (0 – learn, 1 – test)

In our first run we will try to predict the continuous target Y1 using the continuous predictors X1–X10 and categorical predictors Z1\$, Z2\$. We chose a regression problem for this first example because regression problems are somewhat simpler and produce fewer reports than classification problems. The scatter matrix for select continuous variables follows.



The plots suggest only a strong nonlinear relationship between X1 and Y1. It is not clear whether there is any relationship between the remaining predictors and the target.

An attempt to run a stepwise multiple linear regression of Y1 on X1–X10 selects five predictors, X1, X2, X6, X8, X9. Unfortunately, this model only has R-squared equal to .0179, making it practically useless. Later we will discover that this model fails to pick important predictors and at the same time mistakenly selects some irrelevant predictors.




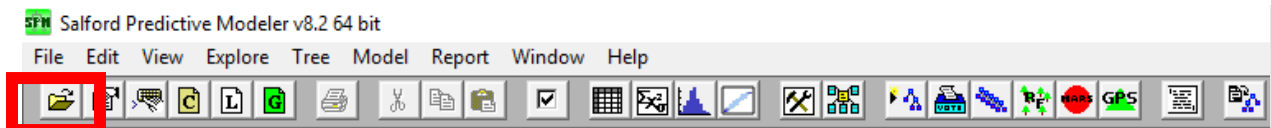
- ✓ Exploiting possible periodic non-linearity in X1 as suggested by the corresponding plot may slightly improve the results, but the remaining relationships remain obscure.

## Reading Data In

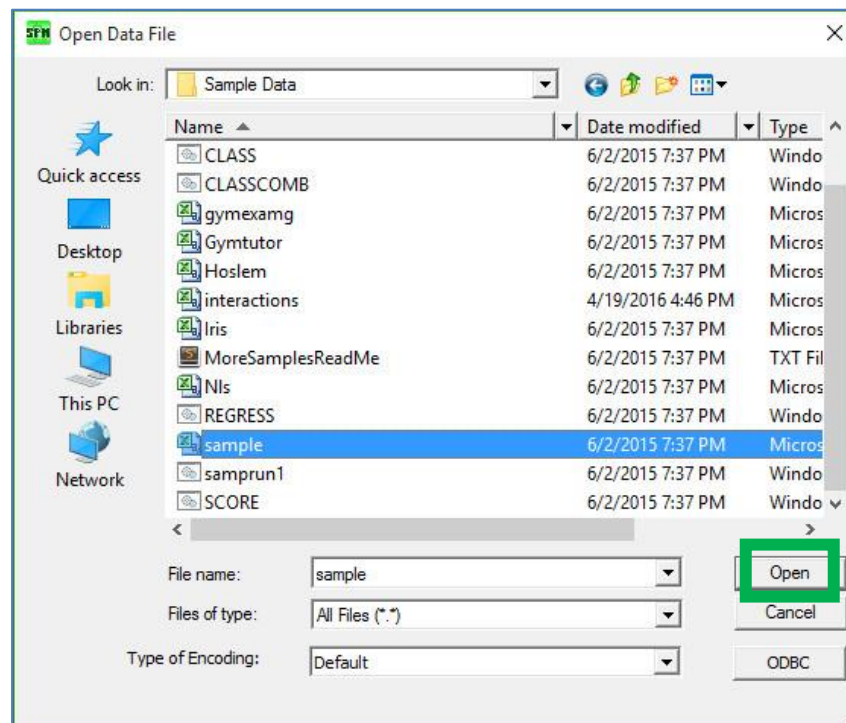
To open the input file `SAMPLE.CSV`:

Select **File>Open>Data File...** from the **File** menu.

- ✓ An alternative is to click on the  button in the toolbar



Use the **Open Data File** dialog window to navigate into the *Sample Data* folder in the SPM installation directory.



Choose **Delimited (\*.csv,\*.dat,\*.txt)** in the **Files of type:** selection box.

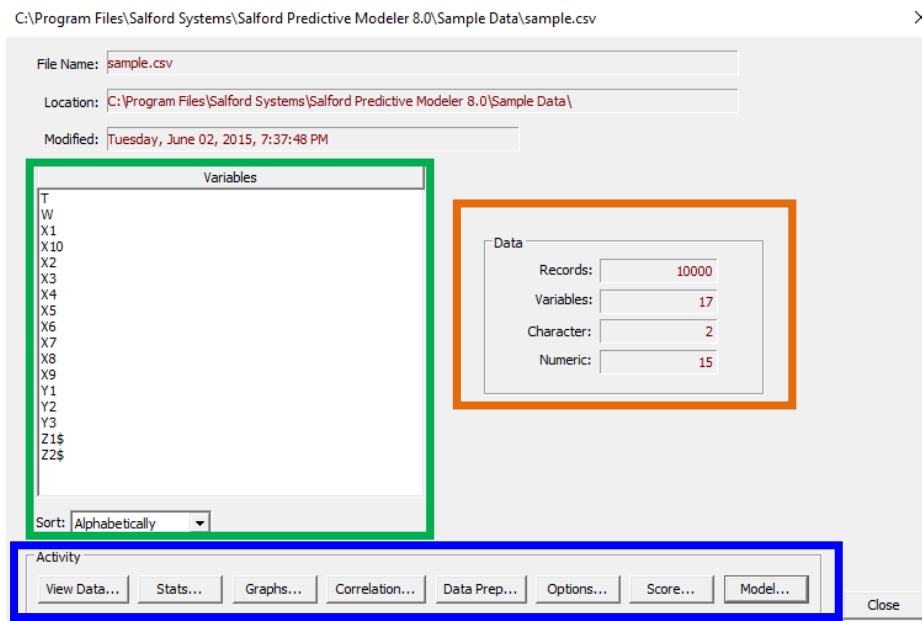
Highlight `SAMPLE.CSV`.

Click the **Open button** to load the data set into TreeNet.

- ☛ Make sure that the data set is not currently being accessed by another application. If it is, the data loading will fail. This is a common source of problems when dealing with Excel and ASCII files.

After you click “Open” you will see the “Activity Window”:

## The Activity Window



**Activity Window Buttons:** The buttons at the bottom of the activity window have the following functions

- ◆ “**View Data**”... allows the user to see the dataset
- ◆ “**Stats**...” allows the user to compute summary statistics for each variable
- ◆ “**Graphs**...” allows the user to build graphs
- ◆ “**Correlation**...” allows the user to compute measures of correlation and similarity
- ◆ “**Data Prep**...” allows user to perform data preparation operations inside of SPM
- ◆ “**Options**...” allows the user change settings for the software
- ◆ “**Score**...” allows the user to use a previously saved model or a model currently open inside of SPM to generate predictions for a dataset
- ◆ “**Model**...” takes the user directly to the model setup

### Data

- ◆ **Records:** number of observations = number of rows in the dataset
- ◆ **Variables:** total number of variables = total number of columns in the dataset
- ◆ **Character:** number of variables with character values (Example: a variable with values “Female” or “Male”)
- ◆ **Numeric:** number of variables with only numeric values
- ◆ \*\*\*Note: Variables = Character + Numeric

### Variables

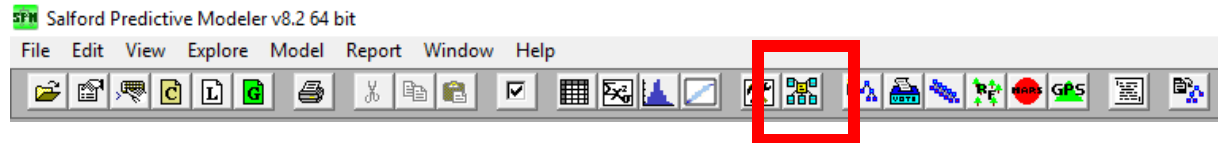
- ◆ Provides a list of the variables in alphabetical order. You can change this to the file order by clicking on the dropdown box (see the bottom of the green rectangle above)

## Setting up a TreeNet Model for Regression

To open the model setup, click the model setup shortcut button

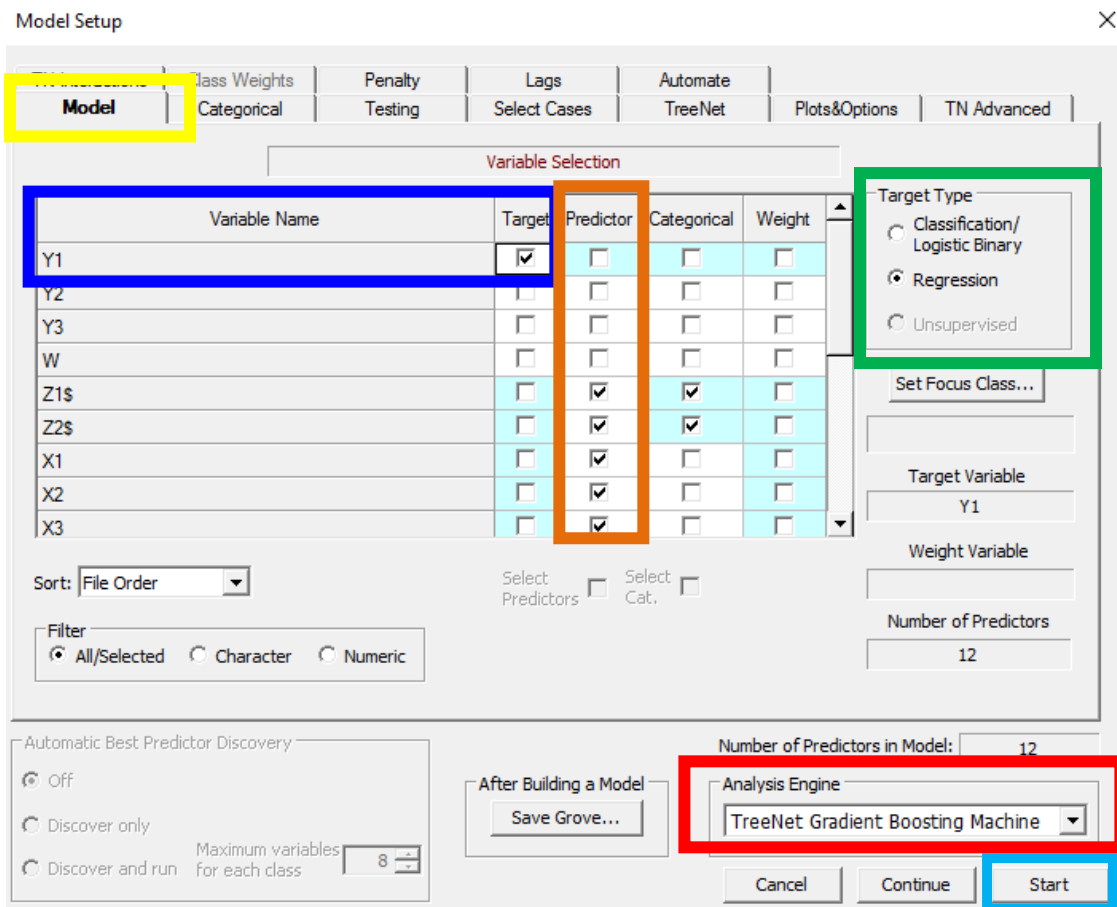


(red rectangle below)



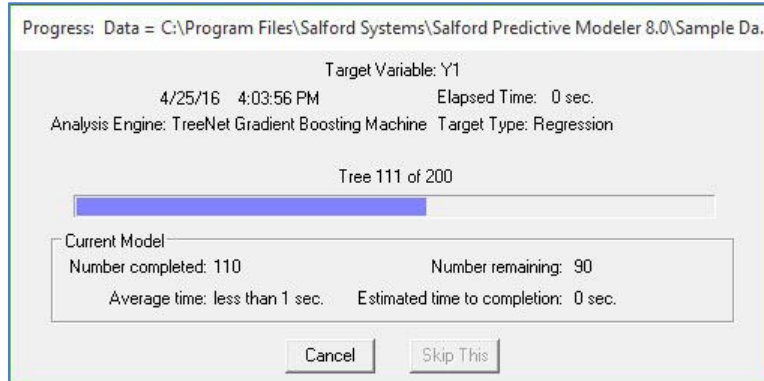
After clicking the model setup shortcut button you will see the model setup dialog pictured below. The first tab that you will see is the **Model Tab** (yellow rectangle below)

1. Set the Analysis Engine to **TreeNet Gradient Boosting Machine**.
2. Set the Target Type to **Regression**.
3. Click the checkbox for the desired variable in the target variable (**blue rectangle below**).
  - a. A **target variable** is the variable that you are trying to predict (i.e. the dependent variable; this is "Y").
4. Click the checkboxes for the desired predictor variables (**orange rectangle below**).
5. Choose the settings in the other tabs (see the [TreeNet Settings section](#) for more information) and then click the **Start button** to build the TreeNet model.



## Running TreeNet

After you click **Start button** in the picture above, you will see the following progress indicator on the screen while the model is being built. The progress indicator lets you know how much time the analysis should take and approximately how much time remains.

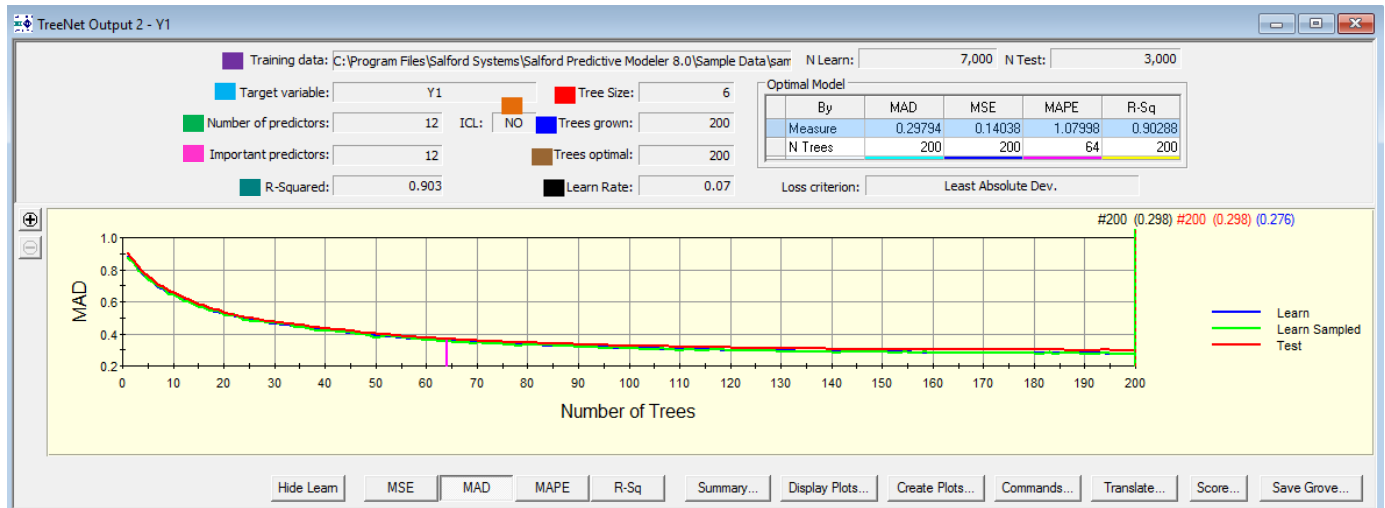


After analysis is complete, text output will appear in the **Classic Output** window, and a new window, the **TreeNet Output**, opens. See the [TreeNet Regression: Interpreting Results section](#) to see an example of the TreeNet output and how to interpret the results.

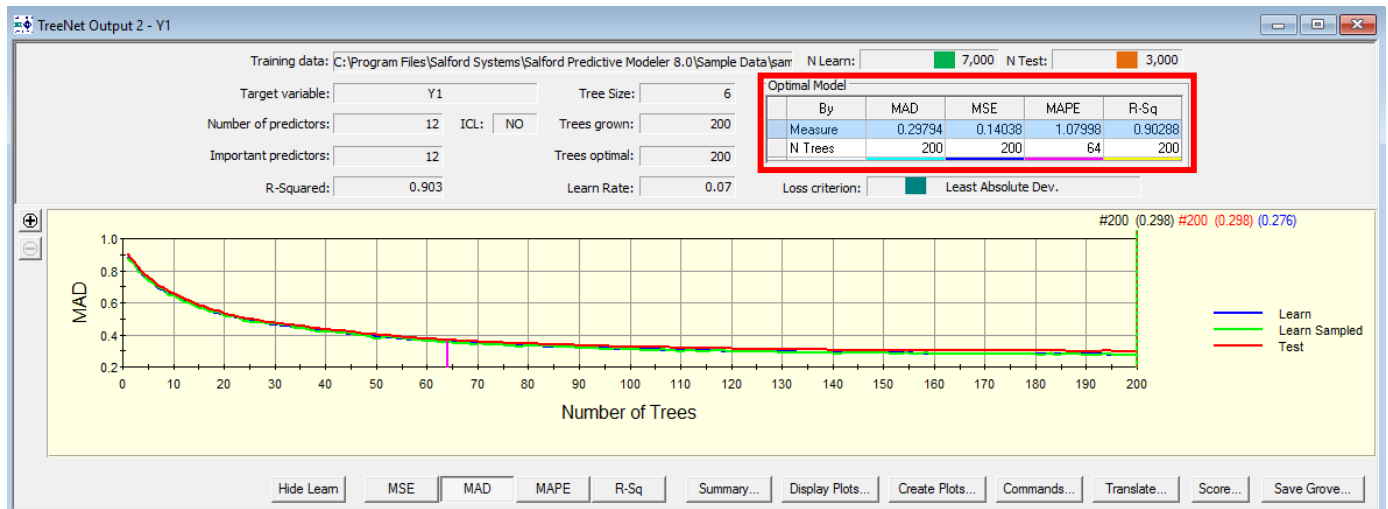
## TreeNet Regression: Interpreting Results

### Initial Output

After the model is complete, you will see the following:

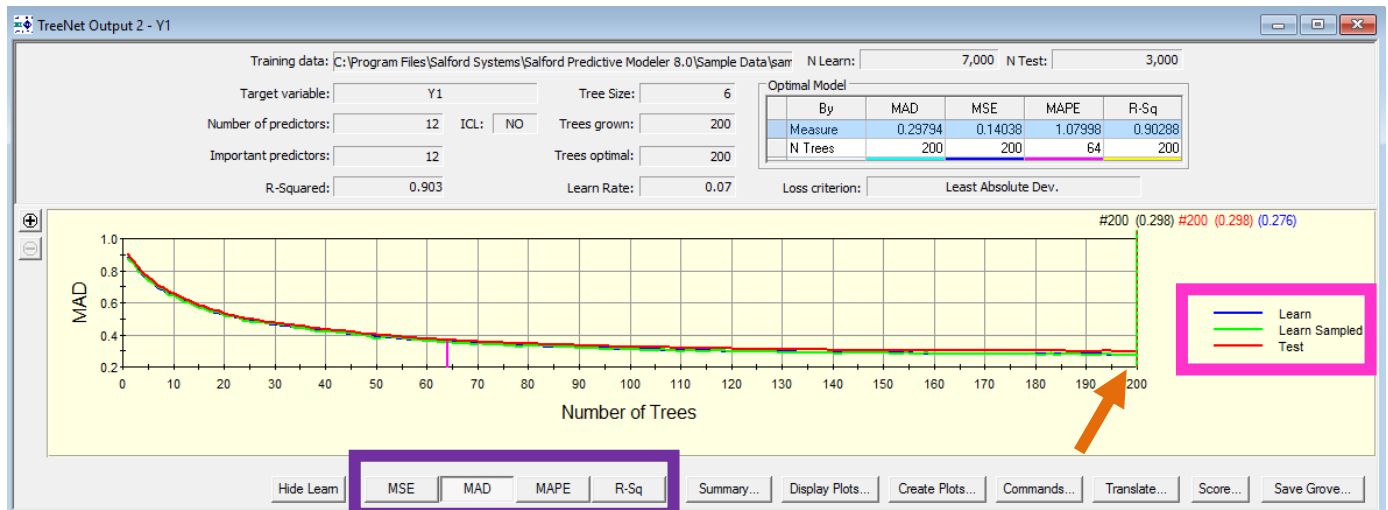


- **Training data:** file path of your dataset
  - **Target variable:** the name of your target variable
  - **Number of predictors:** number of predictors available to be used in the model (i.e. the number of predictor boxes checked during the model setup)
  - **Important predictors:** number of predictors actually used in the model
  - **R-Squared:** R-Squared value corresponding to the optimal model
  - **ICL:** “ICL” denotes the “Interaction Control Language”; See the [TN Interactions Tab](#) entry for more details on ICL. Here we can have two values: “YES” or “NO”
    - ✓ **NO** - means that the ICL was not used (i.e. all variables are allowed to interact)
    - ✓ **YES** - mans that the ICL was used (i.e. some constraint was added)
  - **TreeSize:** maximum number of terminal nodes specified in each tree during the model setup
  - **Trees grown:** maximum setting for the number of trees specified during the model setup
  - **Trees optimal:** optimal number of trees (Note: this can be different depending on the model criterion being used like MAD, MSE, MAPE, or R-Sq)
  - **Learn Rate:** the learning rate specified during the model setup
- ✓ The various summary reports, including Variable Importance rankings, R-squared, and gains charts and all the 2D and 3D graphs displayed above apply to a SINGLE model. In the example above this is the model that consists of *exactly* 193 trees.
- ✓ If you decide to work with a cut-back model, you can view graphs and displays for the smaller model, but you may have to rerun the model and either stop its evolution early or explicitly request that details be saved for many sub-models.



- **N Learn** –number of records in the learning data (the learning data is sometimes called the “training data”)
- **N Test** –number of records in the testing data (the testing data is sometimes called the “validation data”)
- **Optimal Model** – shows the optimal value for four model evaluation criteria along with the corresponding number of trees for each
  - ✓ **\*\*\*Note that the optimal number of trees can be different depending on the measure of used\*\*\***
  - ✓ **MAD** – Median Absolute Deviation; The optimal MAD value is .29368 and occurs at tree 200
  - ✓ **MSE** – Mean Squared Error; The optimal MSE value is .13556 and occurs at tree 200
  - ✓ **MAPE** – Mean Absolute Percentage Error; The optimal MAPE value is 1.73434 and occurs at tree 64
  - ✓ **R-Sq** – R-Squared; The optimal R-Squared value is .90765 and occurs at tree 200
- **Loss criterion:** the loss function specified during the model setup
- ✓ Note that the R-squared of the optimal model on TEST data is 0.903

## Error Curves



The curve in the picture above shows the Median Absolute Deviation (MAD) on the y-axis and the Number of Trees (i.e. the number of iterations) on the x-axis. Notice that the key on the right (**pink rectangle above**) shows the colors that correspond to one of the three curves:

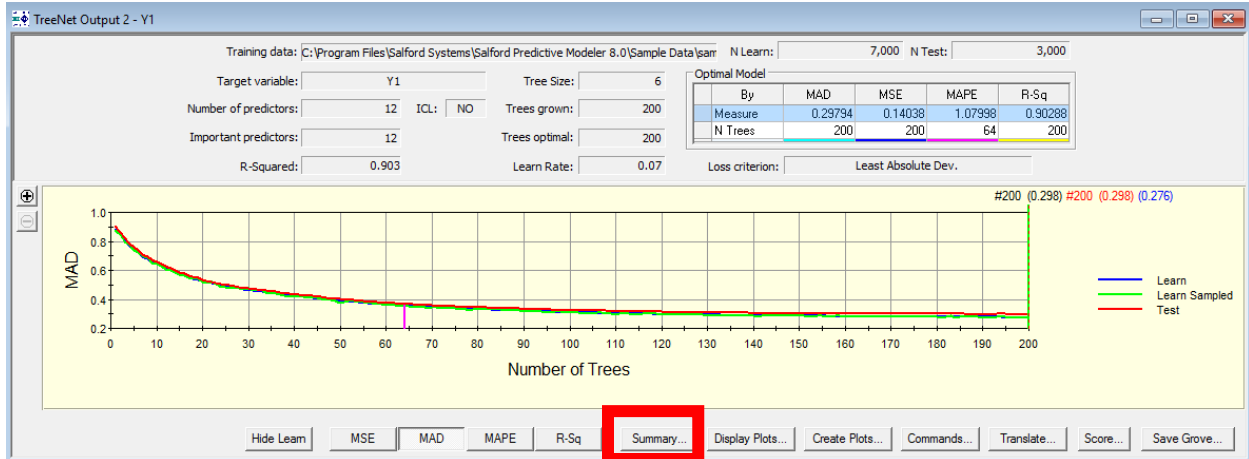
1. **Learn Error Curve**
  - a. The learn error at iteration K is the error for the entire learning sample using the first K trees.
2. **Learn Sampled Error Curve**
  - a. Recall: we take a random subsample in each iteration and each iteration corresponds to fitting a CART tree to the generalized residuals. The learn sampled error at iteration K is the error for the subsample taken at the Kth iteration using the first K trees.
3. **Test Error Curve**
  - a. The test error at tree K is the error for the entire test sample using the first K trees.

The above picture is for the MAD curve, but you can view any of the following curves by clicking the desired button along the bottom (**purple rectangle above**; notice the MAD button is already selected)

1. Mean Squared Error (MSE)
2. Median Absolute Deviation (MAD)
3. Mean Absolute Percentage Error (MAPE)
4. R-Squared

The exact numeric minimum of the **Mean Absolute Deviation (MAD)** on the test data was attained at the 200-tree model highlighted by the green beam (**orange arrow above** points to the green beam at tree 200).

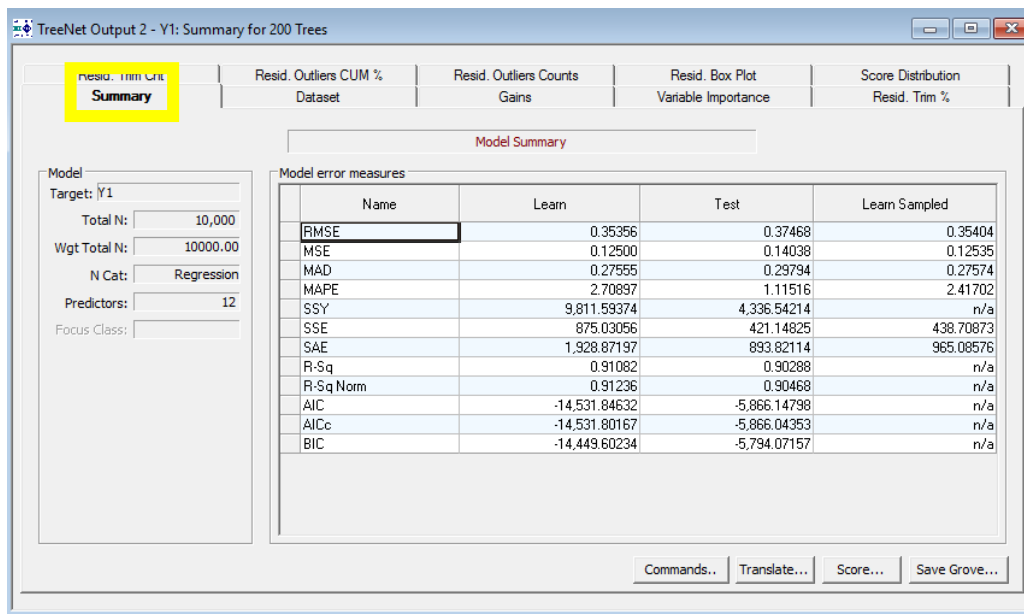
## Summary Button



The overall performance of the TreeNet regression model is summarized in model summary reports. To access the reports, click the “Summary...” button (red rectangle above). This yields a detailed information about the optimal TreeNet model (i.e. the one with 200 trees).

## Summary Tab

The Summary Tab (yellow rectangle below) provides model error measures for the learn, test, and learn sampled data (see the [Error Curve](#) section of this manual for the definition of the learn sampled error).





## Dataset Tab

The Dataset Tab provides information concerning the learn and test samples, record deletion, and the distribution of the target variable in the learn and test samples.

The screenshot shows the 'Dataset' tab in the TreeNet software. The window title is 'TreeNet Output 1 - Y1: Summary for 200 Trees'. The 'Dataset' tab is selected and highlighted in yellow. Below the tab, there are several sections:

- Sample Partition** (highlighted with a red box): A table showing the distribution of data between learn and test samples.
- Record Deletion** (highlighted with a blue box): A table showing the number of records deleted for each sample, categorized by target missing status and predictor status.
- Target Variable Statistics** (highlighted with an orange box): A table showing summary statistics (Mean, Min, 1%, Q1, Median) for the target variable in both learn and test samples.

Sample Partition	N	Pct
Learn	7,000	70.00%
Test	3,000	30.00%
Total	10,000	100.00%

Record Deletions	Target Missing	SELECT	BASIC	Missing Predictors
Learn	0	0	0	0
Test	0	0	0	0
Total	0	0	0	0

Sample	Mean	Min	1%	Q1	Median
Learn	0.77287	-1.82933	-1.17611	-0.11523	0.61151
Test	0.77270	-1.75993	-1.21000	-0.15476	0.57871

### Sampling Partition Information

Provides information about the learn and test data. The learn sample has 7,000 observations that comprise 70% of the total number of records whereas the test sample has 3,000 records that comprise 30% of the total number of records.

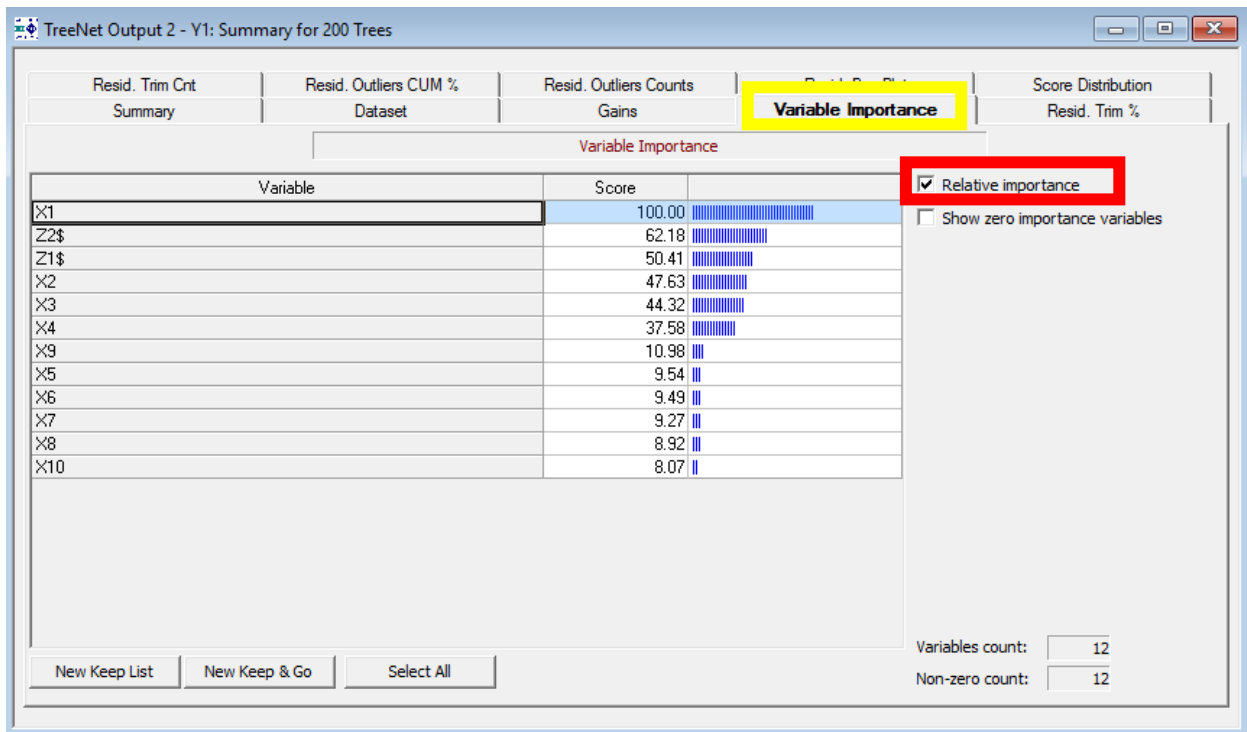
### Record Deletions

TreeNet models automatically handle missing values in the predictor variables, but if there are missing values in the target variable then these records are deleted (this is why we have a column called "Target Missing").

### Distribution of the Target Variable

Provides summary statistics for the target variable in the learn sample (i.e. the first row) and summary statistics for the target variable in the test sample (i.e. the second row).

## Variable Importance Tab



## Computation & Interpretation

Recall that in each iteration of the gradient boosting algorithm we fit a CART tree to generalized residuals. In CART, all possible split values are evaluated for every variable. The split appearing in the tree is the variable and split point combination that has the largest split improvement value. Each time a variable is used in a TreeNet model, the split improvement is recorded.

1. **Raw Variable Importance:** computed as the cumulative sum of improvements of all splits associated with the given variable across all trees up to a specific model size. Note that smaller models will typically involve fewer variables; thus, limiting the number of variables with non-zero importance scores, whereas larger models will often utilize a larger number of predictors.
2. **Relative Variable Importance:** (red rectangle above) rescales the variable importance values so that they are on a scale of 0 to 100 (the most important variable always gets a value of 100). All other variables are rescaled to reflect their importance relative to the most important variable.
  - a. **Interpretation:** in the above picture, the variable X1 is the most important variable and Z1\$ has a relative importance of 50.41, so we can say that X1 is around twice as important as Z1\$. Similarly, we can say that the variable X1 is over 9 times as important as the variable X9 ( $10.98 \times 9 \sim 99$  which is close to X1's variable importance value of 100)
  - b. **Relative Importance Rescaling:** divide each variable's raw importance score by the largest raw importance score and multiply by 100

✓ This conclusion is further supported by the analysis of contribution plots described in a later section.

## New Keep List

The variable importance lists are interactive: you can highlight a subset of variables and then quickly rebuild a model using only the highlighted variables.

Variable	Score	Relative Importance
X1	100.00	100%
Z2\$	62.18	62.18%
Z1\$	50.41	50.41%
X2	47.63	47.63%
X3	44.32	44.32%
X4	37.58	37.58%
X9	10.98	10.98%
X5	9.54	9.54%
X6	9.49	9.49%
X7	9.27	9.27%
X8	8.92	8.92%
X10	8.07	8.07%

Variables count: 12  
Non-zero count: 12

### New Keep List

A KEEP list is a list of variables that you want to select as predictors in the model. KEEP lists are an SPM command line feature that are very useful when you have a large number of variables. Clicking [New Keep List](#) above opens the SPM Notepad that includes KEEP list commands.

```

SPM SPM Notepad: (Untitled 1)

REM Selected 6 variables from: "TreeNet Output 2 - Y1: Summary for 200 Trees"

MODEL Y1
KEEP X1, Z2$, Z1$, X2, X3, X4
  
```

Notice how the variables in the list correspond to the variables that are highlighted. Press **CTRL W** to submit the commands. After you submit the commands, open the model setup again. Notice how only the variables you entered in the KEEP list are now selected as predictors.

Model Setup X

TN Interactions | Class Weights | Penalty | Lags | Automate |  
**Model** | Categorical | Testing | Select Cases | TreeNet | Plots&Options | TN Advanced

Variable Selection

Variable Name	Target	Predictor	Categorical	Weight
W	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Z1\$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Z2\$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
X1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X4	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X5	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X6	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Sort: File Order  Select Predictors  Select Cat.

Filter:  All/Selected  Character  Numeric

Target Type:  
 Classification/ Logistic Binary  
 Regression  
 Unsupervised

Set Focus Class...  
 Target Variable: Y1  
 Weight Variable:  
 Number of Predictors: 6

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

After Building a Model: Save Grove...

Number of Predictors in Model: 6  
 Analysis Engine: TreeNet Gradient Boosting Machine

Cancel Continue Start

## New Keep & Go

The screenshot shows the 'TreeNet Output 2 - Y1: Summary for 200 Trees' window. The 'Variable Importance' tab is selected and highlighted with a yellow box. The window displays a table of variable importance scores and a bar chart. The 'New Keep & Go' button is highlighted with a red box.

Variable	Score	Relative Importance
X1	100.00	100%
Z2\$	62.18	62.18%
Z1\$	50.41	50.41%
X2	47.63	47.63%
X3	44.32	44.32%
X4	37.58	37.58%
X9	10.98	10.98%
X5	9.54	9.54%
X6	9.49	9.49%
X7	9.27	9.27%
X8	8.92	8.92%
X10	8.07	8.07%

Variables count: 12  
Non-zero count: 12

### New Keep & Go

First, highlight the variables in the variable importance list by clicking and dragging. Clicking “**New Keep & Go**” will immediately build a TreeNet model using only the highlighted variables.

## Select All

Variable	Score
X1	100.00
Z2\$	62.18
Z1\$	50.41
X2	47.63
X3	44.32
X4	37.58
X9	10.98
X5	9.54
X6	9.49
X7	9.27
X8	8.92
X10	8.07

### Select All

Click **Select All** to highlight all the variables in the variable importance list (**red rectangle above**). The Select All feature is useful if you wish to quickly build a new TreeNet model using **New Keep & Go** when you have a large number of variables. To accomplish this:

1. Uncheck the “**Show zero importance variables**” checkbox so that the unimportant variables are not included in the list.
2. Click the “**Select All**” button.
3. Click the “**New Keep & Go**” button.

Click “New Keep & Go” to build a TreeNet model using only the variables highlighted (in this case only the variables with a positive variable importance score).

### Variables count and Non-zero count

Provides quick summary information.

**Variables count:** number of variables selected as predictors

**Non-zero count:** number of variables with a variable importance score greater than zero

- ✓ The various summary reports, including Variable Importance rankings, R-squared, and gains charts apply to a SINGLE model. In the example above, this is the model that consists of *exactly* 200 trees because the model with 200 trees was the optimal model.

- ✓ Although the best model is the one normally recommended for further study, there is nothing preventing you from looking at other models. In fact, you can access results for any model in the TreeNet sequence of models, which means that you can review results for models with 1, 2, 3, 4, etc. trees.
- ✓ To economize on resources, we display the overall performance for every size of model beginning with the one-tree model, but we only display detailed summary reports and graphs for select models. You can control how many such models are accompanied with details.

## Resid. Trim % Tab

TreeNet Output 1 - Y1: Summary for 200 Trees

Resid. Trim Cnt	Resid. Outliers CUM %	Resid. Outliers Counts	Resid. Box Plot	Resid. Trim %						
Summary	Dataset	Gains	Variable Importance							
Performance By Largest (absolute value) Residuals Trimming										
Percentile	N Residuals	R-Sq	MSE	MAD	RMSE	MAPE	SSE	AIC	AICc	BIC
100.00%	3,000	0.90288	0.14038	0.29794	0.37468	1.11516	421.14825	-5,866.14798	-5,866.04353	-5,794.07157
99.00%	2,970	0.90819	0.12896	0.28963	0.35911	1.10810	383.02166	-6,059.23032	-6,059.12481	-5,987.27451
98.00%	2,940	0.91151	0.12132	0.28303	0.34831	1.09714	356.68240	-6,177.39818	-6,177.29159	-6,105.56421
97.50%	2,925	0.91324	0.11809	0.28004	0.34365	1.09725	345.42366	-6,224.61345	-6,224.50630	-6,152.84085
97.00%	2,910	0.91496	0.11515	0.27721	0.33934	1.09673	335.08412	-6,266.04287	-6,265.93517	-6,194.33197
96.00%	2,880	0.91786	0.10987	0.27188	0.33147	1.08865	316.43824	-6,336.24234	-6,336.13351	-6,264.65579
95.00%	2,850	0.92070	0.10516	0.26688	0.32428	1.04616	299.69614	-6,395.06976	-6,394.95979	-6,323.60887
90.00%	2,700	0.93267	0.08595	0.24446	0.29318	0.98057	232.07832	-6,601.61683	-6,601.50072	-6,530.80475
80.00%	2,400	0.95226	0.05967	0.20719	0.24428	0.91698	143.21695	-6,741.27225	-6,741.14154	-6,671.87356
75.00%	2,250	0.95845	0.05015	0.19102	0.22395	0.85001	112.84452	-6,709.51782	-6,709.37835	-6,640.89359
70.00%	2,100	0.96482	0.04232	0.17612	0.20571	0.82254	88.86677	-6,617.36406	-6,617.21456	-6,549.56775
60.00%	1,800	0.97506	0.02956	0.14810	0.17192	0.62527	53.20050	-6,314.65335	-6,314.47876	-6,248.70685
50.00%	1,500	0.98252	0.02005	0.12231	0.14160	0.58231	30.07451	-5,840.31345	-5,840.10364	-5,776.55481
40.00%	1,200	0.98846	0.01268	0.09738	0.11260	0.47880	15.21332	-5,217.48635	-5,217.22350	-5,156.40542
30.00%	900	0.99336	0.00704	0.07269	0.08389	0.41717	6.33446	-4,436.75076	-4,436.39901	-4,379.12202
25.00%	750	0.99546	0.00482	0.06035	0.06945	0.31733	3.61770	-3,976.67557	-3,976.25223	-3,921.23469
20.00%	600	0.99700	0.00311	0.04851	0.05580	0.27561	1.86803	-3,439.22714	-3,438.69562	-3,386.46398
10.00%	300	0.99917	0.00081	0.02466	0.02847	0.07520	0.24324	-2,111.24150	-2,110.15440	-2,066.79611
5.00%	150	0.99981	0.00019	0.01217	0.01394	0.03678	0.02914	-1,257.92002	-1,255.64265	-1,221.79240

Precision: 5      N Learn: 7,000      N Test: 3,000      Learn      Test      Holdout

The **Resid. Trim %** tab shows you how the performance of the model is affected after you trim a specific number of residuals according to their absolute size. The absolute value of the residuals (i.e. “absolute residuals”) is computed and then the absolute residuals are sorted in descending order. The residuals are then trimmed according to a certain percentile (i.e. the top 1 percent of the absolute residuals, the top 2 percent of the absolute residuals, etc.)

