



Salford Predictive Modeler®

## **Introduction to Random Forests®**

*This guide provides a brief introduction to Random Forests®.*

© 2018 by Minitab Inc. 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, Inc. in the United States and other countries. Additional trademarks of Minitab, Inc. can be found at [www.minitab.com](http://www.minitab.com). All other marks referenced remain the property of their respective owners.

## What is a Random Forest®?

We begin to answer this question with the precise definition below:

*“Random Forest® is a collection of decision trees grown and combined using the computer code written by Leo Breiman for this purpose.*

*No other combination of decision trees may be described as a Random Forest either scientifically or legally. The only commercial version of Random Forests software is distributed by Salford Systems. Random Forests are sometimes also referred to variously as RF, Random Forests, or Random Forest, and all these terms are trademarks of the creators of Random Forests technology, Leo Breiman and Adele Cutler, and licensed exclusively to Salford Systems.*

*This commercial release is developed in collaboration with Leo Breiman and Adele Cutler. It includes a number of proprietary features, extensions, and enhancements developed by Salford Systems, Leo Breiman and Adele Cutler which are exclusive to Salford Systems software.”*

## How Random Forests work and what does a Random Forests model look like?

Random Forests technology, which represents a substantial advance in data mining technology, is based on novel ways of combining information from a number of decision trees. It is inspired by and builds on the earlier work of Breiman, Friedman, Olshen, and Stone on the CART® decision tree and Breiman’s “Bagger” (bootstrap aggregation). In Random Forests we analyze a database by first selecting a target attribute, dependent variable, or database column, as the subject of study. In business applications the subject of study may be the final status of a loan or credit card account (current vs. default), the fraud status of an insurance claim (fraud vs. not), the outcome of a marketing campaign to win a new customer (success vs. failure), or any of a large number of similar topics commonly seen in the business intelligence (BI) and customer relationship management (CRM) literature. In bioinformatics and biomedical research the subject of study is commonly a disease status (healthy vs. diseased, normal vs. abnormal), etc. The analyst proceeds by specifying which columns of data may be legitimately used to develop a predictive model of the target. The list of legitimate predictor columns can be huge and one objective of the analysis may be to reduce this list to a relatively small number of important factors. It is commonplace to find credit risk scoring analyses beginning with hundreds or even thousands of potential predictors and ending up with a core set of 20 or fewer predictors. In genetic DNA microarray data analysis the initial list of predictors can exceed 30,000 while the final list may contain fewer than 10 predictors (genes). It is important to understand that the analyst is not responsible for winnowing the list of predictors as this can be done by Random Forests. The analyst is responsible only for identifying the eligible predictors. This is because the list of eligible predictors is likely to be influenced by subject matter knowledge and business rules or regulatory or other constraints, and only the analyst is likely to be aware of them.

Once the target and the eligible predictors are identified, Random Forests begins by growing a CART-like decision tree. There are several important ways in which this tree differs from a standard CART tree, however. First, we do not make use of all the available training data to grow the tree. Instead we use a bootstrap sample (described in more detail below). A bootstrap sample is similar to a sample that includes only 2/3 of the original training data. Next, we introduce a further essential element of randomness during the growing process: the selection of the variable used to split a node in the tree construction is accomplished partly or wholly at random. An extreme example of this is a tree in which every node splitter has been selected entirely at random. In standard Random Forests practice the trees are not completely random, but chance does influence the selection of every splitter. Finally, the decision tree is grown out to

the largest possible size and then left unpruned.

Growing an overly large decision tree at random and then leaving it unpruned does not appear to be an especially wise strategy for good modeling practice, and in fact it is not. However, if the process is conducted in precisely the right way, and repeated a large number of times, the results of many such trees can be combined to yield powerful predictors and unique insight into data. In fact, the results can be so good they can be superior to non-random methods such as neural networks, solo decision trees, logistic regression, or support vector machines (SVM). How is such remarkable performance possible? As always in high technology fields the devil is in the details and we will describe some of those details here. (Some algorithmic details remain proprietary and are embedded exclusively in the commercial release of Random Forests.)

### **What are the main advantages of Random Forests (RF) as a modeling tool?**

- ✓ Automatic predictor selection from any number of candidates (potentially thousands):
  - The analyst does not need to do any variable selection or data reduction.
  - RF will ultimately identify the best predictors automatically.
- ✓ Ability to handle data without preprocessing:
  - Data do not need to be rescaled, transformed, or modified in any way.
  - Resistance to outliers in any column (predictors or target)
- ✓ Automatic handling of missing values
- ✓ Accuracy:
  - RF models are often considerably more accurate than a single tree.
  - The accuracy achieved is often competitive with the best alternative methods.
- ✓ Resistance to Overtraining:
  - Growing a large number of RF trees does not create a risk of overfitting.
  - Each tree is a completely independent random experiment.
- ✓ Built-in self testing using “Out of Bag” Data:
  - Self testing is based on an extension of cross validation that is repeated several hundred times. Self tests provide highly reliable assessments of the reliability of the RF model.
- ✓ Speed:
  - Trees are grown at high speed because few variables are in use at any one time.
- ✓ RF Cluster identification:
  - RF can be used to generate tree-based clusters that are metric free.
  - Variables defining clusters are chosen automatically and can vary across clusters.
- ✓ Visualization:
  - RF offers novel graphical displays that can yield new insights into data.

## What is the technology underlying Random Forests and how does it differ from other tree-combining methods?

### CART Decision Trees

A conventional decision tree attempts to divide a heterogeneous database into homogenous segments by searching for splitter variables. These are variables that best separate the data into segments so that the target variable values are quite similar within a segment and different between segments. For example, the tree would look for variables that best separate good risks from bad risks, buyers from window shoppers, or normal from abnormal samples. The conventional decision tree accomplishes this task in a stage-wise manner, using one variable at a time. The CART decision tree begins by attempting to divide the entire database into two segments using one variable to accomplish the division. The tree examines every eligible predictor for its splitting power (measured by one of several mathematical criteria) and affects a database partition using the best-performing predictor. A tree is grown by repeating this process, analogous to a game of 20 questions. The database is first partitioned into two segments. Each of the two segments is in turn partitioned into two further segments using the “best” single variable splitter for the segment in question. The best splitter for a node is found by an exhaustive examination of every eligible splitter at that node and every node is searched in the same way. In CART and Random Forests the splitting or subdividing process is repeated until no further split is possible due to lack of data.

### Random Splitting

Random Forests introduces a new wrinkle to this process. It alters the tree-growing process by narrowing its focus during split selection. Instead of considering all eligible predictors in the search for the next splitter, Random Forests searches among a randomly-selected subset of predictors. For example, if the database contains 100 columns usable for prediction, Random Forests would begin by randomly selecting 10 variables and then selecting the best splitter from among that list of 10 predictors. Once the data have been split into two subsets the process would be repeated by splitting each of the two subsets partly at random. To split one of the two subsets Random Forests would select a different set of 10 predictors at random and would use the best of these for further split. Although in Random Forests tree growing we always split data using the best splitter available, we first randomly limit the list of available splitters to a small number. This means that the selected splitter is best only in the limited sense of being best in the randomly-selected list. If the randomly-selected list contains strong predictors then the best splitter is likely to be useful, but if the randomly-selected list contains only irrelevant or weakly predictive variables, then the split will add little to the overall value of the tree. Observe that in a single Random Forests tree any two distinct nodes are likely to be looking at completely different set of variables. If we have 100 eligible variables but only consider 10 in any one node, then the probability of a variable being included in any pair of nodes is  $1/10 * 1/10 = 1/100$ . This means that Random Forests trees are likely to involve a rather large number of different splitters.

How can this procedure yield useful results? Keep in mind that while only a limited subset of variables are examined at any one time, after a number of splits have been generated it is likely that the most relevant variables will have been selected as splitters. In the example of 100 eligible predictors with 10 considered at any one time, after 30 splits it is very probable that 96 of the 100 predictors will have been considered at least once. Thus the relevant variables can appear in the tree, albeit slowly. We now need to ask how this is leveraged into a useful model.

A key insight here is that each of the trees in the Random Forest is a form of dynamically-constructed nearest neighbor classifier. Nearest neighbor classifiers are generally competent performers. While we do not have high expectations of a single tree in the forest we can be confident that the single tree is not in any way misleading. The tree structure ensures that the classification results are at least a faithful though

partial description of the training data. What is also plain is that the single tree will be a rather “weak” predictive model. In isolation the single tree will not be an especially accurate predictor when applied to new data.

### **Voting Trees**

Random Forests gain their strength from the combination of a large number of individually weak models. Since the models are generated by a doubly random process, with random selection of data and partially random selection of predictors, we can be sure of an adequate supply of substantially different trees. These substantially different trees are then combined through a voting or averaging process. For example, if we are trying to predict a “yes/no” outcome and have generated 500 Random Forest trees, then the output of the aggregate model is the number of “yes” votes received and a decision rule can be based on this number. In this case, we may elect to offer loans to anyone receiving more than 400 “yes” votes out of a possible 500. *(Technically, this is an oversimplification but captures the essence of the process. The RF mechanism actually uses a weighted voting scheme.)*

The notion of combining separate models into a “committee of experts” or “ensemble” has been explored in the data mining and machine learning literature for at least 15 years and the field has reached the conclusion that combining models can be highly effective.

### **So what is new about the Random Forests approach?**

The critical point is that in prior methods the individual models are intended to be rather good standalone models. Indeed, in earlier combining technology the mechanism for generating any of the individual models was identical to the process of building the best possible single model. Random Forests is different in that it introduces an entirely new way of generating individual component models. Paradoxically, strong standalone models do not combine effectively into high performance aggregate models. Weaker standalone models tend to combine more effectively!

*There are two reasons for this:*

First, combining or averaging of models does not accomplish significant accuracy improvement if the individual models are too similar to each other. In the extreme case, if each model is an exact copy of every other then there is no benefit at all to averaging. Thus, it should be clear that the models must be different from each other if the combining technology is to yield any benefits. In Breiman’s earlier work on the bagger, he induced differences across models by randomly selecting different subsets of data for the generation of each tree. While that method yielded benefits in terms of accuracy improvement, the benefits were limited by the fact that the individual trees tended to be different in minor details only, while remaining quite similar in their selection of key splitting variables. In Random Forests the two-part randomization process makes the individual models quite different from each other.

A second reason for the benefits of weak standalone models is that by enforcing weakness on the solo trees we practice a form of “slow learning” discussed by Jerome Friedman in his papers on MART and by Steinberg, Golovnya, and Cardell (2003). By generating the trees partly at random each tree uncovers a small fragment of the underlying patterns and by being partly random the tree is prevented from drawing rapid conclusions about what really matters in the process at hand. When new trees are grown, somewhat different aspects of the data structure are illuminated. Because the trees are grown independently of each other with no single tree depending on any other tree for its generation, adding trees does not create a risk of overfitting. Whereas conventional statistical and most data mining methods need to be limited in their complexity to yield satisfactory results, Random Forests can be expanded indefinitely without a performance loss. If an important pattern genuinely exists in the data, portions of it will be uncovered by

different trees and genuine patterns will be detected repeatedly by different trees. In contrast, accidental patterns will not be repeatedly detected and will generally be washed out in the process of averaging results.

### **What is the Random Forests track record?**

Random Forests was first introduced in 1999 by University of California, Berkeley Professor Leo Breiman, one of the authors of CART. The Random Forests technology has been tested in a broad range of industrial and research settings and has demonstrated considerable benefits. Random Forests often reveal insights into data structure not detected by other methods. As the technology is more widely deployed we expect to disseminate real world performance assessments in our publications and newsletter.

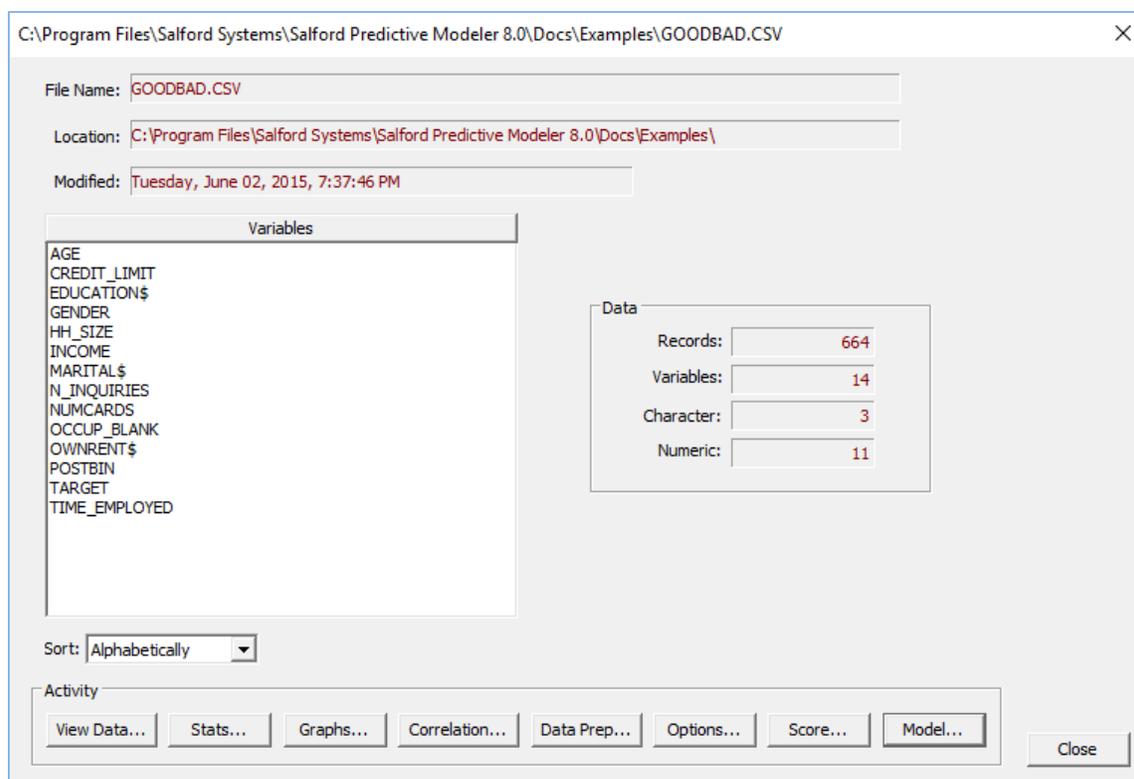
### **How does Random Forests fit into the Salford Systems data mining solution?**

The Salford Systems data mining solution currently rests on four pillars: CART for clear, interpretable classification via decision trees and rules; MARS® for clear, interpretable predictive models via regression and logistic regression, TreeNet/MART for situations in which the analyst may be willing to sacrifice interpretability in favor of very high levels of accuracy, and Random Forests for an alternative technology for combining many trees into a powerful predictive engine. Even in circumstances where interpretability and transparency are mandatory and a model must be expressible in the form of rules, Random Forests can serve a useful function through its visualization tools and its ability to select a few core variables from thousands that might be available.