**Percentile:** the percentile of the distribution of the absolute residuals (any value less than 100% means that trimming has occurred)

**N Residuals:** the number of total absolute residuals considered

**Interpretation** (the interpretation for the other rows is the same)

**First Row:** the first row shows the model performance measures without any residual trimming

**Second Row:** if you trim the top 1% of the residuals (this corresponds to removing the top 30 residuals so that there are  $3,000 - 30 = 2,970$  residuals remaining), then the model will have the following performance: R-Sq = .90819, MSE = .12896, MAD = .28963, RMSE = .35911, MAPE = 1.10810, SEE = 383.02166, AIC = -6,059.23032, AICc = -6,059.12481, and BIC = -5,987.27451

**Third Row:** if you trim the top 2% of the residuals (this corresponds to removing the top 60 residuals so that there are  $3,000 - 60 = 2,940$  residuals remaining), then the model will have the following performance: R-Sq = .91151, MSE = .12132, MAD = .28303, RMSE = .34831, MAPE = 1.09714, SSE = 356.68240, AIC = -6,177.39818, AICc = -6,177.39818, and BIC = -6,105.56421

**\*\*\*Focus on the change in the performance metrics. For instance, the change in the R-sq after trimming the top 2% of absolute residuals is  $.91151 - .90288 = .008$ . Look for large changes.\*\*\***



## Resid. Trim %: Additional Controls

TreeNet Output 1 - Y1: Summary for 200 Trees

Resid. Trim Cnt	Resid. Outliers CUM %	Resid. Outliers Counts	Resid. Box Plot	Score Distribution						
Summary	Dataset	Gains	Variable Importance	Resid. Trim %						
Performance By Largest (absolute value) Residuals Trimming										
Percentile	N Residuals	R-Sq	MSE	MAD	RMSE	MAPE	SSE	AIC	AICc	BIC
100.00%	3,000	0.90288	0.14038	0.29794	0.37468	1.11516	421.14825	-5,866.14798	-5,866.04353	-5,794.07157
99.00%	2,970	0.90819	0.12896	0.28963	0.35911	1.10810	383.02166	-6,059.23032	-6,059.12481	-5,987.27451
98.00%	2,940	0.91151	0.12132	0.28303	0.34831	1.09714	356.68240	-6,177.39818	-6,177.29159	-6,105.56421
97.50%	2,925	0.91324	0.11809	0.28004	0.34365	1.09725	345.42366	-6,224.61345	-6,224.50630	-6,152.84085
97.00%	2,910	0.91496	0.11515	0.27721	0.33934	1.09673	335.08412	-6,266.04287	-6,265.93517	-6,194.33197
96.00%	2,880	0.91786	0.10987	0.27188	0.33147	1.08865	316.43824	-6,336.24234	-6,336.13351	-6,264.65579
95.00%	2,850	0.92070	0.10516	0.26688	0.32428	1.04616	299.69614	-6,395.06976	-6,394.95979	-6,323.60887
90.00%	2,700	0.93267	0.08595	0.24446	0.29318	0.98057	232.07832	-6,601.61683	-6,601.50072	-6,530.80475
80.00%	2,400	0.95226	0.05967	0.20719	0.24428	0.91698	143.21695	-6,741.27225	-6,741.14154	-6,671.87356
75.00%	2,250	0.95845	0.05015	0.19102	0.22395	0.85001	112.84452	-6,709.51782	-6,709.37835	-6,640.89359
70.00%	2,100	0.96482	0.04232	0.17612	0.20571	0.82254	88.86677	-6,617.36406	-6,617.21456	-6,549.56775
60.00%	1,800	0.97506	0.02956	0.14810	0.17192	0.62527	53.20050	-6,314.65335	-6,314.47876	-6,248.70685
50.00%	1,500	0.98252	0.02005	0.12231	0.14160	0.58231	30.07451	-5,840.31345	-5,840.10364	-5,776.55481
40.00%	1,200	0.98846	0.01268	0.09738	0.11260	0.47880	15.21332	-5,217.48635	-5,217.22350	-5,156.40542
30.00%	900	0.99336	0.00704	0.07269	0.08389	0.41717	6.33446	-4,436.75076	-4,436.39901	-4,379.12202
25.00%	750	0.99546	0.00482	0.06035	0.06945	0.31733	3.61770	-3,976.67557	-3,976.25223	-3,921.23469
20.00%	600	0.99700	0.00311	0.04851	0.05580	0.27561	1.86803	-3,439.22714	-3,438.69562	-3,386.46398
10.00%	300	0.99917	0.00081	0.02466	0.02847	0.07520	0.24324	-2,111.24150	-2,110.15440	-2,066.79611
5.00%	150	0.99981	0.00019	0.01217	0.01394	0.03678	0.02914	-1,257.92002	-1,255.64265	-1,221.79240

Precision: 5      N Learn: 7,000      N Test: 3,000      Learn Test Holdout

**Precision:** number of decimal places displayed (the default is 5)

**N Learn:** total number of records in the learn sample

**N Test:** total number of records in the test sample

**Learn, Test, and Holdout Buttons:** view residual trimming statistics for the learn, test, and holdout samples

**Resid. Trim Cnt Tab**

TreeNet Output 1 - Y1: Summary for 200 Trees

Summary		Dataset	Gains				Variable Importance	Resid. Trim %		
<b>Resid. Trim Cnt</b>		Resid. Outliers CUM %	Resid. Outliers Counts				Resid. Box Plot	Score Distribution		
Performance By Largest (absolute value) Residuals Trimming										
Percentile	N Residuals	R-Sq	MSE	MAD	RMSE	MAPE	SSE	AIC	AICc	BIC
100.00%	3,000	0.90288	0.14038	0.29794	0.37468	1.11516	421.14825	-5,866.14798	-5,866.04353	-5,794.07157
99.97%	2,999	0.90327	0.13973	0.29756	0.37380	1.11517	419.04948	-5,878.16751	-5,878.06302	-5,806.09510
99.83%	2,995	0.90392	0.13756	0.29618	0.37089	1.11539	411.98167	-5,917.24338	-5,917.13875	-5,845.18699
99.67%	2,990	0.90484	0.13543	0.29469	0.36801	1.11424	404.94214	-5,953.86056	-5,953.75576	-5,881.82422
99.17%	2,975	0.90732	0.13042	0.29083	0.36114	1.10746	388.00108	-6,036.04879	-6,035.94345	-5,964.07279
98.33%	2,950	0.91004	0.12366	0.28513	0.35166	1.10734	364.80155	-6,142.11048	-6,142.00425	-6,070.23576
96.67%	2,900	0.91569	0.11333	0.27540	0.33664	1.09562	328.64374	-6,290.72598	-6,290.61790	-6,219.05638

Precision: 5      N Learn: 7,000      N Test: 3,000      Learn      Test      Holdout

The absolute value of the residuals, called “absolute residuals,” is computed and then the absolute residuals are sorted in descending order. The residuals are then trimmed according to a certain count (i.e. the largest absolute residual, the top 5 largest absolute residuals, etc.). The **Resid. Trim Cnt Tab** shows you how the performance of the model is affected after you trim a particular number of the absolute residuals.

**Percentile:** percentile of the distribution of the absolute residuals (any value less than 100% means that trimming has occurred)

**N Residuals:** total number of absolute residuals considered (Note the first row in this column corresponds to no trimming)

**Interpretation**

**First Row:** no trimming

**Second Row:** trimming the largest absolute residual results in following metrics for the model: R-sq of .90327, MSE = .13973, MAD=.29756, RMSE=.37380, MAPE = 1.11517, SSE = 419.04948, AIC = -5,878.16751, AICc = -5,878.06302, and BIC = -5,806.09510

**Third Row:** trimming the 5 (= 3,000 – 2,995) largest absolute residual results in following metrics for the model: R-sq of .90392, MSE = .13756, MAD=.29618, RMSE=.37089, MAPE = 1.11539, SSE = 411.98167, AIC = -5,917.24338, AICc = -5,917.13875, and BIC = -5,845.18699

## Resid. Trim Cnt: Additional Controls

TreeNet Output 1 - Y1: Summary for 200 Trees

Summary		Dataset	Gains	Variable Importance	Resid. Trim %					
<b>Resid. Trim Cnt</b>		Resid. Outliers CUM %	Resid. Outliers Counts	Resid. Box Plot	Score Distribution					
Performance By Largest (absolute value) Residuals Trimming										
Percentile	N Residuals	R-Sq	MSE	MAD	RMSE	MAPE	SSE	AIC	AICc	BIC
100.00%	3,000	0.90288	0.14038	0.29794	0.37468	1.11516	421.14825	-5,866.14798	-5,866.04353	-5,794.07157
99.97%	2,999	0.90327	0.13973	0.29756	0.37380	1.11517	419.04948	-5,878.16751	-5,878.06302	-5,806.09510
99.83%	2,995	0.90392	0.13756	0.29618	0.37089	1.11539	411.98167	-5,917.24338	-5,917.13875	-5,845.18699
99.67%	2,990	0.90484	0.13543	0.29469	0.36801	1.11424	404.94214	-5,953.86056	-5,953.75576	-5,881.82422
99.17%	2,975	0.90732	0.13042	0.29083	0.36114	1.10746	388.00108	-6,036.04879	-6,035.94345	-5,964.07279
98.33%	2,950	0.91004	0.12366	0.28513	0.35166	1.10734	364.80155	-6,142.11048	-6,142.00425	-6,070.23576
96.67%	2,900	0.91569	0.11333	0.27540	0.33664	1.09562	328.64374	-6,290.72598	-6,290.61790	-6,219.05638

Precision: 5      N Learn: 7,000      N Test: 3,000      Learn Test Holdout

**Precision:** number of decimal places displayed (the default is 5)

**N Learn:** total number of records in the learn sample

**N Test:** total number of records in the test sample

**Learn, Test, and Holdout Buttons:** view residual trimming statistics for the learn, test, and holdout samples

**Resid. Outliers CUM % Tab**

TreeNet Output 2 - Y1: Summary for 200 Trees

Summary	Resid. Outliers CUM %	Gains	Variable Importance	Resid. Trim %
Resid. Trim Cnt	Resid. Outliers Counts	Resid. Box Plot	Score Distribution	
Percentage of Error Statistics due to Largest (absolute value) Residuals				
% Residuals	N Residuals	% MSE	% MAD	% MAPE
1.00%	30	9.05301	3.75975	32.77155
2.00%	60	15.30716	6.90291	42.34766
2.50%	75	17.98051	8.35675	45.59834
3.00%	90	20.43559	9.74994	48.35527
4.00%	120	24.86298	12.39570	52.69871
5.00%	150	28.83833	14.90276	56.26085
10.00%	300	44.89391	26.15650	67.93914
20.00%	600	65.99370	44.36828	79.62950
25.00%	750	73.20551	51.91511	83.32589
30.00%	900	78.89893	58.62122	86.29256
40.00%	1,200	87.36775	70.17445	90.80362
50.00%	1,500	92.85892	79.47371	94.00749
60.00%	1,800	96.38766	86.92673	96.33140
70.00%	2,100	98.49591	92.68084	97.98123
75.00%	2,250	99.14099	94.93581	98.60776
80.00%	2,400	99.55644	96.74377	99.10877
90.00%	2,700	99.94224	99.17222	99.77044
95.00%	2,850	99.99308	99.79585	99.94040
96.00%	2,880	99.99638	99.86779	99.96185

Precision: 5      N Learn: 7,000      N Test: 3,000      Learn      Test      Holdout

The absolute value of the residuals, called “absolute residuals,” is computed and then the absolute residuals are sorted in descending order. The **Resid. Outliers CUM%** shows you the percentage of the sorted absolute residuals that account for a certain percentage of the model error (i.e. MSE, MAD, or MAPE in the regression setting).

**Interpretation**

**First Row:** the top 1% of absolute residuals (1% 3,000 is 30; there are 3,000 residuals for a test sample) account for 9.05301% of the MSE, 3.75975% of the MAD, and 32.77155% of the MAPE

**Third Row:** the top 2.5% of absolute residuals (2.5% of the 3,000 is 75; there are 3,000 residuals for a test sample) account for 17.98051% of the MSE, 8.35675% of the MAD, and 45.59834% of the MAPE

And so on...

**Precision:** number of decimal places

**N Learn:** number of records in the learn data

**N Test:** number of records in the test data

**Learn, Test, and Holdout Buttons:** click the desired button to see cumulative percentage statistics for the residuals

**Resid. Outliers Counts Tab**

TreeNet Output 2 - Y1: Summary for 200 Trees

Summary	Dataset	Resid. Outliers Counts	Variable Importance	Resid. Trim %
Resid. Trim Cnt	Resid. Outliers CUM %		Resid. Box Plot	Score Distribution
Percentage of Error Statistics due to Largest (absolute value) Residuals				
% Residuals	N Residuals	% MSE	% MAD	% MAPE
0.03%	1	0.49835	0.16208	4.23057
0.17%	5	2.17657	0.75634	13.66439
0.33%	10	3.84808	1.42000	20.07174
0.83%	25	7.87066	3.20151	30.31210
1.67%	50	13.37930	5.89492	39.84550
3.33%	100	21.96483	10.64777	49.96134

Precision: 5

N Learn: 7,000 N Test: 3,000

Learn Test Holdout

The absolute value of the residuals, called “absolute residuals,” is computed and then the absolute residuals are sorted in descending order. The **Resid. Outliers Counts** shows you the number of residuals (largest absolute residual, top 5 absolute residuals, and so on) that account for a certain percentage of the model error (i.e. MSE, MAD, or MAPE in the regression setting).

**Interpretation**

**First Row:** the largest absolute residual accounts for .49835% of the MSE, .16208% of the MAD, and 4.23057 of the MAPE

**Second Row:** the 10 largest absolute residuals account for 3.8408% of the MSE, 1.42% of the MAD, and 20.07174% of the MAPE

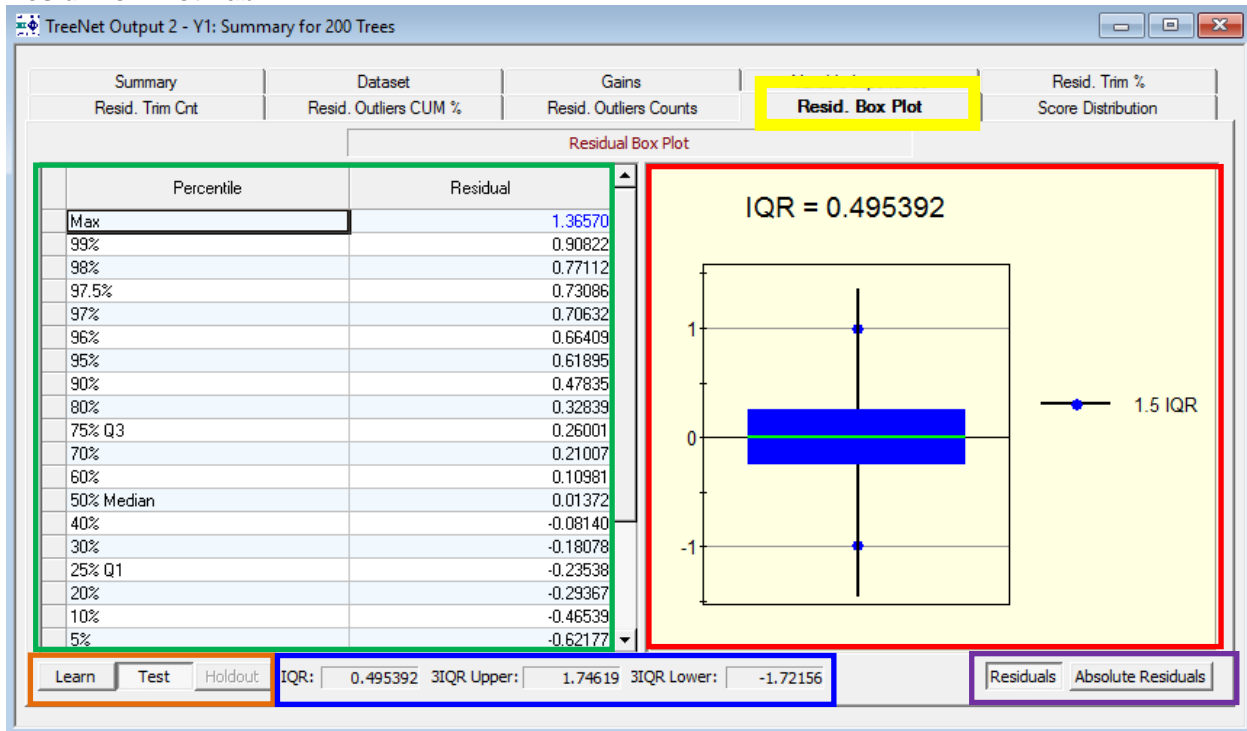
And so on...

**Precision:** number of decimal places

**N Learn:** number of records in the learn data

**N Test:** number of records in the test data

**Learn, Test, and Holdout Buttons:** click the appropriate button to see cumulative percentage statistics for the residuals

**Resid. Box Plot Tab**

The **Resid. Box Plot** tab shows a box plot of the residuals and associated statistics.

**Percentile and Residual:** The percentiles of the distribution of the residuals or the absolute residuals (see the “**Residual and Absolute Residual Buttons**” entry below).

**Learn, Test, and Holdout Buttons:** In the picture above, the Test button is selected and shows the distribution of the residuals for the test sample (you can also see the distribution for the absolute residuals; see the “**Residual and Absolute Residual Buttons**” entry below). For example, clicking the “Learn” button will show the boxplot and associated percentile statistics for the residuals (or absolute residuals) for the learning data.

**IQR, 3IQR Upper, and 3IQR Lower:**

$IQR = Q3 - Q1$  where Q3 is the third quartile and Q1 is the first quartile.

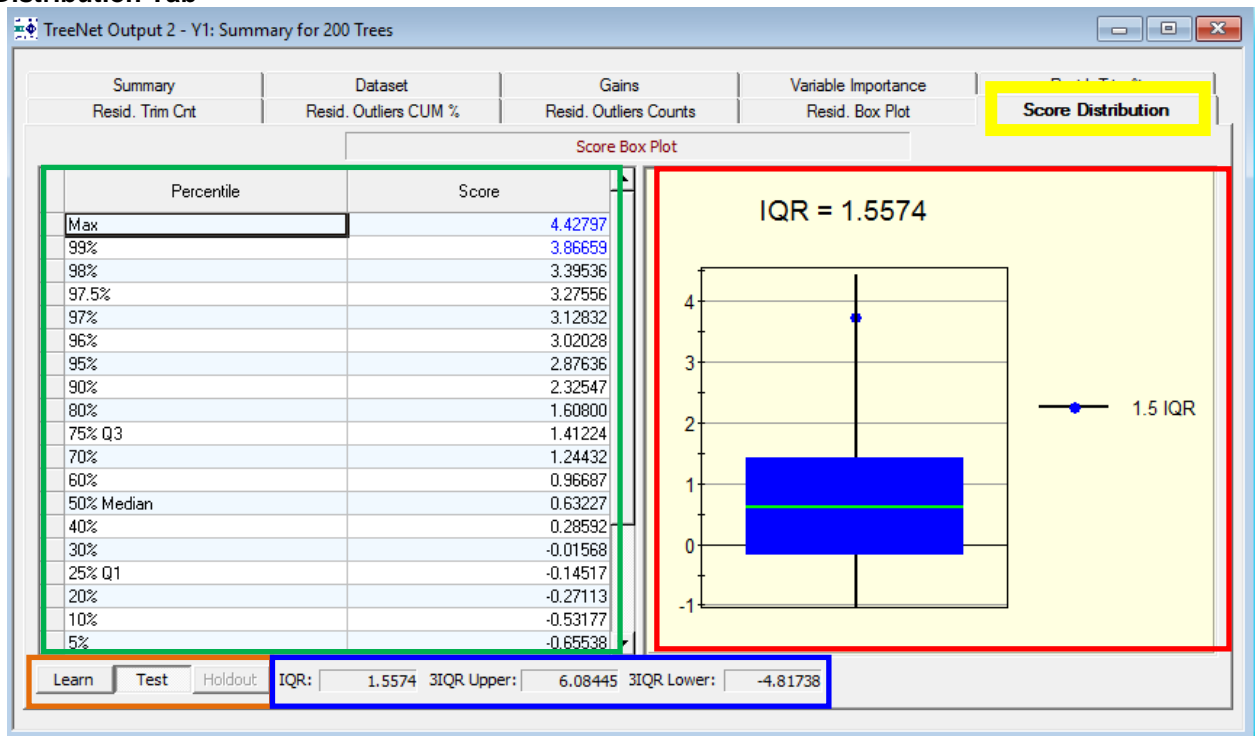
$3IQR\ Upper = Q3 + |3 \cdot IQR|$

$3IQR\ Lower = Q1 - |3 \cdot IQR|$

**Boxplot of the Residuals:** shows the boxplot of the residuals or the absolute residuals (see the “**Residual and Absolute Residual Buttons**” entry directly below).

**Residual and Absolute Residual Buttons:** click either the “Residual” button or the “Absolute Residual” button to see the desired boxplot, percentiles, and statistics

## Score Distribution Tab



The **Score Distribution Tab** shows you the boxplot of the predicted values and associated statistics.

**Percentile and Residual:** percentiles of the distribution of the predicted values.

**Learn, Test, and Holdout Buttons:** In the picture above, the test button is selected and shows the distribution of the predicted values for the test sample. For example, clicking the “Learn” button shows the boxplot and associated percentile statistics for the predicted values using the learn data.

**IQR, 3IQR Upper, and 3IQR Lower:**

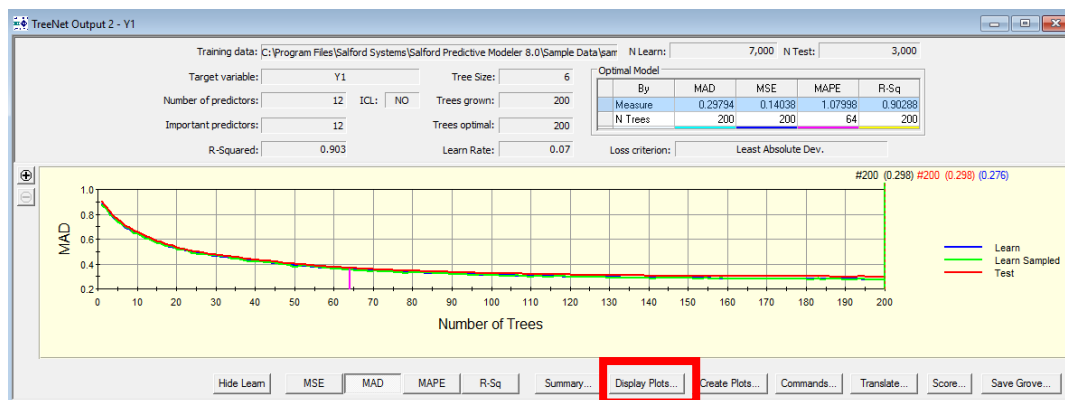
$IQR = Q3 - Q1$  where Q3 is the third quartile and Q1 is the first quartile.

$3IQR\ Upper = Q3 + |3 * IQR|$

$3IQR\ Lower = Q1 - |3 * IQR|$

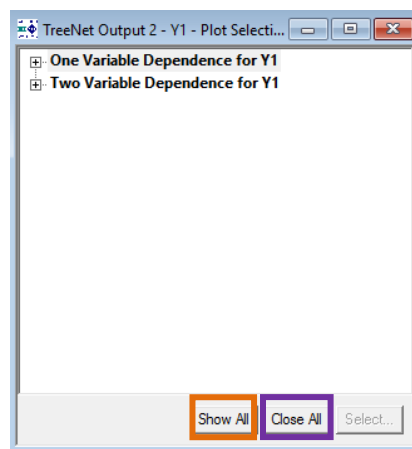
## Display Plots Button

In TreeNet software you can specify that partial dependency plots (PDPs) are created before the model is built (see the [Plots & Options Tab](#) section above; to create PDPs after the model has been built, see the [Create and Viewing Dependency Plots](#) section below). To view the partial dependency plots, click the “Display Plots...” button (see **red rectangle below**)



After you click the “Display Plots...” button you will see the following menu.

## View All Plots



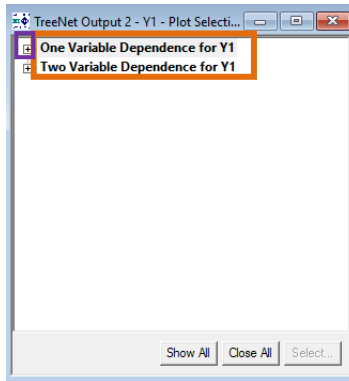
Click “**Show All**” to display all plots (this may take a while if you requested many plots; see the [Warning: Number of PDPs](#) section for information on the number of plots generated).

Click “**Close All**” to close all plots.

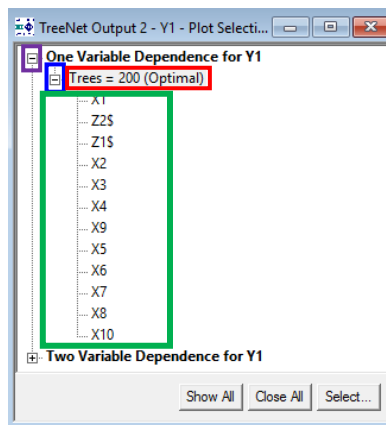


**View All One (or Two) Variable PDPs Only**

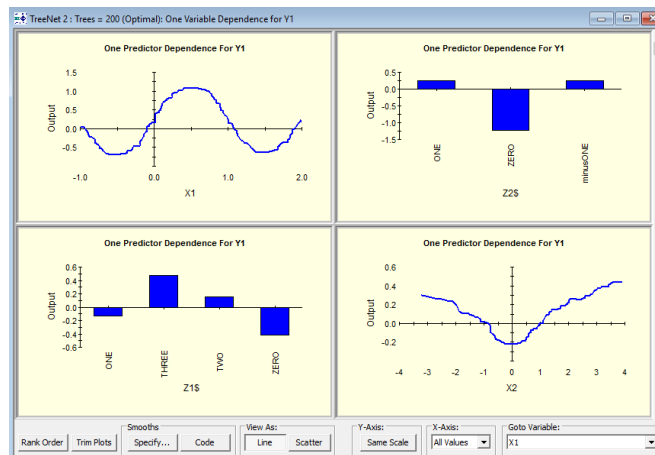
Click either one of the “plus” sign + for the one or two-variable partial dependency plots in the **orange rectangle below**:



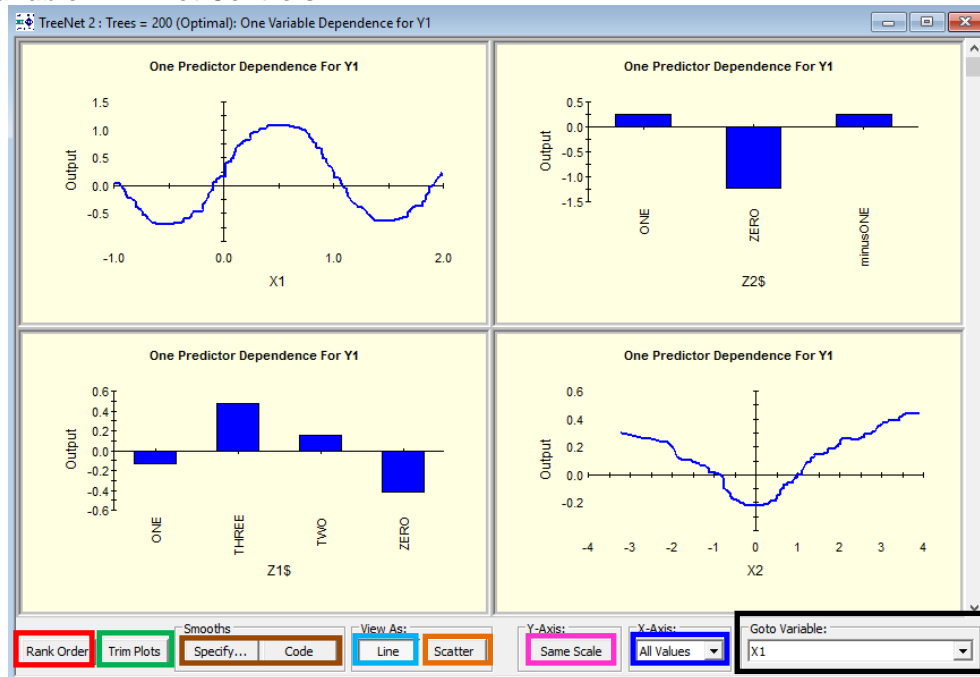
Click the other plus sign (**blue rectangle below**) to expand the list of variables with one variable PDPs.



Click the “**Trees = 200 (Optimal)**” label to view all one variable PDPs or click the name of an individual variable to view a partial dependency plot (any name in the **green rectangle above**) for a single variable. Here is the result of clicking “**Trees = 200 (Optimal)**”



## All One Variable PDP Plot Controls



**Rank Order**– “de-scales” the horizontal axis such that all sampled points are spaced uniformly. This helps viewing “congested” areas of plots, especially when outliers are present.

**Trim Plots**– truncate plot tails usually due to outliers.

**Line**– view the PDPs as shown in the picture above

**Specify & Code**– controls for spline approximations to the partial dependency plots (see the [Spline Approximations](#) section below for more details)

**Scatter**– view individual points that were sampled in plots construction. This is useful in identifying possible outliers or “empty” areas of data.

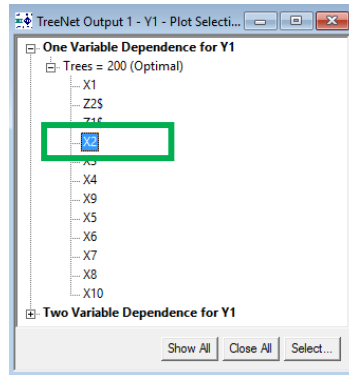
**Y-Axis: Same Scale** – imposes an identical scale on all vertical axes and allows you to see the comparative effect of different variable contributions

**X-Axis: All Values or Zoomed** – zoom in on the x-axis if desired (Zoomed option) otherwise the “All Values” option that is shown above will be used

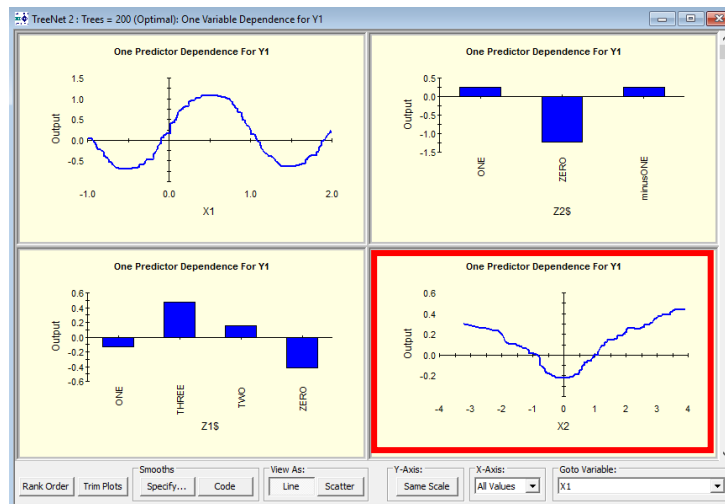
**Goto Variable** – use to navigate directly to the partial dependency plot of a particular variable of interest. This is useful when there is a large number of predictors (in that case searching through all of the plots can be quite tedious)

**View an Individual PDP**

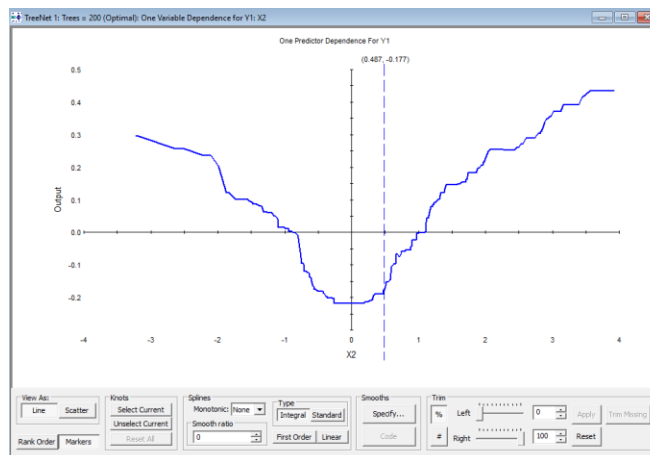
1. Click the desired variable name in the Plot Selection setup (**green rectangle below**) after clicking the desired plus signs +



2. Double click on the desired plot if viewing all PDPs (**red rectangle below**; this is applicable when viewing either all one variable plots or all two variable plots)



The result of using either (1) or (2) is the following:



## Partial Dependency Plots (PDPs)

Partial dependency plots represent a very powerful interpretational side of TreeNet. They were designed to help understand how individual variables or pairs of variables contribute to the predicted response once the effects of all remaining predictors have been accounted for.

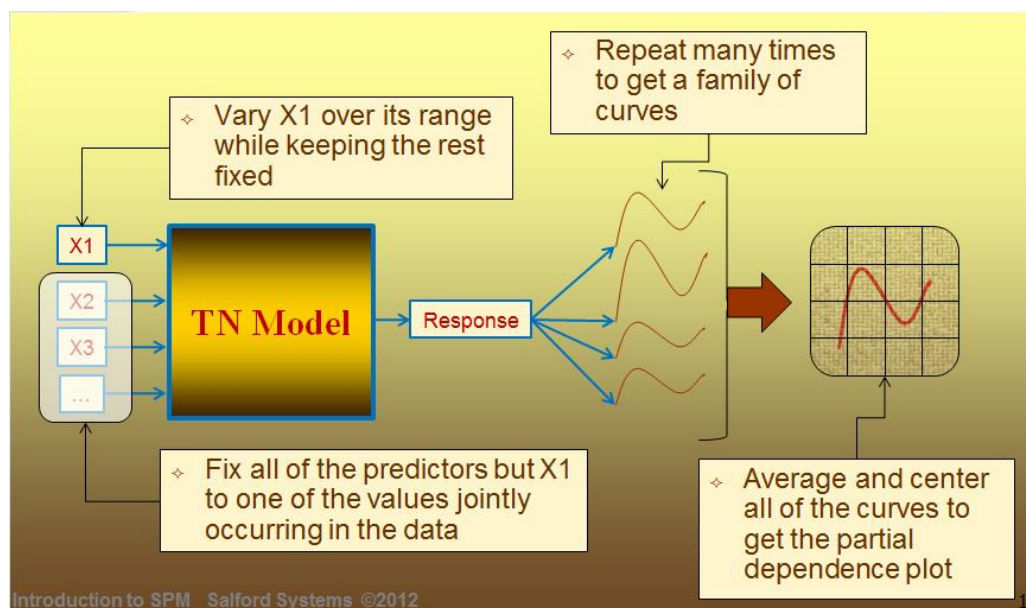
Depending on the complexity of a TreeNet model, the plots can be either exact or at the very least provide useful visual approximations to the nature of dependencies. The former case arises for truly additive models or models with completely isolated interacting pairs of variables, for example, utilizing the ICL language introduced later in this manual.

### Constructing One Variable PDPs

As an example, let's say that we have built a TreeNet model and we want to construct a partial dependency plot for a predictor variable X1. To generate the partial dependency plot for X1 we follow the following steps:

1. Randomly select a record
2. Plug the record from Step 1 into the model. This produces a predicted value
3. Hold all values for all variables in the record constant except change the value for X1. Plug this new artificial record into the model which produces a predicted value.
  - a. By "change the value for X1" we mean just use other values occurring in the data
4. Repeat Step 3 for every unique value of X1. This generates multiple predicted values that are plotted against the variable X1 and then connected to form one curve.
5. Repeat Steps 1-4 many times (the default in SPM is 500). This produces multiple curves.
6. Average and center the curves to get the partial dependency plot for the variable X1.

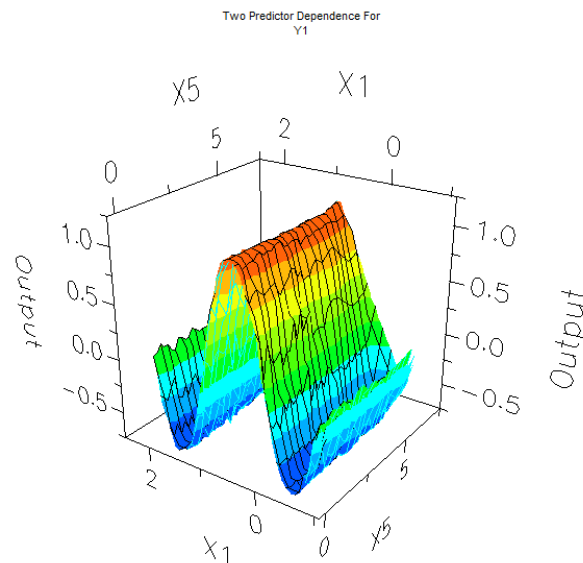
This process is illustrated in the picture below:



### Constructing Two Variable PDPs

To construct two-variable partial dependency plots, follow the same procedure discussed above in the Constructing One Variable PDPs section except that when plotting two variables, the result is a surface instead of a line. As an example, let's say that we have built a TreeNet model and we want to construct a partial dependency plot for predictor variables X1 and X5. To generate the partial dependency plot for X1 and X5 follow these steps:

1. Randomly select a record.
2. Plug the record from Step 1 into the model. This produces a predicted value.
3. Hold all values for all variables in the record constant except change the value for X1 and X5. Plug this new artificial record into the model which produces a predicted value.
  - a. By "change the value for X1 and X5" we mean just use other values that occur for X1 and X5 in the data.
4. Repeat Step 3 for every unique combination of X1 and X5. This generated multiple predicted values which are plotted against both X1 and X5 to form one surface.
5. Repeat Steps 1-4 many times (the default in SPM is 500). This produces multiple surfaces.
6. Average and center the surfaces to get the two-variable partial dependency plot for the variables X1 and X5.



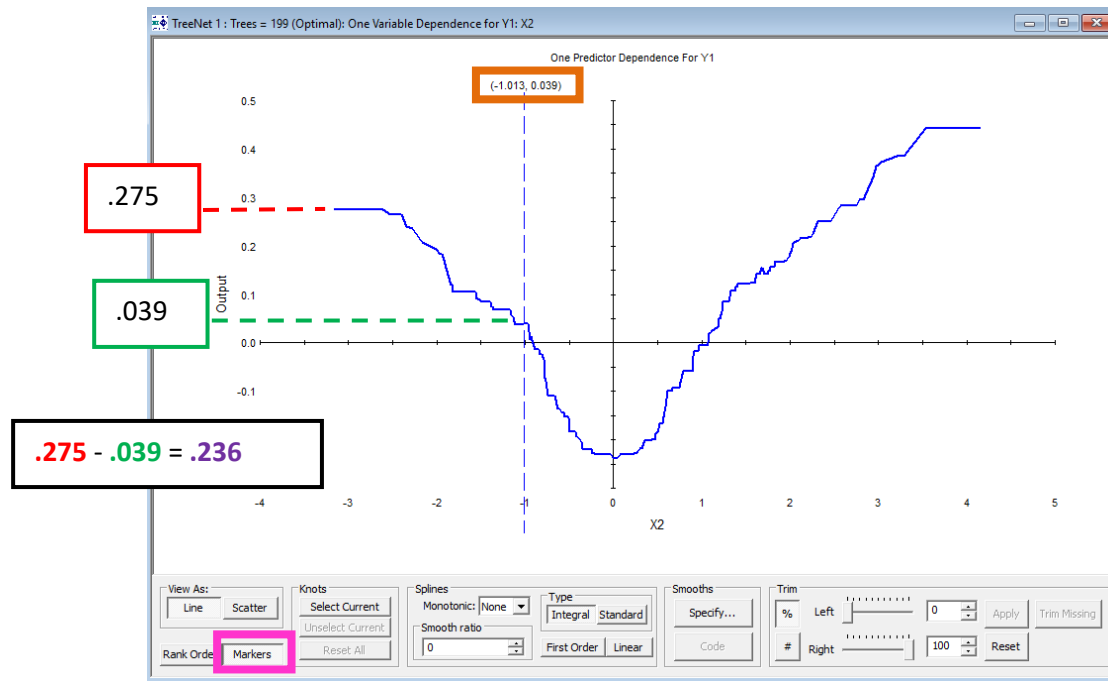
## Interpreting One Variable PDPs

Important things to remember when interpreting Partial Dependency Plots (PDPs) include:

1. **The scale of the Y-axis:** the average of the scale used to make predictions during the modeling process
  - a. **Example:** All predictions in the binary classification setting are on the  $\frac{1}{2}$  log odds scale during the modeling run (Note: SPM will automatically convert this to probabilities for the output you see and when you score the model), and thus the Y-axis for PDPs in the binary classification setting is the average  $\frac{1}{2}$  log odds.
  - b. **Example:** All predictions in the Least Squares, Least Absolute Deviation, and Huber-M are made on the scale of the target variable, and thus the Y-axis for PDPs when using Least Squares, Least Absolute Deviation, and Huber-M is the average target variable scale.
  - c. **Why is the Y-axis an average?** In constructing the partial dependency curve we average multiple curves to obtain the final single curve shown to the user and thus the Y-axis is an average response (see the [Constructing One Variable PDPs](#) section for a detailed description of the construction of one variable PDPs).
2. **PDPs are marginal plots:** should only be interpreted in terms of the change in Y, given a change in the predictor variable(s).

## LAD Loss & Continuous Predictors

Consider the following partial dependency plot for a predictor variable called X2 that was used in a TreeNet model with a continuous target variable named Y1.



Notice that TreeNet has captured a (marginal) nonlinear relationship between the predictor variable X2 and the target variable Y1. The red, green, and pink rectangles were generated outside of SPM.