## Getting Started With Random Forests®

Random Forests is a remarkably flexible learning machine capable of working with both continuous and categorical dependent (target) variables, including multiclass targets. Random Forests is also surprisingly effective with relatively small learning data sets and also with very large numbers of potential predictors. This guide attempts to cover all aspects of the software and therefore includes many sections.

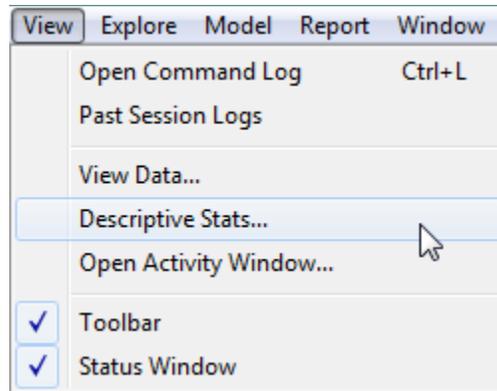
We begin with the GOODBAD.CSV data set with a binary (0/1) target to set up our first analysis. You should be able to locate this file among the examples included with your installation package of SPM®.



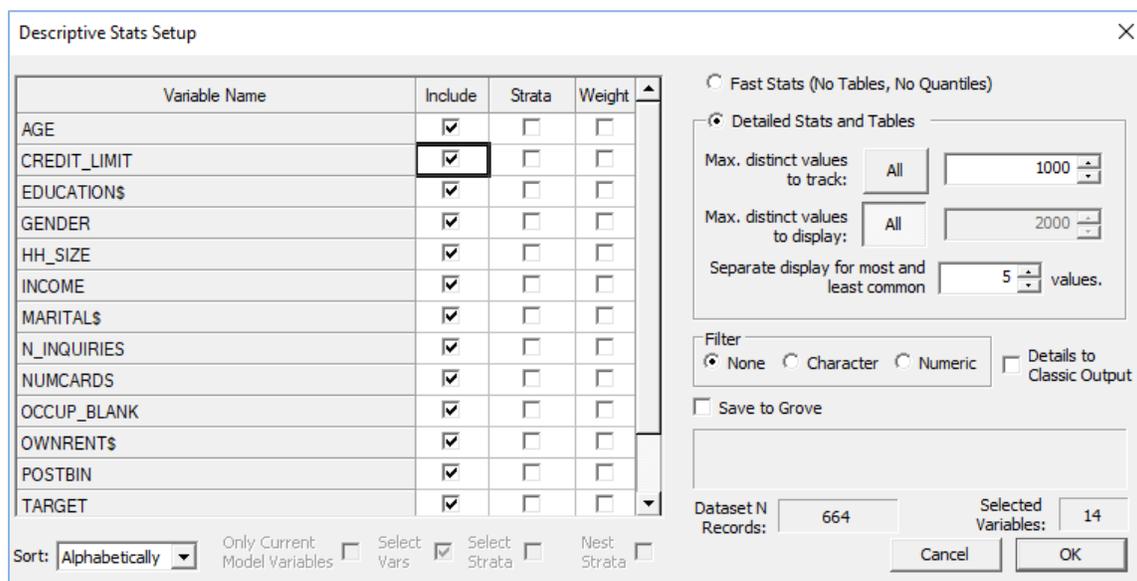
If SPM does not automatically display this “Activity” window when you open a file, you can reach it by clicking on the SPM toolbar. The icon is surrounded by the red square below.



This credit risk dataset has a binary target with TARGET=1 representing a bad outcome, TARGET=0 representing a good outcome, with 203 examples of “bad” and 461 of “good”. We can see this by requesting detailed statistics for the data as illustrated in other parts of this documentation. From the activity window just click on the “Stats” button or from the “View” menu item pull down to reach the “Descriptive Stats” selection.



Here, we decided to also request the classic text tables by checking the “Details to Classic Output” box (also observe the selection of “Detailed Stats and Tables” in the upper right portion of the set up):



This will send a conventional frequency table to the classic output:

TARGET	N	%	Cum %	Wgt	Count	%	Cum %
0	461	69.43	69.43		461	69.43	69.43
1	203	30.57	100.00		203	30.57	100.00
Total	664		100.00		664		100.00

We produce these tables only on request to avoid overly lengthy reports when variables have thousands of possible values.

Model Setup

Class Weights | Penalty | Lags | Automate

**Model** | Categorical | Testing | Select Cases | Random Forests | RF Advanced

Variable Selection

Variable Name	Target	Predictor	Categorical	Weight
MARITAL\$	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
N_INQUIRIES	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
NUMCARDS	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
OCCUP_BLANK	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
OWNRENT\$	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
POSTBIN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
TARGET	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
TIME_EMPLOYED	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Sort:

Filter:  All/Selected  Character  Numeric

Target Type:  Classification/Logistic Binary  Regression  Unsupervised

Set Focus Class...

Target Variable: TARGET

Weight Variable:

Number of Predictors: 13

Automatic Best Predictor Discovery:  Off  Discover only  Discover and run

Maximum variables for each class: 8

After Building a Model:

Analysis Engine: RandomForests Tree Ensembles

Note the selections made on the Model Setup dialog. Choose Random Forests as the Analysis Engine, Classification/Logistic Binary as the Target Type, and TARGET in the Target column.

For testing, we go with the Random Forests default setting of “Out of bag data used for testing”. You might be tempted to consider cross-validation (CV) but for Random Forests, the “Out of Bag” method is essentially a superior version of CV (and hence CV is not even offered).

The screenshot shows the 'Model Setup' dialog box with the 'Testing' tab selected. The 'Random Forests' sub-tab is active. The 'Select Method for Testing' section has 'Out of bag data used for testing' selected. The 'Fraction of cases selected at random' is set to 0.2000, with 'Fast' selected over 'Exact'. 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 13. The 'Analysis Engine' is set to 'RandomForests Tree Ensembles'. A list of variables is visible in a scrollable area: AGE, CREDIT\_LIMIT, EDUCATION\$, GENDER, HH\_SIZE, INCOME, MARITAL\$, and N\_INQUIRIES.

The beauty of the Random Forests methodology is that you genuinely do not need a test partition to obtain reliable estimates of model performance on unseen data, especially important when learn data sets are small.

We visit the Random Forests tab to ensure that key options are set correctly. Here we opt for the default 200 trees, 3 variables selected at random for split consideration at each node and allowing trees to grow to maximum possible size (meaning that nodes containing as few as 2 records are eligible for splitting).

## Random Forests Tab

The screenshot shows the 'Model Setup' dialog box with the 'Random Forests' tab selected. The 'Random Forests Options' section is expanded, showing the following settings:

- Options:**
  - Number of trees to build: 200
  - N predictors: Exactly, 3
  - Frequency of progress reports: 10
  - Number of proximal cases to track: Auto
  - Bootstrap sample size: Auto
  - Parent node minimum cases Regression Recommended Min: 5
  - Permutation Based Variable Importance
  - Random Split Points
- Save Results to Files
  - Save Base Name: [ ]
  - Parallel Coordinates (50) Most Likely in each class (\_parcoor)
  - Multidimensional Scaling Coordinates (\_scaledim; \_outlier)
  - Probabilities And Class Predictions for every record in data (\_pob)
  - Partial Proximity (\_partprox)
  - Full Proximity (\_fullprox)
  - Advanced Missing Value Imputation (\_imputed)
  - Prototypes N: 25 (\_proto)
- Create Full Proximity Matrix
  - If the number of records is less than or equal to: 10000
- Post-processing
  - Suppress all post-processing for Class models EXCEPT varimp
  - Advanced missing value imputation (2) Iterations
- Automatic Best Predictor Discovery
  - Off
  - Discover only
  - Discover and run Maximum variables for each class: 8
- Number of Predictors in Model: 13
- Analysis Engine: RandomForest Tree Ensembles
- Buttons: Save Grove..., Cancel, Continue, Start, Defaults

### Number of trees to build

RF TREES=<n>

This control specifies the number of trees built in the Random Forests model. The default is 200 and the minimum allowable value is 3.

### N predictors

RF PREDs=<x> | SQr | SQr2 | SQr5 | ALL

This control specifies the number of randomly-selected predictors used in each node. The default is 3. You can specify that the engine use either the square root of the number of available predictors, twice the square root, or half the square root. You also have the option to use ALL predictors at each node.

### Bootstrap sample size

RF BOOTSTRAP=<n>

This control specifies the size of the bootstrap sample drawn at each iteration of the model. By default, the engine estimates an optimal size.

AUTOMATE RFBOOTSTRAP automatically varies the bootstrap sample size. See the Automation manual for details.

### Parent node minimum cases

LIMIT ATOM=<n>

Specifies the minimum size below which a node will not be split. By default, ATOM=2 for classification models and ATOM=5 for regression models.

### Permutation Based Variable Importance

RF PMETHOD = <0 | 1>

Classification models only. Clicking this checkbox will result in variable importance measures computed using the permutation method. Leaving this checkbox unchecked will result in variable importance measures that are computed by summing the GINI split improvements for each time a variable is used in the forest. Note that requesting the permutation variable importance will result in longer runtimes.

### Random Split Points

RF RANDOMSPLITS = <YES | NO>

Click this checkbox to have RF choose the split points randomly.

### Post-processing

RF POST=<YES | NO>

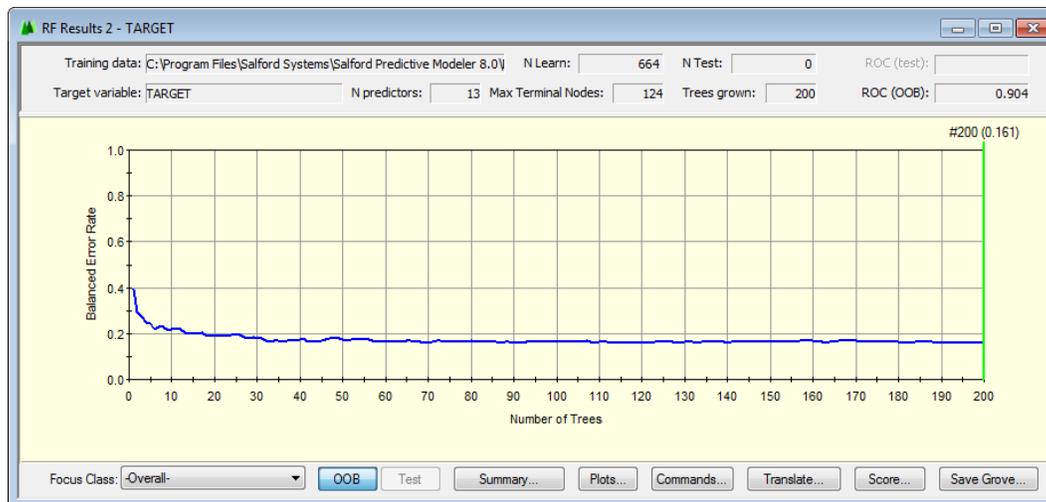
Specifies whether post-processing is turned on for classification models. See later section on post-processing for more details on when to use and expected results.

### Advanced missing value imputation

RF MISSING=<FAST | ADVANCED>

Specifies which of two methods RF should use to impute missing values. FAST uses medians for continuous predictors and modes for discrete predictors. ADVANCED builds ancillary models to determine the best imputation values. ADVANCED requires computation of a proximity matrix and is supported for classification models only.

Clicking the **[Start]** button will initiate the Random Forests analysis. Random Forests trees grow quickly, much faster than legacy CART trees, and you should have the following report in a matter of seconds.



Observe, in the top panel, the size of the largest tree grown, which in this run had 124 nodes. The classic output also reports that average tree size was 88 nodes.

Characterization of tree dimensionality:

```
Smallest:    4 terminal nodes
Largest :   124 terminal nodes
Average :    88 terminal nodes
```

## Post-Processing ON or OFF?

Random Forests generates its trees very quickly but this represents only the first stage of a complete Random Forests analysis. The second, and optional stage, involves considerable post-processing of the learn and test data, and this could take far longer to complete than the first stage. When setting up a Random Forests model you have the option of turning the second stage post-processing on or off.

We recommend that you leave post-processing OFF if...

- You are in the earliest stages of an analysis and you are experimenting with different model set up options looking for the best configuration
- You are principally interested in predictive accuracy and are not concerned with issues such as record clustering
- You are working with very large data sets and expect to be running several to many models and want fast turn-around
- You are content to work with the original CART-style variable importance ranking which does not involve predictor shuffling or post-model sensitivity testing
- Conserving RAM during your run is at a premium since post-processing can use up substantial Windows resources. This will allow you to keep many more SPM windows open during a session.

We recommend that you turn post-processing ON if...

- You have arrived at preferred model configuration and want to explore all possible insights the model might reveal for your data
- You are especially interested in the Random Forests data shuffling method for establishing Variable Importance. This requires post-processing.
- You are working with relatively small data sets and do not mind waiting an extra minute or two for every run to complete
- You want to display the proximities of the learn data records to each other

Post-processing is controlled via the check-box on the Random Forests tab.

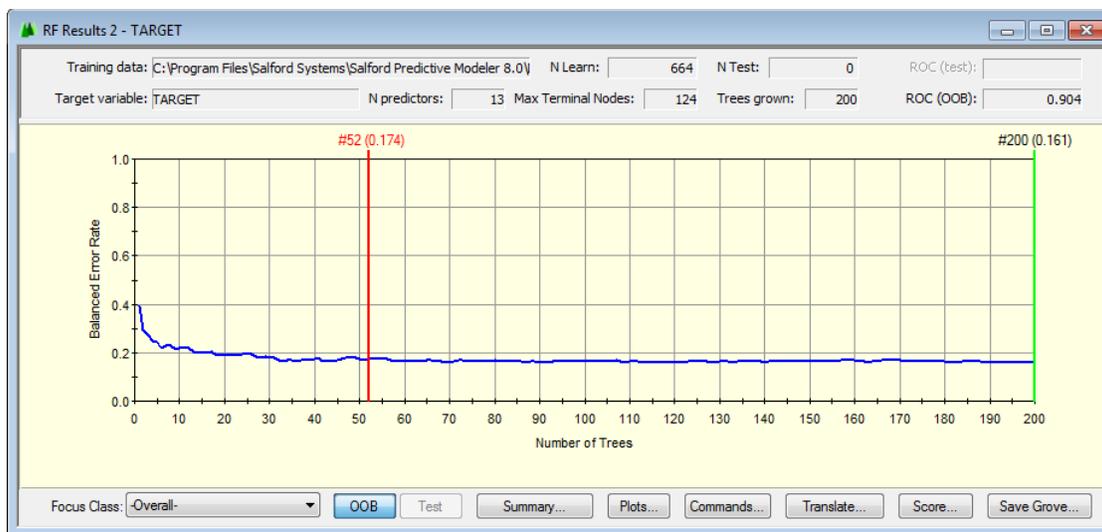
The screenshot shows the 'Model Setup' dialog box with the 'Random Forests' tab selected. The 'Random Forests Options' section is visible, containing various settings for model building. A red box highlights the 'Post-processing' section, which includes the following options:

- Save Results to Files
- Parallel Coordinates (50) Most Likely in each class (\_parcoor)
- Multidimensional Scaling Coordinates (\_scaledim; \_outlier)
- Probabilities And Class Predictions for every record in data (\_oob)
- Partial Proximity (\_partprox)
- Full Proximity (\_fullprox)
- Advanced Missing Value Imputation (\_imputed)
- Prototypes N: 25 (\_proto)
- Create Full Proximity Matrix
- Suppress all post-processing for Class models EXCEPT varimp
- Advanced missing value imputation 2 Iterations

Other visible settings include: Number of trees to build: 200; N predictors: Exactly 3; Frequency of progress reports: 10; Number of proximal cases to track: Auto; Bootstrap sample size: Auto; Parent node minimum cases: 5; Regression Recommended Min: 5; Permutation Based Variable Importance: unchecked; Random Split Points: unchecked; Automatic Best Predictor Discovery: Off; Discover and run: unchecked; Maximum variables for each class: 8; Number of Predictors in Model: 13; Analysis Engine: RandomForest Tree Ensembles.

A crude rule of thumb is that you can expect about  $N/2$  terminal nodes in your largest tree if you have  $N$  records in the data. In this example, the rule of thumb gives an over-estimate, but the sizes of Random Forests trees can vary substantially both within a forest and across data sets of the same size. The rule is good to keep in mind if you are working with larger data sets. A file with 500,000 records can be expected to build trees in the range of 250,000 terminal nodes if you work with the Random Forests defaults. Even if this is an overestimate, the trees can be expected to be huge. Actual tree sizes will be determined by the pattern of the data.

The Results window reports the progress of the forest as it progressively adds more trees. Clicking anywhere on this display will reveal a red beam with the error rate achieved by the forest if it had been stopped at that number of trees. (See the next section for an explanation of the Error rate).



Above, the red beam points to performance measures for the 52-tree forest which is about 8% worse than the full 200-tree forest. While this smaller 52-tree forest may suffice for predictive modeling, it is nowhere near big enough to support the post-forest analytics and data insight reports that we discuss below. Here, we are calling attention to the fact that you can click on the performance graph and use the arrow keys to read the performance of any specific size of forest; we are not recommending that you scale back the number of trees you grow unless the larger forests are taking too long to grow and you do not need to review forest post-processing.

### RF Advanced Tab

**Seed**

 RF SEED = <n>

Sets the RF random number seed to a value of your choosing. The default value is 17395. The value entered must be positive and nonzero. The SEED value will affect the random partitioning of learn and test samples during data preprocessing.

**OOB Record Count Limited to**

 RF SAMPLEAMOUNT = <AUTO | X | N>

A Random Forest model is the result of combining hundreds or even thousands of individual CART trees that are grown on bootstrap samples of the data. The default is “Auto” and results in the usual behavior of RF models. The “Proportion of data” option specifies the portion of the data to be used as out of bag with the value specified by the user to be between 0 and 1. The “Number of records” option sets the size of the OOB sample to be the value specified by the user.

**OOB count applies separately to each class**

 RF SAMPLEBYCLASS = <YES | NO>

Choosing the “Yes” option enforces the OOB Record Count Limited to option (see directly above) for each target class. Choosing the “No” option only enforces the OOB Record Count Limited to option for the learning data as a whole.

**Require that every record is IN BAG at least once in first three trees**

 RF ADJUSTBOOTSTRAP = <YES | NO>

Choosing the “Yes” option will adjust the bootstrap size if it is deemed to be too small relative to the number of trees. If you wish to deliberately use very small bootstrap sizes, then use the “No” option.

**Require that every record is IN BAG at least once in first three trees**

 RF PROXACCUM = <ALL | OOB1 | OOB2>

“Use all trees regardless of OOB status” means that any two records that reach the same terminal node of a given tree may contribute to the proximity matrix. “At least one records of each pair must be OOB” means that at least one of the two records reaching a common terminal node in a given tree be OOB whereas the “Both records in each pair must be OOB” means that both records must be OOB.

**Hierarchical Clustering of Proximity Matrix**

 RF JOIN <YES | NO>

 RF CLUSTERS = <n1, n2, ...>

 RF LINKAGE=< ALL | SINGLE | COMPLETE | CENTROID | AVERAGE | MEDIAN | WARD>

Click the “Hierarchical Clustering of Proximity Matrix” checkbox to request hierarchical clustering on the

RF proximity matrix and then specify the desired number (you can specify more than one value) of clusters. Users can choose from the following linkage methods: Single, Complete, Centroid, Average, Median, or Ward.

## OOB: Out of Bag

This is a fundamental concept for Random Forests and you must understand it in order to properly understand Random Forests results. Random Forests starts with a learn data set but will never use all of the learn data at one time when growing a tree. Instead, Random Forests uses a special form of sampling to create a synthetic learn data sample before growing a tree, and new synthetic learn sample is generated separately and independently for every tree. This means, among other things, that no trees will ever be grown on exactly the same synthetic learn data, although every such sample is constructed from the same master learn data set.

We can think of the synthetic learn sample construction operating as follows:

- ✓ Randomly select 37% of the data to ignore. This portion of the data will not be used at all in the construction of a tree.
- ✓ Randomly assign integer (whole number) weights to remaining 63% of the data so that the sum of weights is equal to the number of records in the master learn data set. The average weight will be about 1.58, but the weights will be the whole numbers 1,2,3,4,5,6,7. (Only rarely will the random weight generation process will produce weights greater than 7).

This sample generation process produces legitimate variations of the learn sample known as “bootstrap” samples.

For our current discussion we want to call attention to the 37% of the data not appearing in the synthetic (bootstrap) sample. Breiman called this data partition the ‘Out of Bag’ or OOB partition and made extensive use of it when evaluating a Random Forests model. To evaluate the predictive accuracy of a forest we start with a specific learn data record and note for which trees in the forest it was OOB. On average any one record will have been OOB for 37% of the trees grown, or about 185 trees of a 500-tree forest. For honest performance assessment we will use only these 185 trees as a record-specific sub-forest to generate predictions (class assignment and predicted probabilities). We will do the same for every record in the learn sample as every record is expected to have been “in bag” for 63% of the samples generated and “out of bag” for the remaining 37%.

- ✓ Note: when evaluating a Random Forests using OOB measures, every record in the learn data is assigned predictions from a different subset of the trees actually grown. This is true only for OOB measurement. When applied to future unseen data we can and will use every tree in the forest.

One reason for growing what might appear to be an overly large forest is that when using OOB measures we leverage only slightly more than 1/3 of the trees for prediction and assessment.

This approach is valid because every Random Forests tree is independently and identically distributed and we rely on averaging results from many draws of these randomly generated objects.

## Random Forests Prediction (Class Assignment)

Any specific Random Forests tree makes its predictions via voting for a single class. In other words, no single tree is able to generate a probability; instead, each tree (actually each terminal node) commits to a specific class. In our example, the vote will be for class 0 or class 1. There are several reasons for this but the most critical is that Random Forests trees are intended to be grown to their maximum possible size which leads to possibly many terminal nodes containing only one learn sample record. With only one record in the terminal node we have no way of generating probabilities from a single tree and thus Random Forests trees vote.

Although any one tree commits to just one of the possible target classes an entire forest has far more flexibility. If a forest of 500 trees is used to make predictions it will first generate votes for each class (and you can save the votes from each tree if you like when you SCORE data with a Random Forests). In Breiman's first 1999 version of Random Forests he used simple voting to generate predictions (predicting both class assignments and predicted probabilities). Later, he modified this approach to assign specific weights to each terminal node in each tree. The important point is that a forest will assign probabilities of class membership for each class.

Random Forests predicted probabilities may be used to make class assignments generated by the entire forest. These will be driven by the threshold we use for class assignments, which for the 0/1 binary target will be 0.50. This is important to remember as some SPM data mining engines default to a different threshold. In particular, TreeNet® defaults to a data driven threshold: the threshold determined by the fraction of the data in the focus class (typically class 1).

## Thresholds and Class Assignments

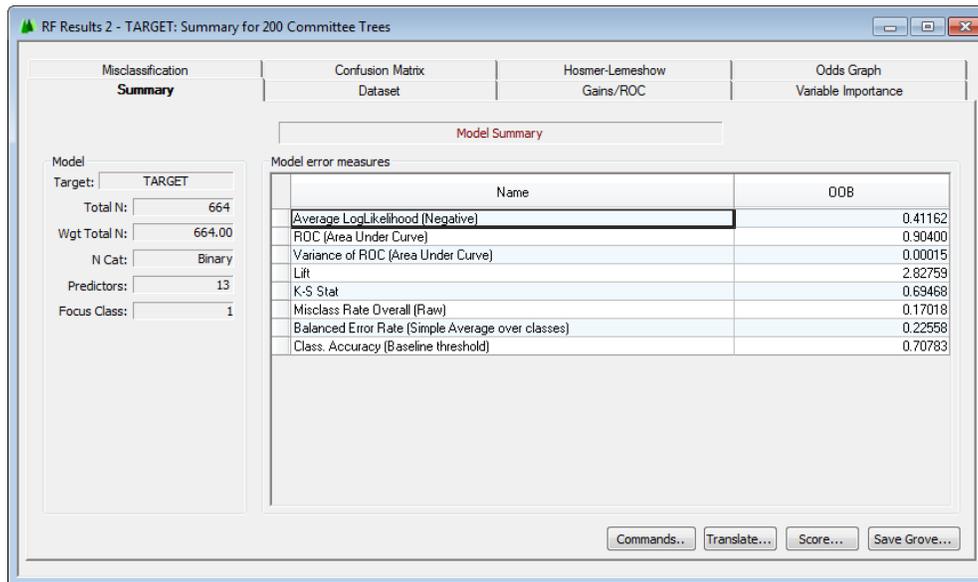
Some learning machines, such as some of the earliest decision trees, are only able to generate class assignments. In other words, if your target was 0/1 the machine would generate a 0 or a 1 for every data record it was asked to classify. No probabilities of class membership are generated. Other learning machines use a two stage process and first generate probabilities of class membership for each target class. This is followed by class assignment which will be determined by a threshold for probabilities. No one would be surprised by the use of 0.50 threshold for a binary target since the assignment rule is equivalent to assigning to the most probable outcome. But this is not always the best rule to use.