### Interpretation

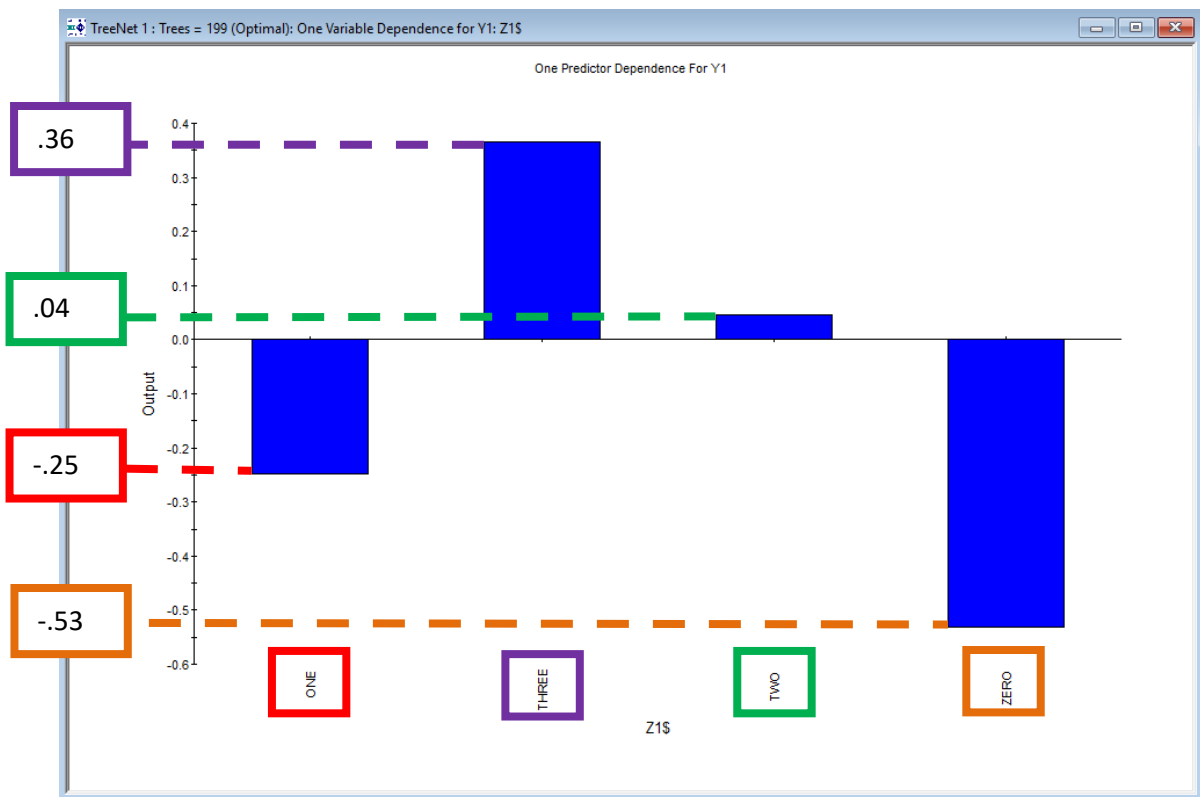
If X2 is increased from -3 to -1, then the target variable Y1 decreases, on average, by approximately **.236** ( $.236 = .275 - .039$ ) after accounting for the other variables in the model

See the picture above. Click the “**Markers**” button above to display the **dotted blue line** shown in the picture. The **dotted blue line** has a **pair of coordinates (x-value, y-value)** that allow you to obtain the values used in the calculation  $.236 = .275 - .039$ .

If X2 is increased from 0 to 2, then the target variable Y1 increases, on average, by approximately **.434** ( $.434 = .196 - [-.238]$ ) after accounting for the other variables in the model

If X2 is increased from 2 to 3, then the target variable Y1 increases, on average, by approximately **.173** ( $.173 = .369 - .196$ ) after accounting for the other variables in the model

## LAD Loss & Categorical Predictors



In this example, we have a continuous target variable called Y1 and a categorical predictor Z1 (Note: you can see the Z1\$ label in the bottom of the plot; the dollar sign \$ means that the variable has character values). The dotted lines and rectangles shown in the picture were created outside of TreeNet software.

### Interpretation:

If Z1\$ has a value of “**ONE**”, then the target variable Y1 **decreases by approximately .25** after accounting for the other variables in the model

If Z1\$ has a value of “**THREE**”, then the target variable Y1 **increases by approximately .36** after accounting for the other variables in the model

If Z1\$ has a value of “**TWO**”, then the target variable Y1 **increases by approximately .04** after accounting for the other variables in the model

If Z1\$ has a value of “**ZERO**”, then the target variable Y1 **decreases by approximately .53** after accounting for the other variables in the model

The Y-axis for the Least Absolute Deviation, Least Squares, Huber, and Quantile loss functions have Y-axis scales that are the average of the actual target variable.

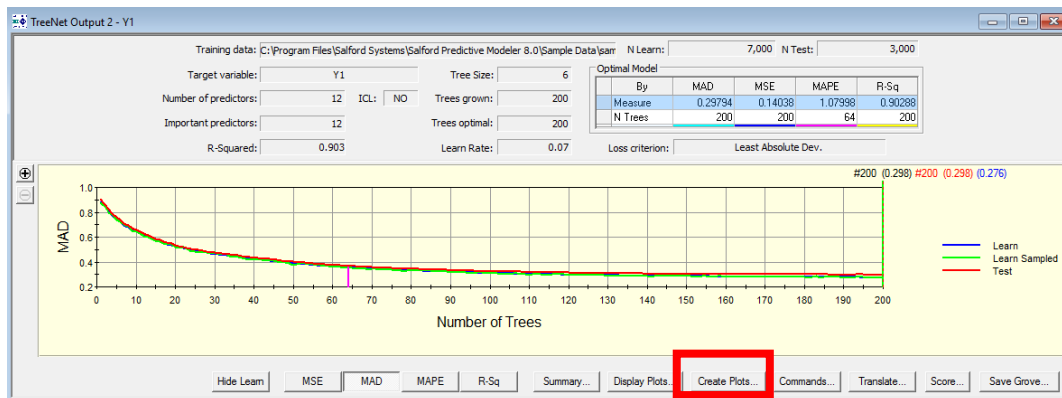


## Spline Approximations to the PDPs

Users have the option to produce spline approximations to the one variable partial dependency plots for continuous predictor variables. See the [Spline Approximations to the Partial Dependency Plots section](#) for more information.

## Create Plots Button

TreeNet has a powerful facility to generate partial dependency plots for any variable, or pair of variables, and any model size selected by the user. Click the **Create Plots...** button in the TreeNet output window (this is only available for most recent TreeNet modeling run). See the [Create Plots Button section](#) for more information.



## Example: Building a Classification/Logistic Binary Model

TreeNet “Logistic Binary” is the recommended way to model any binary outcome and is probably the most popular use of TreeNet. Although most analysts think of this type of model as CLASSIFICATION, in TreeNet we consider it as a special type of regression. This is because we want to produce a PROBABILITY of response for and not just a simple prediction of “YES” or “NO.” TreeNet’s ability to produce extremely accurate rankings of data from “most likely” to “least likely” underpins its strong performance in many data mining settings.

The first point to make is that this section is concerned exclusively with the binary or two-valued target variable. This variable is often coded as “0” and “1” or “YES” and “NO” or “RESPONSE” and “NONRESPONSE,” etc. Such models are by far the most common in data mining applications and are used to predict:

- ◆ Default in credit risk scoring
  - ◆ Response in targeted marketing campaigns
  - ◆ Fraud in insurance, credit card transactions, and loan applications
  - ◆ Disease (or abnormal versus normal) in biomedical settings
- ✓ If you want to analyze such data, then the logistic regression style of TreeNet is likely to give you the best results.

There is no such thing as a “best model” absent a context and an objective. When you run TreeNet, the software builds its model in stages, adding a tree at each stage in an effort to refine performance. Typically, the first tree yields a modest performance, the second tree improves on it, the third tree improves further, and so on. At some point in the process, adding further trees does no good and may even do some harm! The entire set of trees grown is called the model sequence, and after the specified number of trees is generated, we want to determine how many of the trees to keep. To assist in this process we track the following four performance criteria at each stage of the modeling sequence and identify the best number of trees to keep for each of the performance criteria:

**Classification Accuracy:** based on a straightforward tally of how often the model tags a record correctly or incorrectly. If you intend to use the model to simply tag data as either “YES” or “NO” then this criterion will help you select the best model.

**Area under the ROC curve:** most commonly-used model criterion in machine learning and is a measure of overall model performance tied closely to the ability of the model to correctly RANK records from most likely to least likely to be a “1” or a “0.”

**Average Log-Likelihood:** similar to ROC but emphasizes the probability interpretation of the model predictions. This is an option for technical users who may prefer this measure for theoretical reasons.

**Lift in the top X-percentile:** focuses on the ability of the model to concentrate the responders or the “1”s in the highest ranking records. This criterion is useful if the model is intended to exclusively identify, say, the top 1%, 5% or 10% of a data set. In this case, you can safely ignore model performance in the lower percentiles.

- ✓ It is not possible to tell in advance how many trees will be optimal.
- ✓ The best number of trees can vary quite a bit depending on which performance criterion you select.
- ✓ Typically, the best model for classification accuracy has many fewer trees than the best model for best probability estimates (Ave LL) or overall performance (ROC). Best lift is also usually accomplished with relatively few trees.

- ✓ The best model is normally determined by performance on test data or via cross validation. When no independent testing is specified (exploratory runs), the optimal model is always set to the largest number of trees.
- ✓ It is possible to manually select a non-optimal model of any size for graphical display, obtaining summary reports, scoring and translation.

The following section describes how to set up a logistic regression run in TreeNet, and we also provide detailed comments on every setup option available.

## Creating a Sample Example Model Data Set

A sample data set, SAMPLE.CSV, is supplied as part of the TreeNet installation. It is located in the *Sample Data* folder.


The following table lists the major groups of variables in the data set.

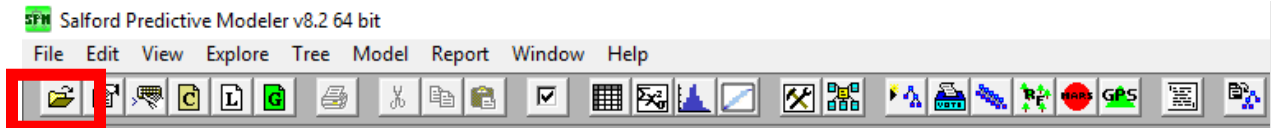
Variable Names	Description
X1, ... , X10	Continuous predictors
Z1\$, Z2\$	Categorical predictors (character)
Y1	Continuous target
Y2	Binary target coded +1 and -1
Y3	Categorical target with 4 levels: 1, 2, 3, and 4
W	Weight variable
T	Learn/Test dummy indicator (0 – learn, 1 – test)

## Reading Data In

To open the input file `SAMPLE.CSV`:

Select **Open>Data File...** from the **File** menu.

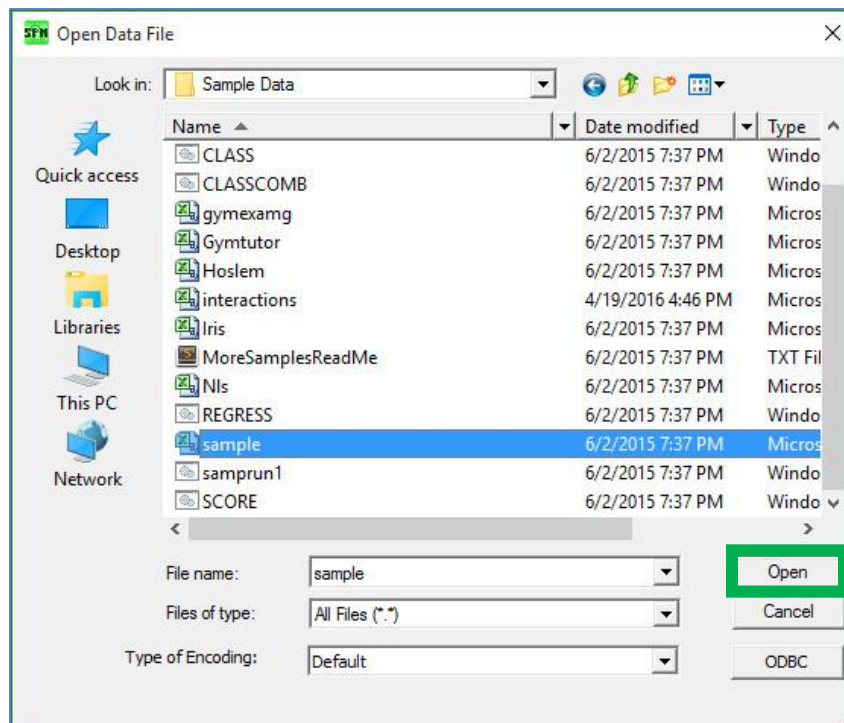
- ✓ You may simply click on the  button in the toolbar



Use the **Open Data File** dialog window to navigate to the *Sample Data* folder in the SPM installation directory.

Choose **Delimited (\*.csv,\*.dat,\*.txt)** in the **Files of type:** selection box.

Highlight `SAMPLE.CSV`.

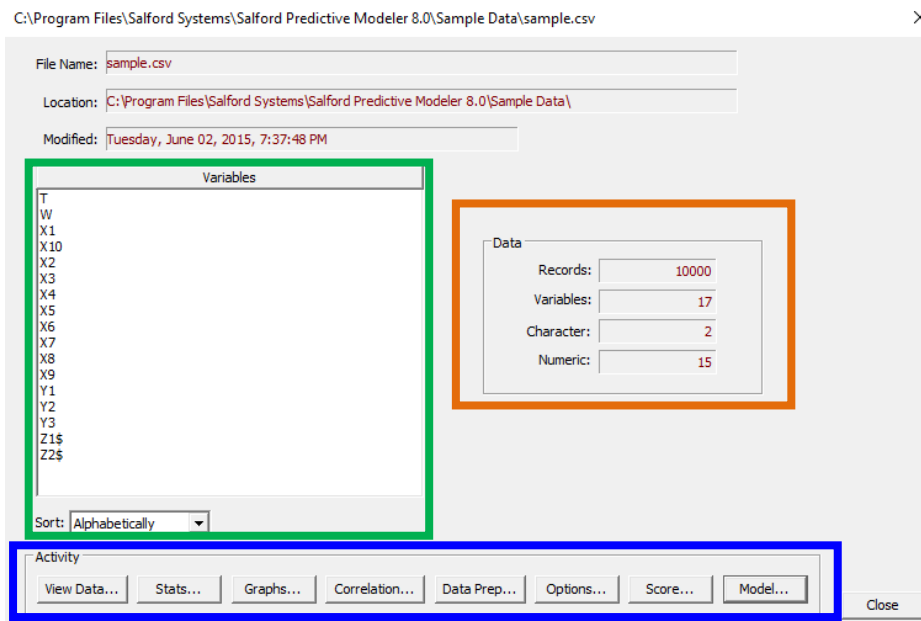


Click the **Open** button to load the data set into TreeNet.

- ⚠ Make sure that the data set is not currently being accessed by another application. If it is, the data loading will fail. This is a common source of problems when dealing with Excel and ASCII files.

After you click “Open” you will see the “Activity Window”:

## The Activity Window



**Activity Window Buttons:** The buttons at the bottom of the activity window have the following functions

- View Data...** allows the user to see the dataset
  - Stats...** allows the user to compute summary statistics for each variable
  - Graphs...** allows the user to build graphs
  - Correlation...** allows the user to compute measures of correlation and similarity
  - Data Prep...** allows user to perform data preparation operations inside of SPM
  - Options...** allows the user change settings for the software
  - Score...** allows the user to use a previously saved model or a model currently open inside of SPM to generate predictions for a dataset
  - Model...** takes the user directly to the model setup
- ✓ The activity window step can be skipped using the **Options** dialog.

### Data

**Records:** number of observations = number of rows in the dataset

**Variables:** total number of variables = total number of columns in the dataset

**Character:** number of variables with character values (Example: a variable with values “Female” or “Male”)

**Numeric:** number of variables with only numeric values


\*\*\*Note: Variables = Character + Numeric

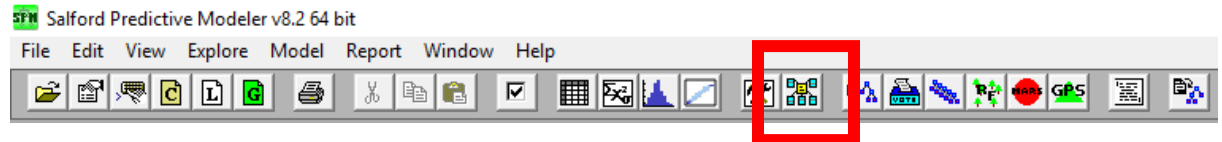
### Variables

Provides a list of the variables in alphabetical order. You can change this to the file order by clicking on the dropdown box (see the bottom of the green rectangle above)

## Setting up the Classification Model

We will now setup a TreeNet model for a binary classification problem. In the **Model** tab we specify our core model variables. Variables specified in this dialog include a target (dependent), predictors (independent), categorical designation, and optionally a single case-weight variable.

- i. To open the model setup click the model setup shortcut button  (red rectangle below) or click “Model” in the Activity Window (see above). A data set must be currently open in the SPM for this operation to work.



After clicking the model setup shortcut button you will see the model setup dialog pictured below.

Model Setup ×

TN Interactions | Class Weights | Penalty | Lags | Automate | Plots&Options | TN Advanced

**Model** | Categorical | Testing | Select Cases | TreeNet

Variable Selection

Variable Name	Target	Predictor	Categorical	Weight
Y1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Y2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Y3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
W	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Z1\$	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Z2\$	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
X1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Sort: File Order

Filter:  All/Selected  Character  Numeric

Select Predictors  Select Cat.

Target Type:  Classification/Logistic Binary  Regression  Unsupervised

Set Focus Class...

Target Variable: \_\_\_\_\_

Weight Variable: \_\_\_\_\_

Number of Predictors: 17

Automatic Best Predictor Discovery:  Off  Discover only  Discover and run

Maximum variables for each class: 8

Number of Predictors in Model: 17

After Building a Model: Save Grove...

Analysis Engine: TreeNet Gradient Boosting Machine

Cancel Continue Start

## Model Tab

Model Setup

TN Interactions | Class Weights | Penalty | Lags | Automate | Plots&Options | TN Advanced

**Model** | Categorical | Testing | Select Cases | TreeNet

Variable Selection

Variable Name	Target	Predictor	Categorical	Weight
Y1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Y2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Y3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
W	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Z1\$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Z2\$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
X1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Sort: File Order

Filter:  All/Selected  Character  Numeric

Automatic Best Predictor Discovery:  Off  Discover only  Discover and run

Maximum variables for each class: 8

After Building a Model: Save Grove...

Analysis Engine: TreeNet Gradient Boosting Machine

Number of Predictors in Model: 12

Target Type:  Classification/Logistic Binary  Regression  Unsupervised

Set Focus Class...

Target Variable: Y2

Weight Variable:

Number of Predictors: 12

Cancel Continue Start

The first tab that you will see is the “Model” tab (yellow rectangle)

1. Set the Analysis Engine to **TreeNet Gradient Boosting Machine**.
2. Set the Target Type to **Classification/Logistic Binary**.
  - a. Select Classification/Logistic Binary when your target is binary (two classes) or multinomial targets (more than 2 classes). TreeNet will automatically determine if the target is binomial or multinomial.
3. Click the checkbox for the desired variable in the target variable: in this case **the target variable is the binary variable called Y2**. A **target variable** is the variable that you are trying to predict.
4. Select potential predictors under the **Predictor** column.
  - a. You may highlight multiple variables first by using a left mouse click and/or the <Shift> or <Ctrl> keys while clicking in the Variable Name column. Once selected you may either check or uncheck the corresponding check marks simultaneously for the selected variables by using the check box at the bottom of the appropriate column (i.e. Select Predictors or Select Cat. Checkboxes).
5. Indicate which predictors are categorical by placing check marks under the **Categorical column** for the corresponding variables. Variables with character values, are automatically set to Categorical have a dollar sign \$ appended to the variable name.
6. Optionally, specify a weight variable using a check box under the **Weight column** (these are observation weights; only one weight variable may be active at a time).
7. **Sort** option: order the variables listed alphabetically or by their file order.
8. Click the **Set Focus Class...** button to specify the event of interest (see the next section for more information on the focus class).

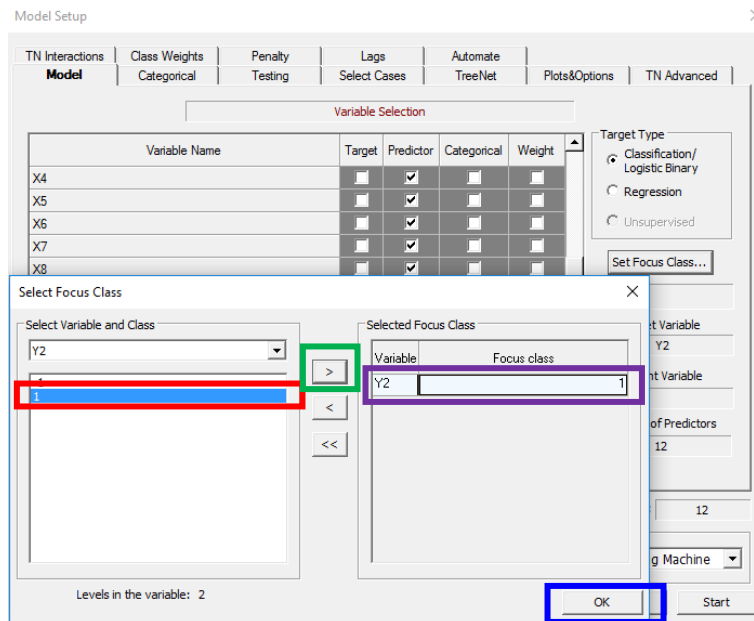
- ✓ Correctly declaring variables as categorical is less important for TreeNet than for CART or other modeling methods. TreeNet is capable of uncovering the true relationship between the target and a categorical predictor whether the declaration is made or not. However, some graphs will be much easier to read if the declarations are made. For high level categorical predictors, however, it may be advantageous to treat such predictors as numeric.
- ✓ All character variables (variables ending in \$) are automatically checked as categorical and cannot be unchecked.
- ✓ Using high-level categorical variables may significantly increase the running time and impose severe memory requirements. Make sure that a continuous variable has not been accidentally checked as categorical.
- ✓ Consider letting TreeNet treat a high-level categorical variable as a continuous predictor if it has more than about 30 levels. This is unlikely to hurt the model and may actually improve results quite a bit. This is possible only if the HLC is coded as numeric rather than text.

### Focus Class: Specifying the Event of Interest

The “event of interest” (called the “focus class” in SPM) is the event that you wish to see predicted probabilities for. For instance, if your target variable Y is coded as 0 or 1 with 0 representing “No Default” and 1 representing “Default” and you want the predicted value to be the probability of “Default” then set the focus class to 1.

- ✓ Setting the Focus Class does not change the essence of the modeling itself but it will affect the presentation of the graphs and various reports.

To setup the event of interest in SPM click the “Set Focus Class” button which will yield



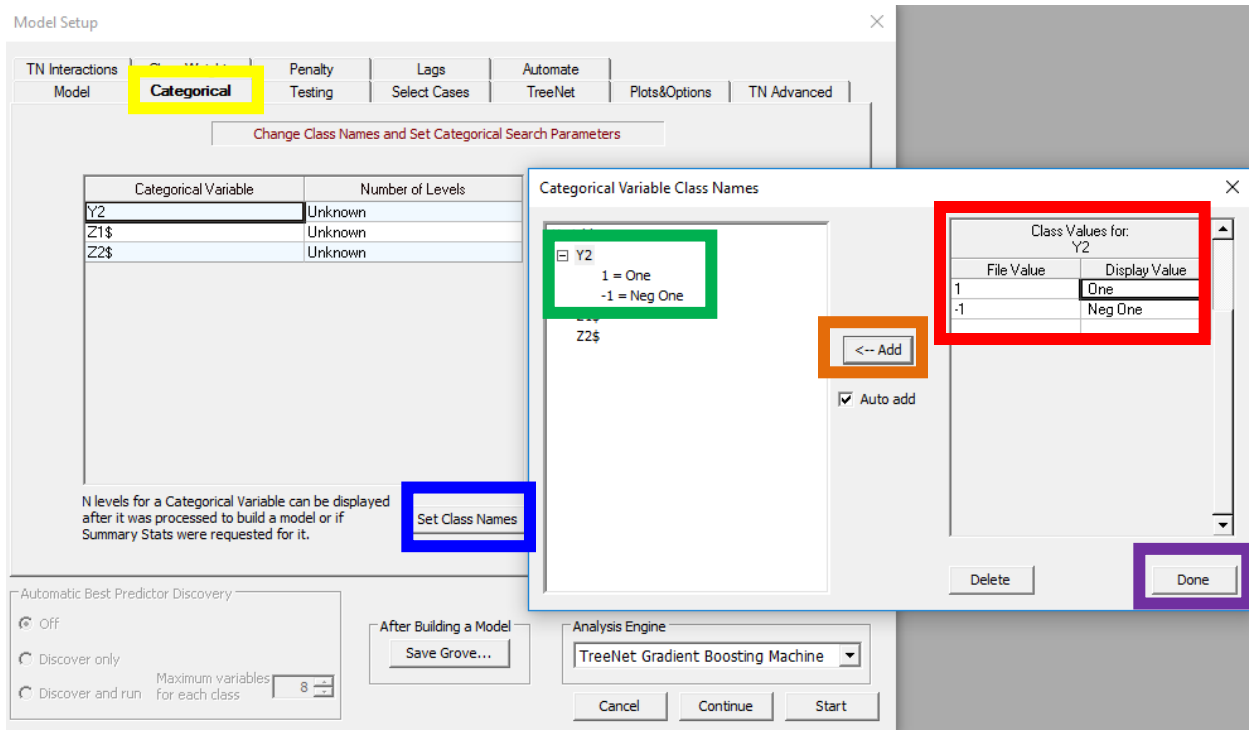
Here we are going to set the Focus Class to be 1:

1. Click on the **class of interest in the left pane**
2. Click the **right arrow > button**
  - a. You will see the **selected class appear in the right pane**
3. Click the **OK button**



## Categorical Tab

In the “Categorical” tab you will see the “Number of Levels” (**blue rectangle below**) for each categorical variable here if you compute summary statistics first, otherwise you will see “Unknown” as seen here (we did not compute summary statistics before opening the model setup).



## Set Class Names for the Target Variable

After you click the “Set Class Names” button, you will see the following menu that allows you to add labels for particular values of categorical variables:

1. **Class Values Menu**
    - a. “File Value” is the original value of variable in the dataset
    - b. “Display Value” is the label that you want to add to a specific value of the original value
  2. Click the **<-- Add button** to add the label for the variable Y2 and **you will see the labels added on the left.**
  3. Click the **Done button** when you are finished adding labels
- ✓ You will see the class labels in the output.
  - ✓ The choice of the Focus Class does not change the essence of the modeling itself, but it does affect the interpretation of the graphs and various reports.

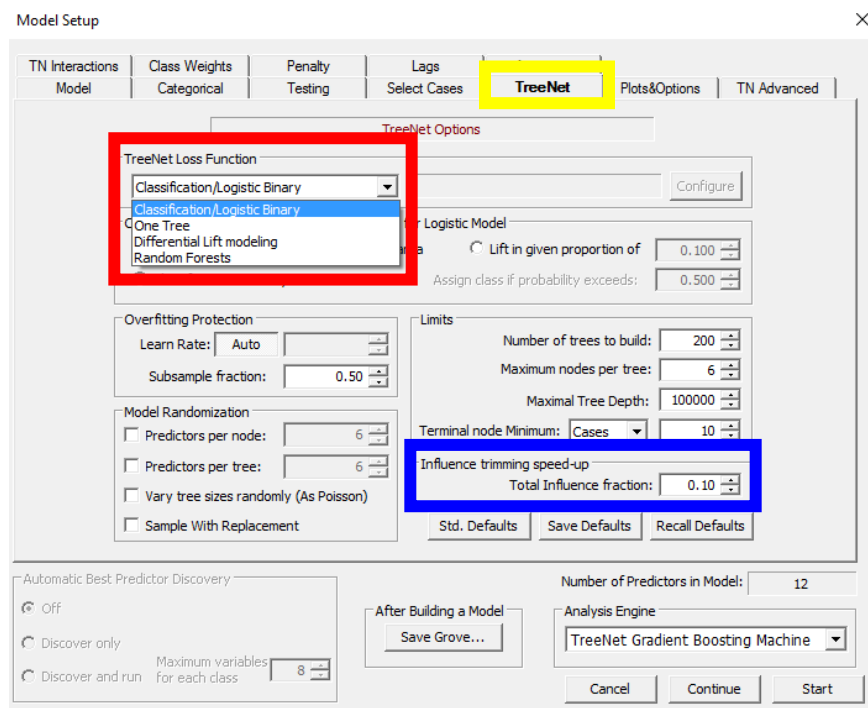
### TreeNet Parameters Specific to Classification

Some TreeNet options are specific to the choice of the Target Type in the Model Tab (i.e. the target type can be either “Classification/Logistic Binary” or “Regression”). The following parameters are specific to TreeNet in the classification setting and should be considered when building a TreeNet classification model (TreeNet parameters that are not included in this list affect both “Classification/Logistic Binary” and “Regression” models). Click each of the links below to learn more:

1. [Classification Loss Functions](#)
2. [Criterion Determining Number of Trees Optimal for Logistic Model](#)
3. [Total Influence Fraction](#)
4. [Subsample Separately by Target Class](#)
5. [Influence trimming Limited to Focus Class and/or Correctly Classified](#)
6. [Class Weights](#)
7. [TreeNet Calibration](#)

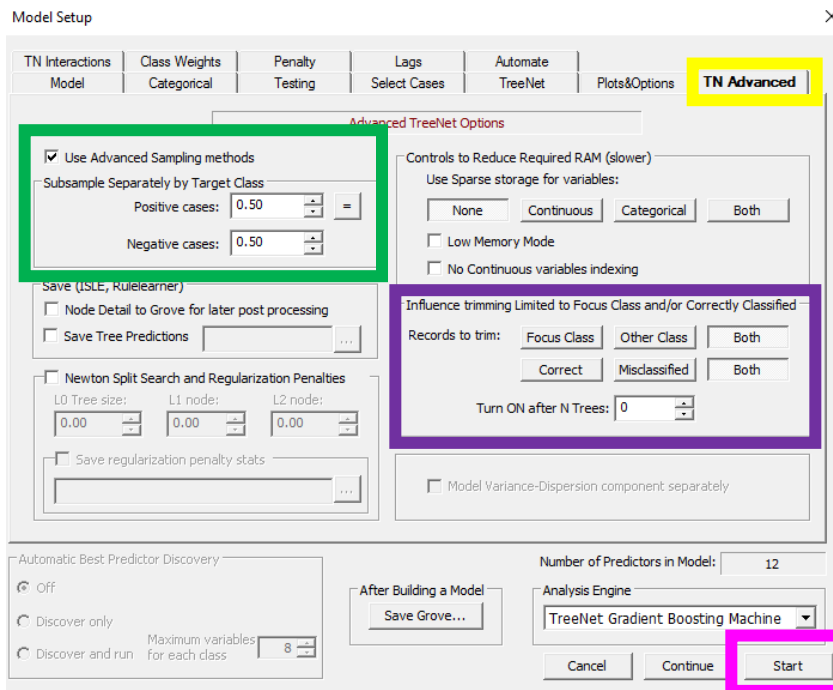
### Choosing the Classification Loss Function

Setting the Target Type to be “Classification” in the [Model Tab section](#) changes the types of loss functions **and are shown below**. For this example, we will set the loss function to “Classification/Logistic Binary” because our target variable is a binary variable (i.e. it has only two values: -1 and 1), and “Classification/Logistic Binary” handles both binary (two classes) and multinomial (more than two classes) classification problems. See the [Classification Loss Functions](#) section for more detail on the loss functions and see the [TreeNet Tab](#) section for more detail on the other settings in this tab.



We will leave the **total influence fraction at its default value of 0.10**.

## Advanced Options for Classification Models



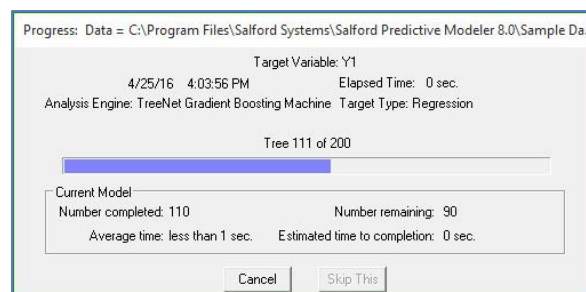
In the TN Advanced Tab you can

1. **Subsample Separately by Target Class.** This is helpful when one of the two classes is very rare. Here we will leave this setting at its default setting of 50% subsampling fractions for each class.
  - a. See the [Advanced Sampling Methods section](#) for more information.
2. **Customize the influence trimming options.** Here we will leave this setting at its default setting of trimming both the Focus Class and Other Class as well as both Correct and Misclassified records.
  - a. See the [Influence Trimming Limited to Focus Class and/or Correctly Classified section](#) for more information.

Click the **Start button** to run the model (you can click the **Start button** from any of the other tabs).

## Running TreeNet

After you click the **Start button** in the picture above, the following progress indicator appears while the model is being built. The progress indicator lets you know how much time the analysis should take and approximately how much time remains.



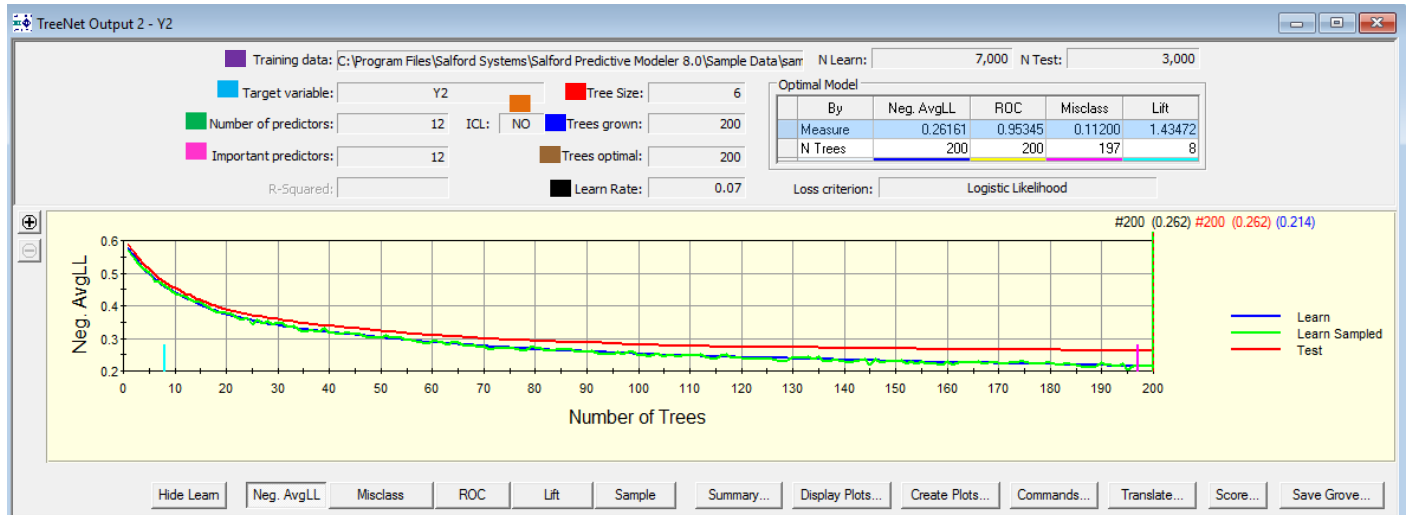
After the analysis is complete, text output appears in the **Classic Output** window, and a new window, the **TreeNet Output**, opens. See the [Classification/Binary Logistic: Interpreting Results](#) section to see an example of the TreeNet output and how to interpret it.

## Classification/Binary Logistic: Interpreting Results

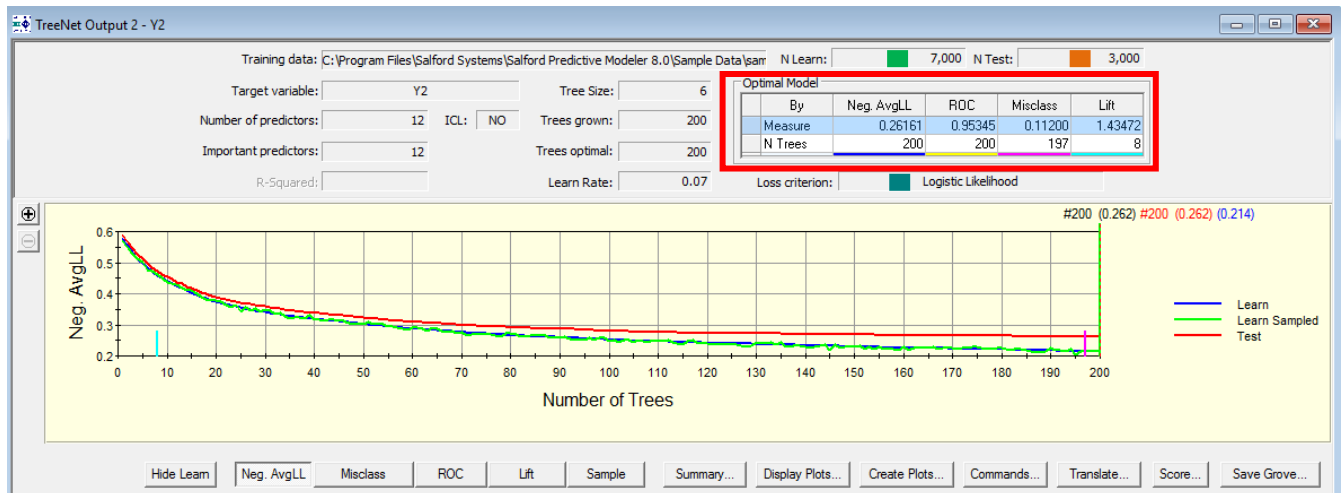
In this section and the subsections that follow we will explain the output of a TreeNet model.

### Initial Output I

When the model is complete, you will see the following:



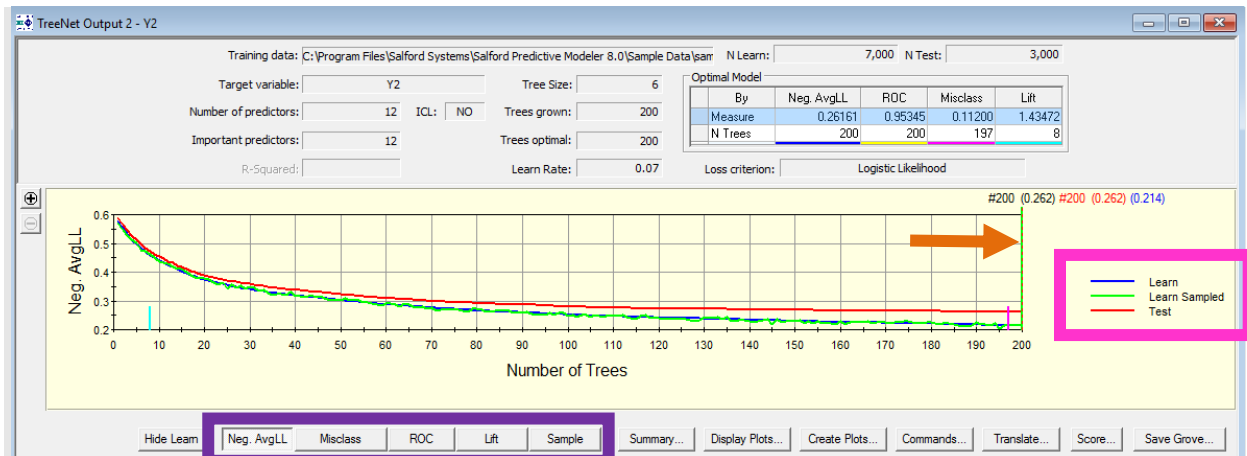
- **Training data:** file path of your dataset
- **Target variable:** name of your target variable
- **Number of predictors:** number of predictors available to be used in the model (i.e. the number of predictor boxes checked during the model setup)
- **Important predictors:** number of predictors actually used in the model
- **ICL:** “ICL” denotes the “Interaction Control Language”; See the [TN Interactions Tab](#) entry for more details on ICL. Here we can have two values: “YES” or “NO”
  - ✓ **NO** - means that the ICL was not used (i.e. all variables are allowed to interact)
  - ✓ **YES**- means that the ICL was used (i.e. some constraint was added)
- **TreeSize:** maximum number of terminal nodes specified in each tree during the model setup
- **Trees grown:** maximum settings for the numbers of trees specified during the model setup
- **Trees optimal:** optimal number of trees (Note: this can be different depending on the model criterion being used like Neg. AvgLL, ROC, Misclass, or Lift)
- **Learn Rate:** learning rate specified during the model setup



- **N Learn** –number of records in the learning data (the learning data is sometimes called the “training data”)
- **N Test** –number of records in the testing data (the testing data is sometimes called the “validation data”)
- **Optimal Model** – shows the optimal value for four model evaluation criteria along with the corresponding number of trees for each
  - ✓ **\*\*\*Note that the optimal number of trees can be different depending on the measure of used\*\*\***
  - ✓ **Neg. AvgLL** – Negative Average Log Likelihood (smaller values are preferred) on the test sample. The optimal Neg. AvgLL value is .26161 and occurs at tree 200. The Negative Average Log Likelihood is similar to ROC but emphasizes the probability interpretation of the model predictions. This is offered for technical users who may prefer this measure for theoretical reasons.
  - ✓ **ROC** – Area Under the ROC Curve for the test sample. The optimal ROC value is .93345 and occurs at tree 200. This is the most commonly-used model criterion in machine learning and is a measure of overall model performance tied closely to the ability of the model to correctly RANK records from most likely to least likely to be a “1” or a “0.”
  - ✓ **Misclass**– Misclassification Rate for the test sample. The optimal misclassification rate is .112 (i.e. 11.20%) and occurs at tree 197. The misclassification rate is based on a straightforward tally of how often the model tags a record correctly or incorrectly. If you intend to use the model to tag data as either “YES” or “NO” then this criterion will help you select the best model.
  - ✓ **Lift** – the optimal Lift for the test sample (top 10%) is 1.43472 and occurs at tree 8. Lift focuses on the ability of the model to concentrate the responders or the “1”s in the highest ranking records. This criterion is useful if the model is intended to be used exclusively to identify, say, the top 1%, 5% or 10% of a data set. In this case, we can safely ignore model performance in the lower percentiles.
- **Loss criterion**: the loss function specified during the model setup

- ✓ It is not possible to tell in advance how many trees will be optimal.
- ✓ The best number of trees can vary quite a bit depending on which performance criterion you select.
- ✓ Typically, the best model for classification accuracy has many fewer trees than the best model for best probability estimates (Ave LL) or overall performance (ROC). Best lift is also usually accomplished with relatively few trees.
- ✓ The best model is normally determined by performance on test data or via cross validation. When no independent testing is specified (exploratory runs), the optimal model is always set to the largest number of trees.
- ✓ It is possible to manually select a non-optimal model of any size for graphical display, obtaining summary reports, scoring, and translation.

## Error Curves



The curve in the picture above shows the Negative Average Log Likelihood (Neg. AvgLL) on the y-axis and the Number of Trees (i.e. the number of iterations) on the x-axis. Notice that the key on the right (**pink rectangle above**) shows the colors that correspond to one of the three curves:

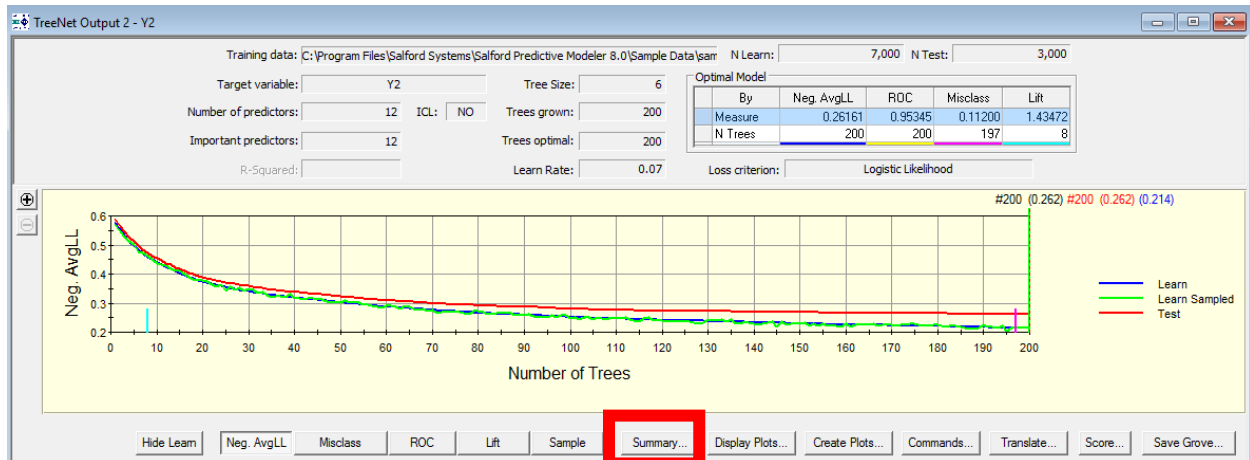
1. **Learn Error Curve** (the view of this curve is almost completely obstructed by the other two curves)
  - b. The learn error at iteration K is the error for the entire learning sample using the first K trees.
2. **Learn Sampled Error Curve**
  - c. Recall: we take a random subsample in each iteration and each iteration corresponds to fitting a CART tree to the generalized residuals. The learn sampled error at iteration K is the error for the subsample taken at the Kth iteration using the first K trees as the model.
3. **Test Error Curve**
  - a. The test error at tree K is the error for the entire test sample using the first K trees.

The above picture is for the Negative Average Log Likelihood (Neg. AvgLL) curve, but you can view any of the following curves simply by clicking the desired button along the bottom (**purple rectangle above**; notice how the Neg. AvgLL button is already selected)

- Negative Average Log Likelihood (Neg. AvgLL)
- Misclassification Rate (Misclass)
- Area Under the ROC Curve (ROC)
- Lift
- Sample

The exact numeric minimum of the **Negative Average Log Likelihood (Neg. AvgLL)** on the test data was attained at the 200-tree model highlighted by the green beam (**orange arrow above** points to the green beam at tree 200).

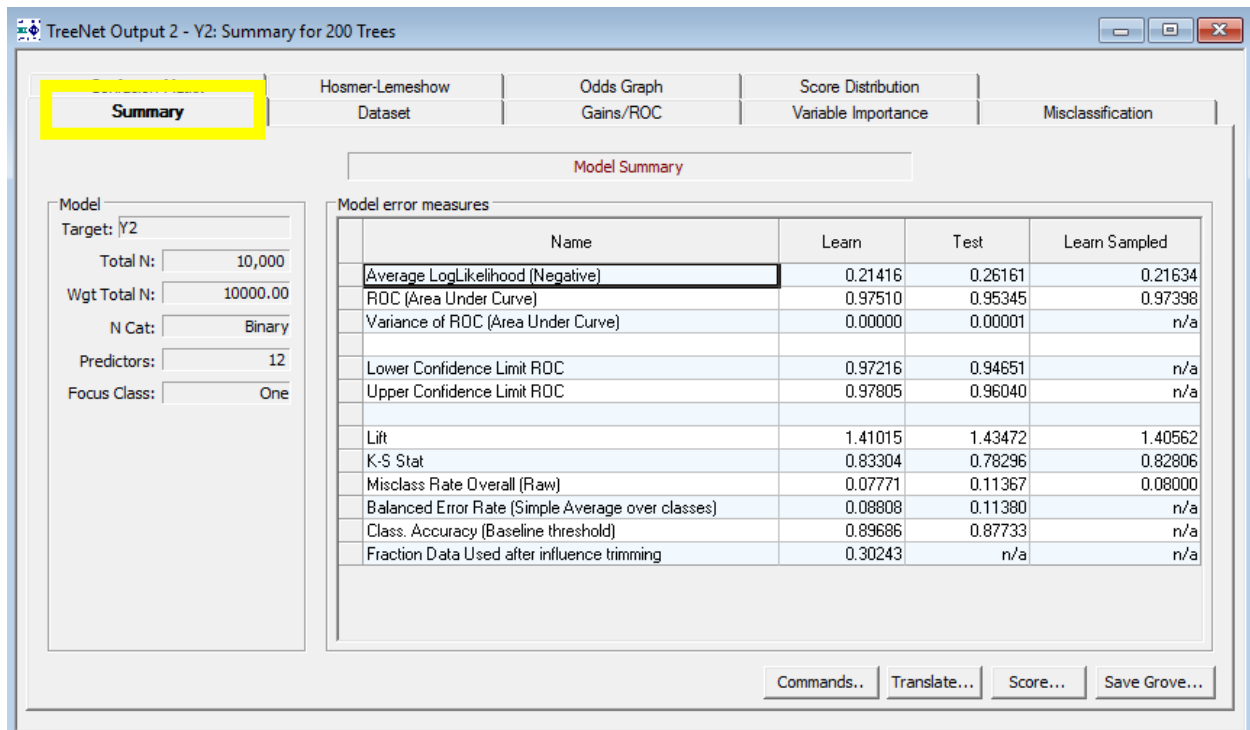
## Summary Button



Additional performance information about the TreeNet classification model can be accessed by clicking the **Summary... button**.

## Summary Tab

The Summary tab provides model error measures for the learn, test, and learn sampled data (see the [Error Curve](#) section of this manual for the definition of the learn sampled error).





## Dataset Tab

The Dataset tab provides information about the learn and test samples, record deletion, and the distribution of the target variable in the learn and test samples.

The screenshot shows the 'Dataset' tab in the TreeNet software. The 'Dataset' tab is highlighted in yellow. Below it, three tables are highlighted with colored boxes: 'Sample Partition' (red), 'Record Deletion' (blue), and 'Target Variable Statistics' (orange). The 'Outliers' section is empty.

Sample Partition	N	Pct
Learn	7,000	70.00%
Test	3,000	30.00%
Total	10,000	100.00%

Record Deletions	Target Missing	SELECT	BASIC	Missing Predictors
Learn	0	0	0	0
Test	0	0	0	0
Total	0	0	0	0

Class	Sample	N	Pct
Neg One	Learn	2,036	29.09%
	Test	909	30.30%
	Total	2,945	29.45%
One	Learn	4,964	70.91%
	Test	2,091	69.70%
	Total	7,055	70.55%

### Sampling Partition Information

Provides information about the learn and test data. The learn sample has 7,000 observations that comprise 70% of the total number of records whereas the test sample has 3,000 records that comprise 30% of the total number of records.

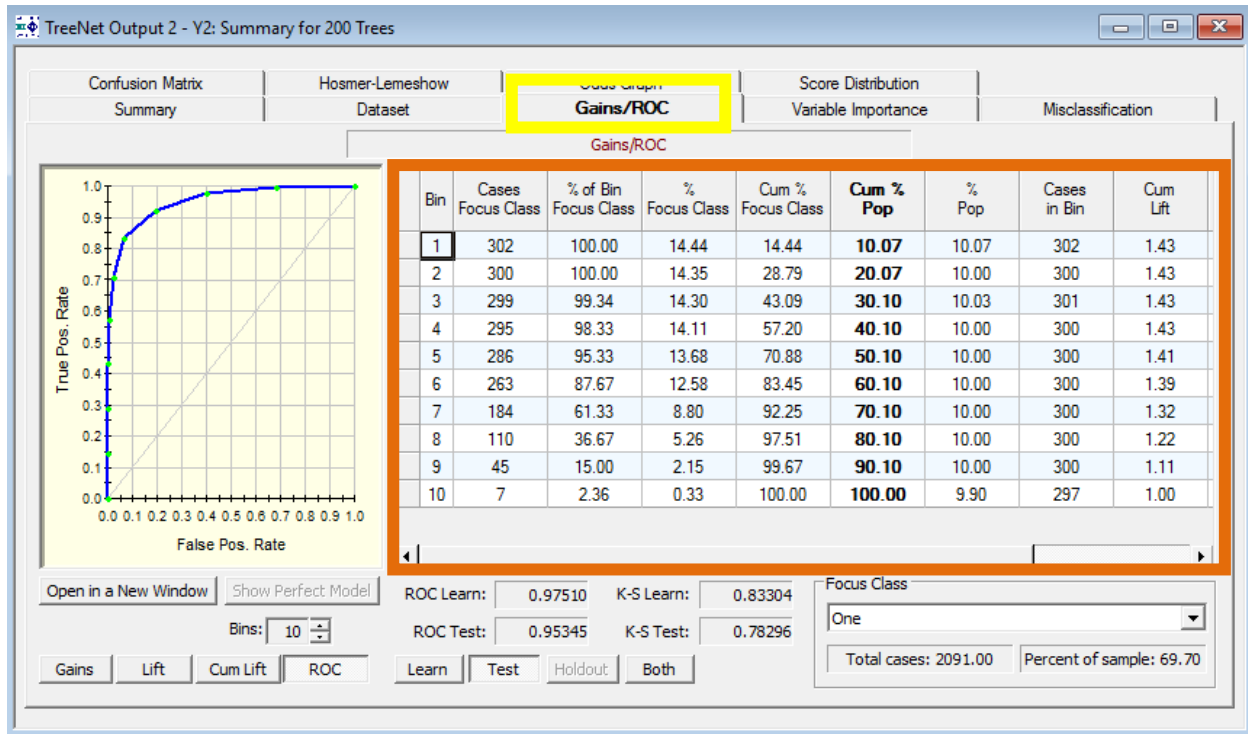
### Record Deletions

TreeNet models automatically handle missing values in the predictors variables, but when there are missing values in the target variable record deletion occurs for observations with missing target values (that is why we have a column called “Target Missing”).

### Target Variable Statistics

Provides summary statistics for the target variable in the learn sample and test sample for each class. Note that the class labels that were defined in the model setup, “Neg One” and “One,” are used instead of the actual values -1 and 1.

## Gains/ROC Tab



### Procedure for Generating the Gains Table

**TARGET** denotes the actual response (0 or 1, YES or NO, etc.) and it must be known for each case.

**RESPONSE** denotes the scores predicted by the given model.

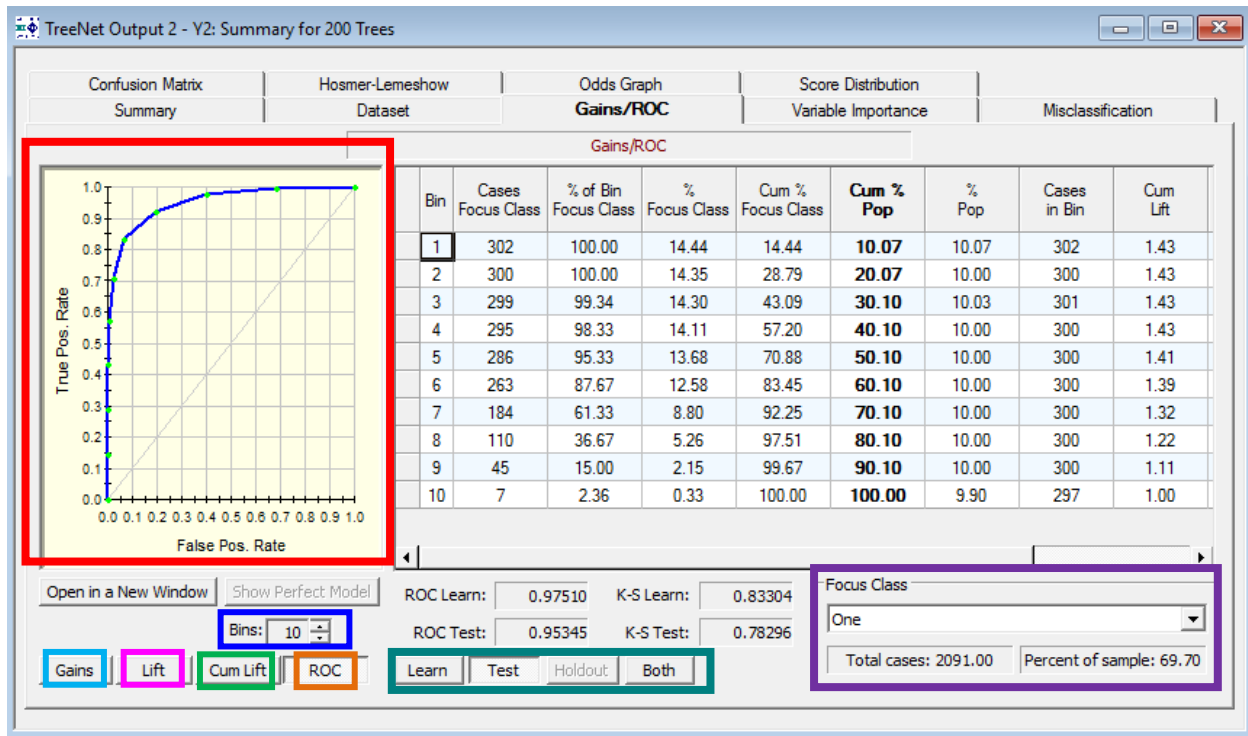
**K** denotes the predetermined number of bins.

1. Sort the data by descending **RESPONSE**.
2. Divide the data into **K** bins such that each bin has an approximately equal number of weighted cases.

The following table defines the columns of this report:

### Gains Tables

- **% of Bin Focus Class** – the percentage of 1s in the given bin.
- **% Focus Class** – the weighted number of class 1 cases in the bin divided by the total weighted count of class 1 cases.
- **Cum % Focus Class** – the simple cumulate of the **% Focus Class** column. This is also known as **sensitivity**, assuming that the classification threshold is set at the corresponding bin boundary.
- **Cases in Bin** – the weighted number of cases in the given bin.
- **% Pop** – the weighted number of cases in the given bin divided by the total weighted number of cases.
- **Cum % Pop** – the simple cumulate of the **% Pop** column.
- **Cum Lift** – **Cum % Focus Class** divided by **Cum % Pop**.
- **Lift Pop** – **% Focus Class** divided by **% Pop**.



The **graph on the left** displays:

When the **Gains button** is pressed – **Cum % Focus Class** versus **Cum % Pop**.

When the **Lift button** is pressed – **Lift Pop** versus **Cum % Pop**.

When the **Cum. Lift button** is pressed – **Cum Lift** versus **Cum % Pop**.

When the **ROC button** is pressed – **Cum % Focus Class** versus **(1 – specificity)** (The specificity is defined similarly to the sensitivity but with respect to the class 0.). This is commonly referred to as the **ROC curve**.

The **number of bins** can be changed using the **Bins: 10** control.

The number of bins can be set independently when the **Learn** or **Test buttons** are pressed. When the **Both button** is pressed, you will see both curves and the grid will show Test data. This allows extra flexibility in displaying the results over different data partitions.

The class in focus (i.e. the event class; see the [Focus Class: Specifying the Event of Interest](#) section for more information and how to set this up; can be selected using the **Focus Class selection box**.

- ✓ The ROC curve, while similar to the gains curve, has a number of theoretical advantages, especially when considering the model performance on the dominant class (in which case the gains curve may degenerate into the 45-degree line).

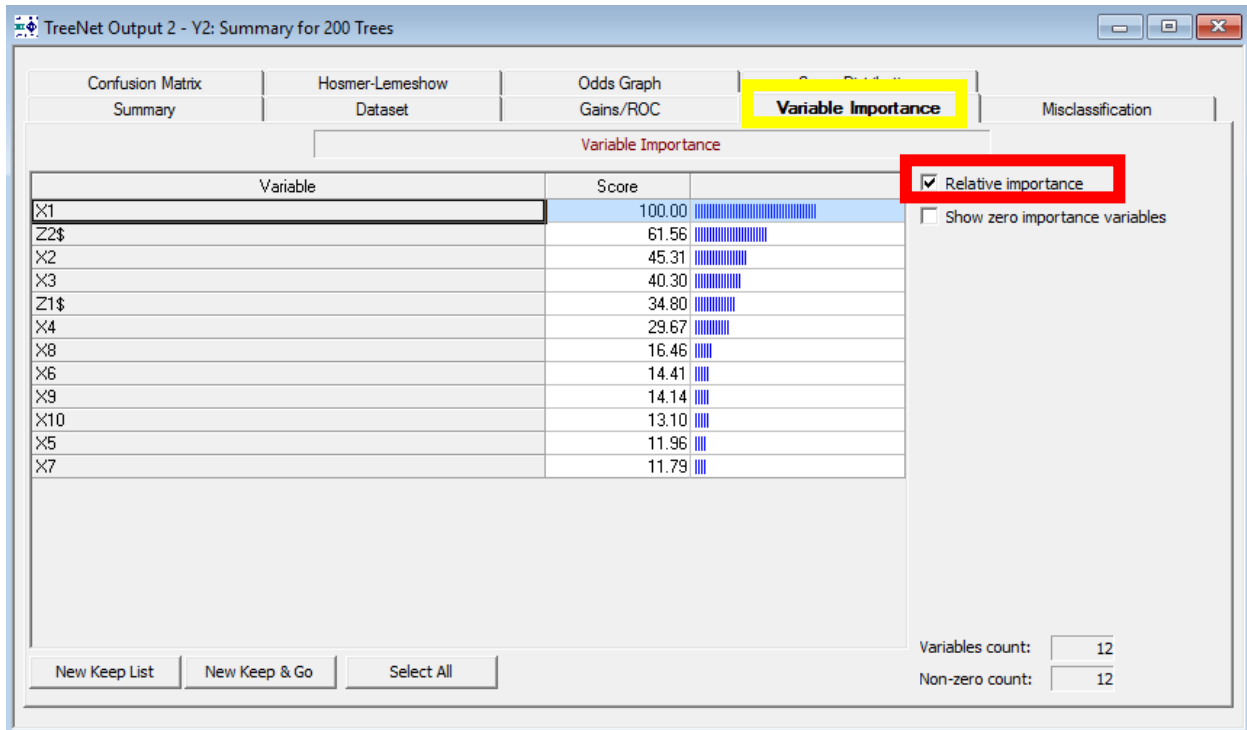
**Learn, Test, or Both buttons:** when the test data are available, both learn and test gains as well as ROC charts can be displayed using the Learn, Test, or Both buttons

## Interpreting Gains

You can use the following simple interpretation of gains. Assume that a scoring model is built to predict potential responders to a certain mail order. If every member of the population is given the offer, all responders would be targeted. Now assume that you want to send mail offers to only 10% of the population. When a model is not available and offers are mailed randomly, we can expect to collect only about 1/10<sup>th</sup> of the responders. In the presence of a model, instead of random mailing, one could send offers to the 10% of the population in the top bin (the highest scores). In this case the expected percent of covered responders would be **Lift Pop** times 1/10<sup>th</sup>. The interpretation is similar for the other bins and cumulative lift.

- ✓ This is a special case (0/1 response) of the regression gains.
- ✓ Gains/ROC based on the test data are usually more realistic than gains/ROC based on the training data.
- ✓ Test gains/ROC are not available when independent testing is disabled (Testing tab in the Model Setup).

## Variable Importance Tab



## Computation & Interpretation

Recall that in each iteration of the gradient boosting algorithm we fit a CART tree to generalized residuals. In CART, all possible split values are evaluated for every variable. The split appearing in the tree is one variable and split point combination with the largest split improvement value. Each time a variable is used in a TreeNet model, the split improvement is recorded.

1. **Raw Variable Importance:** computed as the cumulative sum of improvements of all splits associated with the given variable across all trees up to a specific model size. Note that smaller models typically involve fewer variables; thus, limiting the number of variables with non-zero importance scores, whereas larger models will often utilize a larger number of predictors.
2. **Relative Variable Importance:** (red rectangle above) rescales the variable importance values so that they are on a scale of 0 to 100 (the most important variable always gets a value of 100). All other variables are rescaled to reflect their importance relative to the most important variable.

**Interpretation:** For example, in the above picture, the variable X1 is the most important variable and Z1\$ has a relative importance of 34.80, so X1 is about twice as important as Z1\$. Similarly, we can say that the variable X1 is over 9 times as important as the variable X9 ( $10.98 \times 9 = 99$ , which is close to X1's variable importance value of 100)

**Relative Importance Rescaling:** divide each variable's raw importance score by the largest raw importance score and multiply by 100.

✓ This conclusion is further supported by the analysis of contribution plots described in a later section.

## New Keep List

The variable importance lists are interactive: you can highlight a subset of variables and then quickly rebuild a model using only the highlighted variables.

Variable	Score
X1	100.00
Z2\$	61.56
X2	45.31
X3	40.30
Z1\$	34.80
X4	29.67
X8	16.46
X6	14.41
X9	14.14
X10	13.10
X5	11.96
X7	11.79

### New Keep List

A KEEP list is a list of variables that you want to select as predictors in the model. KEEP lists are an SPM command line feature and are useful when you have a large number of variables. Clicking [New Keep List](#) above will open the SPM Notepad with KEEP list commands.

```

REM Selected 6 variables from: "TreeNet Output 2 - Y2: Summary for 200 Trees"

MODEL Y2
KEEP X1, Z2$, X2, X3, Z1$, X4

```

Notice how the variables in the list correspond to the variables that are highlighted. Press **CTRL W** to submit the commands. After the commands are submitted, open the model setup again and notice how only the variables you **entered in the KEEP list are now selected as predictors**.

Model Setup

TN Interactions | Class Weights | Penalty | Lags | Automate | Plots&Options | TN Advanced  
**Model** | Categorical | Testing | Select Cases | TreeNet

Variable Selection

Variable Name	Target	Predictor	Categorical	Weight
W	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Z1\$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Z2\$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
X1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X4	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X5	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X6	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Sort: File Order

Filter:  All/Selected  Character  Numeric

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

Number of Predictors in Model: 6

Analysis Engine: TreeNet Gradient Boosting Machine

Buttons: Save Grove..., Cancel, Continue, Start

## New Keep & Go

TreeNet Output 2 - Y2: Summary for 200 Trees

Confusion Matrix | Hosmer-Lemeshow | Odds Graph | Score Distribution | Misclassification  
 Summary | Dataset | Gains/ROC | **Variable Importance**

Variable Importance

Variable	Score
X1	100.00
Z2\$	61.56
X2	45.31
X3	40.30
Z1\$	34.80
X4	29.67
X8	16.46
X6	14.41
X9	14.14
X10	13.10
X5	11.96
X7	11.79

Buttons: New Keep List, **New Keep & Go**, Select All

Variables count: 12  
 Non-zero count: 12

## New Keep & Go

First, highlight the variables in the variable importance list as desired by clicking and dragging. Clicking **"New Keep & Go"** to immediately build a TreeNet model using only the highlighted variables.

## Select All

Variable	Score
X1	100.00
Z2\$	61.56
X2	45.31
X3	40.30
Z1\$	34.80
X4	29.67
X8	16.46
X6	14.41
X9	14.14
X10	13.10
X5	11.96
X7	11.79

Variables count: 12  
Non-zero count: 12

### Select All

Click **Select All** to highlight all the variables in the variable importance list (**red rectangle above**). The Select All feature is useful if you wish to quickly build a new TreeNet model using **New Keep & Go**, but you have a large number of variables. To accomplish this:

1. Uncheck the “**Show zero importance variables**” checkbox.
2. Click the “**Select All**” button.
3. Click the “**New Keep & Go**” button.

Click “New Keep & Go” to immediately build a TreeNet model that uses only the highlighted variables (in this case only the variables with a positive variable importance score).

### Variables count and Non-zero count

This provides quick summary information.

**Variables count:** number of variables selected as predictors

**Non-zero count:** number of variables with a variable importance score greater than zero

- ✓ Although the best model is the one normally recommended for further study there is nothing preventing you from looking at other models. In fact, you can access results for any model in the TreeNet sequence of models, which means that you can review results for models with 1, 2, 3, 4, etc. trees.
- ✓ To economize on resources, we display the overall performance for every size of model beginning with the one-tree model, but we only display detailed summary reports and graphs for select models. You can control how many models are accompanied with details.



## Misclassification Tab

The Misclassification tab provides misclassification statistics for both the learn and test sample.

Learn Sample					Test Sample				
Class	N Cases	N Mis-Classed	Pct. Error	Cost	Class	N Cases	N Mis-Classed	Pct. Error	Cost
Neg One	2,036	114	5.60%	0.05599	Neg One	909	83	9.13%	0.09131
One	4,964	600	12.09%	0.12087	One	2,091	285	13.63%	0.13630

Threshold: 0.700 Balance Baseline Show Table...

### Learn Sample

**Class**- target classes; \*\*\*Note: “Neg One” and “One” are labels that we added to the target variable values -1 and 1, respectively (see the [Set Class Names for the Target Variable](#) section)

**N Cases**- number of cases corresponding to the specific target class in the learn sample. The number of cases for the “Neg One” class is 2,036 and the number of cases for the “One” class is 4,964.

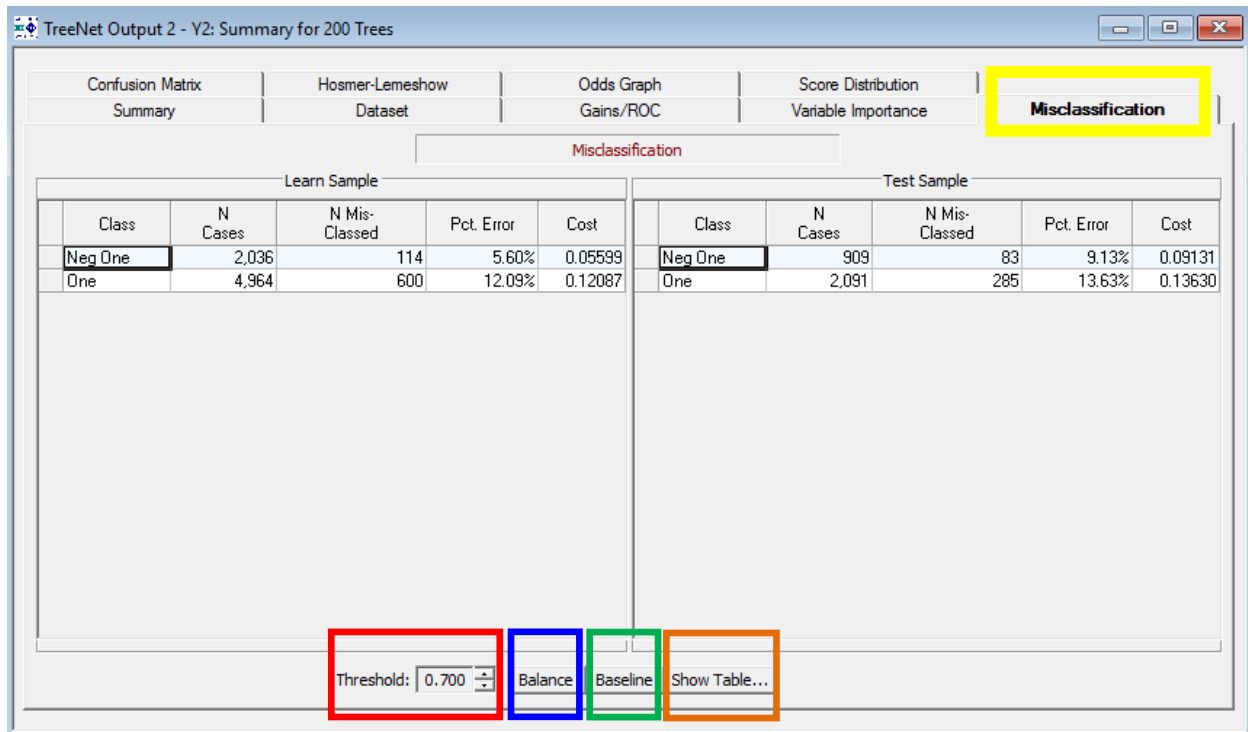
**N Mis-Classed**- number of misclassified cases corresponding to the specific target class in the learn sample. The number of misclassified cases for the “Neg One” class in the learn sample is 114 and the number of misclassified cases for the “One” class in the learn sample is 600.

**Pct. Error**- percent of misclassified cases corresponding to the specific target class in the learn sample. The percent of misclassified cases for the “Neg One” class in the learn sample is 5.60% and the percent of misclassified cases for the “One” class in the learn sample is 12.09%.

$$\text{Pct. Error} = \frac{\text{N MisClassed}}{\text{N Cases}}$$

**Cost**- cost associated with the specific target class in the learn sample (you can specify costs via the command line). The (learn sample) cost of misclassifying the 114 “Neg One” class is .05599 and the (test sample) cost of misclassifying the 600 “One” cases is 0.12087.

**Test Sample**- the interpretation is the same, but has different numbers and the phrase “test sample” instead of “learn sample”



Learn Sample					Test Sample				
Class	N Cases	N Mis-Classed	Pct. Error	Cost	Class	N Cases	N Mis-Classed	Pct. Error	Cost
Neg One	2,036	114	5.60%	0.05599	Neg One	909	83	9.13%	0.09131
One	4,964	600	12.09%	0.12087	One	2,091	285	13.63%	0.13630

Threshold: 0.700 Balance Baseline Show Table...

**Threshold Control:** The threshold is the point at which a case is classified as the focus class (i.e. the focus class is the event of interest). In this example the focus class is the “One” class and a threshold of .70 means that a case is only classified as a “One” if its predicted probability is greater than or equal to .70. The default threshold value is the percent of records in the test data that correspond to the focus class (i.e. the probability of the focus class in the test sample).

✓ For binary target models, class assignments are based on a specific threshold, which can be changed using the Threshold control at the bottom. A higher threshold will decrease sensitivity and increase specificity. The **Base line button** sets the threshold to percent of target class in the sample. The **Balance button** tries to find a threshold that minimizes difference in **Percent Correct** between classes. The **Show Table... button** opens a new window with a table summarizing misclassification stats for each available threshold.

**Balance button:** automatically adjusts the threshold so that the difference in the misclassification rates is minimized (i.e. the misclassification rates for the two class levels will be more similar).

**Base line button:** sets the threshold back to its default value which is the percent of records in the test data that correspond to the focus class (i.e. the probability of the focus class in the test sample).

**Show Table... button:** shows a table in a new window with detailed misclassification statistics for each available threshold.

## Show Table... Button Example

Click the **Show Table...** button in the picture above and you will see the following table:

TreeNet Output 2 - Y2: Summary for 200 Trees: Threshold table

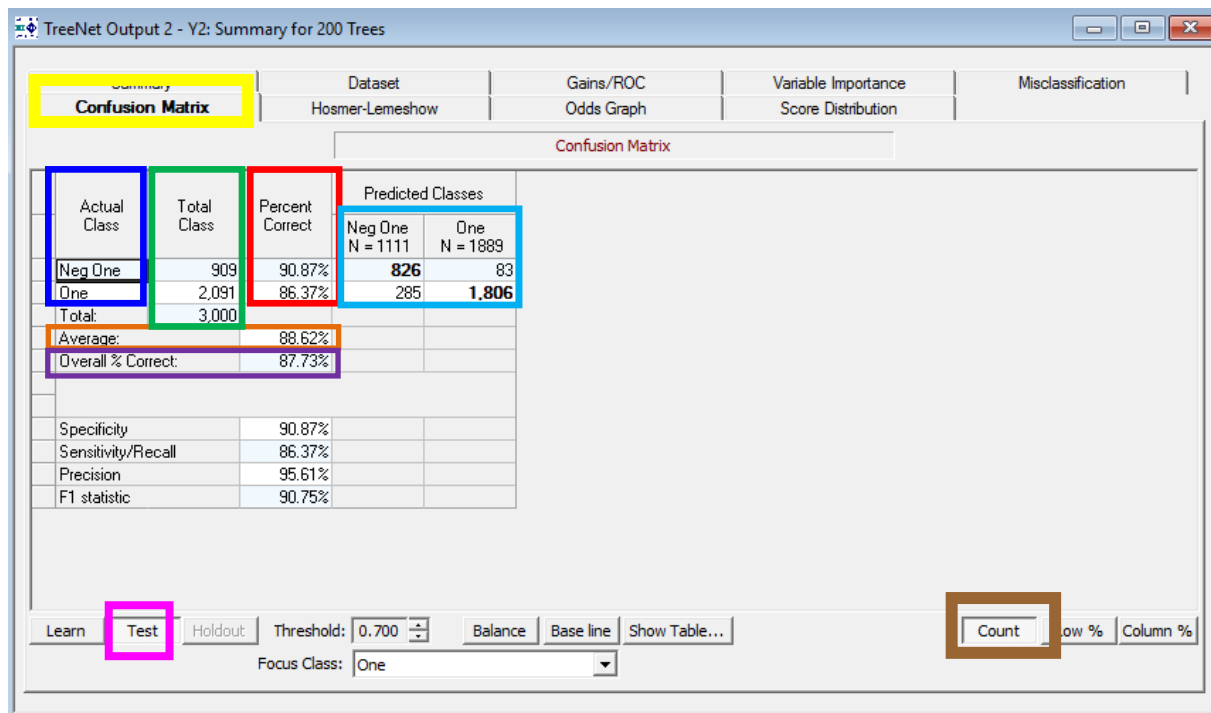
Threshold	Class Neg One					Class One					% Missclass
	N Correct	N Misclass	% Missclass	Sensitivity	Specificity	N Correct	N Misclass	% Missclass	Sensitivity	Specificity	
0.00000	0	909	100.00000	0.00000	1.00000	2,091	0	0.00000	1.00000	0.00000	30.30000
0.01000	0	909	100.00000	0.00000	1.00000	2,091	0	0.00000	1.00000	0.00000	30.30000
0.02000	2	907	99.77998	0.00220	1.00000	2,091	0	0.00000	1.00000	0.00220	30.23333
0.03000	15	894	98.34983	0.01650	1.00000	2,091	0	0.00000	1.00000	0.01650	29.80000
0.04000	50	859	94.49945	0.05501	0.99952	2,090	1	0.04782	0.99952	0.05501	28.66667
0.05000	91	818	89.98900	0.10011	0.99952	2,090	1	0.04782	0.99952	0.10011	27.30000
0.06000	122	787	86.57866	0.13421	0.99904	2,089	2	0.09565	0.99904	0.13421	26.30000
0.07000	153	756	83.16832	0.16832	0.99857	2,088	3	0.14347	0.99857	0.16832	25.30000
0.08000	177	733	80.59895	0.19130	0.99809	2,088	3	0.14347	0.99857	0.19130	24.50000
0.09000	208	701	77.11771	0.22882	0.99809	2,087	4	0.19130	0.99809	0.22882	23.50000
0.10000	257	672	73.32733	0.28073	0.99761	2,086	5	0.23912	0.99761	0.28073	22.30000
0.11000	258	651	71.61716	0.28383	0.99761	2,086	5	0.23912	0.99761	0.28383	21.86667
0.12000	302	607	66.77668	0.33223	0.99522	2,081	10	0.47824	0.99522	0.33223	20.56667
0.13000	320	589	64.79648	0.35204	0.99283	2,076	15	0.71736	0.99283	0.35204	20.13333
0.14000	345	564	62.04620	0.37954	0.99283	2,076	15	0.71736	0.99283	0.37954	19.30000
0.15000	367	542	59.62596	0.40374	0.99187	2,074	17	0.81301	0.99187	0.40374	18.63333
0.16000	381	528	58.08581	0.41914	0.99139	2,073	18	0.86083	0.99139	0.41914	18.20000
0.17000	394	515	56.65567	0.43344	0.98996	2,070	21	1.00430	0.98996	0.43344	17.86667
0.18000	425	484	53.24532	0.46755	0.98900	2,068	23	1.09995	0.98900	0.46755	16.90000
0.19000	440	469	51.59516	0.48405	0.98661	2,063	28	1.33907	0.98661	0.48405	16.56667

Learn Test Holdout

**Interpretation:** If the threshold is set to .09, then class “Neg One” will have 208 correctly classified cases and 701 misclassified which corresponds to a class “Neg One” misclassification rate of 77.11771%, a sensitivity of .22882, and a specificity of .99809. For class “One,” the threshold of .09 leads to 2,087 correctly classified cases, 4 misclassified cases which corresponds to a class “One” misclassification rate of .19130%, a sensitivity of .99809, and a specificity of .22882. The overall misclassification rate is 23.5%.

## Confusion Matrix Tab

The confusion matrix is a standard summary for classifiers of all kinds and is used to assess statistical models such as logistic regression as well as more exotic data mining models. We call it the Confusion Matrix (in previous versions of SPM it is called the “Prediction Success” table) following Nobel Prize-winning economist Daniel McFadden’s 1979 paper on the subject. Confusion matrices based on the learn sample are usually too optimistic. You should always use confusion matrices based on the test sample (or on cross validation, when a separate test sample is not available) as fair estimates of a Classification Model performance.



The **Actual Class column** lists actual levels of the Target Variable: “Neg One” and “One”. \*\*\*Note: instead of the actual target values -1 and 1 we see the labels: “Neg One” and “One,” because we specified these labels earlier (see the [Set Class Names for the Target Variable section](#))

**Total Class column** shows total number of observations for each class in the **test sample**. In the example above there are 909 “Neg One” class cases and 2,091 “One” class cases for a total of 3,000 cases.

**Percent Correct column** shows percent of the class observations that were classified correctly in the **test sample**. In the picture above, 90.87% of the “Neg One” cases were classified correctly, 86.37% of the “One” cases were classified correctly.

**Average:** the average of the percent correct values based on the **test sample**. In the example above the average of 90.87% and 86.37% is  $\frac{90.87\% + 86.37\%}{2} = 88.62\%$ .

**Overall % Correct:** overall percent correct for both classes for the **test sample**. In the example above this is can be calculated from the **confusion matrix count values** (notice that the **count button is selected in the lower right-hand corner**) and multiplying by 100 to obtain a percentage value:

$$\frac{\text{Number of Correctly Classified Cases}}{\text{Total Number of Cases}} = \frac{826 + 1806}{826 + 1806 + 83 + 285} * 100 = 87.73333333\%$$

The screenshot shows the 'Confusion Matrix' window in Salford Predictive Modeler. The window title is 'TreeNet Output 2 - Y2: Summary for 200 Trees'. The 'Confusion Matrix' tab is active. The window is divided into several sections:

- Summary:** A table showing overall performance metrics.
- Dataset:** Hosmer-Lemeshow
- Gains/ROC:** Odds Graph
- Variable Importance:** Score Distribution
- Misclassification:** (Empty)

The main area displays the 'Confusion Matrix' table:

Actual Class	Total Class	Percent Correct	Predicted Classes	
			Neg One N = 1111	One N = 1889
Neg One	909	90.87%	826	83
One	2,091	86.37%	285	1,806
Total:	3,000			
Average:		88.62%		
Overall % Correct:		87.73%		
Specificity		90.87%		
Sensitivity/Recall		86.37%		
Precision		95.61%		
F1 statistic		90.75%		

At the bottom of the window, there are control buttons and a dropdown menu:

- Learn, Test, Holdout:** Buttons to select the data source for the confusion matrix.
- Threshold:** A numeric input field set to 0.700.
- Balance, Base line, Show Table...:** Buttons for threshold selection and table display.
- Count, Row %, Column %:** Buttons to toggle the display of raw counts or percentages.
- Focus Class:** A dropdown menu currently set to 'One'.

**Learn, Test, and Holdout buttons-** view the confusion matrix for the learn, test, or holdout samples by selecting the desired button

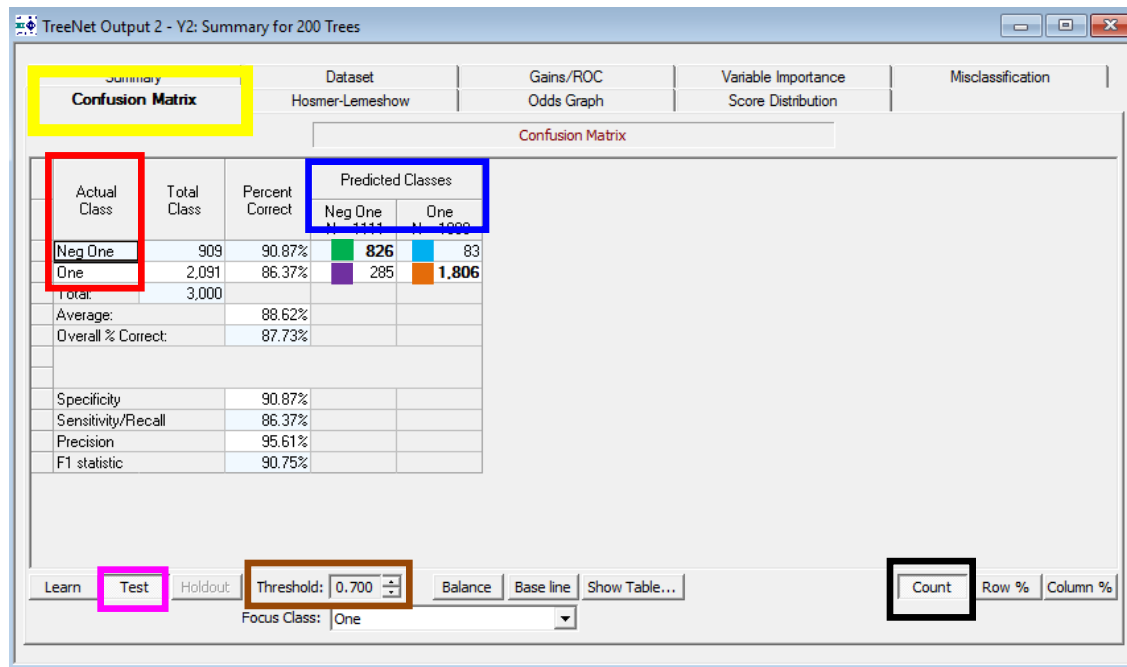
**Threshold:** The threshold is the point at which a case is classified as the focus class (i.e. the focus class is the event of interest). In this example the focus class is the “One” class and a threshold of .70 means that a case is only classified as a “One” if its predicted probability is greater than or equal to .70. The default threshold value is the percent of records in the test data that correspond to the focus class (i.e. the probability of the focus class in the test sample).

✓ For binary target models, class assignments are based on a specific threshold, which can be changed using the Threshold control at the bottom. A higher threshold will decrease sensitivity and increase specificity.