For TreeNet, for example, for a binary 0/1 target, we would normally use the observed fraction of 1s in the training data as the threshold for classifying as a 1. In SPM, this fraction is referred to as the "Baseline" threshold. Imagine that we are analyzing a rare event, occurring just in 1% of our data. The baseline threshold rule would assign a record with a 2% chance of being a 1 to that class. The idea is not that the event is most likely to occur for this record but that the record is far more likely to be a 1 than a randomly selected record from the train data.

## Random Forests Performance Measures: (Classification)

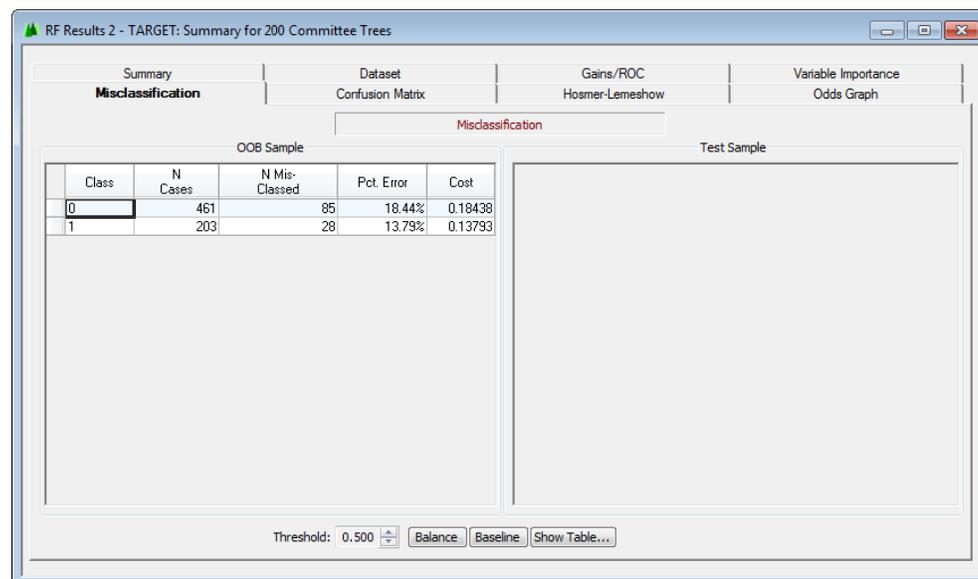
There are several alternative measures of performance available for classification models and the graph displayed shows just one in particular. (We will get to others shortly). It is important to understand how the measure being reported, which was Breiman's preferred, is calculated. Our graph displays an average misclassification rate on OOB or unseen data; in this case, we are seeing a simple average of two classification error rates, one for each of the two classes of the target. If there were five classes then the graph would report a simple average of five misclassification rates.

Clicking on ‘Summary’ will give us the following report. Observe that all results are based on OOB data as described earlier.



The negative Log-Likelihood reported is averaged to facilitate comparisons between learn and test results, and the ROC is calculated using the OOB predicted probabilities for each record.

Classification accuracy is somewhat trickier to manage because it depends on the threshold we use to make a class assignment. In our model the predicted probability of being a TARGET=1 ranges from a low of about 0.028 to a high of 1.0. How large a probability do we need to convince ourselves that a given record should be treated as if it were in fact a TARGET=1? Random Forests simply uses majority or plurality rule and in this case if the predicted probability exceeds 0.50 we classify the record as a 1.



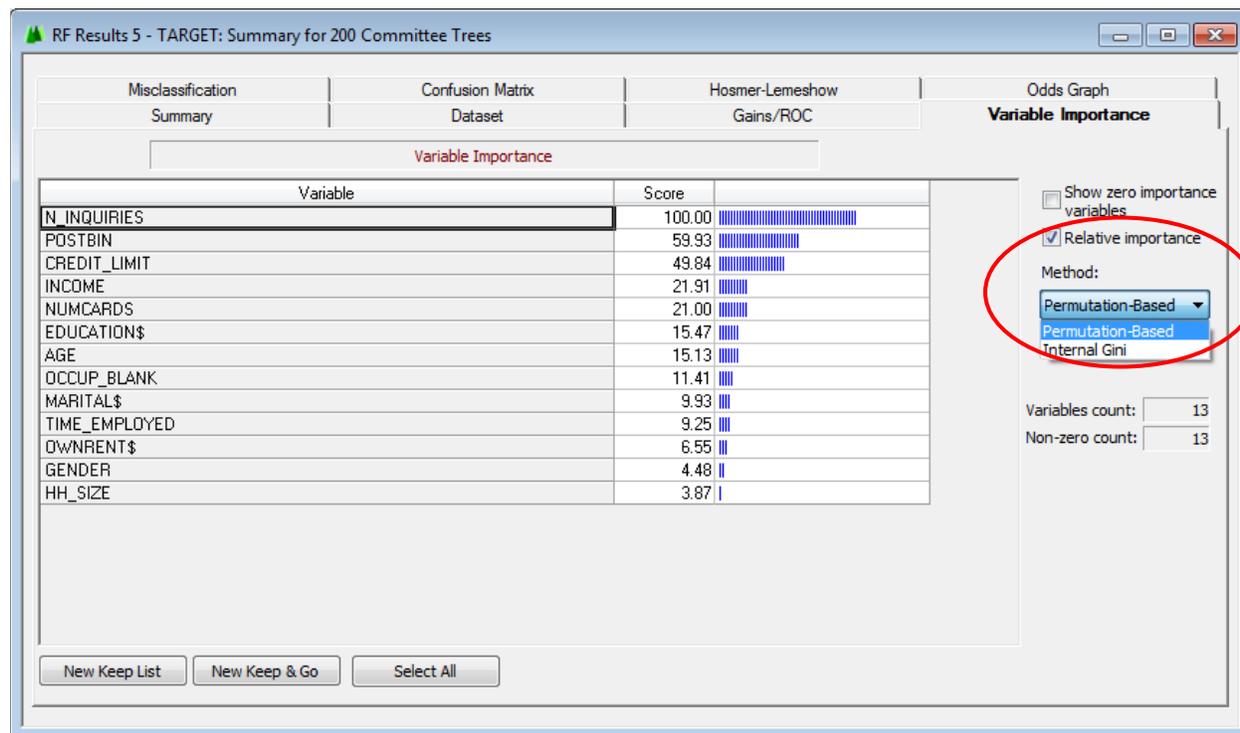
Clicking on the “Misclassification” tab and making sure that the “Threshold” has been set to the Random Forests preferred value of 0.50 we see that we have two different classification error rates for the two classes. The “Balanced Error Rate” is the simple arithmetic mean of these two rates and this is what is

plotted on the main Random Forests Results display.

The other tabs in this Summary window display reports that are common across all SPM data mining and predictive modeling engines and are described in separate general sections.

## Variable Importance

Many analysts highly value the ability to rank predictors in a database. It comes down to knowing what matters and what does not. Especially when working with a large number of variables being able to focus on a relatively small number aids decision makers to have confidence in communication with others. In SPM every one of our data mining engines offers a plausible ranking of the available predictors but Random Forests offers a unique twist on this concept.



In our tree-based engines CART and TreeNet variable importance is measured in terms of the role that a variable plays in the construction of a model. Not surprisingly for CART, the variable at the top of the tree that splits the root node is often rated as most important. Technically, variable importance is based on how often a variable is used in a tree (or collection of trees), what fraction of the data passes through a given node being split, and how well the splitter performs in a node as it separates levels of the target variable. In CART there is only one tree to examine; in TreeNet and Random Forests there are many. But the method of computing a variable importance score is the same: compute a score for every split the variable generates, sum the scores across all splits made, and then normalize so that the highest scoring variable is assigned a score of 100. (If you prefer to see raw instead of normalized scores just uncheck the “Relative Importance” box on the Variable Importance display.)

This method is available in Random Forests as well and is labeled the “Internal Gini” method (note that it is one of the choices from the pull-down menu shown above). The key characteristic of this method is that it is essentially a description of the role of variable in a model *as constructed*. But Random Forests also uses

another novel and intriguing method: Random Forests runs a form of simulation to ask: if a specific variable was damaged by being hopelessly scrambled in the data how much would that damage the ability of the model to predict accurately? This is similar to asking what would have happened if we did not have access to the variable in question and were forced to build the model without it? Would losing that variable have hurt our ability to predict? This approach was referred to as the “Standard” method by Breiman and Cutler and is referred to as “Permutation-Based” in SPM (see the screen shot with the “Permutation-Based” and “Internal Gini” options above).

The Random Forests method works as follows: starting with the training data and one variable, scramble the values of the variable in its column. No change is made to the variable other than to shuffle the values among the rows of the database. Then the data is dropped down the first tree to generate the first prediction for every record in the learn data. We then move to the second tree and again reshuffle the same variable, and drop the learn data down this second tree. We continue in this fashion until we reach the last tree. Now we compare our predictive accuracy gained from the original data with the accuracy obtained after repeatedly re-shuffling our variable. If our predictive accuracy drops dramatically we will consider the variable important; if the predictive accuracy does not drop at all then we will consider the variable essentially irrelevant.

The beauty of this methodology is that it is extremely fast, involving only the scoring of the learn database. In a 500-tree forest, each variable is reshuffled 500 times on its way through the forest meaning our results will not be dependent on any particular random shuffle. Although the process needs to be repeated entirely for every variable, the process is vastly faster than the alternative of dropping each variable in turn and building a whole new forest (the “leave-one-variable-out” or LOVO method). The final ranking of the variables is determined by the amount of deterioration in performance induced by the shuffling experiment. The variable suffering the greatest deterioration in classification accuracy will be ranked most important.

To obtain the shuffled-data simulation measure of importance you must have turned this method and post processing ON. Once your Random Forests model is run with post-processing you will see an option as to which method of variable importance calculation to display. The classic output will print both versions if both have been computed.

To invoke this method, simply type the following at the command prompt:

```
RF PMETHOD = 1
```

```
RF GO
```

PMETHOD=0 will disable the algorithm altogether and usually results in a significant speed up; note that the traditional Gini-based variable importance will still be available. PMETHOD=1 turns on permutation-based variable importance calculation; this may cause a significant slowdown on datasets with many observations but usually runs quickly on wide datasets.

## Proximity in Random Forests

One of the most imaginative uses Breiman and Cutler made of their forests was to construct a measure of proximity or similarity between any pair of records in the learn data. Recall that we typically grow very large trees with possibly hundreds or thousands of terminal nodes. The Random Forests proximity 500 trees then the maximum possible match rate is 500 and the minimum is always 0. For an arbitrary pair of records what kind of match rate should we expect?

Given that the trees are constructed at least partially at random the path that a given record follows down one tree could involve a totally different set of variables than are followed down another tree. Thus, for two records to match very frequently they will have to have very similar values for a large number of variables. If two records were to match on all 500 trees they would have to be near clones of each other. If two records never match, even though given 500 chances, they must be very different. This then is the key idea behind “proximity” in Random Forests and the measures are leveraged for several purposes in the post-processing.

- ◆ A key feature of Random Forests-based proximities is that we can obtain these measures
- ◆ Without needing to select variables. Random Forests will accomplish its own variable selection.
- ◆ Using any mix of continuous and categorical predictors
- ◆ Without needing to standardize the predictors or needing to worry about outliers
- ◆ Allowing Random Forests to do its own missing value handling

The Random Forests post-processing uses the proximities to construct several displays, reports, and interesting graphs. These will be accessed from the main results display by clicking on “Summary...” or “Plots”. But before going to these reports we will have a closer look at the raw material Random Forests uses to create these displays. On the model setup dialog we have the option to save five different sets of results (the proximity option may save two files):

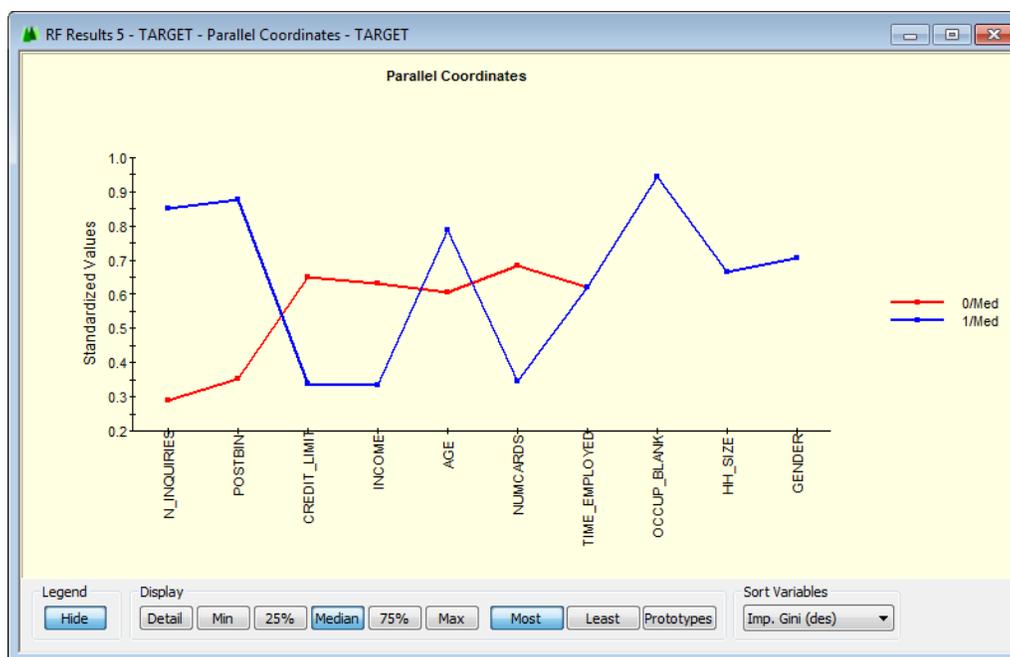
SPM will allow you to specify the first part of the names for these files and then tacks on its own identifier:

- ◆ Parallel Coordinates: Class specific summary of predictors
- ◆ Outliers and Scaling Dimensions
- ◆ OOB predicted probabilities and OOB class assignments made by the forest
- ◆ Full Proximity Matrix: Full NxN matrix (if requested)

- ◆ Partial Proximity: Listing of IDs of nearest neighbors for each learn record
- ◆ Imputed
- ◆ Prototypes

## Parallel Coordinates Plots

The PC Plot is a convenient way to plot group means or percentiles for many continuous measures all on the same graph. The plot style can also be used to display individual records. To permit the simultaneous display of values for several continuous variables with potentially very different scales we first map all values of any variable to the interval between 0 and 1. The minimum value of any variable is always as 0 and the maximum as 1; similarly, the median is represented as .50, the 25<sup>th</sup> percentile as .25 etc. Suppose we have two groups in our data, such as those with TARGET=0 and TARGET=1. We can now plot the median value for the variables used in the model for each group.