**Balance, Base line, and Show Table... buttons:** The **Base line** button will set the threshold to the percent of target class in the sample. The **Balance** button tries to find a threshold that minimizes difference in Percent Correct between classes. The **Show Table... button** opens a new window with a table summarizing misclassification stats for each available threshold (see the [Show Table... Button Example section](#) for an example)

**Count, Row %, and Column % buttons:** The Count button will show the raw count values in the confusion matrix and is the default setting, clicking the Row% button allows you to see a percentage breakdown of the actual class in terms of the predictions (i.e. 91% of the class 1 cases were classified as class1 and 9% of the class 1 cases were classified as class 2; also, the Row% button shows the rounded true negative, false negative, false positive, and true positive percentages), the Column% button shows a percentage breakdown of the predictions class in terms of the actual class (i.e. 96% of the class 1 predictions had an actual value of class1 and 4% of the class 1 predictions had an actual value of class1). See the following three interpretation sections for more detail: [Interpreting the Confusion Matrix: Count](#), [Interpreting the Confusion Matrix: Row %](#), and [Interpreting the Confusion Matrix: Column %](#).

## Interpretation: Count



In the picture above we have the **Predicted Class values** (“Neg One” and “One”) along the top and the **Actual Class values** (“Neg One” and “One”) along the bottom. How many cases were predicted as “Neg One” when the cases have an actual class value of “One”? To find this value, place your right index finger on the class value “Neg One” in the blue rectangle above, and place your left index finger on the class value “One” in the red rectangle above. Now bring your two fingers together: they **intersect at the value 285**. This means that we incorrectly classified 285 cases as “Neg One” (this type of mistake is commonly called a “false negative”).

Using this strategy, we can interpret the above confusion matrix for the **test sample** data

- **826 cases** were **predicted to be class “Neg One”** when the cases were **actually class “Neg One”** (i.e. 826 cases were correctly classified as “Neg One”)
  - These 826 cases are sometimes referred to as “true negatives”
- **285 cases** were **predicted to be class “Neg One”** when the cases were **actually class “One”** (i.e. 285 cases were incorrectly classified as “Neg One”)
  - These 285 cases are sometimes referred to as “false negatives”
- **83 cases** were **predicted to be class “One”** when the cases were **actually class “Neg One”** (i.e. 83 cases were incorrectly classified as “One”)
  - These 83 cases are sometimes referred to as “false positives”
- **1,806 cases** were **predicted to be class “One”** when the cases were **actually class “One”** (i.e. 1,806 cases were correctly classified as “One”)
  - These 1,806 cases are sometimes referred to as “true positives”

The **threshold value is 0.70**. This means that a case is predicted to be the focus class (i.e. the predicted class on the right which in this case is class “One”) if the predicted probability for the focus class is greater than or equal to 0.70. You can adjust the threshold and observe the impact on the confusion matrix.

## Interpretation: Row %

Actual Class	Total Class	Percent Correct	Predicted Classes	
			Neg One N = 1111	One N = 1889
Neg One	909	90.87%	91	9
One	2,091	86.37%	14	86
Total:	3,000			
Average:		88.62%		
Overall % Correct:		87.73%		
Specificity		90.87%		
Sensitivity/Recall		86.37%		
Precision		95.61%		
F1 statistic		90.75%		

Clicking the **Row %** button shows a percentage breakdown of the actual class in terms of the predictions and the resulting values in the confusion matrix above. Note that interpretation below depends on the value of the threshold which is described in the [Interpreting the Confusion Matrix: Count](#) section. Also, the values used in the calculations below are count values and can also be found in this section.

**91%** of the **test sample** cases that are **actually class “Neg One”** were **predicted to be “Neg One”** (i.e. this is the **true negative percentage = specificity** rounded to the nearest percent)

$$\frac{826}{826 + 83} * 100 = 90.869\% \approx 91\%$$

**9%** of the **test sample** cases that are **actually class “Neg One”** were **predicted to be “One”** (i.e. this is the **false positive percentage = 1-specificity** rounded to the nearest percent)

$$\frac{83}{826 + 83} * 100 = 9.13\% \approx 9\%$$

**86%** of the **test sample** cases that are **actually class “One”** were **predicted to be “One”** (i.e. this is the **true positive percentage = sensitivity** rounded to the nearest percent)

$$\frac{1806}{285 + 1806} * 100 = 86.37\% \approx 86\%$$

**14%** of the **test sample** cases that are **actually class “One”** were **predicted to be “Neg One”** (i.e. this is the **false negative percentage = 1-sensitivity** rounded to the nearest percent)

$$\frac{285}{285 + 1806} * 100 = 13.63\% \approx 14\%$$

## Interpretation: Column %

Actual Class	Total Class	Percent Correct	Predicted Classes	
			Neg One	One
Neg One	909	90.87%	74	4
One	2,091	86.37%	26	96
Total:	3,000			
Average:		88.62%		
Overall % Correct:		87.73%		
Specificity		90.87%		
Sensitivity/Recall		86.37%		
Precision		95.61%		
F1 statistic		90.75%		

Click the **Column % button** to see a percentage breakdown of the predicted class in terms of the actual class values and the resulting values in the confusion matrix above. Note that interpretation below depends on the value of the threshold which is described in the [Interpreting the Confusion Matrix: Count](#) section. Also, the values used in the calculations below are count values and can also be found in this section.

**74%** of the **test sample** cases that are **predicted to be “Neg One”** are **actually class “Neg One”**

$$\frac{826}{826 + 285} * 100 = 74.347\% \approx 74\%$$

**26%** of the **test sample** cases that are **predicted to be “Neg One”** are **actually class “One”**

$$\frac{285}{826 + 285} * 100 = 25.65\% \approx 26\%$$

**4%** of the **test sample** cases that were **predicted to be “One”** are **actually class “Neg One”**

$$\frac{83}{83 + 1806} * 100 = 4.393\% \approx 4\%$$

**96%** of the **test sample** cases that were **predicted to be “One”** are **actually class “One”**

$$\frac{1806}{83 + 1806} * 100 = 95.60\% \approx 96\%$$



## Interpretation: Specificity, Sensitivity/Recall, Precision, and F1 statistic

The screenshot shows the 'Confusion Matrix' tab in the TreeNet software. The main table displays the following data:

Actual Class	Total Class	Percent Correct	Predicted Classes	
			Neg One N = 1111	One N = 1889
Neg One	909	90.87%	826	83
One	2,091	86.37%	285	1,806
Total:	3,000			
Average:		88.62%		
Overall % Correct:		87.73%		

Below the main table, a summary table provides the following metrics:

Specificity	90.87%
Sensitivity/Recall	86.37%
Precision	95.61%
F1 statistic	90.75%

The 'Focus Class' dropdown menu is set to 'One'.

For binary targets we provide **values for Specificity, Sensitivity/Recall, Precision, and F1 statistic**. These values are shown for the Focus class **specified in the Focus Class combo box** and the threshold value specified (see the next page for a description of the threshold).

### Sensitivity/Recall

Probability for the Focus class cases to be classified by the model as Focus class ones. Computed as number of correctly classified Focus class cases divided by total number of Focus class cases in the dataset. This measure is also known as Recall.

### Specificity

Probability for the non-Focus class cases to be classified as non-Focus class ones. Computed as number of correctly classified non-Focus class cases divided by total number of non-Focus class cases in the dataset.

### Precision

Computed as number of correctly classified Focus class cases divided by total number of cases classified as Focus class.

### F1 statistic

$2 \times \text{Precision} \times \text{Recall} / (\text{Precision} + \text{Recall})$

### Hosmer-Lemeshow tab

- ✓ This is a binary classification-only feature.

This tab shows the results of Hosmer-Lemeshow goodness of fit test for binary classification models. Responses from model are binned into ten deciles and several statistics are computed for each bin.

For a given bin  $i$  Chi squared component is computed by the following formula.

$$X_i^2 = \frac{(ResponseObserved_i - ResponseExpected_i)^2}{ResponseExpected_i \cdot (1 - \frac{ResponseExpected_i}{n_i})}$$

Under the table we show the Total Chi-Square value (sum of Chi-Sq statistics per each bin) and **P-value of Hosmer-Lemeshow statistics** (HL Stat. P-value) under the grid. The null hypothesis of the Hosmer-Lemeshow Test is that the model fits the data well whereas the alternative hypothesis is that the model does not fit the data well.

		Decile	1	2	3	4	5	6	7	8	9	10
Response	Observed		0	1	1	3	4	2	7	7	9	10
	Expected		1.31	1.39	1.47	1.65	2.11	2.80	4.71	6.85	8.36	9.64
Non-Response	Observed		14	13	13	11	10	11	6	6	4	3
	Expected		12.69	12.61	12.53	12.35	11.89	10.20	8.29	6.15	4.64	3.36
Avg. Observed Prob.			0.00	0.07	0.07	0.21	0.29	0.15	0.54	0.54	0.69	0.77
Avg. Predicted Prob.			0.09	0.10	0.11	0.12	0.15	0.22	0.36	0.53	0.64	0.74
Chi-Sq Component			1.44	0.12	0.17	1.26	2.01	0.29	1.75	0.01	0.14	0.05
Log Odds Observed				-2.56	-2.56	-1.30	-0.92	-1.70	0.15	0.15	0.81	1.20
Log Odds Predicted			-2.27	-2.21	-2.14	-2.01	-1.73	-1.29	-0.57	0.11	0.59	1.05
Records in Bin			14	14	14	14	14	13	13	13	13	13
% Records in Bin			10.37	10.37	10.37	10.37	10.37	9.63	9.63	9.63	9.63	9.63

Precision: 2     Equal width     Balanced    Total Chi-Sq: 7.23231    HL Stat. P-value: 0.511788   

- ✓ For a binary target, the choice of Focus class does not affect the essence of the results.

Response Observed	Sum of Case Weights for Focus Class observations in the bin.
Response Expected	Sum of Model responses for Focus Class observations in the bin.
Non-Response Observed	Sum of Case Weights for non-Focus Class observations in the bin.
Non-Response Expected	Sum of Model responses for non-Focus Class observations in the bin.

Avg. Observed Prob.	<b>Response Observed</b> divided by sum of Case Weights of all observations in the bin.
Avg. Predicted Prob.	<b>Response Expected</b> divided by sum of Model responses for all observations in the bin
Chi-Sq Component	Contains Chi squared component computed by formula described above
Log Odds Observed	Log from <b>Avg. Observed Prob.</b> divided by (1 – <b>Avg. Observed Prob.</b> )
Log Odds Predicted	Log from <b>Avg. Predicted Prob.</b> divided by (1 – <b>Avg. Predicted Prob.</b> )
Records in Bin	Number of cases in the bin
% Records in Bin	Percent of cases in the bin from total number in sample

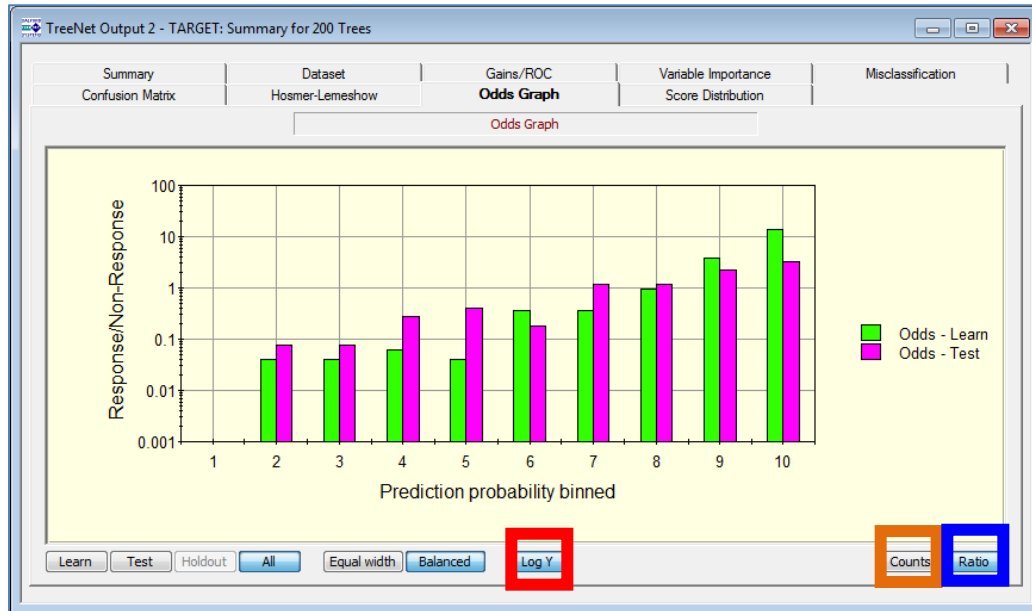
You can switch **between the following alternative breakdowns of records into bins (see picture on the previous page)**

- **Equal width** – each bin is populated with cases with predicted probabilities in the range of a decile (e.g. a decile of probabilities between 0.1 and 0.2).
  - **Balanced** – each bin has the same sum of Case Weights.
- 🔍 CART uses nodes as bins so this condition cannot be satisfied exactly.

### Odds Graph tab

✓ This is a binary classification-only feature.

Odds graph visualizes the **Hosmer-Lemeshow** table as a chart.



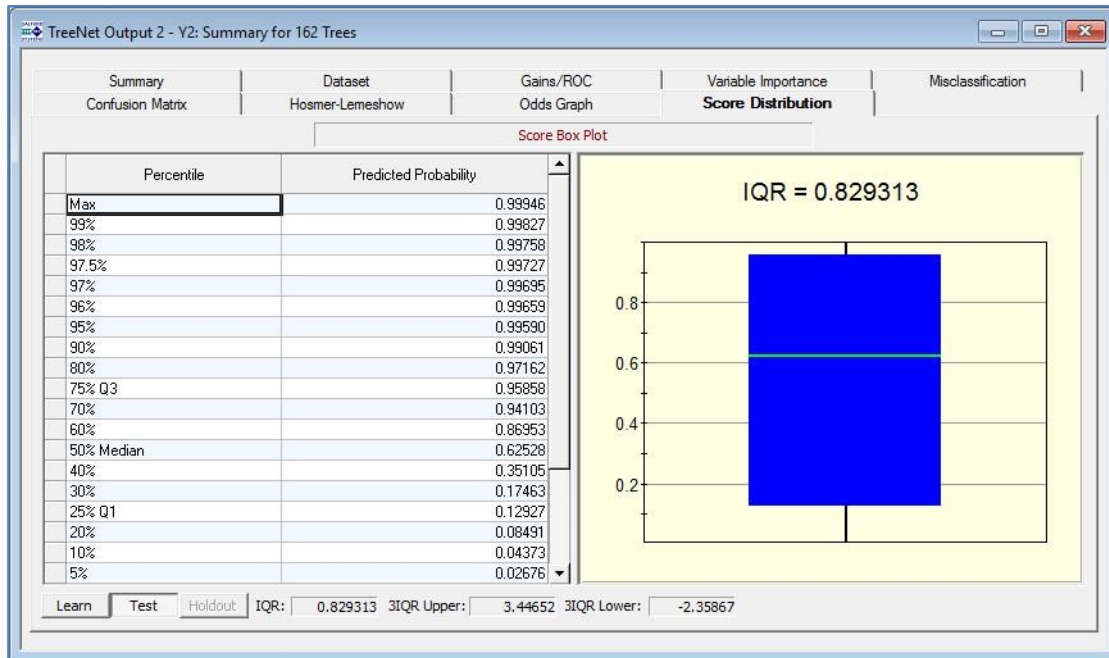
The chart supports the following modes.

**Counts button** – shows just raw sum of weights for cases predicted as focus class (Response\Non-Response Expected rows).

**Ratio button** – shows ratio of Response Expected divided to Non-Response Expected.

**Log Y button** switches Y-axis between normal and logarithmic scales.

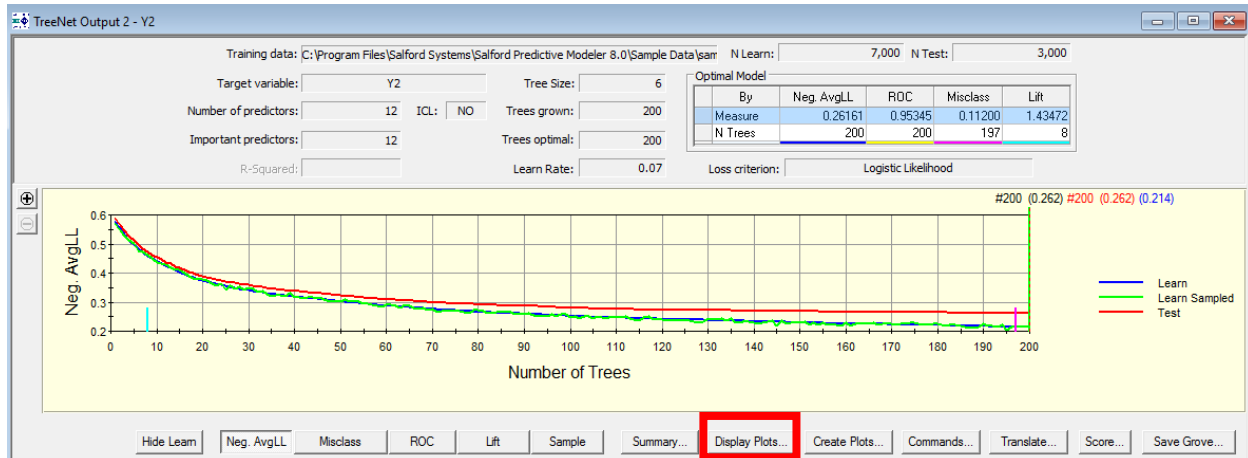
## Score Distribution Tab



In the Score Distribution Tab, user can monitor the spread of predicted probabilities (or scores) that may be too tight in some TreeNet models. Such models rank order well, but do not accurately capture probabilities. This result can be repaired by growing more trees or requesting optional recalibration. See section on [Using Logistic Calibration on TreeNet Results](#) for this option.

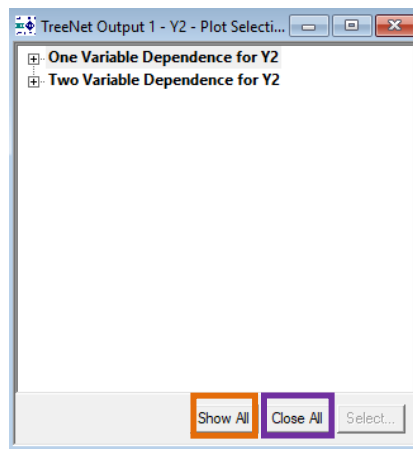
## Display Plots Button: Viewing PDPs

In TreeNet® software you can specify that partial dependency plots (PDPs) are created before the model is built (see the [Plots & Options Tab](#) section above; we discuss how to create PDPs after the model is built in the [Create and Viewing Dependency Plots](#) section). To view the partial dependency plots click the “Display Plots...” button (see **red rectangle below**)



After you click the **Display Plots...** button you will see the following menu.

## View All Plots

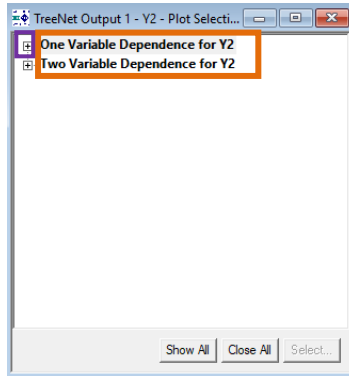


Click **Show All** to display all plots (this may take a while if you requested many plots; see the [Warning: Number of PDPs](#) section for information on the number of plots generated)

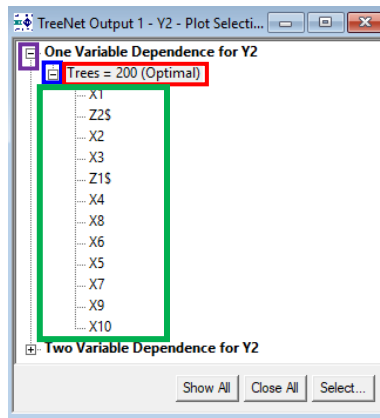
Click **Close All** to close all plots.

### View All One (or Two) Variable PDPs Only

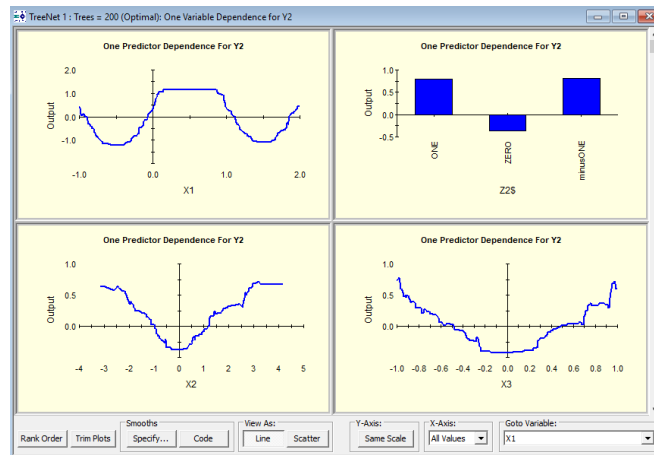
Click either one of the “plus” signs + for the one or two variable partial dependency plots in the **orange rectangle below**:



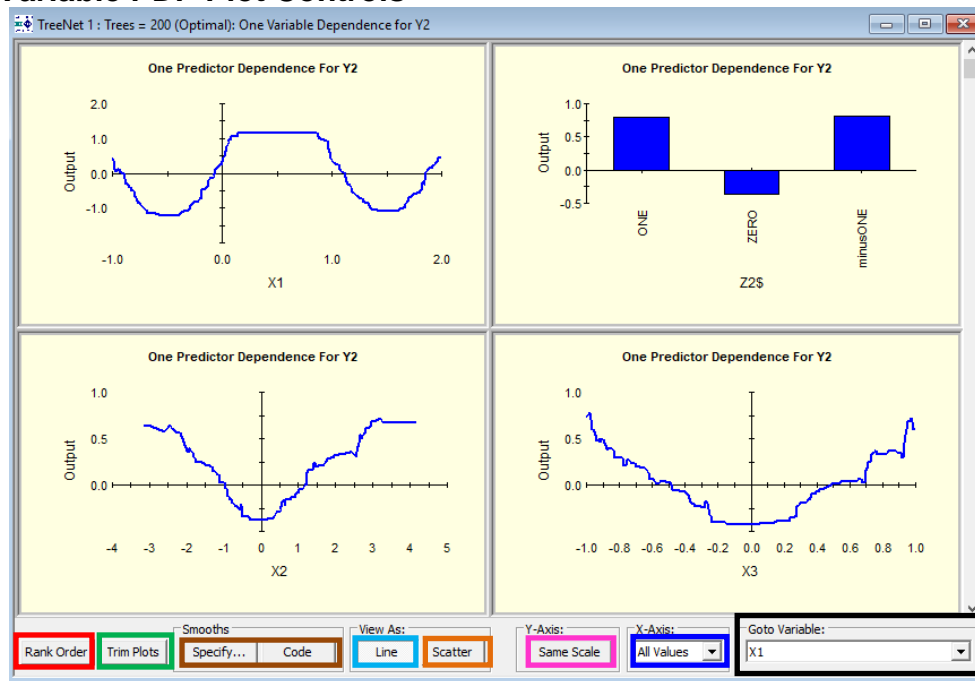
Click the other plus sign (**blue rectangle below**) to expand the list of variables with one variable PDPs:



Click the **Trees = 200 (Optimal)** label to view all one variable PDPs or click the name of an individual variable to view a partial dependency plot (any name in the **green rectangle above**) for a single variable. Here is the result of clicking **Trees = 200 (Optimal)**:



## All One Variable PDP Plot Controls



**Rank Order**– “de-scales” the horizontal axis such that all sampled points are spaced uniformly. This helps viewing “congested” areas of plots, especially when outliers are present.

**Trim Plots**– truncate plot tails usually due to outliers.

**Line**– view the PDPs as shown in the picture above.

**Specify & Code**–controls for spline approximations to the partial dependency plot (see the [Spline Approximations](#) section below for more details).

**Scatter**– view individual points that were sampled in plots construction. This is useful in identifying possible outliers or “empty” areas of data.

**Y-Axis: Same Scale**– imposes an identical scale on all vertical axes and allows you to see the comparative effect of different variable contributions.

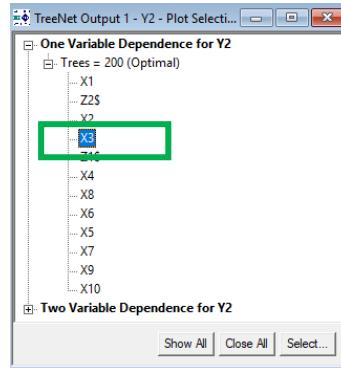
**X-Axis: All Values or Zoomed**– zoom in on the x-axis if desired (Zoomed option); otherwise the “All Values” option that is shown above will be used.

**Goto Variable** – use to navigate directly to the partial dependency plot of a particular variable of interest. This is especially useful when there a large number of predictors (in that case searching through all of the plots can be quite tedious).

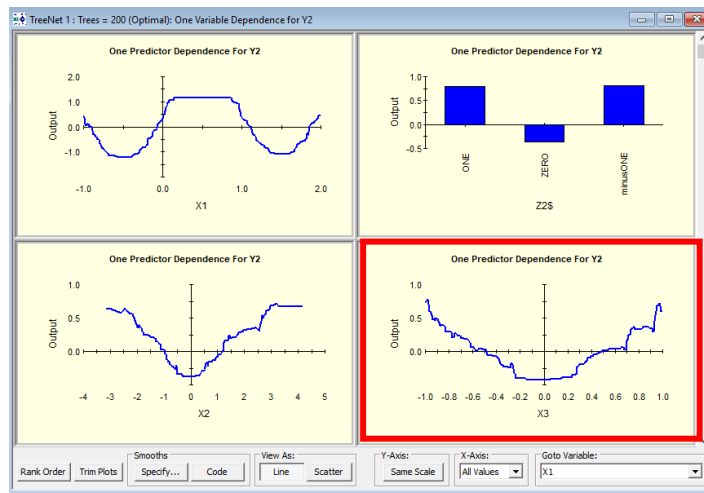


### View an Individual PDP

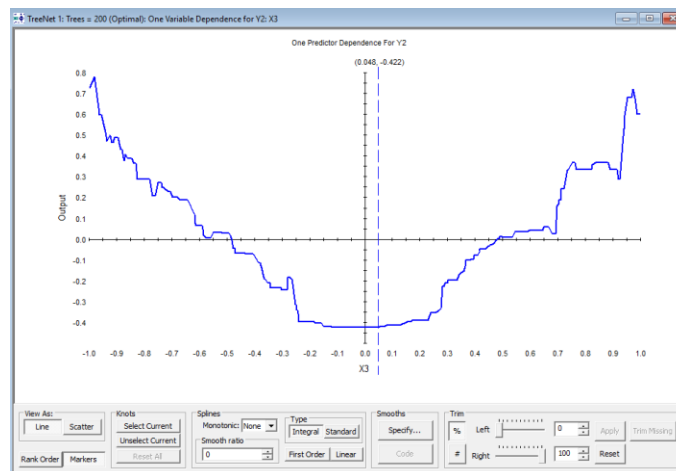
1. Click the desired variable name in the Plot Selection setup (**green rectangle below**) after clicking the desired plus signs +.



2. Double click on the desired plot if viewing all PDPs (**red rectangle below**; this is applicable when viewing either all one variable plots or all two variable plots).



The result of using either (1) or (2) is the following:



## Partial Dependency Plots (PDPs)

Partial dependency plots represent a very powerful interpretational side of TreeNet. They were designed to help understand how individual variables or pairs of variables contribute to the predicted response once the effects of all remaining predictors have been accounted for.

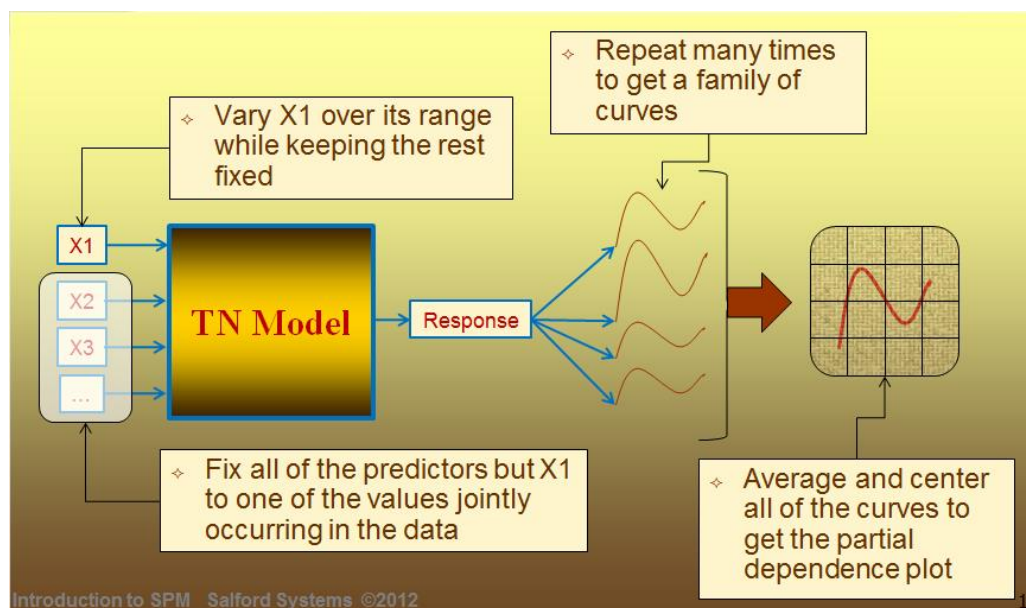
Depending on the complexity of a TreeNet model, the plots can be either exact or at the very least provide useful visual approximations to the nature of dependencies. The former case arises for truly additive models or models with completely isolated interacting pairs of variables, for example, utilizing the ICL language introduced later in this manual.

### Constructing One Variable PDPs

As an example, let's say that we have built a TreeNet model and we want to construct a partial dependency plot for a predictor variable X1. To generate the partial dependency plot for X1 we follow the following steps:

7. Randomly select a record
8. Plug the record from Step 1 into the model. This produces a predicted value
9. Hold all values for all variables in the record constant except change the value for X1. Plug this new artificial record into the model which produces a predicted value.
  - a. By "change the value for X1" we mean just use other values occurring in the data
10. Repeat Step 3 for every unique value of X1. This generates multiple predicted values which are plotted against the variable X1 and then connected to form one curve.
11. Repeat Steps 1-4 many times (the default in SPM is 500). This produces multiple curves.
12. Average and center the curves to get the partial dependency plot for the variable X1.

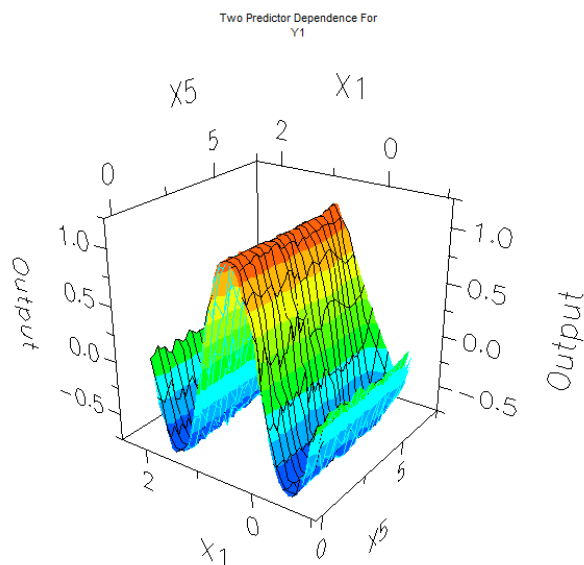
This process is illustrated in the picture below:



## Constructing Two Variable PDPs

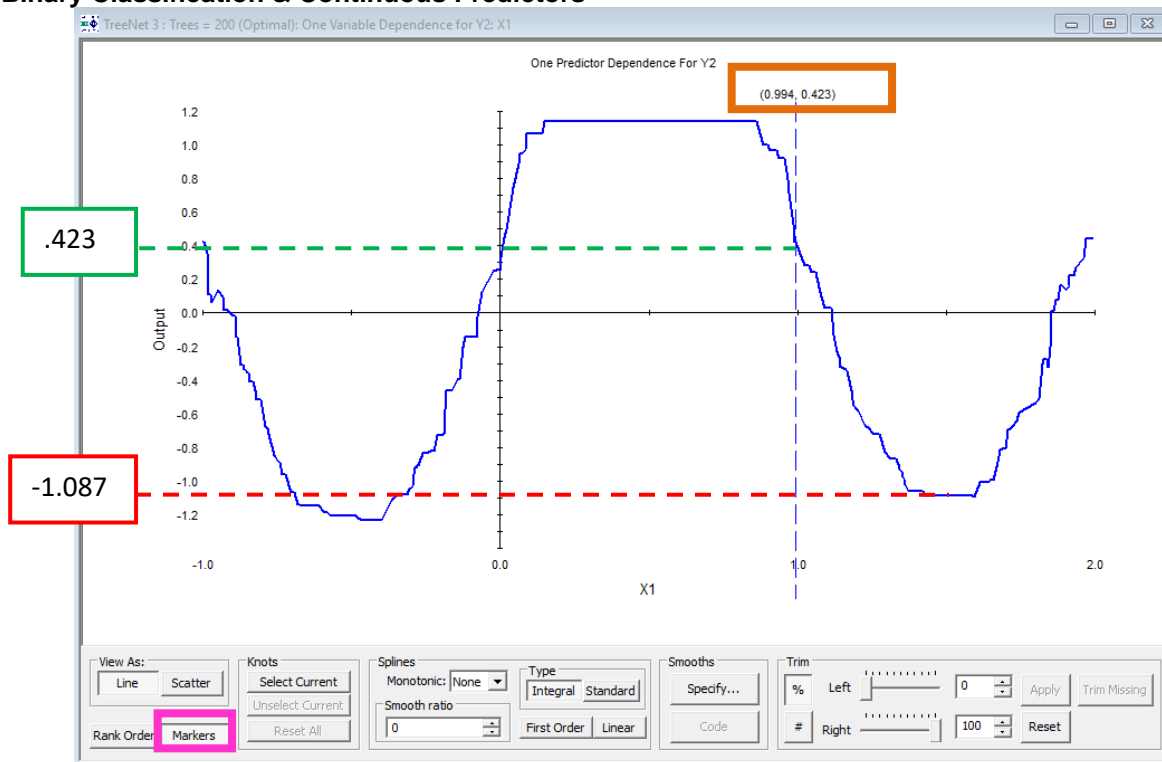
To construct two variable partial dependency plots we follow the same procedure discussed above in the Constructing One Variable PDPs section except that we now have two variables and when we plot we now have a surface instead of a line. As an example, let's say that we have built a TreeNet model and we want to construct a partial dependency plot for predictor variables X1 and X5. To generate the partial dependency plot for X1 and X5 we follow the following steps:

1. Randomly select a record
2. Plug the record from Step 1 into the model. This produces a predicted value
3. Hold all values for all variables in the record constant except change the value for X1 and X5. Plug this new artificial record into the model which produces a predicted value.
  - a. By "change the value for X1 and X5" we mean just use other values that occur for X1 and X5 in the data
4. Repeat Step 3 for every unique combination of X1 and X5. This generated multiple predicted values which are plotted against both X1 and X5 to form one surface.
5. Repeat Steps 1-4 many times (the default in SPM is 500). This produces multiple surfaces.
6. Average and center the surfaces to get the two-variable partial dependency plot for the variables X1 and X5.



## Interpreting PDPs: Binary Classification

### Binary Classification & Continuous Predictors



In this example our target variable is a variable called Y2. Y2 can take two values: 0 or 1, so this is a binary classification problem. The predictor variable is a continuous variable called X1. The **Y-axis for partial dependency plots in the case of binary classification is the average  $\frac{1}{2}$  log-odds of the event** (the user defines the event class and this is referred to the “Focus Class”; see this link for more details). The dotted lines and rectangles shown in the picture were created outside of TreeNet software.

#### Interpretation:

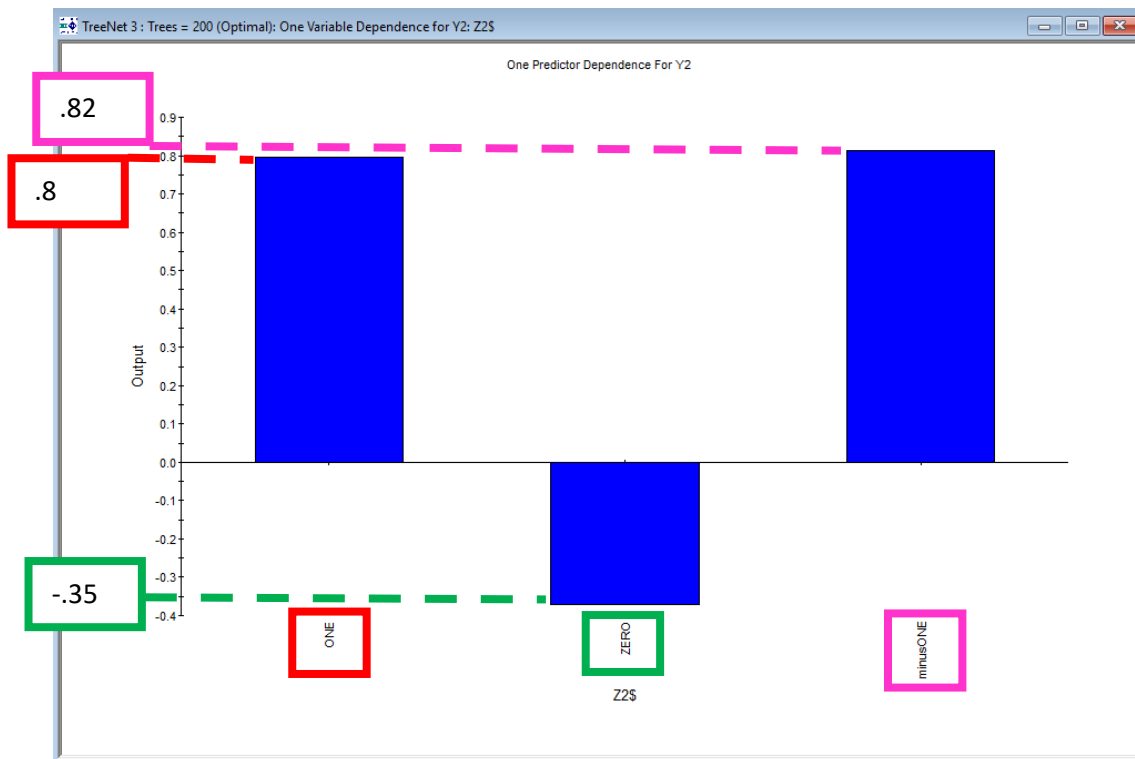
If X1 increases from 1 to 1.5 then the  $\frac{1}{2}$  log-odds of the event, on average, decreases by 1.51 (because  $-1.51 = -1.087 - .423$ ).

See the picture above. Click the “Markers” button above to display the **dotted blue line** shown in the picture. The **dotted blue line** has a **pair of coordinates (x-value, y-value)** that allow you to obtain the values used in the calculation  $-1.51 = -1.087 - .423$ .

If X1 increases from .15 to .864, then the  $\frac{1}{2}$  log-odds of the event, on average, does not change.

If X1 decreases from -.5 to -1, then the  $\frac{1}{2}$  log-odds of the event, on average, increases by 1.628 (because  $1.628 = .424 - [-1.204]$ ).

## Binary Classification & Categorical Predictors



In this example the target variable is a variable called Y2. Y2 can take two values: 0 or 1, so this is a binary classification problem. The predictor variable is a categorical variable called Z2 (the dollar sign \$ denotes that Z2 is categorical variable). Z2 takes three values: “ONE”, “ZERO”, and “minusONE”. The **Y-axis for partial dependency plots in the case of binary classification is the average  $\frac{1}{2}$  log-odds of the event** (the user defines the event class and this is referred to the “Focus Class”; see this link for more details).

### Interpretation

If Z2 = “ONE”, then the  $\frac{1}{2}$  log-odds of the event, on average, **increases by .8** after accounting for the other variables in the model.

If Z2 = “ZERO”, then the  $\frac{1}{2}$  log-odds of the event, on average, **decreases by .35** after accounting for the other variables in the model.

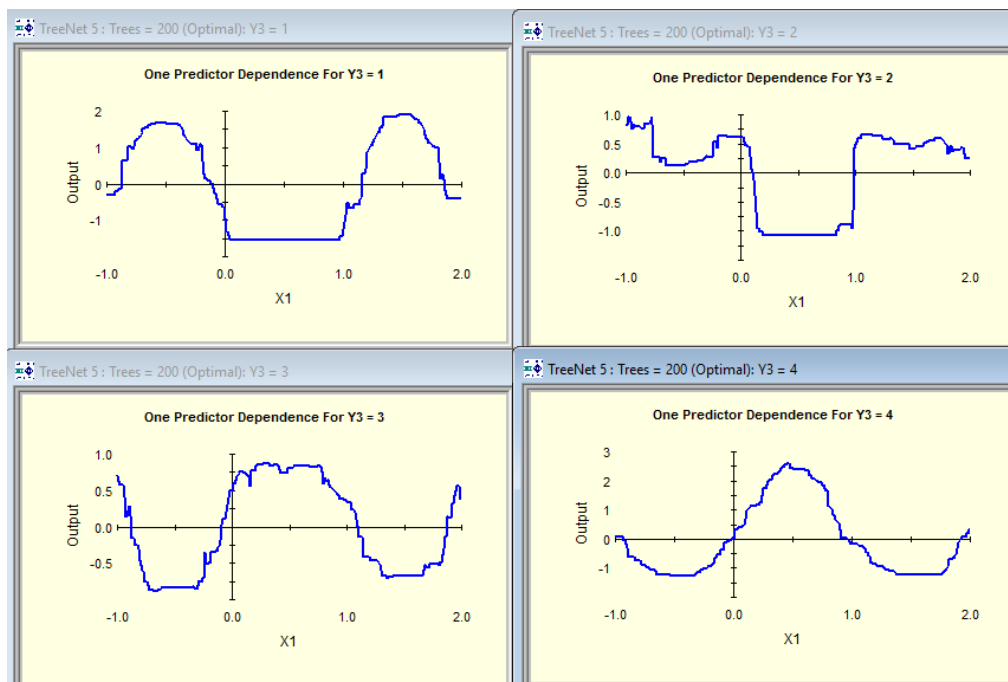
If Z2 = “minusONE”, then the  $\frac{1}{2}$  log-odds of the event, on average, **increases by .82** after accounting for the other variables in the model.