This graph is quite different from those usually encountered in statistical and data mining reporting. It is actually very simple but takes a little getting used to. We will start by describing some essential concepts and go into details subsequently. First, we need to pay attention to the color coding: the blue line corresponds to a group of records predicted to have TARGET=1 and the red corresponds to a group predicted to have TARGET=0. For some predictors such as POSTBIN the red and the blue line are quite far apart, which suggests that POSTBIN can be expected to be quite different (typically) for the two groups. For other variables, the lines are essentially on top of each other meaning that there is no difference at all between the groups so far as these variables are concerned.

The data underlying this plot can be optionally saved, as we saw in the Random Forests model setup dialog, where we elected to save the file as GOODBAD\_RFSAVE\_PARCOOR.CSV. The "PARCOOR" part of the name was added automatically to our selection of the stem of the name (GOODBAD\_RFSAVE\_). The first few rows and columns of this file are shown below:

	A	B	C	D	E	F	G	H
1	TARGET	PREDICTION	PLOT_GROUP	PROB	RECORD_NUMBER	CONTENT	EDUCATION	MARITA
2	0	0	0	0.977212	6	scaled data	College	Married
3	0	0	0	0.975612	7	scaled data	College	Married
4	0	0	0	1	23	scaled data	College	Single
5	0	0	0	1	29	scaled data	College	Single
6	0	0	0	0.981314	36	scaled data	College	Single
7	0	0	0	1	46	scaled data	College	Single
8	0	0	0	0.979207	48	scaled data	College	Married
9	0	0	0	1	56	scaled data	College	Single
10	0	0	0	1	57	scaled data	College	Single
11	0	0	0	1	85	scaled data	College	Single
12	0	0	0	1	100	scaled data	College	Single
13	0	0	0	0.969686	119	scaled data	College	Married
14	0	0	0	1	128	scaled data	College	Married
15	0	0	0	1	131	scaled data	College	Single
16	0	0	0	0.983288	140	scaled data	College	Single
17	0	0	0	0.973709	142	scaled data	College	Married
18	0	0	0	1	209	scaled data	College	Married
19	0	0	0	1	212	scaled data	College	Married
20	0	0	0	1	229	scaled data	College	Single

The first block of rows consist of records with the highest predicted probabilities of having TARGET=0. The records are sorted by RECORD\_NUMBER which is the row number of the record in the original data set. Since we went with the default options and we have sufficient data, this file starts with the 50 records with *highest* probabilities of being in class 0.

Following, we get another 50 records with the *lowest* probabilities of being class 0. For the two-class target, this second group is redundant because low probability of being class 0 automatically means high probability of being class 1. But if we were working with say a 3-class problem, the fact that we know that a record has a low probability of being in class 1 does not tell us if that record has a high probability of being in a specific other class.

Since the first line of the spreadsheet contains the header (variable names) the 50 lines of data appear in rows 2-51. Immediately following, starting in row 52 we get the 50 records with the lowest probabilities of being Class 0. The first few rows of this part of the file are shown below.

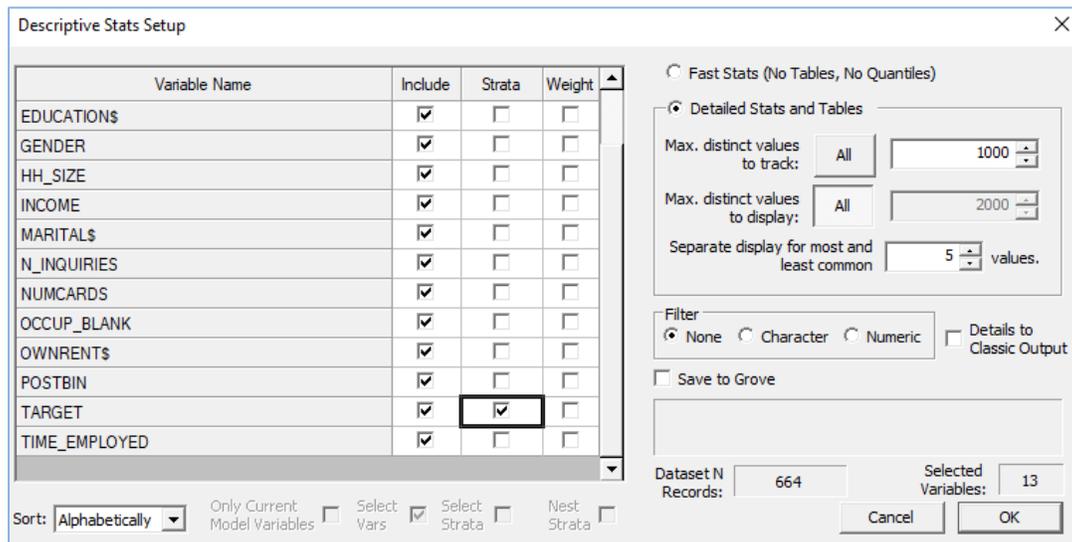
	A	B	C	D	E	F	G	H
1	TARGET	PREDICTION	PLOT_GROUP	PROB	RECORD_NUMBER	CONTENT	EDUCATION	MARR
52	0	1	0	0.117137	50	scaled data	College	Singl
53	0	1	0	0.0697653	102	scaled data	College	Marr
54	1	1	0	0.0225386	104	scaled data	College	Singl
55	1	1	0	0.0948476	116	scaled data	College	Singl
56	1	1	0	0.0874295	122	scaled data	College	Marr
57	1	1	0	0.104476	155	scaled data	HS	Marr
58	1	1	0	0.094917	160	scaled data	College	Marr
59	1	1	0	0.0714171	168	scaled data	College	Marr
60	1	1	0	0.0973242	198	scaled data	HS	Marr
61	1	1	0	0.111955	205	scaled data	College	Marr
62	0	1	0	0.113474	221	scaled data	College	Singl
63	1	1	0	0.115132	227	scaled data	HS	Singl
64	1	1	0	0.0254042	233	scaled data	HS	Marr
65	0	1	0	0.0967186	235	scaled data	HS	Marr
66	1	1	0	0.0331438	255	scaled data	HS	Marr
67	1	1	0	0.0538254	267	scaled data	College	Singl
68	1	1	0	0.0324072	268	scaled data	HS	Singl
69	0	1	0	0.0632789	280	scaled data	College	Marr
70	1	1	0	0.0303911	289	scaled data	College	Marr
71	0	1	0	0.0933953	300	scaled data	HS	Marr

For comparison purposes we ran Descriptive Statistics stratified (grouped) by the value of the variable TARGET and you can see that indeed POSTBIN is different between the two groups. But the way we generate these summaries in conventional statistics and Random Forests is very different as the Random Forests approach uses the predictive model to select the subset of records which are the most likely to belong to a class.

We request the descriptive statistics from the View menu or the from the toolbar (see the circled toolbar icon)



to bring up this dialog where we ask for group specific statistics in the “strata” column.



These are the results for TARGET=1

AGE	187	16	7.88	36	33.79679
CREDIT_LIMIT	203	0	0.00	86	12.898.8719
EDUCATIONS	203	0	0.00	3	
GENDER	197	6	2.96	2	-2.12183
HH_SIZE	184	19	9.36	7	3.12500
INCOME	203	0	0.00	112	3.869.70443
MARITALS	203	0	0.00	3	
N_INQUIRIES	203	0	0.00	20	5.50246
NUMCARDS	203	0	0.00	7	1.27586
OCCUP_BLANK	203	0	0.00	2	0.15271
OWNRENTS	182	21	10.34	4	
POSTBIN	203	0	0.00	4	3.98522
TIME_EMPLOYED	203	0	0.00	22	2.38670

and these are for TARGET=0

Variable	N	N Missing	% Missing	N Distinct	Mean
AGE	393	68	14.75	35	31.14758
CREDIT_LIMIT	461	0	0.00	216	15,877.08021
EDUCATIONS	450	11	2.39	3	
GENDER	392	69	14.97	2	0.03061
HH_SIZE	360	101	21.91	7	3.41111
INCOME	461	0	0.00	268	4,174.58135
MARITALS	460	1	0.22	3	
N_INQUIRIES	461	0	0.00	15	1.87636
NUMCARDS	461	0	0.00	10	2.03254
OCCUP_BLANK	461	0	0.00	2	0.01085
OWNRENTS	339	122	26.46	4	
POSTBIN	461	0	0.00	5	2.67462
TIME_EMPLOYED	461	0	0.00	24	1.46421

While there is often rough correspondence between simple descriptive statistics and the Random Forests-generated parallel coordinates plots, the latter are based on a predictive *model*. Looking again at the Random Forests plots, observe that the variables will be sorted in order of importance using the variable importance ranking method selected at the bottom of the display. Remember that in these plots the groups are defined not by their *actual* class membership but by the Random Forests predicted probability. So in the parallel coordinates plot graph above the blue values represent typical data among the 50 records assigned the highest probabilities of being TARGET=1 (this could include some records that are actually 0 but in an accurate model this will occur rarely). Similarly, the red values represent typical values for those assigned the highest probabilities of being TARGET=0.

- ✓ The default graph does not come with the grid lines but if you right-click on the graph and select “Edit chart” and then “Advanced” you will find many controls allowing customization of the graph. We just went to the Y-axis grid option and set its value to 0.10. More info on Graph Editing appears in a separate section.

If we were to plot the median AGE using *all* the learn data the point would be positioned at the 0.50 level on the Y-axis. The same would be true for all variables and the plot for all variables would just be a horizontal line at the 0.50 level. But when we plot the median for a subset of the data, the median for that subset might fall lower or higher than the median for the overall sample. In the PC plot we define special groups as identified by our forest: records with highest probabilities for a given class. In the 2-class typical binary target situation we have (a) records identified as most likely to be TARGET=1, and, (b) records identified as most likely to be TARGET=0. In the multi-class problem we will have a group of most likely records for each class.

### How many records in each group for parallel coordinates plots?

If you have plenty of data, Random Forests defaults to selecting the 50 most likely records for each class. However, Random Forests never selects more than half the available records for any class. So, if your smallest class only has 30 records then Random Forests will adjust to selecting half this number, or 15 records, and this limit will then apply to all classes.

If you would like to take more direct control over the number of records used for display in the PC plots use the command line to enter a command like the following:

```
RF PARCOOR=20
```

If you have sufficient records to generate at least 20 records per class (meaning that you have at least 40 records in the smallest class) then you will get 20 records extracted for each class for the parallel coordinates plot.

## Out of Bag Predictions

As every record has an approximately 37% chance of not being included in the training sample for a specific tree, within an entire forest each record will have been out of the training sample (OOB) for about 37% of the trees in the forest. This means that in our default 500 tree forest any given record will have been OOB for about 185 trees. If we were to extract the subforest of just those 185 trees for the record in question and use them to make predictions we would be making a prediction for previously unseen data. If we did this for every record in the original learn sample we would obtain a complete set of OOB predictions ideal for honestly assessing the performance of the forest on future unseen data.

It is this feature that makes Random Forests so convenient and effective, especially when dealing with learn databases containing only a few samples (rows), such as encountered in many biomedical studies but also in text classification.

In all likelihood, no two records in the learn data set will be scored by exactly the same subset of trees when we produce OOB predictions. In other words, each record will be scored on a unique subset of trees because the precise pattern of in-bag and out-of-bag will be unique to each record.

- ✓ Because the fraction of trees that can be used to produce OOB predictions is small we recommend growing somewhat more trees than appear to be necessary. In many circumstances, growing 1,000 trees will yield much more reliable OOB results.