## Interpreting PDPs: Multinomial Classification

Let's suppose that we have a target variable Y with K categories and we want to predict the category of Y based on the values of the predictor variables. This type of problem is commonly referred to as a *multinomial classification* problem. The gradient boosting algorithm for multinomial classification problems involves fitting K binary classification models. As an example, let's say the target variable is a categorical variable with 5 values: red, green, blue, yellow, and orange. To build a multinomial TreeNet model we build the following 5 binary TreeNet models: Red vs. Not Red, Green vs. Not Green, Blue vs. Not Blue, Yellow vs. Not Yellow, Orange vs. Not Orange. The predicted class (i.e. red, green, blue, yellow, or orange) is the class that has minimizes the model error or cost.

In the context of partial dependency plots for multinomial classification problems, we will have K plots for each variable with each plot corresponding to an individual binary classification problem (like Red vs. Not Red or Blue vs. Not Blue).

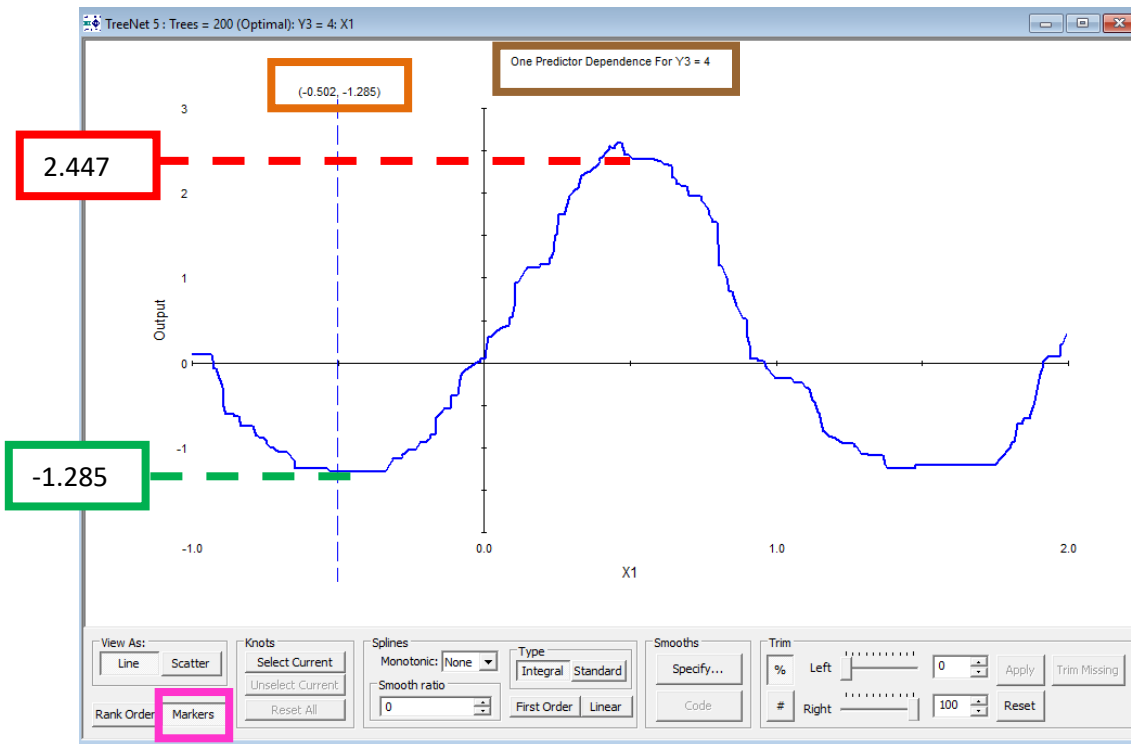
### Multinomial PDPs & Continuous Predictors



The four PDPs above are for a continuous predictor X1 and a multinomial target variable Y3. Y3 can take 4 values: 1, 2, 3, or 4, so this is a multinomial classification problem where we fit K=4 models: 1 vs. Not 1, 2 vs. Not 2, 3 vs. Not 3, and 4 vs. Not 4 (notice the plot labels in the top middle such as “One Predictor Dependence For Y3 = 1” etc.).

The **Y-axis for partial dependency plots in the case of multinomial classification is still the average  $\frac{1}{2}$  log-odds of the event** (in the plots above you can see the event class in the top middle of each plot: “One Predictor Dependence for Y3 = 3”). On the next page we interpret the Partial Dependency Plot for Y3 = 4 (plot in the lower right corner in the picture above)

## Interpretation Example



In this example our plot is for a model for classifying the target variable Y3 as a “4” or “Not 4,” so this is (still) a binary classification problem (we know that this is for the problem Y3 =4 vs Not 4 because at the top of the plot we can see the label “**One Predictor Dependence For Y3 = 4**”). The predictor variable is a continuous variable called X1. The **Y-axis for partial dependency plots in the case of binary classification is (still) the average  $\frac{1}{2}$  log-odds of the event** (in this case the event is Y3 = 4).

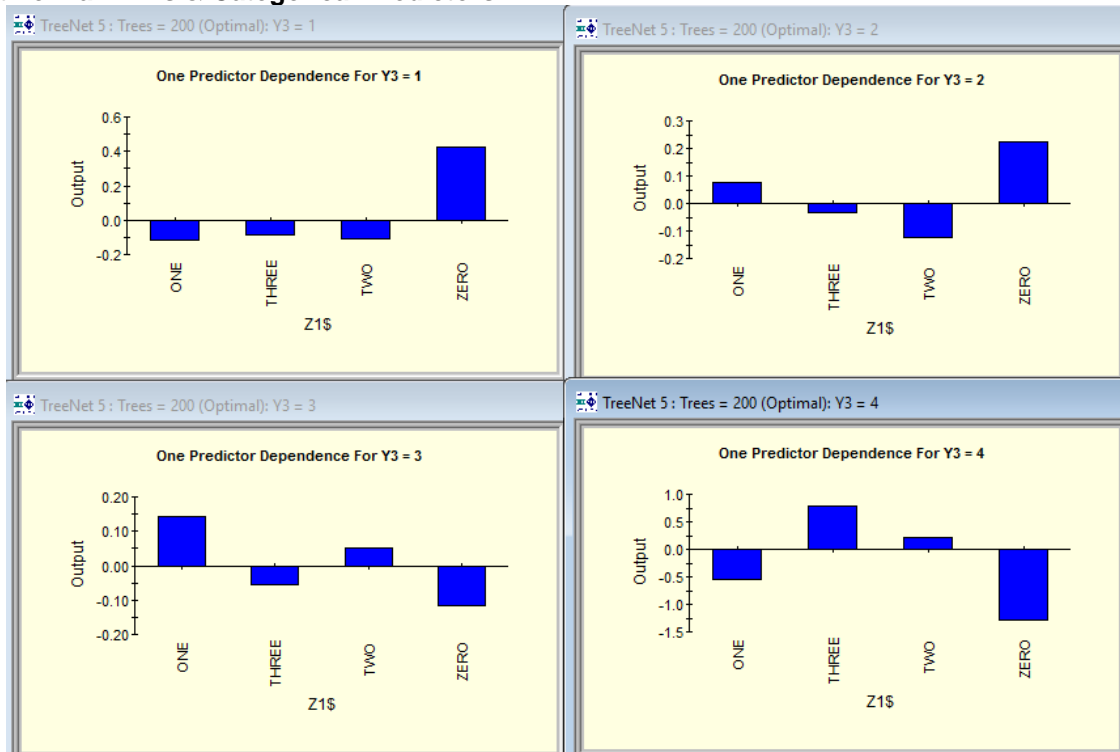
### Interpretation:

If X1 increases from -5. to .5, then the  $\frac{1}{2}$  log-odds of Y3 =4 (versus Y3 not being equal to 4), on average, increases by 3.732 (because  $3.732 = 2.447 - [-1.285]$ ).

See the picture above. Click the “**Markers**” button above to display the **dotted blue line** shown in the picture. The **dotted blue line** has a **pair of coordinates (x-value, y-value)** that are used in the calculation  $-1.51 = -1.087 - .423$ .

The interpretation for the predictor variable X1 for Y3 =4 (vs. Y3 not equal to 4) is the similar to the interpretation for the binary classification model because fundamentally a multinomial classification model is multiple binary classification models. The interpretation for the other three PDPs for the predictor X1 will have an interpretation similar to the one here.

## Multinomial PDPs & Categorical Predictors



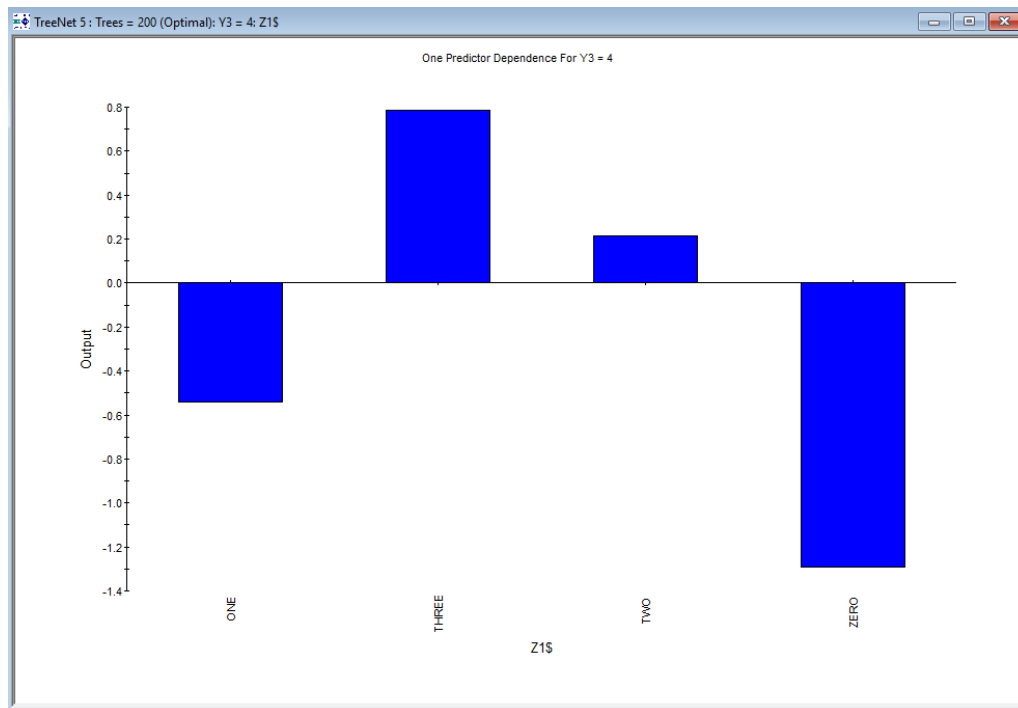
In this example the target variable is a variable called Y3. Y3 can take 4 values: 1, 2, 3, or 4, so this is a multinomial classification problem where we fit K=4 models: 1 vs. Not 1, 2 vs. Not 2, 3 vs. Not 3, and 4 vs. Not 4. The predictor variable is a continuous variable called X1.

The Y-axis for partial dependency plots in the case of multinomial classification is still the average  $\frac{1}{2}$  log-odds of the event (in the plots above you can see the event class in the top middle of each plot: “One Predictor Dependence for Y3 = 3” and so on).

On the next page we will interpret the Partial Dependency Plot for Y3 = 4 (plot in the lower right corner in the picture above)



## Interpretation Example



In this example our plot is for a model for classifying the target variable Y3 as a “4” or “Not 4,” so this is (still) a binary classification problem (we know that this is for the problem Y3 =4 vs Not 4 because at the top of the plot we can see the label “**One Predictor Dependence For Y3 = 4**”). The predictor variable is a categorical variable called Z1\$ (the dollar sign \$ means that Z1 has character values). The **Y-axis for partial dependency plots in the case of binary classification is (still) the average  $\frac{1}{2}$  log-odds of the event** (in this case the event is Y3 = 4 (vs. Y3 Not Equal to 4) ).

### Interpretation:

If Z1 has a value of “ONE” then the  $\frac{1}{2}$  log-odds of Y3 =4 (versus Y3 not being equal to 4), on average, decreases by approximately .55

If Z1 has a value of “THREE” then the  $\frac{1}{2}$  log-odds of Y3 =4 (versus Y3 not being equal to 4), on average, increases by approximately .78

If Z1 has a value of “TWO” then the  $\frac{1}{2}$  log-odds of Y3 =4 (versus Y3 not being equal to 4), on average, increases by approximately .20

If Z1 has a value of “ZERO” then the  $\frac{1}{2}$  log-odds of Y3 =4 (versus Y3 not being equal to 4), on average, decreases by approximately 1.32

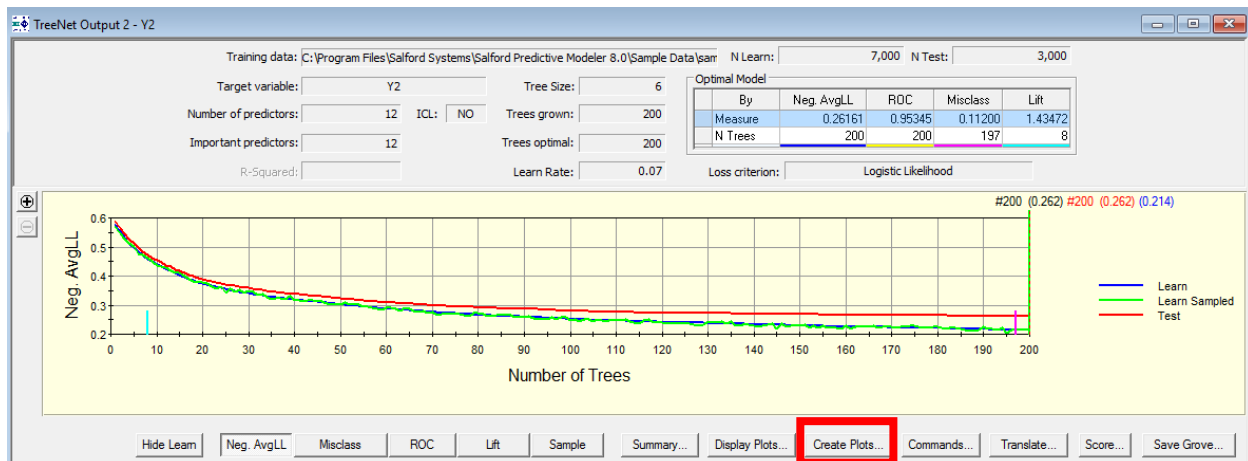
The interpretation for the predictor variable X1 for Y3 =4 (vs. Y3 not equal to 4) is similar to the interpretation for the binary classification model because fundamentally a multinomial classification model is multiple binary classification models. The interpretation for the other three PDPs for the predictor Z1 will have an interpretation similar to the one here.

## Spline Approximations to the PDPs

Users have the option to produce spline approximations to the one variable partial dependency plots for continuous predictor variables. See the [Spline Approximations to the Partial Dependency Plots section](#) for more information.

## Create Plots

TreeNet has a powerful facility to generate partial dependency plots for any variable or pair of variables and any model size selected by the user. Click the **Create Plots...** button in the TreeNet output window (this is only available for most recent TreeNet modeling run). See the [Create Plots Button section](#) for more information.



## Spline Approximations to Partial Dependency Plots

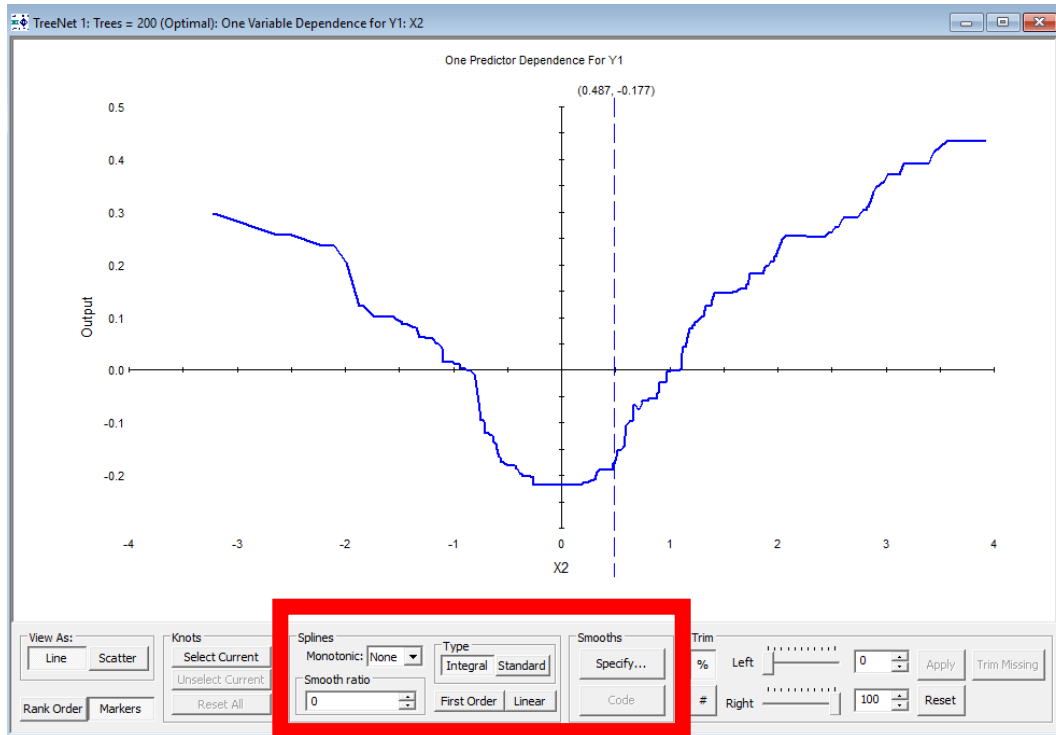
Users have the option to produce spline approximations to the one variable partial dependency plots for continuous predictor variables. The plot approximation machinery was designed to facilitate model extraction based on TreeNet model insights. The idea is that each dependency plot represents a variable transformation; therefore, once a variable is transformed it can be used in the conventional regression modeling as a main effect. Note that the transformation becomes exact in the absence of interactions in which case the TreeNet engine can be used as an ultimate transformation discovery machine.

Plot approximations replace the internally constructed partial dependency plots by simple analytical expressions calculated to have the best fit to the partial dependency plot of interest.

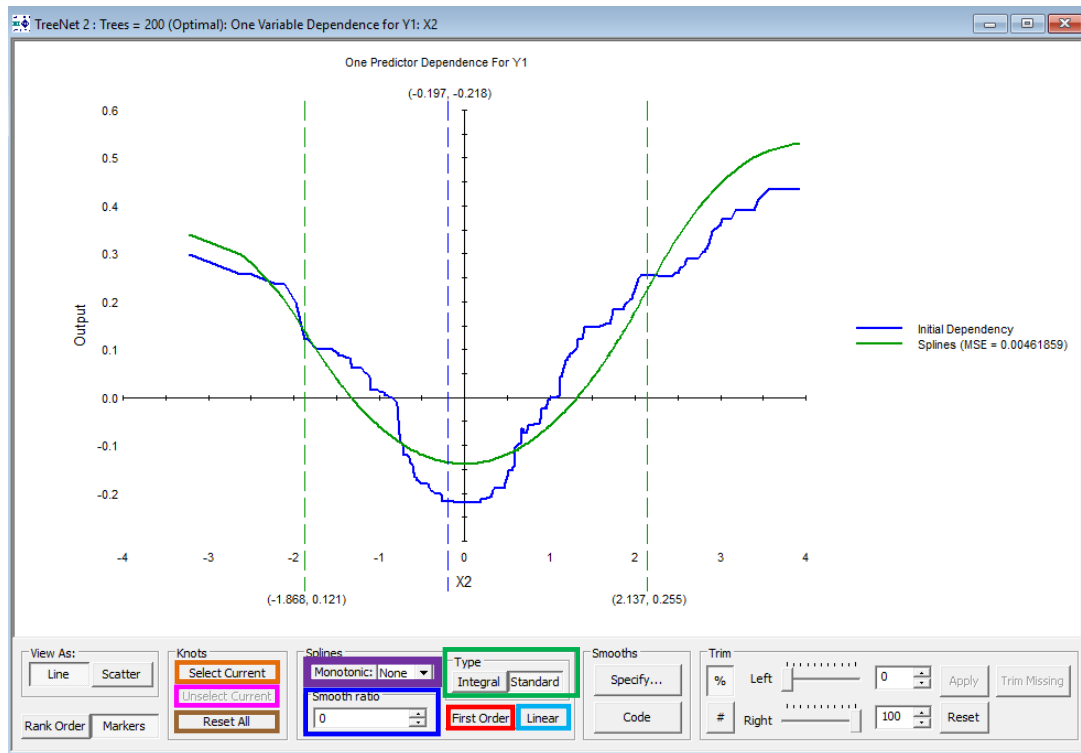
By far, the spline approximations are the most flexible to accomplish this task. Each spline is defined by a set of knots (points on the horizontal X-axis) and its order (first-order spline for piece-wise linear segments and second-order splines for the smooth quadratic segments). The internal calculations ensure the continuity and/or the smoothness of the spline approximation.

The spline controls are located below the plot (**red rectangle below**) and are explained in detail on the next page. The following view of a partial dependency plot can be obtained in a number of ways (see the [View an Individual PDP](#) section).

- ✓ There is no approximations facility for the bivariate plots due to their natural complexity.
- ✓ There is no command line support for the approximations.



## Manual Knot Placement



### Two Options to Add Your Own Knots

- To add a knot, position the vertical slider (the vertical slider is the blue dotted line near the middle of the picture above) at the point of interest and click the **Select Current button**
  - Note: click the **Unselect Current button** to deselect the current point of interest or the **Reset All button** to remove all current knots.
- Double clicking on the desired knot location on the curve.

Note that each added knot is represented by a vertical green dotted line (see the picture above). You can use the mouse pointer to move the existing knots left or right to achieve fine adjustment of the spline fit. In the example above, we show a two-knot spline solution to the X2 partial dependency plot.

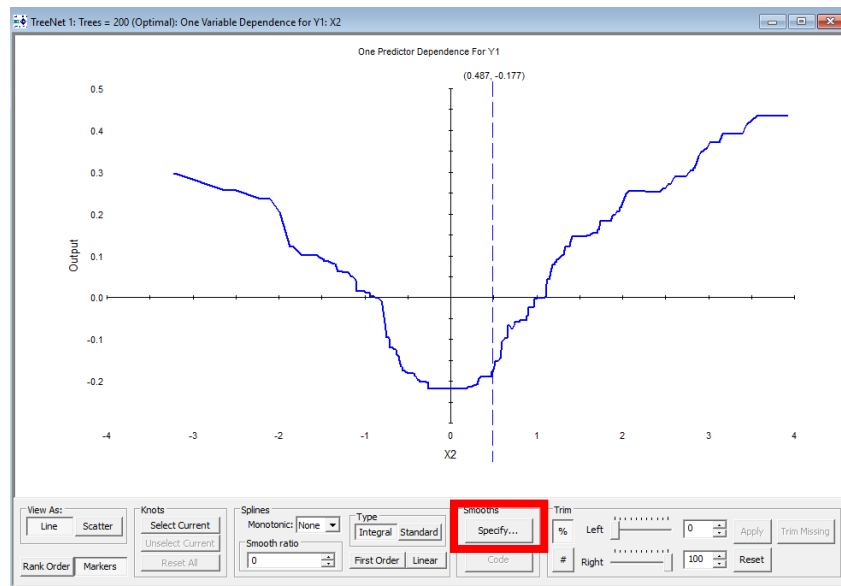
### Splines Section

- Monotonic Control:** enforce monotonicity constraint (Up, Down, or None) on the spline solution.
- Smooth Ratio Control:** enforce additional smoothness on the spline solution.

### Type Section

- Standard vs. Integral** spline type control is no longer supported. All splines are now integral spline because of their superior characteristics.
- First Order button:** By default, second-order splines are calculated. Press the First Order button to replace second-order splines with the first-order splines.
- Linear button:** adds a linear approximation to the plot.

## Automatic Knot Placement & Fixed Approximations



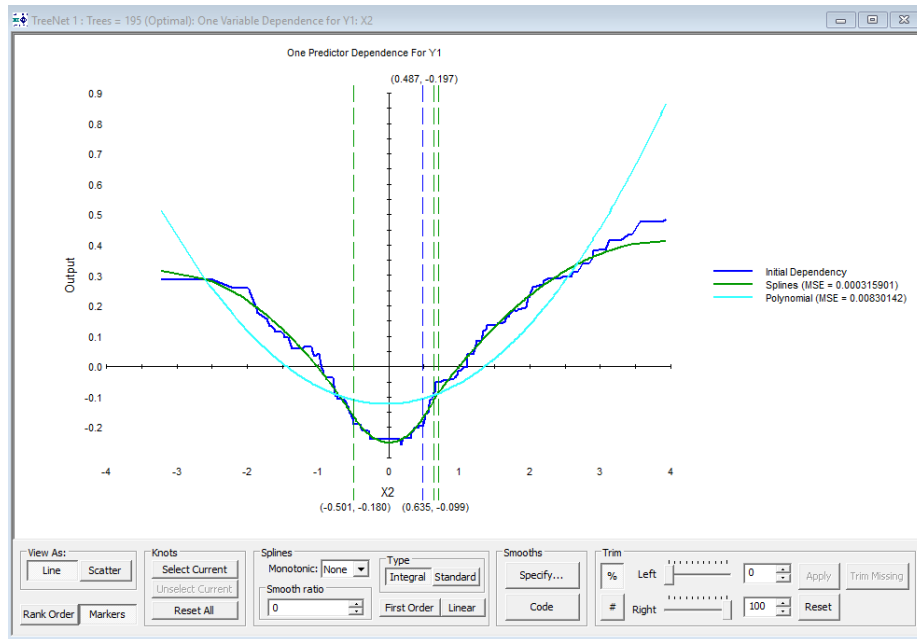
If you want SPM to determine the knot location automatically or to specify parametric approximations to the partial dependency plots then press the **“Specify...”** button. Clicking this button results in the following menu:

The 'TN Dependency Approximations Setup' dialog box is shown. It has a title bar with a close button (X). The 'Total Continuous Variables to approximate:' is set to 1. The 'Splines' checkbox is checked and highlighted with a red box. The 'Number of Knots:' is set to 3 and highlighted with an orange box. The 'Type' is set to 'Integral' and 'Order' is 'First Order'. The 'Monotonic:' is 'None' and 'Smooth rate:' is 0. Below this, there are four unchecked checkboxes: 'Logit', 'Inverse', 'Exponent', and 'Logarithm', each with a corresponding mathematical formula. The 'Polynomial' checkbox is checked and highlighted with a green box, and the 'Degree:' is set to 2 and highlighted with a blue box. At the bottom, there are 'Apply', 'Reset All', 'Cancel', and 'OK' buttons. The 'Trimmed Dependences Approximation' checkbox is checked.

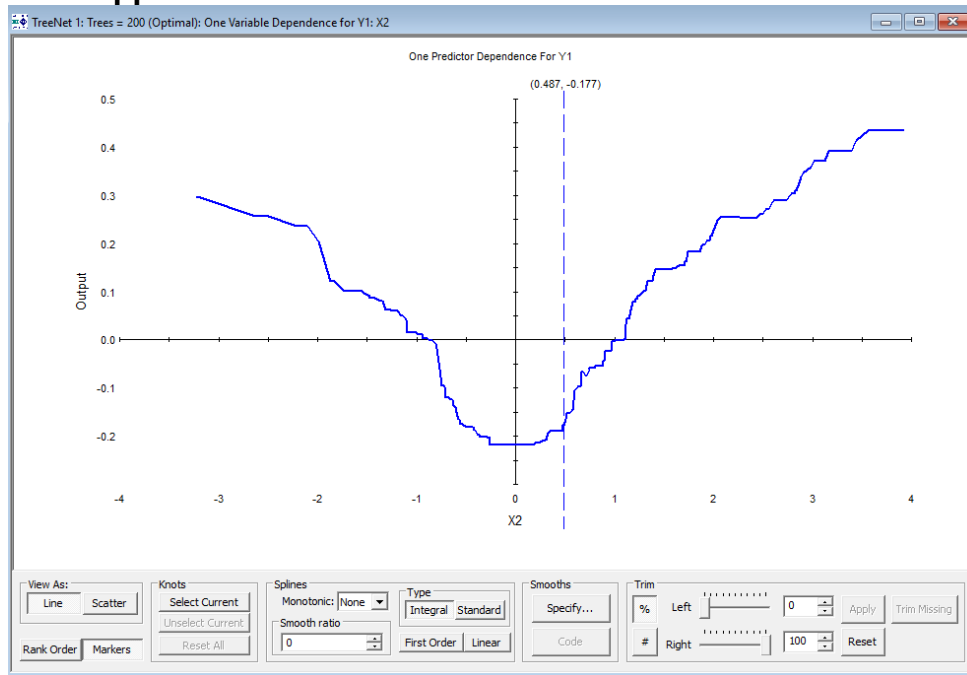
Check the **“Splines”** checkbox and specify the desired **Number of Knots** and modify the other spline controls in the **blue rectangle above** (the other settings are discussed in the [Adding Your Own Knots](#) section). To approximate the partial dependency plot with a parametric form then click any of the checkboxes in the **pink rectangle above**. Here we have checked the **Polynomial checkbox and specified a degree of 2**. The spline approximation with 3 knots and the degree 2 polynomial approximation will be overlaid.

Click the **“Ok”** button or the **“Apply”** button to approximate the plot with the spline. Click **“Reset All”** to remove all splines created for the plot.

The result of clicking the “Ok” button is the following. The spline with the three automatically added knots is in green and the degree 2 polynomial spline is in light blue.



## Using Splines Approximations in a Model

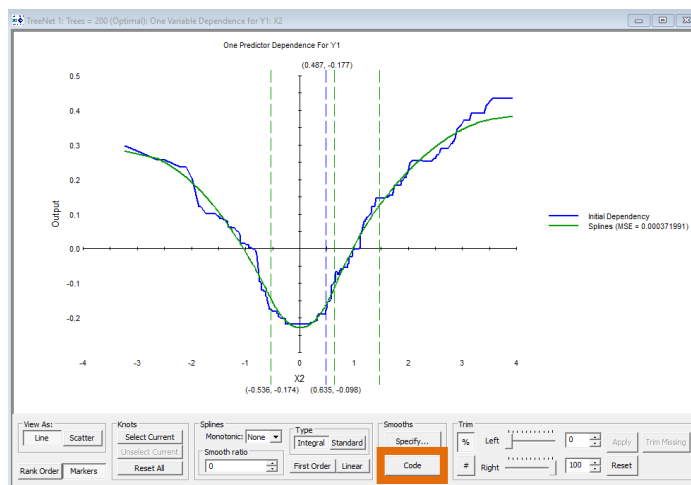


In this example, we approximate this plot with a spline that has three knots.

1. Click the “Specify” button (red rectangle above). Upon clicking the “Specify” button you will see the following menu.

Click the “**Splines**” checkbox and set the **Number of Knots to 3** then click **Ok**.

The result of clicking the “Ok” button is the following:



In the picture above notice that the three dotted lines correspond to the knot placement (we set the number of knots to 3 in the spline setup). To create, save, and add a spline approximation to the list of predictor variables that can be in the model, click the “Code” button above which results in the following window with SPM Basic Code

```

REM Spline Model
% IF X2 < -3.22628 THEN FOR
% LET X2_S1 = 0
% LET X2_S2 = 0
% LET X2_S3 = 0
% NEXT
% ELSE IF X2 < -0.535516 THEN FOR
% LET X2_S1 = (X2 + 3.22628) / 10.3899
% LET X2_S2 = 0
% LET X2_S3 = 0
% NEXT
% ELSE IF X2 < 0.635039 THEN FOR
% LET X2_S1 = 1 - (X2 - 0.635039) / 4.51988
% LET X2_S2 = (X2 + 0.535516) * (X2 - 0.535516) / 2.3575
% LET X2_S3 = 0
% NEXT
% ELSE IF X2 < 1.47849 THEN FOR
% LET X2_S1 = 1
% LET X2_S2 = 1 - (X2 - 1.47849) / 1.69871
% LET X2_S3 = (X2 - 0.635039) * (X2 - 0.635039) / 2.77917
% NEXT
% ELSE IF X2 < 3.93005 THEN FOR
% LET X2_S1 = 1
% LET X2_S2 = 1
% LET X2_S3 = 1 - (X2 - 3.93005) * (X2 - 3.93005) / 8.07791
% NEXT
% ELSE FOR
% LET X2_S1 = 1
% LET X2_S2 = 1
% LET X2_S3 = 1
% NEXT
% LET X2_S = 0.28094 - 0.608845 * X2_S1 + 0.367268 * X2_S2 + 0.342178 * X2_S3

```

**Save:** save the code in a command file (only active when SAS is the code format)

**Submit:** submit the code directly to the command prompt in the Classic Output window

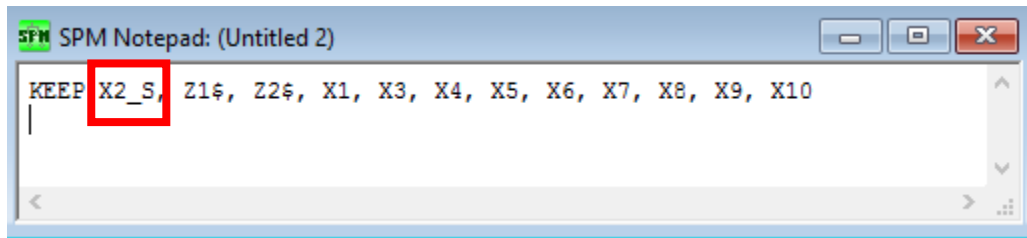
**Submit & Save:** save the new (and original) variables into a new dataset.

**Submit & Keep:** submits the SPM Basic code shown above (green rectangle above; this code creates the spline variable in SPM) and then automatically adds this new spline variable to the list of available predictors in the model. Click “Submit & Save”

**Include All Important Variables** checkbox to add the other important variables in the TreeNet model to keep list in addition to the new spline variable.



After you click the “**Submit & Keep**” button in the picture above, an SPM Notepad opens and allows you to submit commands to SPM:

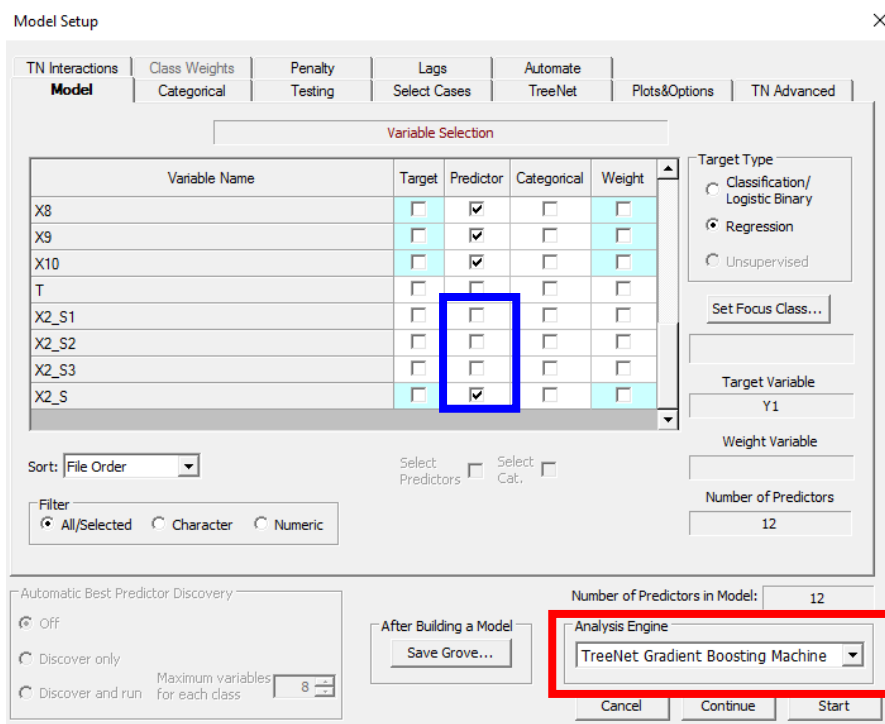


Notice the X2\_S (**red rectangle above**) in the KEEP list. X2\_S is the spline approximation to the partial dependency plot. Click into this window and press **CTRL W** to submit this command.

The result of the command

```
KEEP X2_S, Z1$, Z2$, X1, X3, X4, X5, X6, X7, X8, X9, X10
```

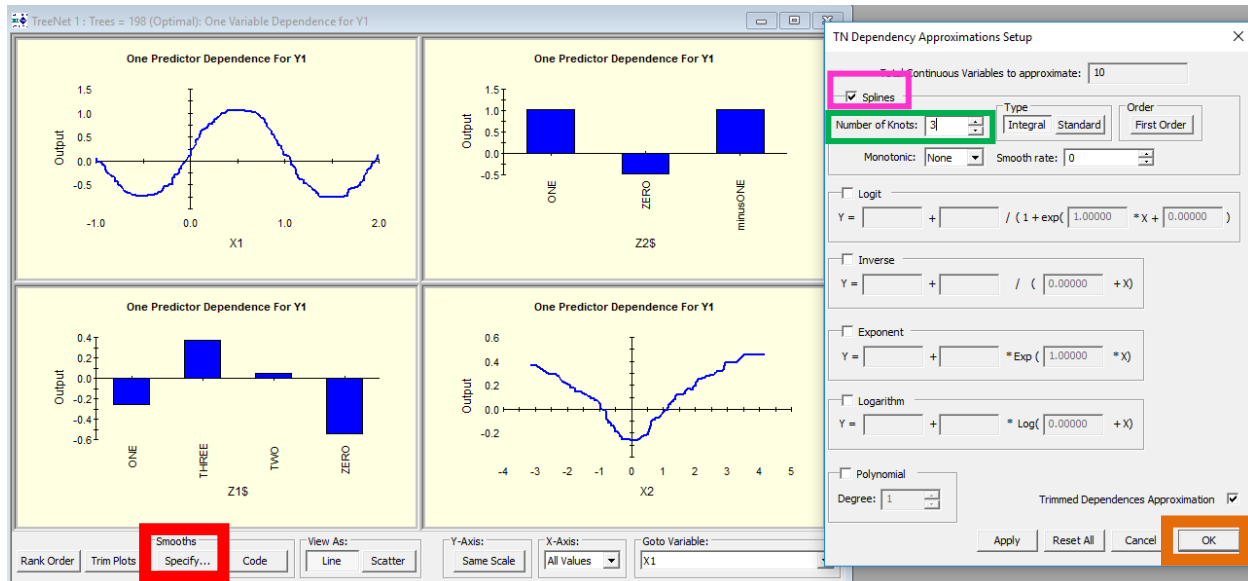
is that all variables in the list, including the spline approximation to the partial dependency plot X2\_S, are now set as predictors in a model. If you go back to the model setup you can see the result:



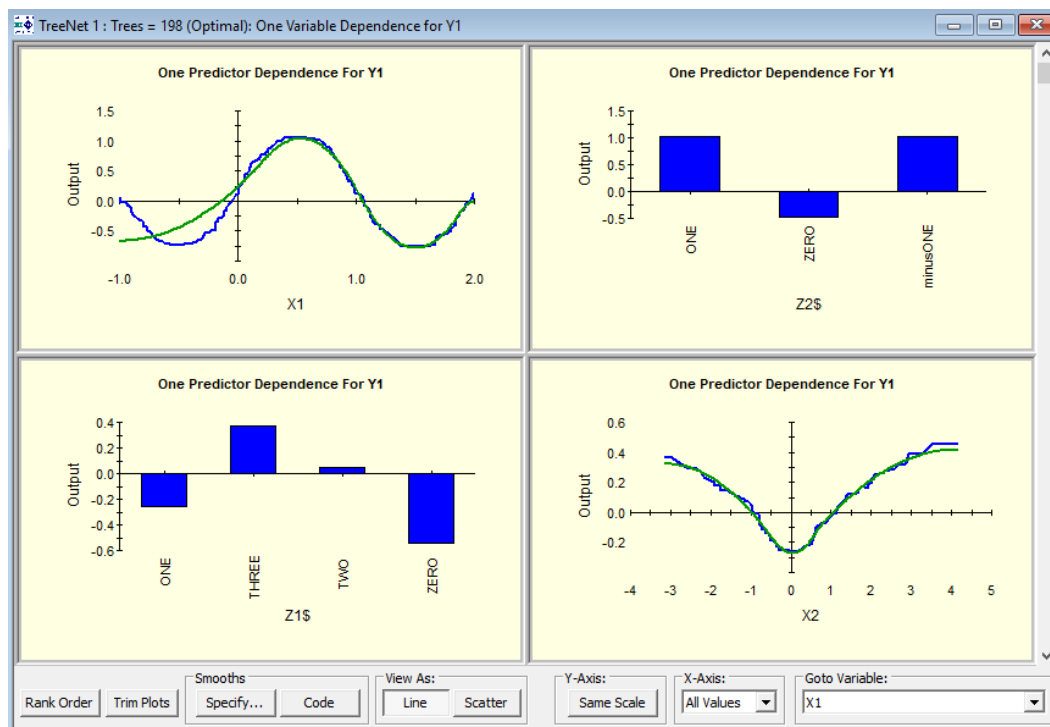
- ✓ X2\_S1, X2\_S2, and X2\_S3 (**blue rectangle above**) are used in the construction of the final spline variable X2\_S. Only X2\_S should be included as a predictor in the model as shown in the **blue rectangle above**.
- ✓ A popular use of these controls is to use TreeNet spline approximations as predictors in a standard logistic regression scenario. Nonlinearities are captured in the splines while traditional regression reporting is preserved. You can build a logistic regression model by changing the Analysis Engine (**red rectangle above**)

## Simultaneous Spline Approximations

You can generate approximations for all univariate graphs simultaneously by clicking the **Specify ...** button on the one variable dependence window (see [View All One \(or Two\) Variable PDPs Only](#) section for more information about seeing all one-variable partial dependency plots in a single window). To let SPM determine the optimal knot placements, click the **Splines** checkbox, specify the **number of knots to be 3**, and click the **OK** button.

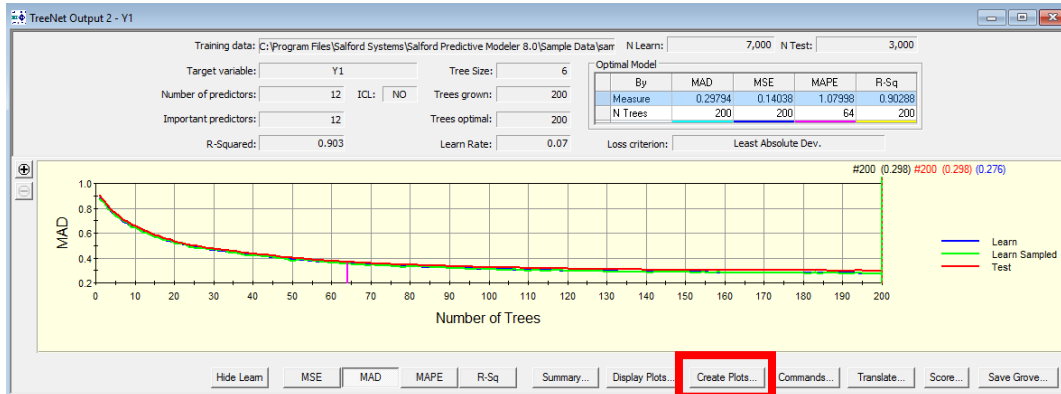


The result of clicking the **OK** button is a spline approximation for every numeric partial dependency plot. Notice that each numeric plot now has a green spline approximation overlaid on the original plot in blue:



## Create Plots Button

TreeNet has a powerful facility to generate partial dependency plots for any variable or pair of variables and any model size selected by the user. Click the **Create Plots...** button in the TreeNet output window (this button is only available for most recent TreeNet modeling run):



## Create Plots Tab

The 'Create Plots' dialog box is shown with the following settings:

- Data:** Data file: C:\Program Files\Salford Systems\Salford Predictive Modeler 8.0\Sample Data\sample
- Grove:** Grove file: TreeNet Output 2 - Y1, Type: Regression
- Select Number Of Trees To Create Plots For:**
  - Nr. of trees: 145 (highlighted with a blue box)
  - Entropy: [empty], Misclass: [empty]
  - Training Sample: LAD: 0.287, LS: 0.135 (highlighted with a green box)
  - Test Sample: LAD: 0.305, LS: 0.148 (highlighted with a green box)
  - Buttons: Optimal Tree (orange box), Select... (blue box)
- Select Types:**
  - One variable dependence,  Two variable dependence
  - 500 points, 5000 points, Centered
- Select Variables To Plot:**
  - Table with columns: Variable, Score
  - Variables: X1 (100.00), Z2\$ (62.36), Z1\$ (50.29), X2 (46.06), X3 (43.33), X4 (35.80), X9 (5.35), X7 (4.68), X8 (4.38), X10 (4.00)
  - Buttons: Use top (3 variables), Including (5 interacting), Importance, Interactions, Select, Select All, Remove, Remove All
  - Number of plots that may be created: 0
- Number of existing plots:** 79, Clear Existing
- Buttons: Cancel, Create Plots

**Nr. of trees:** number of trees that you want the plots to be based on

**Training Sample & Test Sample:** training and test sample error statistics for Least Absolute Deviation (LAD) and Least Squares (LS)

**Optimal Tree button:** sets the **Nr. of trees** to the number that corresponds to the optimal model

**Select button:** sets the **Nr. of trees** to be the size desired by the user and chosen based on clicking on the model error curve that appears.

Create Plots

**Create Plots** | Pairwise Interactions

Data  
Data file: C:\Program Files\Salford Systems\Salford Predictive Modeler 8.0\Sample Data\sample

Grove  
Grove file: TreeNet Output 4 - Y1 Type: Regression

Select Number Of Trees To Create Plots For

Training Sample  
Nr. of trees: 145 Entropy: Misclass: LAD: 0.287 LS: 0.135 **Optimal Tree**

Test Sample  
Sample: Entropy: Misclass: LAD: 0.305 LS: 0.148 **Select...**

Select Types

One variable dependence  Two variable dependence

500 points 5000 points **Centered**

Select Variables To Plot

Variable	Score
X1	100.00
Z2\$	62.36
Z1\$	50.29
X2	46.06
X3	43.33
X4	35.80
X9	5.35
X7	4.68
X8	4.38
X10	4.00

Use top 3 variables  
 Including 5 interacting

Importance Interactions

Select Select All

Remove Remove All

Number of plots that may be created: 0

Number of existing plots: 78 **Clear Existing**

Cancel Create Plots

**One variable dependence checkbox:** create one variable partial dependency plots

**Points:** number of records used in constructing the one variable PDP

**Two variable dependence checkbox:** create two variable partial dependency plots

**Points:** number of records used in constructing the two variable PDP

**Centered:** center the plots (the button is currently selected so the plots will be centered)

**Number of existing plots:** number of plots currently available view (you can specify that TreeNet create partial dependency plots automatically after TreeNet finishes its run; see the [Plots & Options Tab](#) section of this document)

**Clear Existing:** remove all plots available to be viewed

Create Plots

Data file: C:\Program Files\Salford Systems\Salford Predictive Modeler 8.0\Sample Data\sample

Grove file: TreeNet Output 5 - Y1 Type: Regression

Select Number Of Trees To Create Plots For

Training Sample  
Nr. of trees: 145 Entropy: Misclass: LAD: 0.287 LS: 0.135 Optimal Tree

Test Sample  
Sample: Entropy: Misclass: LAD: 0.305 LS: 0.148 Select...

Select Types  
 One variable dependence  Two variable dependence  
500 points 5000 points Centered

Select Variables To Plot

Variable	Score
X1	100.00
Z2\$	62.36
Z1\$	50.29
X2	46.06
X3	43.33
X4	35.80
X9	5.35
X7	4.68
X8	4.38
X10	4.00

Use top 3 variables

Including 5 interacting

Importance Interactions

Select Select All

Remove Remove All

Number of plots that may be created: 15

Number of existing plots: 78 Clear Existing

Cancel Create Plots

**Select Variables To Plot:** click any variable to generate plots for (hold the CTRL key or the SHIFT key if selecting multiple variables) and then click the **Select button** to add these variables to the list of variables for which plots are to be generated

**Use Top \_\_\_ Variables:** check the checkbox and specify the top number of variables to see PDPs for. The variable list (**blue rectangle above**) can be sorted by either the (relative) variable importance score or a variable's whole variable interaction score depending on whether you click the **Importance button** or the **Interactions button** (see the [Whole Variable Interaction Score Report](#) section and/or the [Variable Importance Tab](#) section for more details on the computations of these values).

**Including \_\_\_ interacting:** check the checkbox and specify the top number of variables based on their global interaction score.

**Importance:** selected in the picture; sort the variables by their relative variable importance score (see the [Variable Importance Tab](#) section for more information)

**Interactions:** not selected in the picture; sort the variables by their "Global Score" value (see the [Interpreting the 2-Way Interaction Report](#) section for more information)

**Select:** highlight the desired variables on the left (blue rectangle above) and click "**Select**". The variables will appear in **the box on the right**.

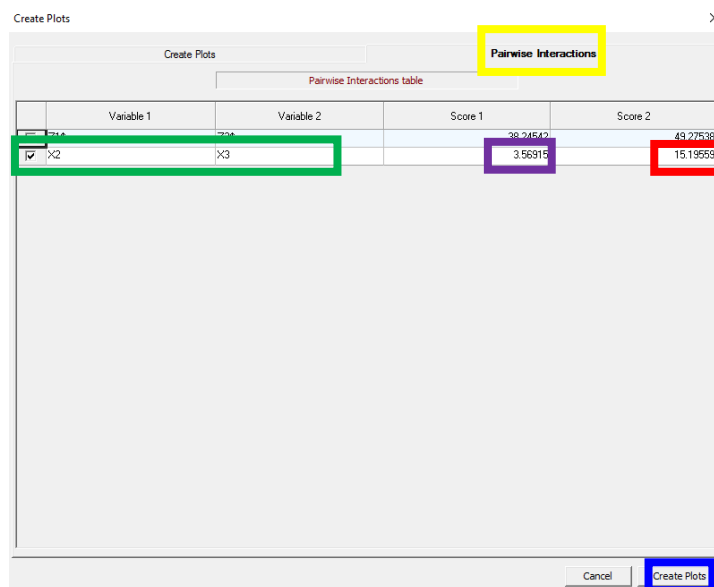
**Select All:** all variables will appear in **the box on the right**

**Remove:** click a variable name in **the box on the right** and then click the **Remove button** to remove the variable

**Remove All:** all variables will be removed from **the box on the right**

## Pairwise Interactions Tab

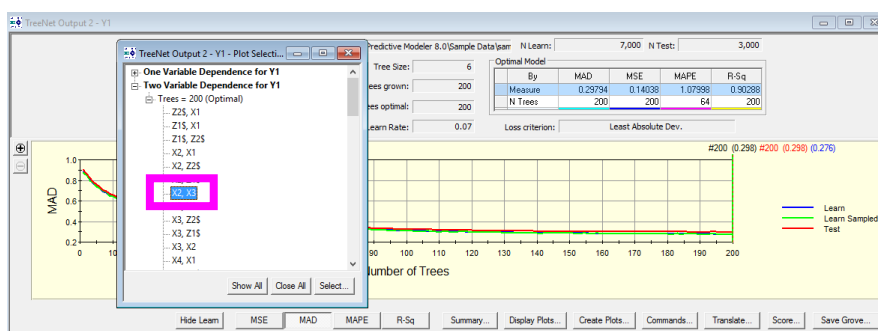
The Pairwise Interactions Tab allows user to directly request two-variable partial dependency plots based on the interaction scores.



**Score Global** (on the % scale) shows the contribution of the suspected interacting pair normalized to the overall response surface (total variation of the predicted response). A value of 3.56915 means that **the interaction between X2 and X3 accounts for 3.56915% of the total variation in the predicted response**. Note: a value of zero means that the interaction under consideration has no effect on the overall model.

**Score Local** (on the % scale) shows the contribution of the suspected interacting pair of variables normalized to the two-variable partial dependency function of the pair of variables under consideration (two variable partial dependency functions are used to generate the two-variable partial dependency plots). A value of 15.19559 means that **15.19559% of the variation in the two-variable partial dependency plot for X2 and X3 is explained by the interaction between X2 and X3**. Note: a value of zero means that there is no interaction between the two variables under consideration (see the [Interpreting the 2-way Interaction Report section](#) for more information on Score Global and Score Local)

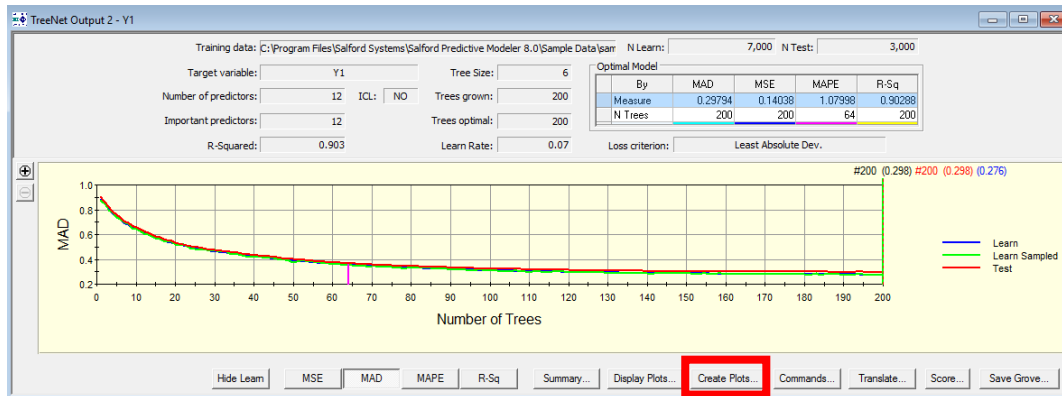
To generate two-variable partial dependency plots **click the checkbox(es) corresponding to the desired pair of variables** and then **click the Create Plots button**. Upon clicking the **Create Plots button** the plot will be constructed (if you request a large number of plots then this may take a while) and you will be shown the following menu. Click the plus signs and double click on the pair of variables that you requested the plot for (in this case X2 and X2: **pink rectangle below**). After double clicking you will be shown the plot.



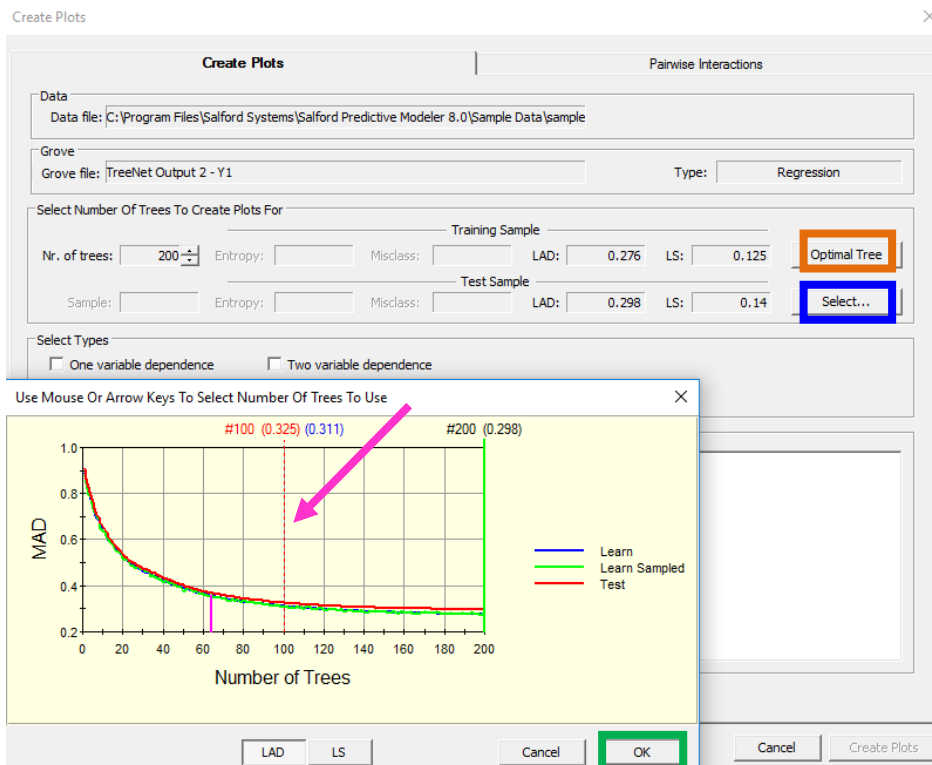
**Example: Top Interacting Plots for the 100 Tree Model**

Let's assume you have built a TreeNet model and you want to view 2 partial dependency plots for the top 5 interacting variables for the TreeNet model with 100 trees.

Click the **Create Pots... button** below:

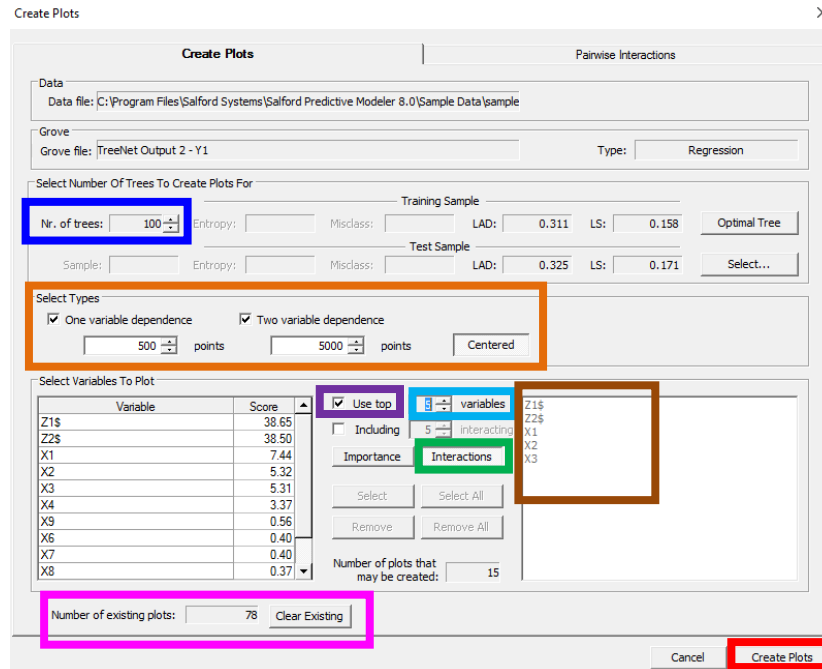


You will see the following menu. Click the **Select... button** to see the error curve below. Click anywhere along the curve that you wish to see Partial Dependency Plots (PDPs) for and then click the **OK button**. In the picture below, the number of trees is set to 100 (see the **pink arrow** pointing to the red dotted line). To view PDPs for the optimal tree then click the **Optimal Tree button**.



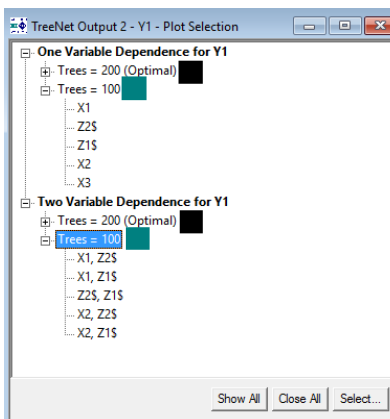
After you click the OK button you will see that the **Nr. of trees has been set to 100**. To view the one-variable dependency plots and two-variable dependency plots **click the desired checkboxes**.





Click the **Interactions** button and then click the **Use top** checkbox and **set the number of variables to 5**. You will see **5 variables appear in the right panel**. Click the **Create Plots** button to build the partial dependency plots. Notice that **78 plots currently exist** so you will have the option of viewing the plots specified here in addition to the plots already created.

After you click the **Create Plots** button the requested plots will be constructed and you will see the following menu (\*\*Note: requesting a large number of plots may take a while for them to be constructed):



Click the plus signs + to expand the options and notice that we can view plots for two models: the optimal model with 200 trees (**black square above**; this was created automatically after the TreeNet run) and **the 100 tree model** (this is what we specified in the steps earlier in this example). To view an individual plot, double click the variable name in the desired section. To view all of the one or two variable plots for the 100 tree model double click the “Tree=100” label (**highlighted in blue; next to the green square**) or “Trees=200 (Optimal)” label.

This is the result of clicking “Trees=100” in the two variable dependence section:



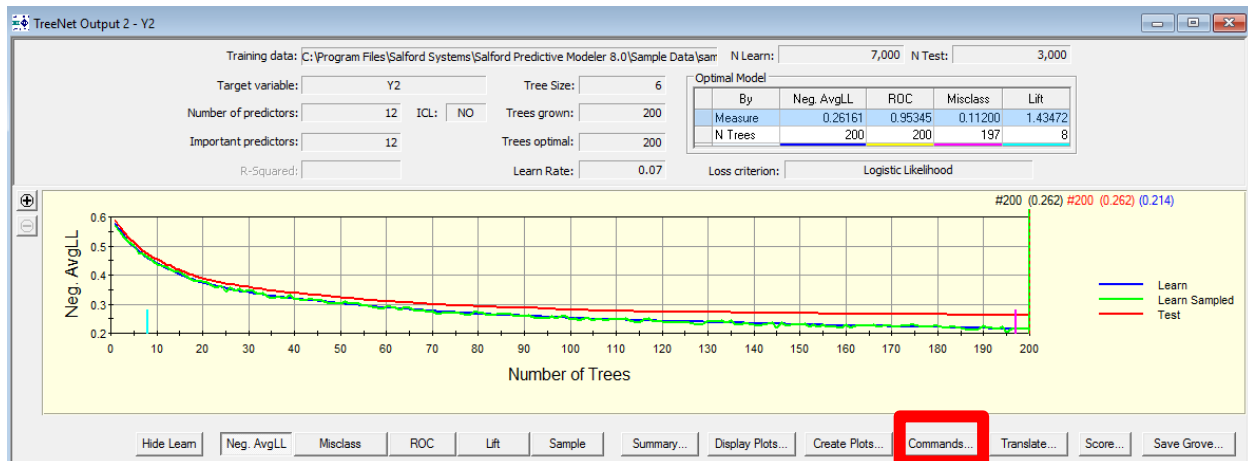
These are the two variable partial dependency plots for the 100 tree model.

## Viewing Results for Non-optimal Trees

The optimal model(s) are always accompanied by a comprehensive set of diagnostics, reports and graphs. In addition, TreeNet includes complete reports for select other models and partial reports for all models. In the case of regression, complete reports are always available for two potentially different models: the model with the number of trees yielding the optimal Least Squares, and the model containing the number of trees yielding the optimal LAD. A partial model summary is computed for all models but contains only core performance measures and the variable importance ranking. You may select any model by pressing the <Ctrl> + arrow keys.

- ✓ It is possible to increase the number of models with full summaries in the TN Advanced tab of the Model Setup dialog.
- ☛ Requesting too many complete summaries might result in a significant program slowdown and may cause memory problems. Remember that each model might require 50 to several hundred graphs and if you save them all for 2000 models you can easily bog down your machine.

## Commands



Every time you click a setting in SPM a corresponding command is created. The commands allow you to create scripts that you can run in SPM perform a variety of tasks including building models, preparing data, performing experiments, and more. Clicking the **Commands...** button above will show the commands used to generate the TreeNet model as well as the commands leading up to the creation of the model like reading in data, computing summary statistics, and the creation of past models:

```

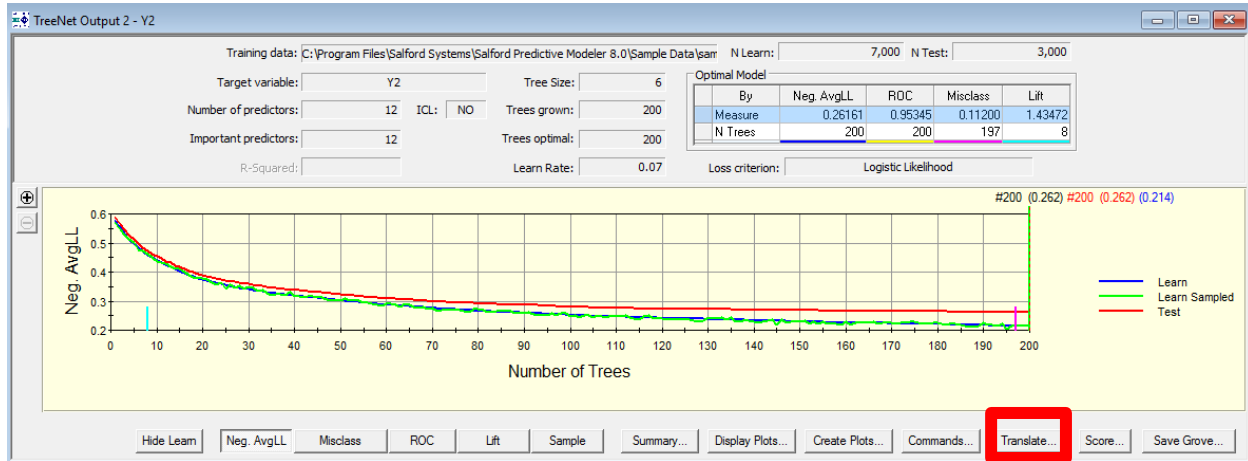
Command File - TreeNet Output 1 - Y2

FORMAT = 5
DISCRETE MAX = 1000,1000
REM ***Setting CART default options
LOPTIONS, NOPRINT = NO, PS = NO
BOPTIONS SURROGATES = 5 PRINT = 5, COMPETITORS = 5 CPRINT = 5, TREELIST = 10,
  BRIEF
SEED 13579, 12345, 131, NORETAIN
RF SEED 17395
THREADS = 4
REM ***Setting MARS default options
BOPTIONS PENALTY = 0.000000, SPEED = 4, INTERACTIONS = 1, MINSPAN = 0, BASIS = 15
MARS SEED = 987654321
BOPTIONS OLS = YES
PRINT = SHORT
USE "C:\Program Files\Salford Systems\Salford Predictive Modeler 8.0\Sample Data\sample.csv"
REM ***Setting General options
LOPTIONS MEANS = NO, PREDICTION_SUCCESS = NO, TIMING = YES, GAINS = NO, ROC = NO, PLOTS = NO
FORMAT = 5
DISCRETE MAX = 1000,1000
REM ***Setting CART options
LOPTIONS, NOPRINT = NO, PS = NO
BOPTIONS SURROGATES = 5 PRINT = 5, COMPETITORS = 5 CPRINT = 5, TREELIST = 10,
  BRIEF
REM ***Setting MARS default options
BOPTIONS PENALTY = 0.000000, SPEED = 4, INTERACTIONS = 1, MINSPAN = 0, BASIS = 15
MARS SEED = 987654321
BOPTIONS OLS = YES
PRINT = SHORT
CATEGORY
AUXILIARY
MODEL Y2
KEEP
KEEP Z16, Z26, X1, X2, X3, X4, X5, X6, X7, X8, X9, X10
LOPTIONS UNS = NO
CATEGORY Y2
TREENET GO

```

## Translating TreeNet Models

You have the option of translating TreeNet models into 4 languages: C, PMML, Java, and SAS compatible code. You can also output the classic output contents to a file, the history, the rules in the TreeNet model, or plot data.



Click the **Translate...** button to open the Model Translation menu:

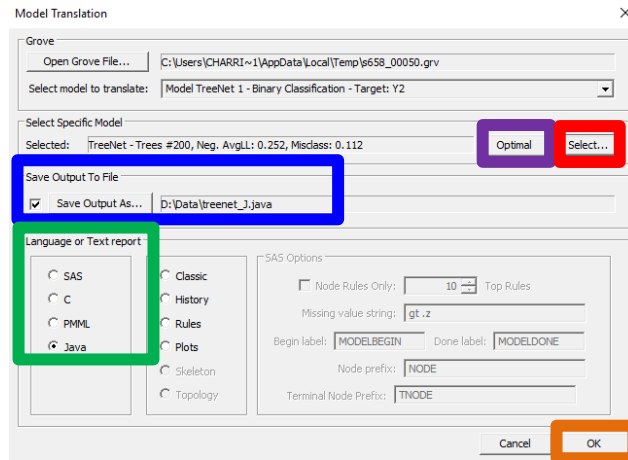
The 'Model Translation' dialog box is shown with the following settings:

- Grove: C:\Users\CHARRI~1\AppData\Local\Temp\p658\_00050.grv
- Select model to translate: Model TreeNet 1 - Binary Classification - Target: Y2
- Select Specific Model: Selected: TreeNet - Trees #200, Neg. AvgLL: 0.252, Misclass: 0.112
- Save Output To File:  Save Output As...
- Language or Text report:
  - SAS
  - C
  - PMML
  - Java
  - Classic
  - History
  - Rules
  - Plots
  - Skeleton
  - Topology
- SAS Options:
  - Node Rules Only: 10 Top Rules
  - Missing value string: gt.z
  - Begin label: MODELBEGIN Done label: MODELDONE
  - Node prefix: NODE
  - Terminal Node Prefix: TNODE

Buttons: Cancel, OK

### Example: Translating the Optimal TreeNet into Java Code

1. Click the **Java radio button**
2. Click the **Save Output As...** checkbox and specify a file path to save the translation code.
3. Click the **Optimal button** if you want a translation for the model with the optimal number of trees; otherwise click the **Select... button** and select the number of trees desired.
4. Click the **OK button** to translate the model.

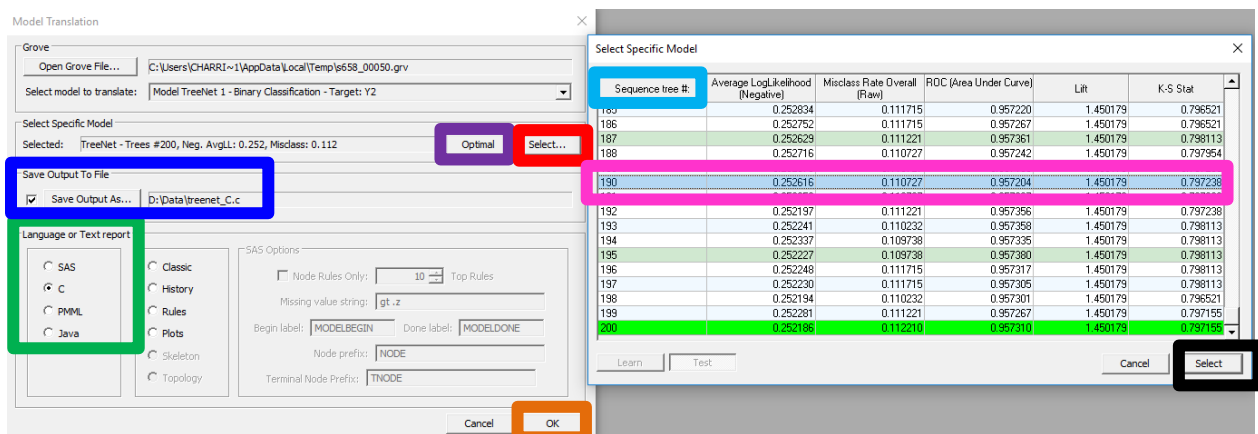


5. Here is a snapshot of the file that is located in the file path defined in Step 2:

```
tree_1 tree_1_obj=new tree_1 ();
tree_2 tree_2_obj=new tree_2 ();
tree_3 tree_3_obj=new tree_3 ();
tree_4 tree_4_obj=new tree_4 ();
tree_5 tree_5_obj=new tree_5 ();
tree_6 tree_6_obj=new tree_6 ();
tree_7 tree_7_obj=new tree_7 ();
tree_8 tree_8_obj=new tree_8 ();
tree_9 tree_9_obj=new tree_9 ();
tree_10 tree_10_obj=new tree_10 ();
tree_11 tree_11_obj=new tree_11 ();
tree_12 tree_12_obj=new tree_12 ();
tree_13 tree_13_obj=new tree_13 ();
tree_14 tree_14_obj=new tree_14 ();
tree_15 tree_15_obj=new tree_15 ();
tree_16 tree_16_obj=new tree_16 ();
tree_17 tree_17_obj=new tree_17 ();
tree_18 tree_18_obj=new tree_18 ();
tree_19 tree_19_obj=new tree_19 ();
tree_20 tree_20_obj=new tree_20 ();
```

## Example: Translating a Non-Optimal Model into C code

1. Click the **C radio button**
2. Click the **Save Output As... checkbox** and specify a file path to save the translation code
3. Click the **Optimal button** if you want a translation for the model with the optimal number of trees; otherwise click the **Select... button** and select the number of trees desired.
  - a. Here we will click the **Select... button** so that we can select a model with a non-optimal number of trees.
  - b. To select a non-optimal number of trees, click the desired row and note that each row corresponds the number of trees in the model (we call this **Sequence tree #**).
  - c. In this example, we will click the **row for the model with 190 trees** and then click the **Select button (black rectangle in the picture below)**.
4. Click the **OK button** to translate the model



5. Here is a snapshot of the file that is located in the file path defined in Step 2. Note that the number of trees is 190 as selected in the step above.

```

/* Target: Y2 */
/* N trees: 190 */
/* N target classes: 2 */

double target, net_response = 0.0;
int node, done;
net_response+=INIT;

/*****
/* Class-specific treenets */
*****/

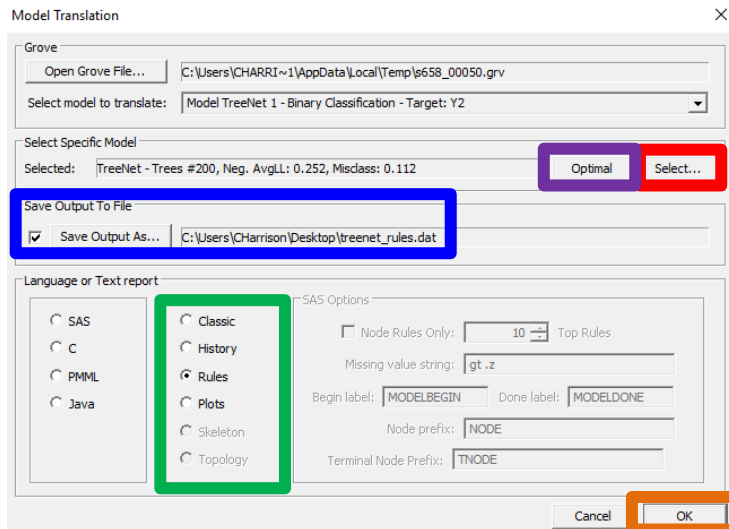
double expsum = 0.0;
double prob0, score0; /* Y2 = 1 */
double prob1, score1; /* Y2 = -1 */

/*****
/* The following predictors had no missing data in

```

## Example: Translating the Rules to a .dat file

1. Click the **Rules** radio button.
2. Click the **Save Output As...** checkbox and specify a file path to save the translation code.
3. Click the **Optimal** button if you want a translation for the model with the optimal number of trees; otherwise click the **Select...** button and select the number of trees desired.
4. Click the **OK** button to translate the model.



5. Here is a snapshot of the file that is located in the file path defined in Step 2.

```

/* Rule 1: response = 0.25846, Tree = 1, Depth = 2, Node 2 */
tree_1_node_2 = (Z2 = "ZERO");

/* Rule 2: response = -0.30559, Tree = 1, Depth = 3, Node 3 */
tree_1_node_3 = (Z2 = "ZERO") and (X1 < -0.136451542377 and not
missing(X1));

/* Rule 3: response = 0.43457, Tree = 1, Depth = 4, Terminal
Node 1 */
tree_1_tnode_1 = (Z2 = "ZERO") and (X1 < -0.136451542377 and not
missing(X1)) and (X1 < -0.761756956577 and not missing(X1));

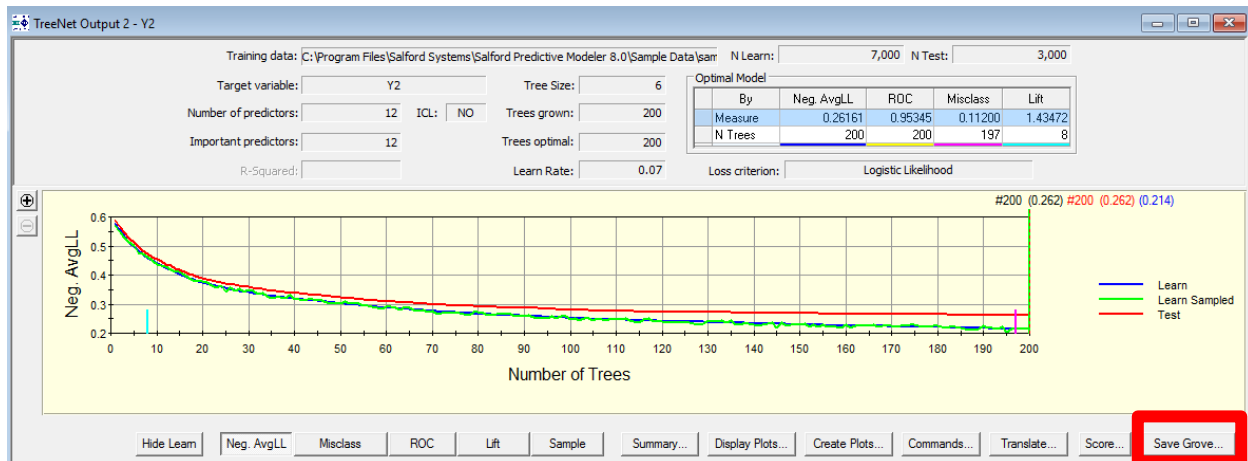
/* Rule 4: response = 0.35488, Tree = 1, Depth = 4, Terminal
Node 2 */
tree_1_tnode_2 = (Z2 = "ZERO") and (X1 < -0.136451542377 and not
missing(X1)) and (X1 >= -0.761756956577 or missing(X1));

/* Rule 5: response = 0.50740, Tree = 1, Depth = 3, Node 4 */
tree_1_node_4 = (Z2 = "ZERO") and (X1 >= -0.136451542377 or
missing(X1));

/* Rule 6: response = 0.49568, Tree = 1, Depth = 4, Terminal

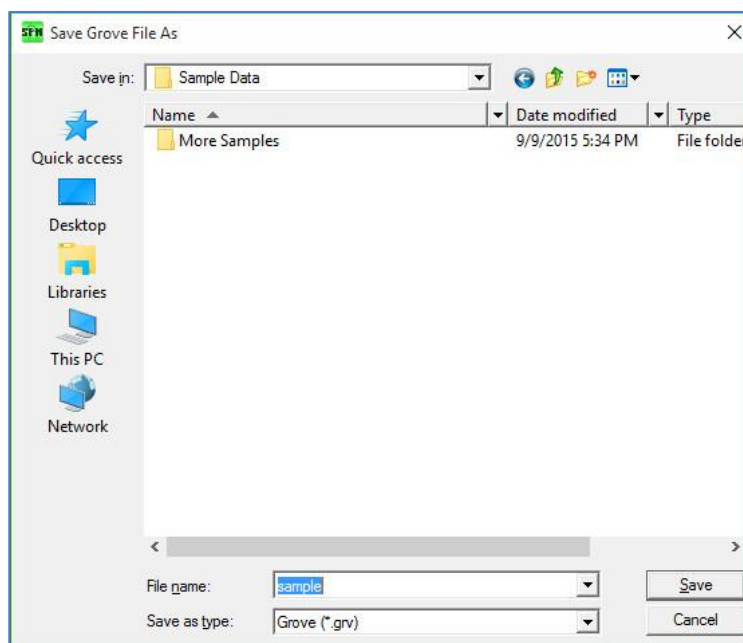
```

## Saving TreeNet Models: Grove File



Any time during model setup, you can use the **Save Grove...** button to save the currently active TreeNet model to a grove file. The grove file, a binary file that contains all the information about the TreeNet model, must be saved if you want to apply the model to a new data set or translate the model into one of the supported languages at a later time.

After the **Save Grove...** button is pressed, the **Specify Grove File** dialog window appears:



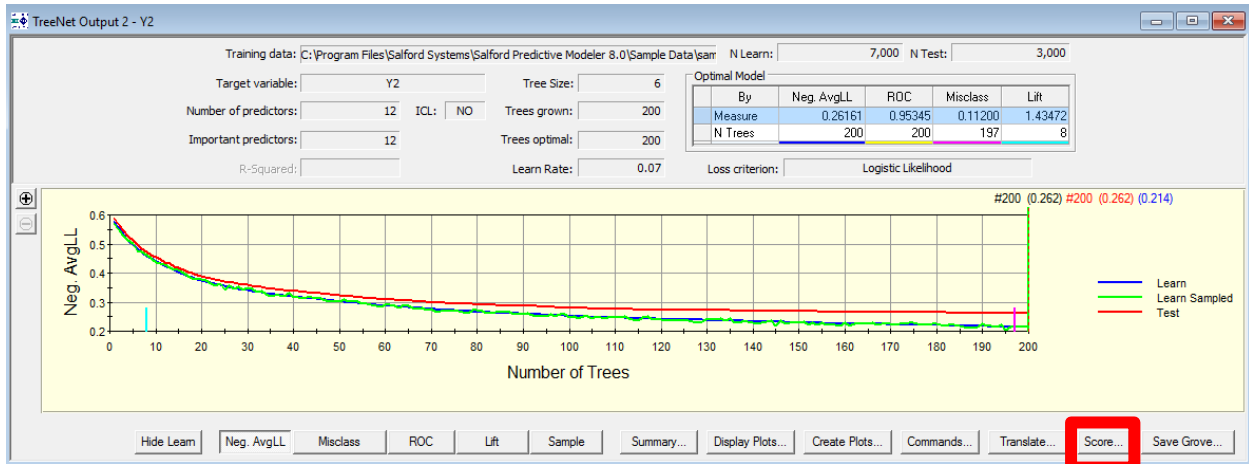
Type in the name of the grove file to be created and click the **[Save]** button.

- ☛ The grove file is created on your hard drive only after the model-building process is completely finished. You cannot create more plots using interactive GUI controls after you save the model.
- ✓ You can also save the grove file from the Results window (see below) after the run is finished.



### Scoring TreeNet Models (Generate Predictions)

Scoring refers to the generation of predictions for a specific model on new data or data used to build the model.



You can score a TreeNet model directly from an open grove (TreeNet Output window) by clicking the **Score... button**. After clicking the **Score... button** you will see the Score Data menu:

Score Data

General | Select Cases | Advanced

Data: Open Data File... C:\Program Files\Salford Systems\Salford Predictive Modeler 8.0\Sample Data\sample.csv

Grove: Open Grove File... C:\Users\CHARRI~1\AppData\Local\Temp\s55o\_00050.grv

Select model to score: Model TreeNet 1 - Regression - Target: Y1

Save score results

Save Scores As...  Save all model related values  Save all variables on input dataset to score dataset

Save Grove As...

Select Specific Model

Selected: TreeNet - Trees #198, RMSE: 0.363, MSE: 0.132, MAD: 0.289 [Optimal] [Select...]

Target, Weight and ID Variables

Target Variable: [Select] [Clear]

Weight Variable: [Select] [Clear]

Treatment Variable: [Select] [Clear]

Sort: Alphabetically [Select] [Remove]

Buttons: Cancel, Continue, Score

## General Tab

Click the **Open Data File...** button and specify the dataset that you would like to generate predictions for.

Click the **Open Grove File...** button if you want to generate predictions using a previously saved model.

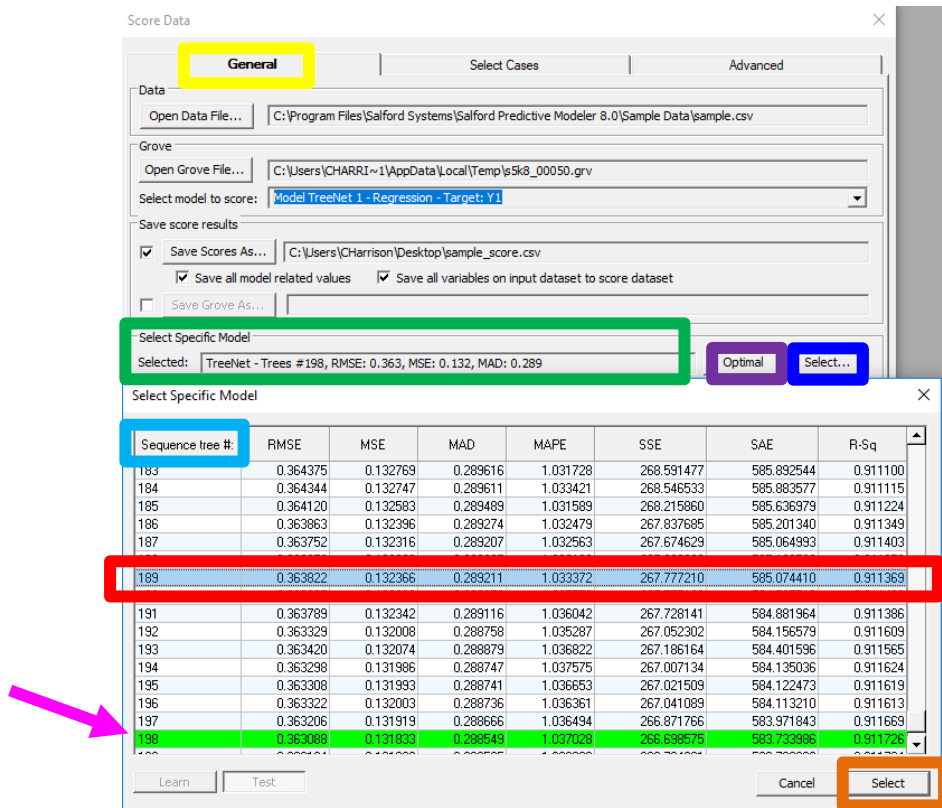
Click the **Select model to score dropdown box** to select the model you want to use to generate predictions (only applicable if the model is currently open in SPM)

Click the **Save Score As...** checkbox to specify a save location for the predictions.