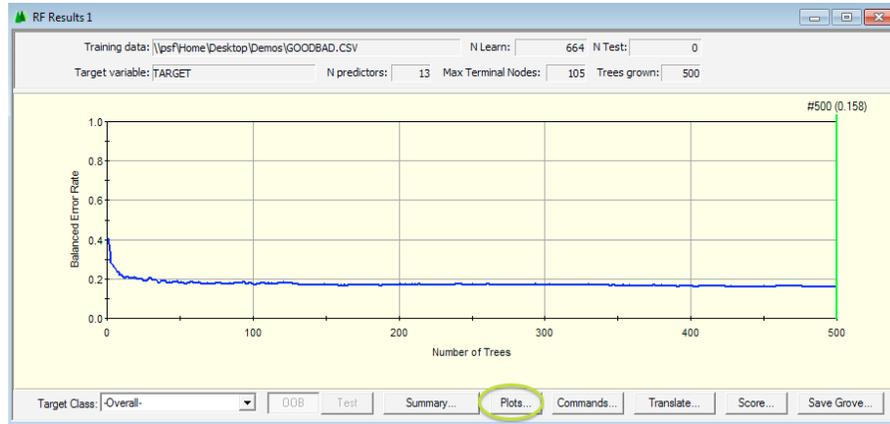
This is advisable only if you are interested in Random Forests post-processing and the detailed investigation of the original learning data. If your only interest is in prediction on new data you may safely work with smaller forests (fewer trees).

The OOB predictions will be written to a file on request, as we showed earlier on the Model Setup Random Forests tab. The OOB predictions file is given the suffix OOB and looks like this for our example forest:

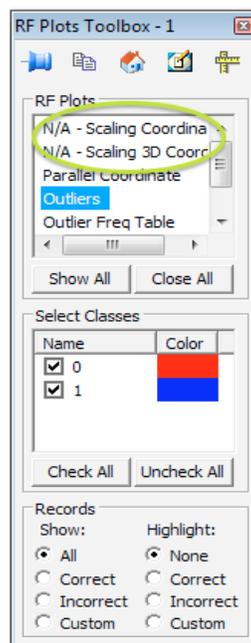
The fields in this file include a RECORD ID (row number), a set of OOB predicted probabilities for each class of the original target, the Random Forests OOB class assignment (prediction of class membership), and the "margin". The MARGIN is simply the difference between the largest and the second largest OOB probability for that record and can be ignored for two class targets. For multi-class targets the MARGIN is considered a measure of how confident Random Forests is in its prediction of class membership, or alternatively, in how clearly separable the classes in question might be.

## How many trees make a forest?

The post-processing of the Random Forests yields fascinating insight into the structure of your data and is revealed in the plots and derived data sets produced after the forest is complete. To gain access to the plots just click on the “Plots” button on the Random Forests results display:



This will reveal a floating toolbar we will discuss in detail below. Here observe that two of the graphs are “N/A” which usually means the forest is too small.



We therefore reran the identical model but with 999 trees and this manages to solve the problem. Now we are able to produce the graphs.



## Proximity Matrices

Random Forests introduces a novel and extraordinarily easy to use clustering methodology based on Random Forests's internal distance measures. Start with a pair of arbitrarily selected records and note the terminal node each lands in for every tree. Imagine that they both land in the same terminal node every time. This could only happen if the two records happened to be near clones of each other as they have to traverse a potentially very large tree and each must take the same left/right decision in every node and in every tree. Conversely,

Imagine that the two records never end up in the same terminal node. Then the two records must have at least a few variables with much different values ensuring that somewhere, in every tree, the records part company at an internal node.

The Random Forests measure of proximity is then simply the fraction of available trees for which a pair of records land in the same terminal node. Matching on the terminal node is an all or nothing affair; we have no concept of a "partial" match for two cases that go all the way down a deep tree together and part company only at the last split. Two records that match on 85% of the available trees will be assigned a mutual proximity of .85. Obviously, the proximity measure must be greater than or equal to zero and less than or equal to one.

## Proximity and OOB trees

In the current version of SPM Random Forests the proximity matrix is based on all trees for all pairs of records. In a future release we plan to allow the option of requiring that at least one of the two records must be OOB for a tree to be available. This will mean that the number of available trees will vary randomly across pairs of records, but that on average each pair will obtain a proximity measure from about 60% of the trees in a given forest (to be rejected a tree must be in-bag for both records which has a probability of  $.63 \times .63 = .3969$  and  $1 - .3969 = .60$ ).

The proximity matrix will have as many rows and columns as records in the original learn data set. It is useful to keep in mind how large this matrix might become:

Rows Learn Data	Number of Cells in Matrix	Size
100	10,000	40 KB
1,000	1,000,000	4 MB
10,000	100,000,000	400 MB
100,000	10,000,000,000	40 GB
1,000,000	1,000,000,000,000	4 TB

From the table it should be obvious that trying to work with the full proximity matrix with much more than 10,000 records will be impractical if not impossible in many environments.

This does not mean that Random Forests cannot be used for predictive modeling for larger data sets but it does mean that we will want to be sure that we are not requesting the creation of the full proximity matrix.

## Partial Proximity Matrix

During the time that Leo Breiman was actively developing Random Forests with Adele Cutler they were focused on relatively small data sets, whereas, at Salford Systems, we were working with much larger data sets principally drawn from banking and direct marketing applications. We therefore suggested that Breiman work with an abbreviated or truncated proximity matrix that would track only the top  $K$  nearest neighbors for every record in the learn sample. This would require counting the number of matches for every pair of records in the available trees in the forest, but after calculation of every proximity for a single record we would remember and store only a relatively small amount of information: the identity of the  $K$  nearest neighbors and the proximity values. This partial proximity information is controlled by the circled option in the model setup dialog below.

The screenshot shows the 'Model Setup' dialog box with the 'Random Forests Options' tab selected. The 'Number of proximal cases to track' field is highlighted with a red box and set to 'Auto'. Other settings include 'Number of trees to build' at 200, 'N predictors' at 3, 'Frequency of progress reports' at 10, and 'Create Full Proximity Matrix' checked. The 'Analysis Engine' is set to 'Random Forests Tree Ensembles'.

For a very large data set we might set this “proximity depth” to a number such as 100. If we were working with a 10,000 record learn data file the resulting partial proximity data file would contain 100 output records for each original input record. Such a proximity matrix is shown next, listing the top 10 nearest neighbors for each of the first three rows of the learn data. We will have one row for each near neighbor identified and we record:

- ID of the learn data record
- Rank of the near neighbor (1<sup>st</sup>, 2<sup>nd</sup> 3<sup>rd</sup> etc)
- ID of the near neighbor
- Number of terminal node matches

The last measure should be divided by the number of trees used to calculate the normalized proximity measure.

	A	B	C	D
1	CASE	K	LOZ	PROX
2	1	1	1	999
3	1	2	291	402
4	1	3	265	288
5	1	4	561	247
6	1	5	54	241
7	1	6	135	236
8	1	7	451	217
9	1	8	361	203
10	1	9	49	202
11	1	10	26	177
12	2	1	2	999
13	2	2	597	278
14	2	3	542	266
15	2	4	604	257
16	2	5	91	233
17	2	6	81	210
18	2	7	244	207
19	2	8	89	200
20	2	9	607	195
21	2	10	19	192
22	3	1	3	999
23	3	2	607	479
24	3	3	71	313
25	3	4	466	265
26	3	5	649	251
27	3	6	290	250
28	3	7	143	249
29	3	8	606	247
30	3	9	41	227

## Outliers

Breiman and Cutler define an outlier as a record that is far away from the bulk of the other members of its *own* target class. Breiman and Cutler frequently make use of the concept of average proximities and leverage that idea here. Imagine the classic extreme outlier. Such a record is far away from all of the data and thus of course is on average not close to anything. By definition such an outlier has no near neighbors and is easy to identify. But what do we do about groups of relatively isolated records or outlier clusters? By using average proximities we can deal effectively with outlying clusters. Since the members of such a cluster may all be very close to each other, they will all have near neighbors. But when looking at average distances or proximities we will see that such an outlying cluster is obviously exceptional.

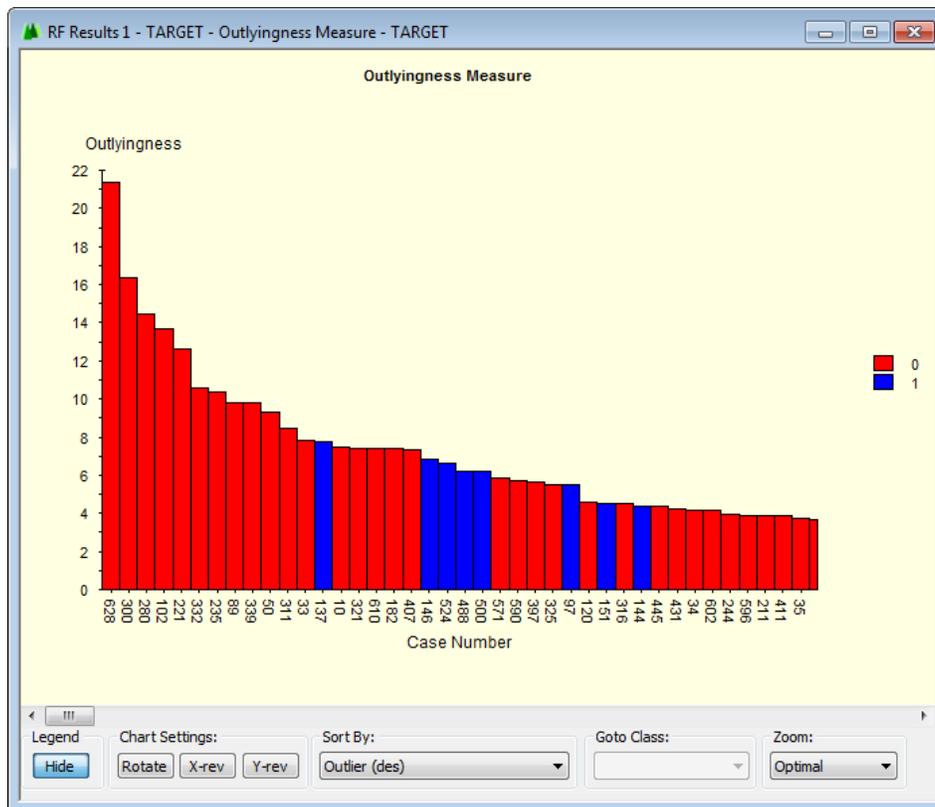
The standard measure of outlyingness in Random Forests is based on this average distance of a record from all other members of its class. Using the median and IQR of these distances an outlier measure is calculated. Here we show part of the output file containing the outlier scores here sorted in descending order:

	A	B	C	D
1	CASE	TARGET	PRED	OUTLIER
2	628	0	1	26.8792896
3	182	0	1	12.6818514
4	235	0	1	11.4411058
5	300	0	1	10.8315029
6	321	0	1	10.5611763
7	102	0	1	8.86011219
8	407	0	1	8.69936085
9	332	0	1	8.13937664
10	89	0	1	7.82913733
11	221	0	1	7.67611551
12	339	0	1	7.51490116
13	311	0	1	6.76945591
14	146	1	0	6.65484476

Record number 628 has the largest score by far, and we can confirm this in part by taking a close look at the full proximity matrix. There we can see that in fact Record 628 is not close to any other record; taking an average over all the proximities in the matrix we obtain .079 whereas for record 628 we have an average of .021. The average maximum proximity in the entire matrix is .595 whereas for record 628 the closest record has a proximity of .186. Not surprisingly record 628 is misclassified by Random Forests as it looks more like a 1 than a 0, but this record is not really close to any other 1s either.

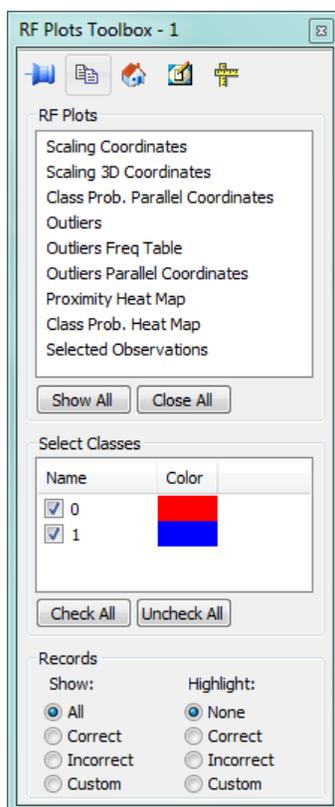
We can review the file displayed above only if we requested that it be saved. However, providing that we permit post-processing of the forest, we can always see a graphical representation of the outlier scores by selecting the “Outlier” graph from the floating toolbar.

### 999 Tree Forest



## Working with Random Forests Plots

Random Forests provides you with a rich set of plots intended to provide deeper insight into your data than is obtained from performance measures and variable importance rankings. There is a lot of ground to cover and we start here with just the most elementary controls.



The plots control center is a floating toolbar listing the possible plot types in the upper box (note the circled title). At the moment there are seven available:

- **Scaling Coordinates:** 2D plots based on multidimensional scaling (MDS) of the full proximity matrix. Plots selected pairs of MDS coordinates.
- **Scaling 3D Coordinates:** plots all learn sample data points using the first 3 coordinates extracted via MDS
- **Class Prob. Parallel Coordinates:** plots of continuous predictors ordered by importance
- **Outliers:** score bar chart for all learn sample records
- **Outliers Freq. Table:** score frequency chart
- **Proximity Heat Map:** based on the full proximity matrix. Matrix is limited to the size displayable on screen using one pixel per learn sample record.
- **Class Probability Heat Map:** display of OOB predicted probability of class membership by class. Rows are learn sample records and “columns” are target variable classes.
- **Selected Observations:** displays list of observations in selected area when “Lasso” is applied in either heat map (right-click > Lasso, right-click > Apply Lasso).

Double-clicking on any plot type will bring up a specific plot, while clicking on the “Show All” button will bring up all available plots.

## Cluster Insights: Full Proximity Matrix

The full proximity matrix contains the core measures from which all cluster and group insights will be derived (when combined with actual class membership, predicted class membership, and predicted probabilities). To obtain this matrix you must enable Random Forests *post-processing* on the Random Forests model setup tab.

We set this option to OFF by default because post-processing can require more computer time than building the forest. Turn post-processing on after you are sure you are using good control parameters and you have tuned your model to obtain reasonable results and performance.

Saving the full proximity matrix is not strictly necessary, but if you wish to make use of the Random Forests between-learn-record distances in your own post-processing you must request that the matrix be saved.

You specify the file name under which the full proximity matrix will be saved on the model set up dialog. There you also indicate the upper limit on learn sample records that can be used when creating such a matrix.

To see the matrix displayed on screen, your learn sample data set must contain fewer records than horizontal pixels available.

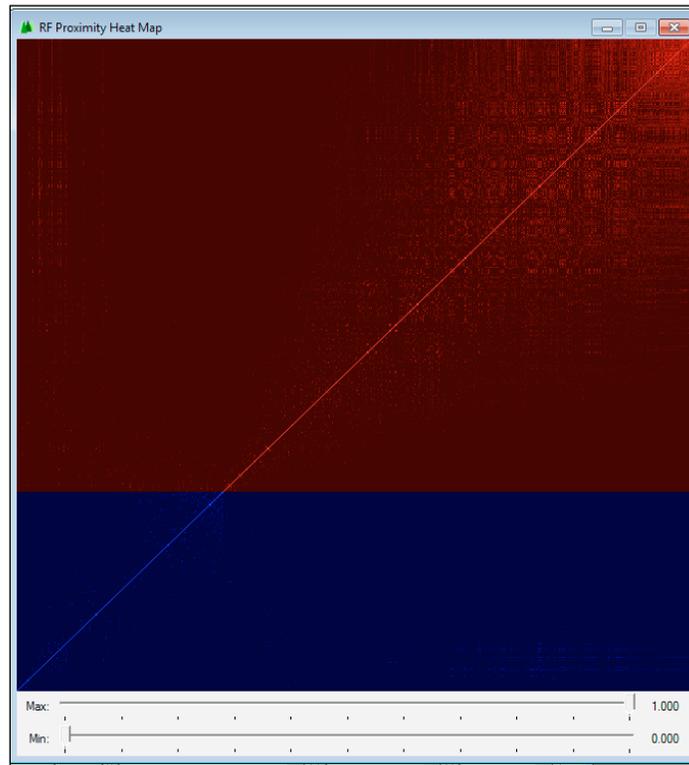
The default display of our full proximity matrix is shown below. The matrix is organized as a square, with record ID as an implicit label for the rows and columns. The records are sorted first by actual class membership and then by average proximity to all learn sample records. The records that are likely to be near the center of a cluster will appear towards the upper right corner of each class.

The brightness of a point reflects the proximity of the pair records represented by that point. Since on the diagonal line we display the proximity of any point to itself, we see that this line is the brightest region of the display. We see what appear to be two areas of brightness among the class 0 records (red), one a small square very close to the upper right corner and a larger square a little to the left along the diagonal.

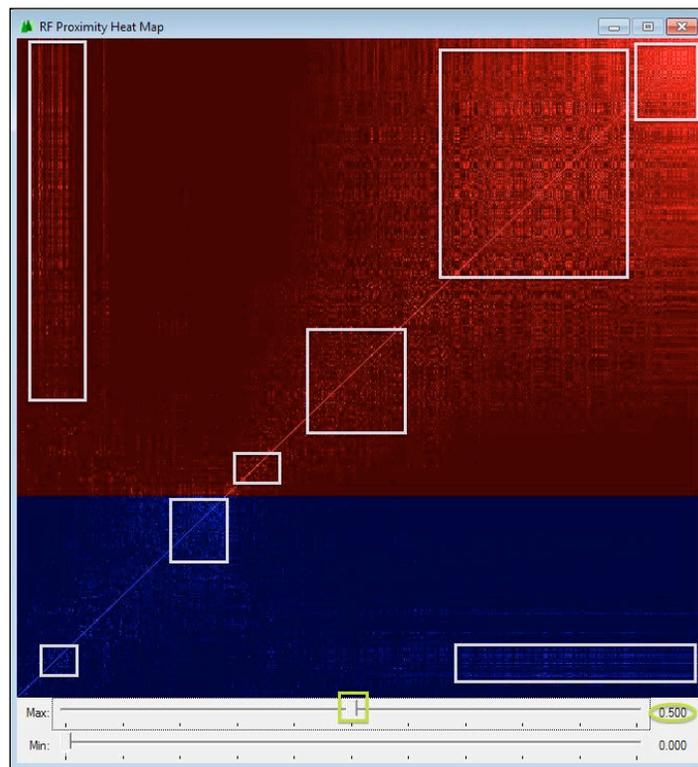
To make spotting clusters easier we can manipulate the two slider controls. The upper slider determines how close two records need to be in order to be displayed brightly and in the modified display we lowered this control from 1.0 to 0.50. This turns many more points bright and allows us to identify possibly interesting areas in the data. Areas that are the diagonal blocks defining each class reveal records that are close to records of the “wrong” class.

The third display also manipulates the lower slider which determines the lowest proximity which will display any level of brightness. This helps to clarify the image further.

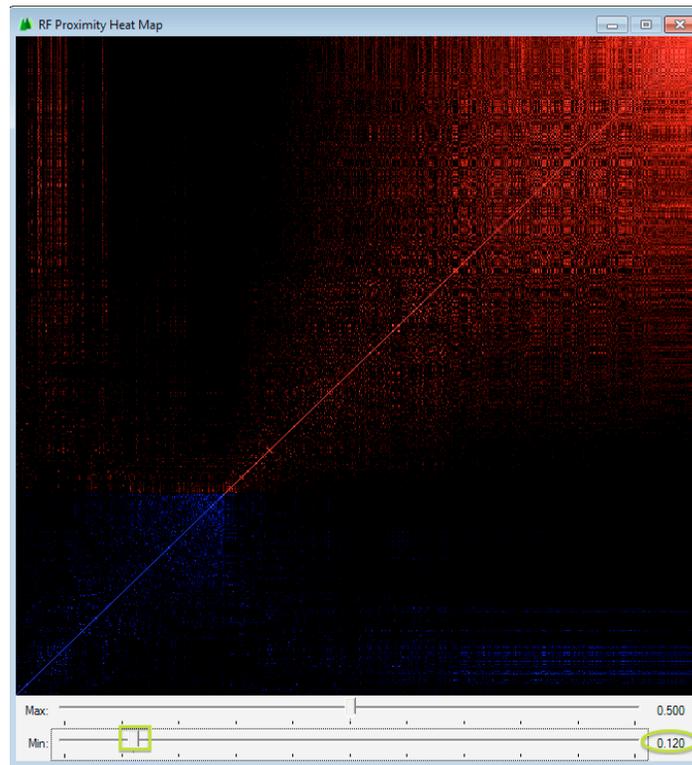
### Default Full Proximity Heat Map



Display with lowered max setting for points to display brightly



## Display with increased minimum proximity required to display

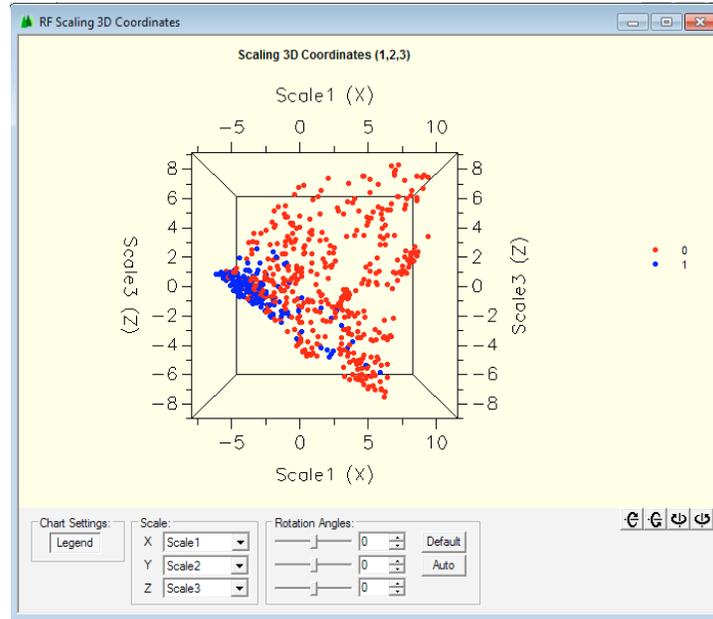


## Multi-Dimensional Scaling of the Proximity Matrix

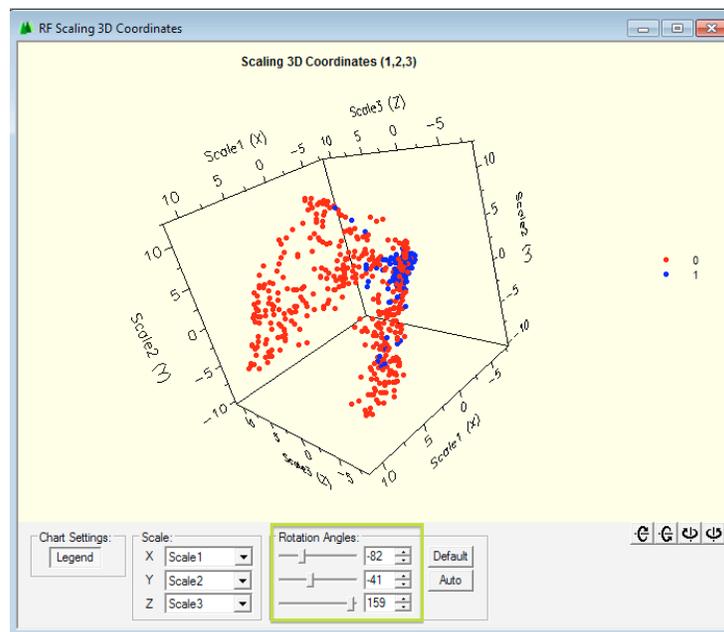
The MDS processing of the full proximity matrix is intended to display a simplified and possibly over-simplified version of the distances between all the points in the learn data. The resulting display is spatially meaningful; points close to each other on the MDS display are supposed to be close to each other as defined by the proximities, and, points far away from each other are supposed to be distant from each other as defined by the proximity matrix. In other words the proximity matrix is the “truth” and the MDS display is intended to be a convenient if possibly oversimplified display.

The default 3D MDS display is shown next. The first use we will make of this display is to look for evidence of clustering. While the TARGET=1 group appears to be packed into a relatively small region of the graph, the TARGET=0 group is very widely dispersed and suggests the possibility of several class 0 clusters. But the display below is intended to be 3D which means that we can rotate it in space to see if a different spatial perspective might be more useful.

The best next step is to click on the “Auto” button and allow the image to spin automatically. When we see something interesting we click on “Auto” again to stop the spinning.



Default perspective 3D graph: note all three rotation angles set to 0.0

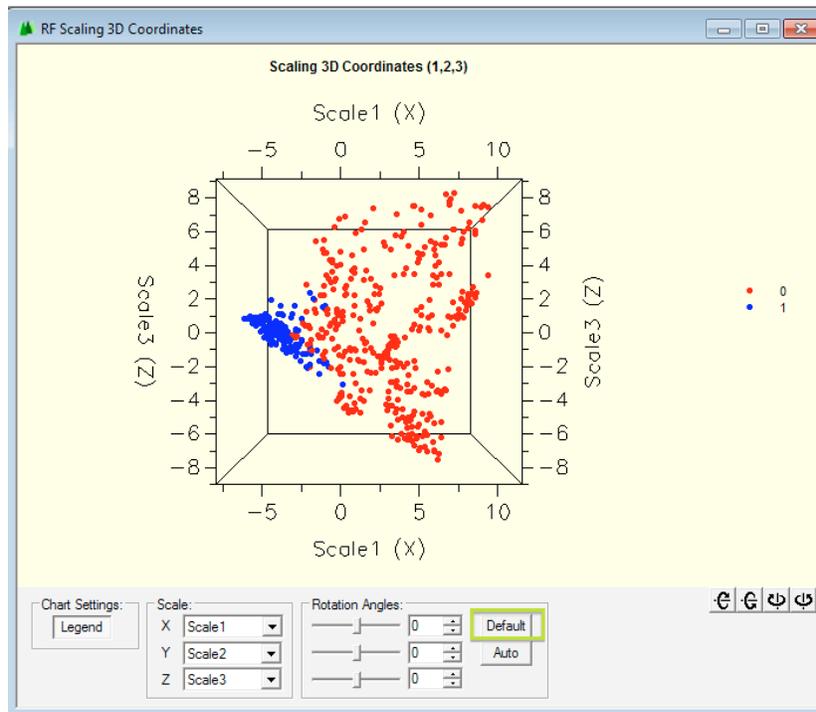


To reproduce the angle displayed you will need to enter the XYZ coordinates highlighted in the green rectangle above. From the angle selected we see that the TARGET=0 group lies on two major branches (left and right) and we might be persuaded that there are possibly five clusters here (three on the left and two on the right). No matter how we spin this display the TARGET=1 group appears to lie mostly in just two dimensions and to consist of perhaps one central cluster and some isolated points.