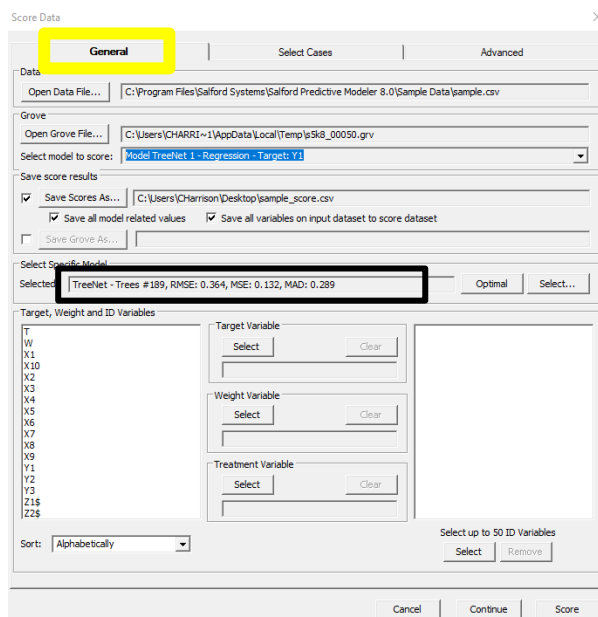
Click the **Save all model related values checkbox** to save all values related to the model.

Click the **Save all Variables on input dataset to score dataset checkbox** if you want to save the predictor variables to the dataset with the predictions.

Click the **Save Grove As...** button if you want to save the recently built model that you want to score.



Click the **Select...** button to select the TreeNet model size (i.e. number of trees which is referred to above as “Sequence tree #”) to be used to generate predictions. The default number of trees is the number that corresponds to the optimal model and is highlighted in green (the pink arrow above points to the highlighted row). To select a model size that is different from the optimal model, click the desired row. In the picture above, the row corresponding to a TreeNet model with 189 trees has been clicked (red rectangle above). Click the **Select** button to finalize this choice. Notice that the model now selected is the one with 189 trees (see the black rectangle in the picture below).



The variables listed on the left (**black rectangle above**) are the variables contained in the dataset that was specified after clicking the **Open Data File...** button.

Click the **target variable name** and then click the **Select button** to set this variable as the target variable. If your dataset does not have a target variable then do nothing (the predictions will still be generated, but you will not see model scoring statistics).

If you have a case weight variable, then click the variable name from the list on the left (**black rectangle above**) and then click the Select button to set this variable as the Weight Variable.

If you have a treatment variable, then click the variable name from the list on the left (**black rectangle above**) and then click the Select button to set this variable as the Treatment Variable.

If you have an ID variable(s), then click the variable name(s) from the list on the left (**black rectangle above**) and then click the Select button to set this variable as the ID variable.

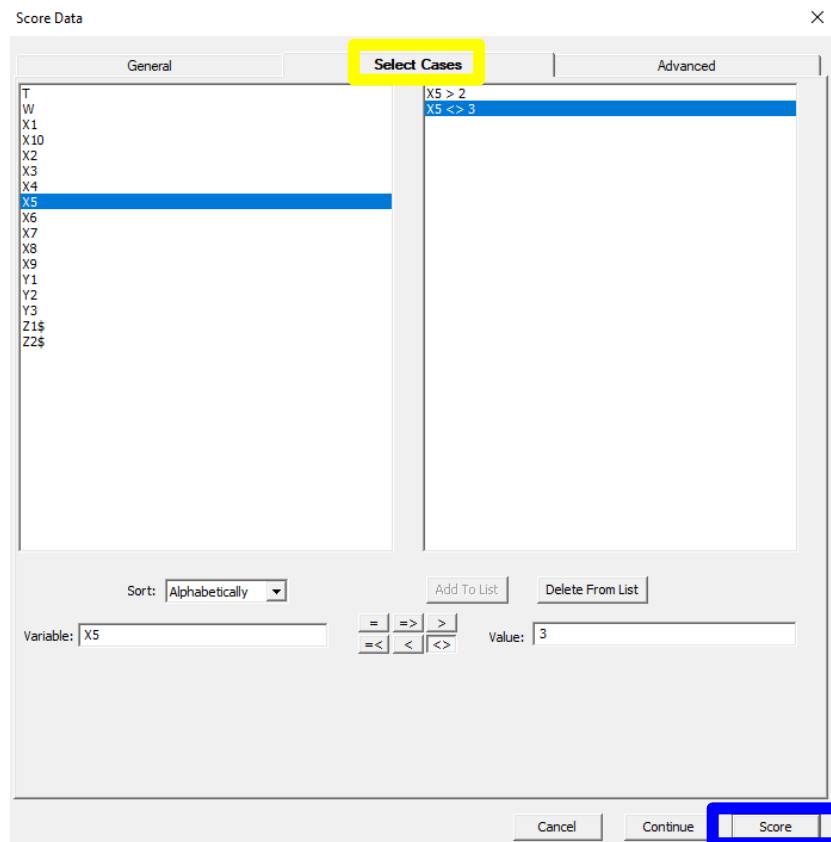
Click the **Cancel button** to exit this menu without saving the settings.

Click the **Continue button** to exit this menu and save the settings.

Click the **Score button** to generate predictions for the dataset specified after clicking the **Open Data File...** button.

## Select Cases Tab

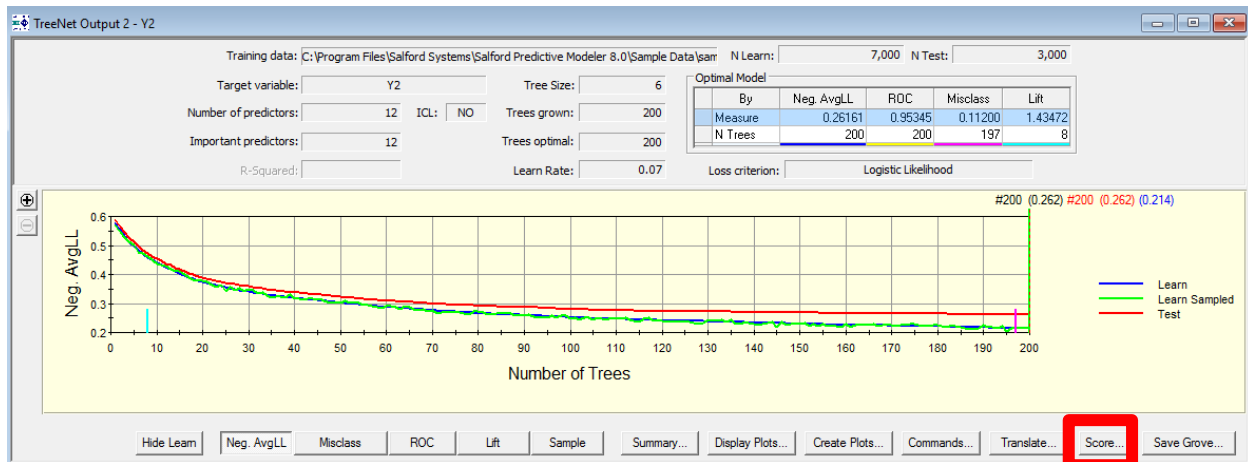
The Select Cases tab allows users to define a filter for the observations that have predictions generated.



In the example above we have defined a filter with two conditions:  $X5 > 2$  and  $X5 \neq 3$  (i.e.  $X5$  is not equal to 3). If you click the **Score button** with this filter defined then the only cases that have a predicted value are those observations whose value for  $X5$  is greater than 2 and  $X5$  is not equal to 3. See the [Select Cases Tab section](#) for more information about the Select Cases functionality.

## Generating Predictions for Learn and Test Partitions

Sometimes it is desirable to see the predicted values for the cases that were used in the Learn sample and Test sample. To accomplish this click the **Score... button**



After you click the **Score... button** you will see the Score Data setup menu:

Score Data

General | Select Cases | Advanced

Data: Open Data File... C:\Users\CHarrison\Desktop\sample.csv

Grove: Open Grove File... C:\Users\CHARRI~1\AppData\Local\Temp\sdtk\_00050.grv

Select model to score: Model TreeNet 1 - Binary Classification - Target: Y2

Save score results:

- Save Scores As... C:\Users\CHarrison\Desktop\Salford Systems\sample\_score.csv
- Save all model related values
- Save all variables on input dataset to score dataset

Select Specific Model: Selected: TreeNet - Trees #200, Neg. AvgLL: 0.262, Misclass: 0.114

Target, Weight and ID Variables:

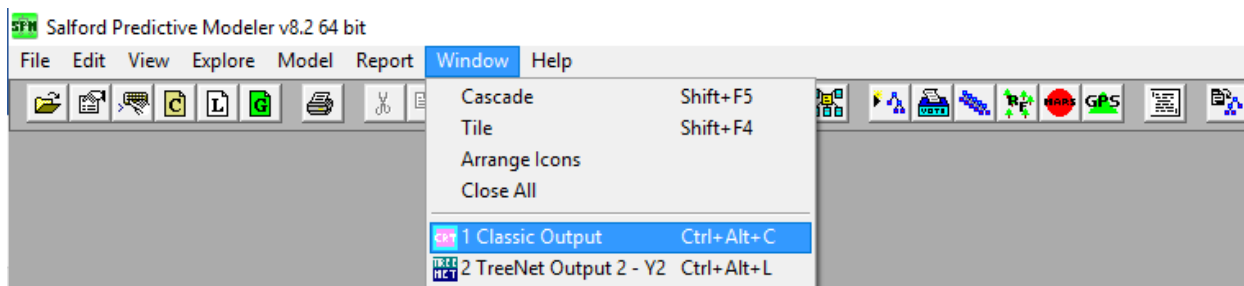
- Target Variable: Y2
- Weight Variable: (empty)
- Treatment Variable: (empty)

Sort: Alphabetically

Buttons: Cancel, **Continue**, Score

To specify the save location for the model predictions by clicking the **Save Score As... checkbox**. Also, click the **Save all model values checkbox** and if you wish to save the predictor variables along with the predictions then click **Save all variables on input dataset to score dataset**. Specify the target variable by clicking Y2 (**purple rectangle below**) and then clicking the **Select button** for the Target Variable section. Click the **Continue button** to save the settings.

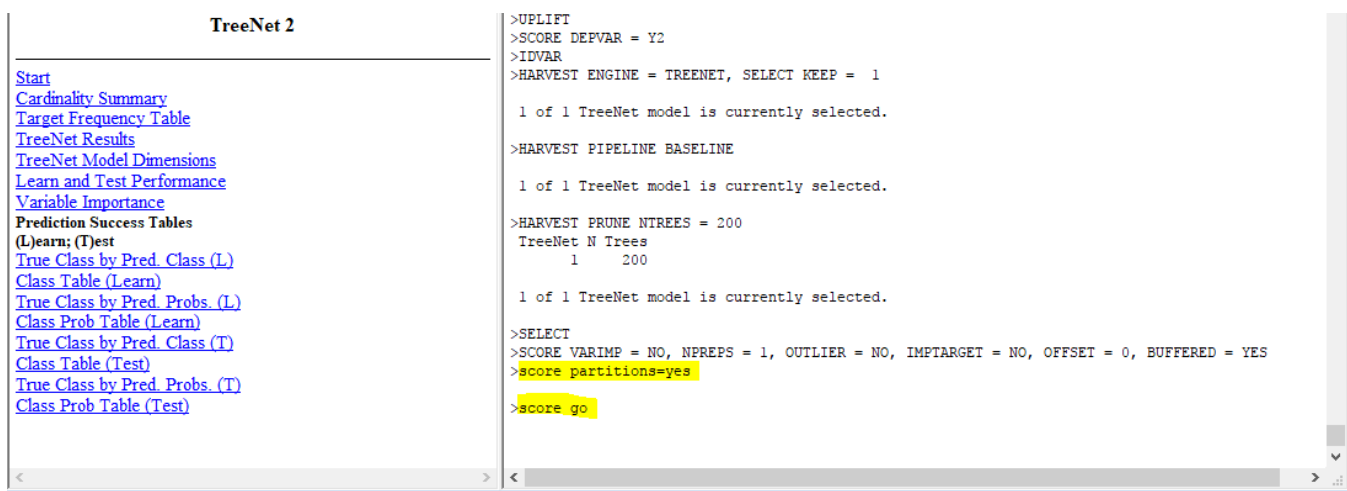
Now we need to tell SPM to preserve the Learn and Test sample partitions used in the model. This is accomplished using the SPM command line. To navigate to the SPM command line click Window > Classic Output Window



After clicking “Classic Output,” the Classic Output Window will appear. Click into the Classic Output Window and type the following two commands (SPM is not case sensitive). Press **ENTER** after typing each command.

```
SCORE PARTITIONS=YES
SCORE GO
```

Your screen will look similar to the following (both commands are highlighted in yellow below)



The `SCORE PARTITIONS=YES` command tells SPM to preserve the learn and test partitions used during the model building whereas the `SCORE GO` command tells SPM to start the scoring process.

After you type the SCORE GO command and press the ENTER key you will see the Score results menu:

Notice that you can view model statistics for both the learn and test sample partitions (red and blue rectangles in the picture, respectively). The predictions are stored in the **Output Scores file path**. Navigate to this location and open this file. Here is a snapshot of the file:

CASEID	SAMPLE	RESPONSE	PROB_NEG_ONE	PROB_ONE	PREDICTION	CORRECT	Y2	Z1	Z2	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
9996	LEARN	-0.016268	0.5081	0.4919	-1	0	1	TWO	ZERO	-0.10	0.20	-0.48	-0.58	1.37	0.06	-0.23	-0.14	1.33	1.74
9997	LEARN	1.269960	0.0731	0.9269	1	1	1	TWO	minusONE	-0.71	-0.34	-0.39	1.60	0.93	-1.10	-1.41	-0.79	-1.64	-1.40
9998	LEARN	1.288810	0.0706	0.9294	1	1	1	THREE	ONE	1.82	1.71	0.05	-0.17	1.13	-0.37	-0.15	0.62	-0.75	0.22
10000	LEARN	-1.253910	0.9247	0.0753	-1	1	-1	ZERO	ZERO	-0.37	-0.78	0.46	-0.65	0.44	-1.81	-0.70	-0.18	-1.02	0.09
4	TEST	-0.116732	0.5581	0.4419	-1	0	1	ONE	ZERO	-0.92	-0.59	-0.19	-0.90	0.24	-0.20	1.22	-0.94	-1.11	-0.41
6	TEST	0.261883	0.3720	0.6280	1	0	-1	THREE	ZERO	-0.95	-0.99	-0.04	0.15	1.31	-0.37	0.43	-0.63	-0.09	0.76
7	TEST	1.088940	0.1018	0.8982	1	1	1	TWO	ONE	1.45	0.81	0.70	-0.27	0.70	1.13	-0.47	-0.05	0.42	1.67
9	TEST	-0.442434	0.7078	0.2922	-1	1	-1	ZERO	ZERO	-0.28	-0.46	0.57	2.07	0.61	-0.80	0.51	0.02	-1.04	-0.84

**CASEID:** an ID variable added by SPM (it is the row number in the original dataset).

**SAMPLE:** specifies if the record was in the LEARN sample or the TEST sample.

**RESPONSE:**  $\frac{1}{2}$  log odds of  $Y2=1$  (remember  $Y2$  is a binary target variable with two values: 1 or -1).

**PROB\_NEG\_ONE:** probability of a -1 (remember we set labels for the target variable in the previous steps; see [Set Class Names for the Target Variable section](#) for more detail).

**PROB\_ONE:** probability of a 1 (remember we set labels for the target variable in the previous steps; see [Set Class Names for the Target Variable section](#) for more detail).

**PREDICTION:** predicted class based on the probability (the default cutoff probability is 50% which is the same as predicting the class that has the larger predicted probability in the case of binary targets).

**CORRECT:** if the predicted class PREDICTION is the same as the value for the original target variable  $Y2$ , then CORRECT = 1 (i.e. the prediction was correct). If the predicted class PREDICTION differs from the value for the original target variable then CORRECT = 0 (i.e. the prediction was incorrect).

**Y2:** the original target variable.



## Generating Predictions Using a Saved Model

Previously saved models can be used to generate predictions for other datasets. As an example, you may build a model, and three months later, you may need to use the model to generate predictions for an entirely new dataset. In this example, we will use a previously saved classification model used to predict the target variable Y2 (see the [Example: Building a Classification/Logistic Binary Model](#) section for more information). The dataset we are going to generate predictions for is called “sample\_unlabeled.csv”:

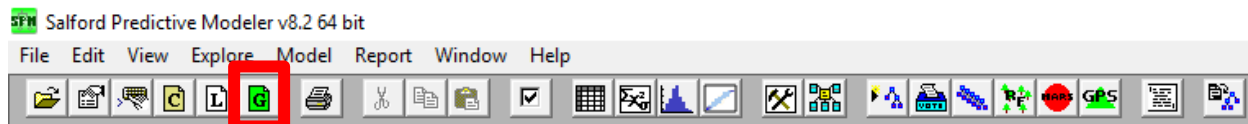
ID	W	Z1\$	Z2\$	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	T
1	1	THREE	ONE	1.770768	1.800651	0.260861	0.031665	0.00409	-0.35627	-1.62307	-0.8572	-0.58907	-0.81788	1
2	1	TWO	ZERO	-0.1008	0.198375	-0.48282	-0.57987	1.374421	0.05805	-0.23008	-0.14364	1.332015	1.739205	1
3	1	TWO	minusONE	-0.7144	-0.33853	-0.39488	1.59934	0.92849	-1.09675	-1.40583	-0.78784	-1.63646	-1.40083	1
4	1	THREE	ONE	1.819314	1.707082	0.046882	-0.17015	1.133787	-0.37498	-0.14705	0.622754	-0.75278	0.218776	1
5	1	THREE	ZERO	-0.12598	-0.71218	0.699241	-0.46433	4.389846	-0.07872	-0.58759	1.083918	0.384606	-0.47347	1
6	2	ZERO	ZERO	-0.37318	-0.78073	0.461346	-0.65427	0.435218	-1.81077	-0.70474	-0.17554	-1.01598	0.093519	1

Notice that there is not a column for the target variable Y2, so we are going to use our model to generate predicted probabilities and predicted classes for all 6 records. There are two primary steps involved in scoring data using a saved model:

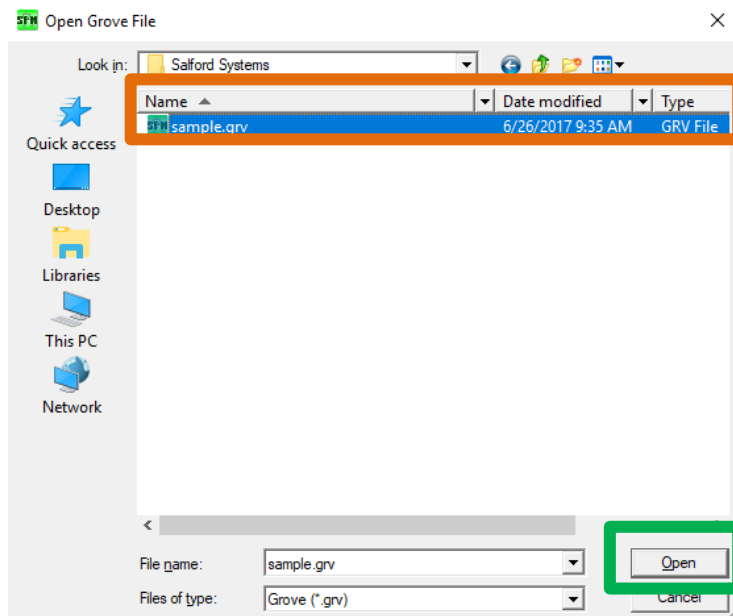
1. Save the model so it can be used later
2. Score the data using the saved model

The steps involved in saving a model are outlined in the [Saving TreeNet Models: Grove File](#) section. Please see this section for more detail. To score data using a saved model:

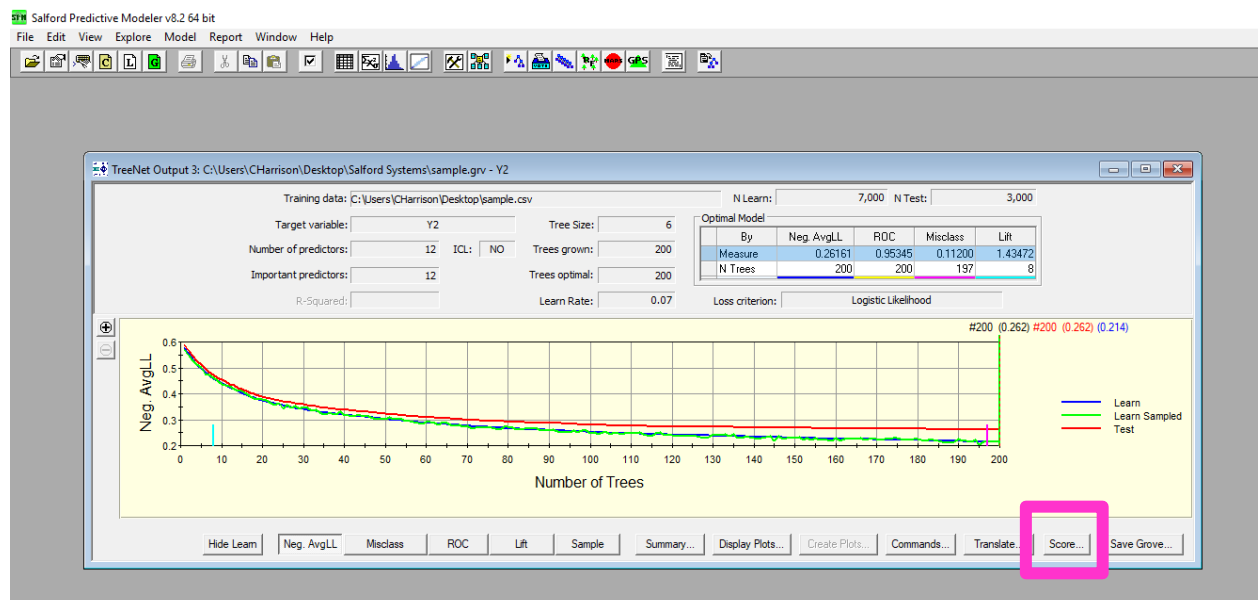
1. Open the saved model in SPM by clicking the “Open Grove” shortcut button **in the red rectangle below** (remember that a “grove” is a just a file that stores models constructed in SPM; see the [Saving TreeNet Models: Grove File](#) section for more information)



Navigate to the saved model (the file extension is .grv), click the file name (**orange rectangle below**) and then click the Open button (**green rectangle below**) to open the saved model in SPM:

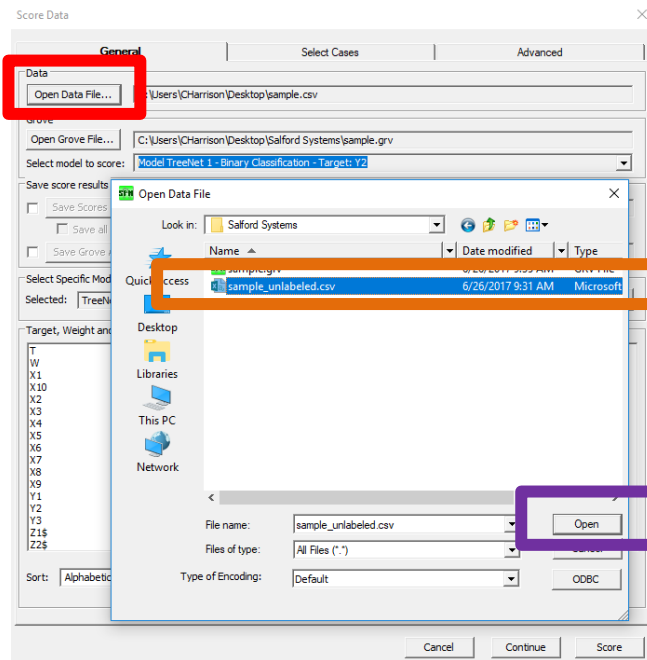


Double click the file name to open the model in SPM and you will see something similar to the following picture. Notice that this is the output window that you see after you construct a TreeNet model.



- To generate predictions for a dataset, click the **Score button in the picture above.**

- Specify the new file that you want to score by clicking the **Open Data File... button** and navigating to the desired file. Once you find the file, click the file name (**orange rectangle below**) and then click the **Open button** (you can also just double click the file name).



4. Specify the save location for the model predictions by clicking the **Save Score As... checkbox**. Also, click the **Save all model values checkbox**. If you wish to save the predictor variables along with the predictions then click **Save all variables on input dataset to score dataset**. Specify the "ID" variable to be an identification variable by clicking ID (**purple rectangle below**) and then clicking the **Select button** for the **ID variable section on the right**. After you click the **Select button** you will see the ID variable name appear in this section. Click the **Score button** to generate predictions for the sampled\_unlabeled.csv dataset.

Score Data

General | Select Cases | Advanced

Data  
Open Data File... C:\Users\CHarrison\Desktop\Salford Systems\sample\_unlabeled.csv

Grove  
Open Grove File... C:\Users\CHarrison\Desktop\Salford Systems\sample.grv

Select model to score: Model TreeNet 1 - Binary Classification - Target: Y2

Save score results

Save Scores As... C:\Users\CHarrison\Desktop\Salford Systems\sample\_unlabeled\_score.csv

Save all model related values

Save all variables on input dataset to score dataset

Save Grove As...

Select Specific Model  
Selected: TreeNet - Trees #200, Neg. AvgLL: 0.262, Misclass: 0.114 Optimal Select...

Target, Weight and ID Variables

ID

W  
X1  
X10  
X2  
X3  
X4  
X5  
X6  
X7  
X8  
X9  
Z1\$  
Z2\$

Target Variable  
Select Clear

Weight Variable  
Select Clear

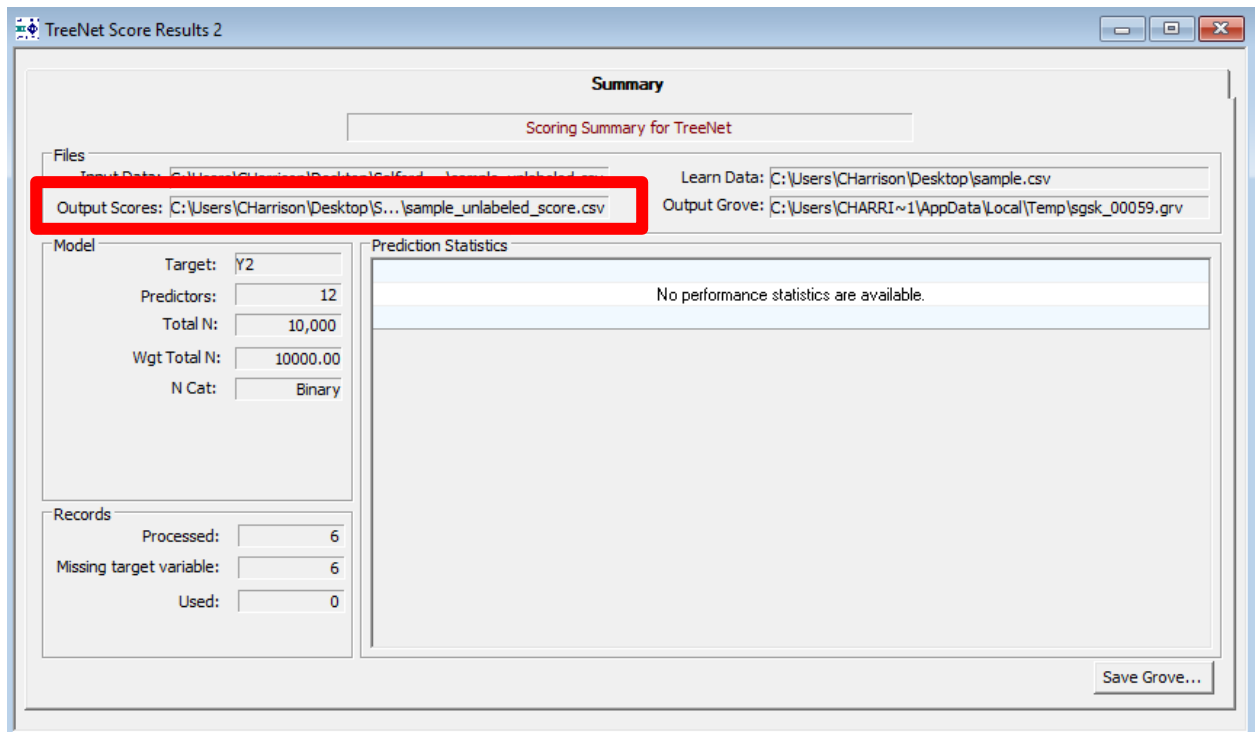
Treatment Variable  
Select Clear

Sort: Alphabetically

Select up to 50 ID Variables  
Select Remove

Cancel Continue Score

5. After the scoring process is complete, you will see something like the following picture. Notice that the file path for the dataset with the predictions is given by the **Output Scores file path**:



6. Here is the file specified in the **Output Scores file path**:

CASEID	RESPONSE	PROB_NEG_ONE	PROB_ONE	PREDICTION	Z1	Z2	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	ID	W	T
1	1.57596	0.0410156	0.958984	1 THREE	ONE		1.77077	1.80065	0.260861	0.031665	0.00409	-0.35627	-1.62307	-0.85721	-0.58907	-0.81789	1	1	1
2	-0.016268	0.508133	0.491867	-1 TWO	ZERO		-0.1008	0.198375	-0.48282	-0.57987	1.37442	0.05805	-0.23008	-0.14364	1.33201	1.7392	2	1	1
3	1.26996	0.0731068	0.926893	1 TWO	minusONE		-0.7144	-0.33853	-0.39488	1.59934	0.92849	-1.09675	-1.40583	-0.78784	-1.63646	-1.40083	3	1	1
4	1.28881	0.0705921	0.929408	1 THREE	ONE		1.81931	1.70708	0.046882	-0.17015	1.13379	-0.37499	-0.14705	0.622754	-0.75278	0.218776	4	1	1
5	-0.0489603	0.524461	0.475539	-1 THREE	ZERO		-0.12598	-0.71218	0.699241	-0.46433	4.38985	-0.07872	-0.58759	1.08392	0.384606	-0.47347	5	1	1
6	-1.25391	0.924689	0.0753113	-1 ZERO	ZERO		-0.37318	-0.78073	0.461346	-0.65427	0.435218	-1.81077	-0.70474	-0.17554	-1.01598	0.093519	6	2	1

**CASEID**: an ID variable created by SPM. The value is the row number (not including the header row).

**RESPONSE**: the predicted  $\frac{1}{2}$  log-odds of the event.

**PROB\_NEG\_ONE**: the predicted probability of the class “Neg One” (remember we set labels for the target variable in the previous steps; see [Set Class Names for the Target Variable section](#) for more detail).

**PROB\_ONE**: the predicted probability of the class “One” (remember we set labels for the target variable in the previous steps; see [Set Class Names for the Target Variable section](#) for more detail).

**PREDICTION**: the predicted class which in this case is either a 1 or -1. The default cutoff probability for class prediction is .5 (this is the same as selecting the predicted class to be the class with the larger predicted probability).

## Rare TreeNet Commands

Most of the engine specific control parameters occur on the TREENET command. Type HELP TREENET in the command line to see a detailed description of every available option. You can also consult extensive command reference.

Here we point out some interesting features available only through the command line and otherwise not directly accessible through the Model Setup window.

```
TREENET FULLREPORT = <YES|NO>
```

Only applies to classification models, turns on additional extensive reporting like confusion matrices, detailed variable importance tables, etc.

```
TREENET GB = <n>
```

Defines the number of bins in gains charts in the text output.

## Using Logistic Calibration on TreeNet Results

TreeNet is a powerful predictive modeling engine to optimize logistic log-likelihood on binary classification tasks. However, best TreeNet practices usually require very small learning rates and large number of trees to reduce the problem of overfitting which is a known issue for any boosting methodology. This may cause the predicted probabilities coming out of an optimal TreeNet model to have a reduced range, this is especially so when area under ROC curve is used for optimal model selection because the criterion itself is invariant under monotone transformations of predicted response, including scale transformation.

To make the proper adjustment to the predicted probability we offer the following approach:

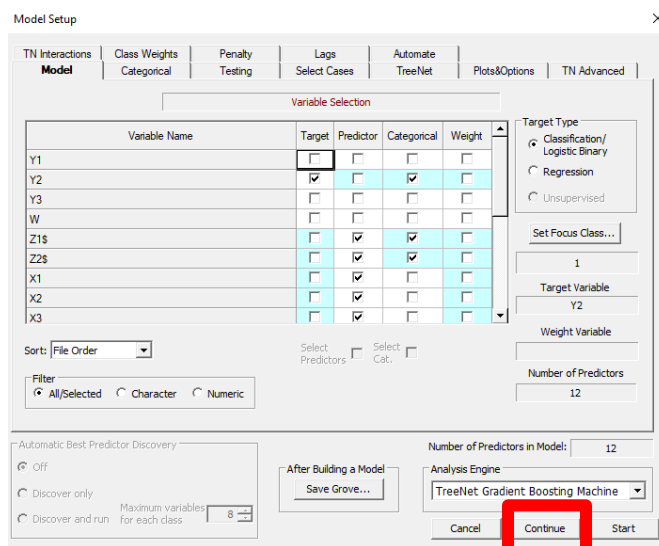
1. Build a binary logistic TreeNet model using either Log-likelihood or ROC-area optimality criterion.
2. Apply this model to the test sample, the output prediction (called RESPONSE) is the half-log-odds of the class in focus.
3. Use the RESPONSE as the sole predictor to build a univariate logistic regression on the test sample.
4. The resulting INTERCEPT and SLOPE coefficients are the final adjustments.

The following command requests test-based calibration of TreeNet raw response for binary logit.

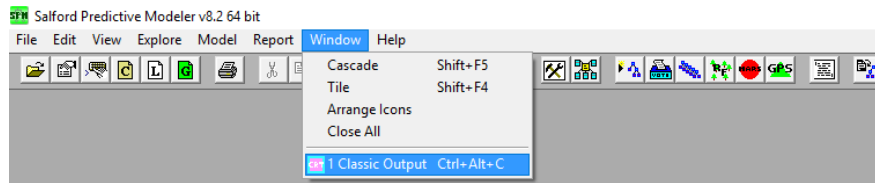
```
📄 TREE NET CALIB=YES
```

The calibrated intercept I and slope S are printed in a separate table for each model size. The calibrated TreeNet response is then  $I + S * (\text{raw response})$ . The calibration option is a command line only feature. To use this feature follow these steps:

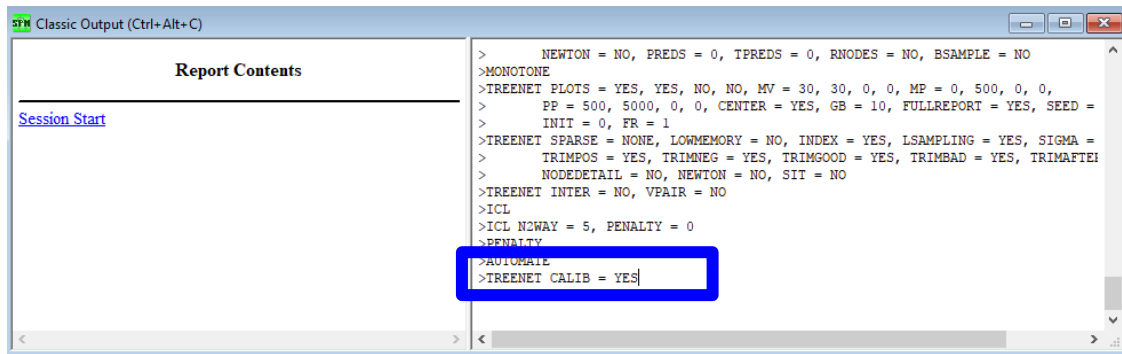
1. Setup your TreeNet model in the GUI as described in the section above (skip this step if using the command line only). Click the **Continue button** to simultaneously exit the model setup and save the model settings:



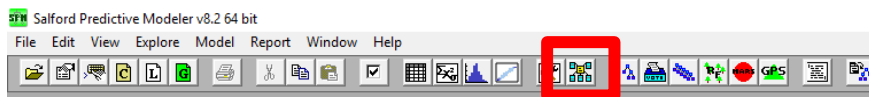
- Now we will tell TreeNet to use logistic calibration. This feature is a command line only feature, so we will first navigate to the command line. To do this click Window > Classic Output Window and the Classic Output Window will appear.



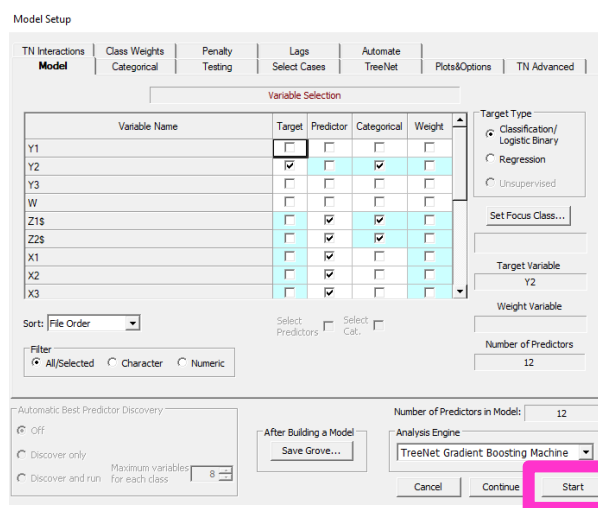
- Type `TREENET CALIB = YES` in the command line (blue rectangle below) and press **ENTER** to submit the command to SPM



- Type `TREENET GO` in the command line and press **ENTER** to run the TreeNet model. Alternatively, click the Model Setup shortcut button (red rectangle below) to go back to TreeNet engine setup.



Clicking this button yields the model setup. To run the model with the logistic calibration, click the **START** button

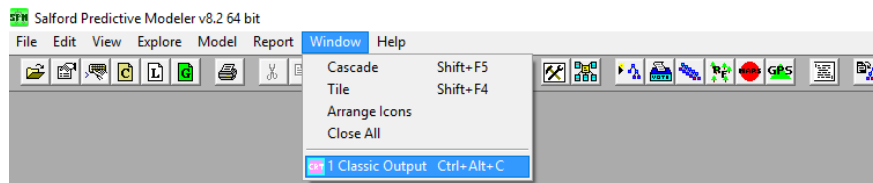




## Viewing the Logistic Calibration Results

The results of the logistic calibration are contained in the Classic Output Window.

1. Navigate to the Classic Output Window by clicking Window > Classic Output Window. The Classic Output Window will appear:



2. Click the **TreeNet Results** link on the left and scroll down to the “LOGIT CALIBRATION” section (green rectangle below)

N-trees	Intercept	Slope
1	-12.6323	30.0603
2	-6.30837	16.3239
3	-4.29419	11.899
4	-3.09069	9.31132
5	-2.49604	8.06901
6	-1.96339	6.86065
7	-1.67764	6.24652
8	-1.52798	5.97221
9	-1.35189	5.62466
10	-1.23181	5.36246
11	-1.07457	4.98861
12	-0.9641	4.78556
13	-0.947305	4.83248
14	-0.859923	4.68822
15	-0.801729	4.56475
16	-0.735804	4.42805
17	-0.654977	4.24727
18	-0.588715	4.07267
19	-0.552657	3.99199
20	-0.537244	4.00854
21	-0.514436	3.93572
22	-0.458079	3.82473
23	-0.446482	3.80651
24	-0.425924	3.78791
25	-0.412493	3.70348
26	-0.392511	3.69719

**Interpretation:** the intercept and slope coefficients are provided for each TreeNet iteration (remember the number of iterations = number of trees because each iteration corresponds to fitting a CART tree to the generalized residuals; in SPM we use “number of trees” instead of “number of iterations”).

- **N-Trees:** number of trees
- **Intercept:** coefficient for the intercept term in the logit model (this is the “I” in the equation provided above)
- **Slope:** coefficient for the raw response term (this is the “S” in the equation provided above)

Ideally, the value for the intercept would be 0 and the value for the slope would be 2 because the only predictor variable is the  $\frac{1}{2}$  log odds of the event and a logit model tries to model the log odds of the event. The actual calibration values indicate the lack of calibration of the original raw RESPONSE.

## Missing Value Handling

TreeNet handles missing values by generating what could be thought of as a 3-way split when using a variable with missing values to grow a tree. First, the records with missing values are moved to their own child node, and the records with good data for the variable move to the other child node. TreeNet is then free to induce a split on the variable using just data with good values.

In order to illustrate this concept, translate your model and view the beginning of the code:

```
/* Root node, NO_1: */
if N_INQUIRIES <= .z then goto NO_5;
  if N_INQUIRIES < 4.5 then goto NO_2;
  else goto NO_5;

NO_2:
if OCCUP_BLANK <= .z then goto TO_4;
  if OCCUP_BLANK < 0.5 then goto NO_3;
  else goto TO_4;

NO_3:
if N_INQUIRIES <= .z then goto NO_4;
  if N_INQUIRIES < 2.5 then goto TO_1;
  else goto NO_4;

TO_1:
  response = -0.41589446;
  goto D0;

NO_4:
if POSTBIN <= .z then goto TO_3;
  if POSTBIN < 3.5 then goto TO_2;
  else goto TO_3;

TO_2:
  response = -0.41744438;
  goto D0;

TO_3:
  response = -0.40304714;
  goto D0;

TO_4:
  response = -0.40031047;
  goto D0;

NO_5:
if POSTBIN <= .z then goto TO_6;
  if POSTBIN < 2.5 then goto TO_5;
  else goto TO_6;

TO_5:
  response = -0.41008216;
  goto D0;

TO_6:
  response = -0.39731206;
  goto D0;
```

The statements in bold check whether the value is missing (`<= .z`) before continuing down the tree. In the root node, if `N_INQUIRIES` is missing, the record is sent to node 5. If not, the condition `N_INQUIRIES < 4.5` is checked.