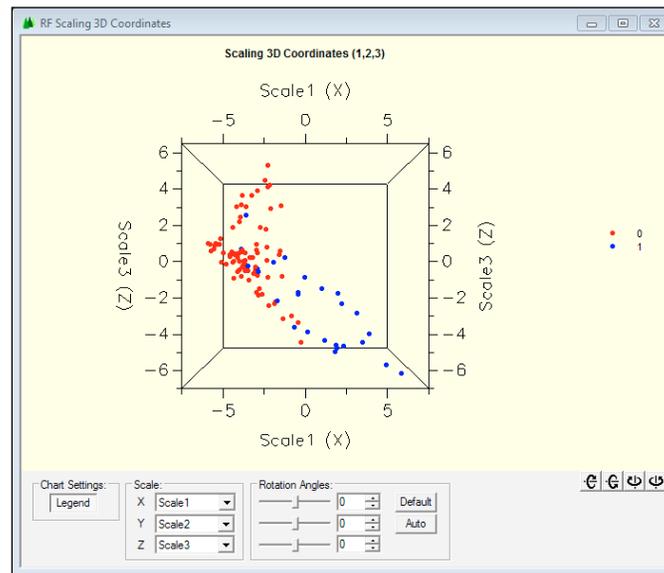
Using the floating toolbar we can now refine the display by choosing to include selected classes (here we have only 0 and 1) which allows us to see more clearly when many points from different classes overlap. We can also elect to display only the records classified correctly or incorrectly.

Records	
Show:	Highlight:
<input type="radio"/> All	<input checked="" type="radio"/> None
<input checked="" type="radio"/> Correct	<input type="radio"/> Correct
<input type="radio"/> Incorrect	<input type="radio"/> Incorrect
<input type="radio"/> Custom	<input type="radio"/> Custom

Displaying correctly classified points only from the Default perspective:



### Displaying points misclassified:

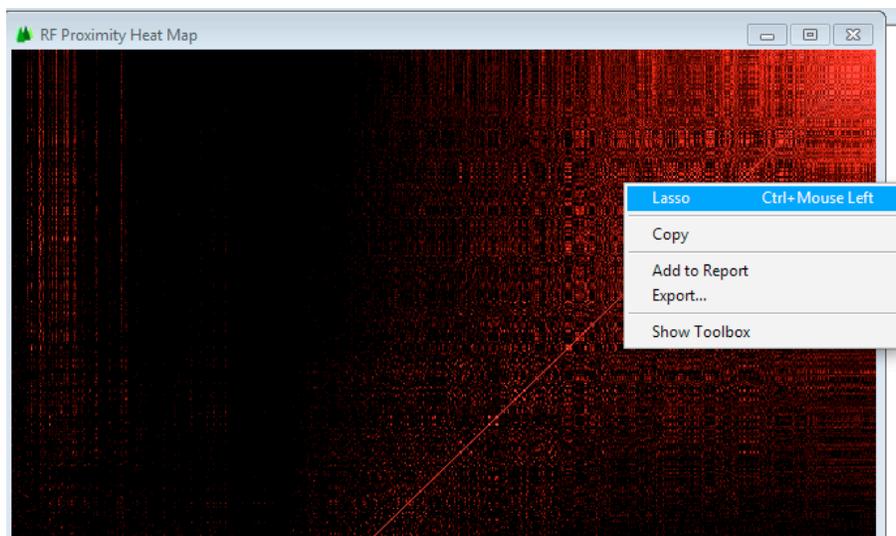


We can see that the misclassified points are located in the “wrong” region; they appear in areas dominated by the other class.

### Linking the Proximity Heat Map and the MDS Display

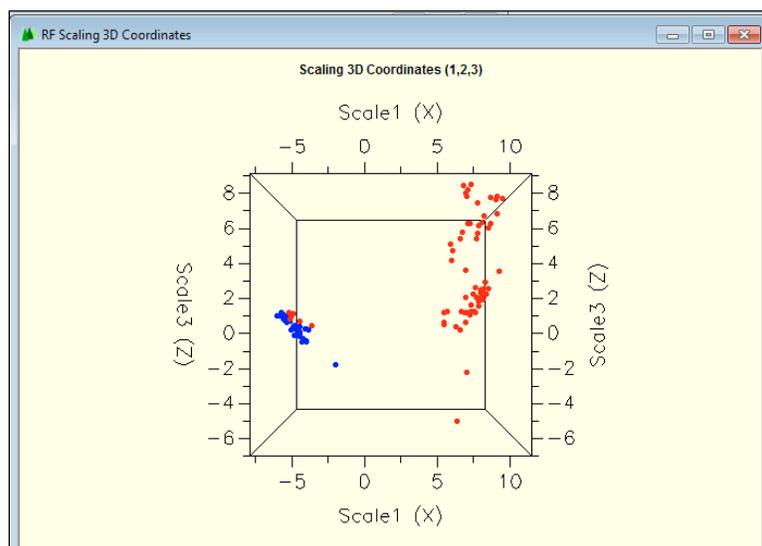
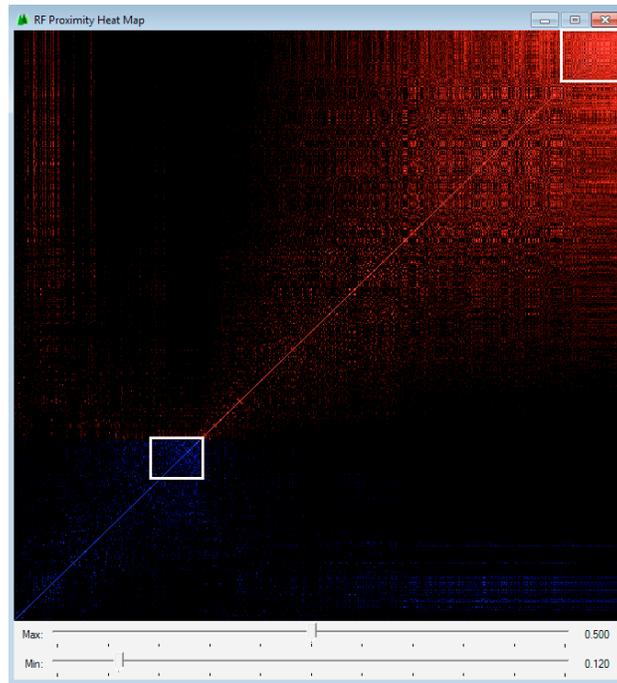
The MDS display allows us to subset records based on class membership or classification status (correct or incorrect). We can also manually select subsets of data to display by marking areas with the mouse on the full proximity display. To accomplish this, return to that display and right mouse click within the display to bring up a new menu:

#### Right mouse-click on Proximity Heat Map to reveal menu:



Now use the “lasso” drawing tool to identify a group of records to highlight. You may select several groups. Once you have selected your subsets of data, right mouse click again to see the option to “Apply” the lasso, which will limit the 3D display to just the selected records.

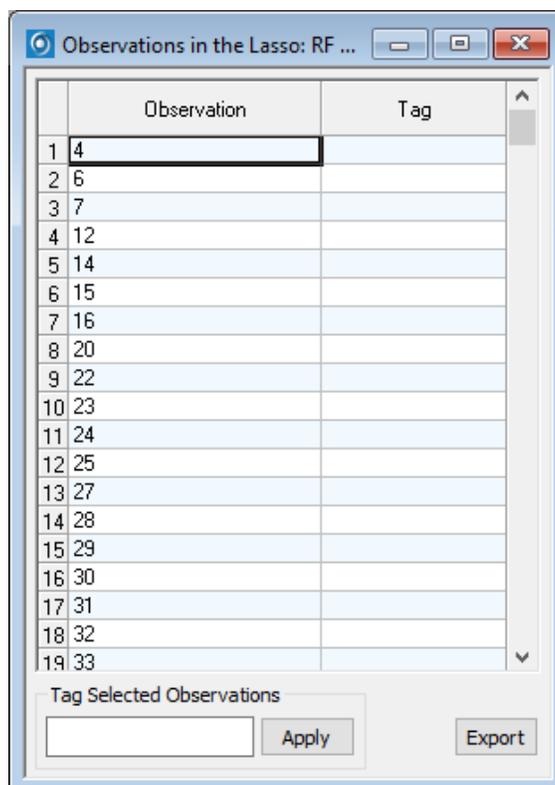
In the screen shot below we show the approximate areas we selected: one group of what appear to be rather tightly clustered class 0 and another similar but smaller group of class 1 records.



To get this display we must “Apply” the drawing lasso and then also select “Custom” on the Plots Toolbox menu. We see two widely separated groups of records but with two surprises: the class 0s appear to fall into at least 2 groups and a few class 0s appear to be very close to the central class 1 region. Could this be right?

To explain what we see we have to point out that records in the proximity matrix are sorted by overall proximity (on average) to all other records. Two adjacent records might each be close to many other records because they are the centroids of their own clusters while these two clusters need not be close to each other. That appears to be the case for this data. Also, if you look closely at the upper right hand area of the class 1 proximity matrix you can see that it is bordered by bright dots belonging to class 0. In other words, the MDS display is doing a good job of representing the full proximity matrix for this data.

After applying a lasso to the heat map, you have the ability to see which observations in the dataset have been selected. Return to the Plots Toolbox and double-click Selected Observations in the plot list:



Use this display to tag selected observations and export the information to an output dataset.

## Automatic Post-Forest Hierarchical Clustering

In SPM 8.0, automatic post-forest clustering can be invoked from the command line or an SPM Notepad. The relevant Random Forests commands are

```
RF JOIN=<YES|NO>
```

This command requests hierarchical clustering. We advise its use directly from inside SPM only for smaller datasets, as this can be time-consuming.

```
RF LINKAGE=<ALL|SINGLE|COMPLETE|CENTROID|AVERAGE|MEDIAN|WARD>
```

This command selects from the seven available ways to accomplish agglomeration as we move from observations to clusters.

 RF SVCLUSTER=<YES | NO>

This final command will save the learn data set with cluster assignments added. Three “solutions” are saved for four, six, and ten clusters.

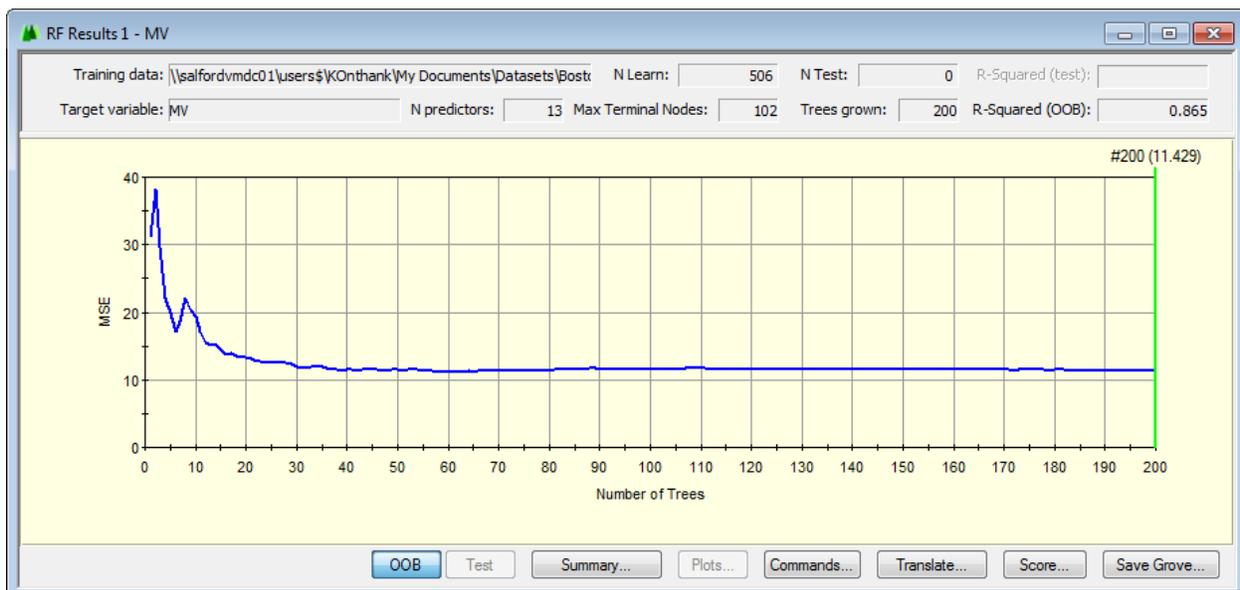
JOIN can be very time-consuming to run which is why it is not visible from the GUI.

## Random Forests Regression

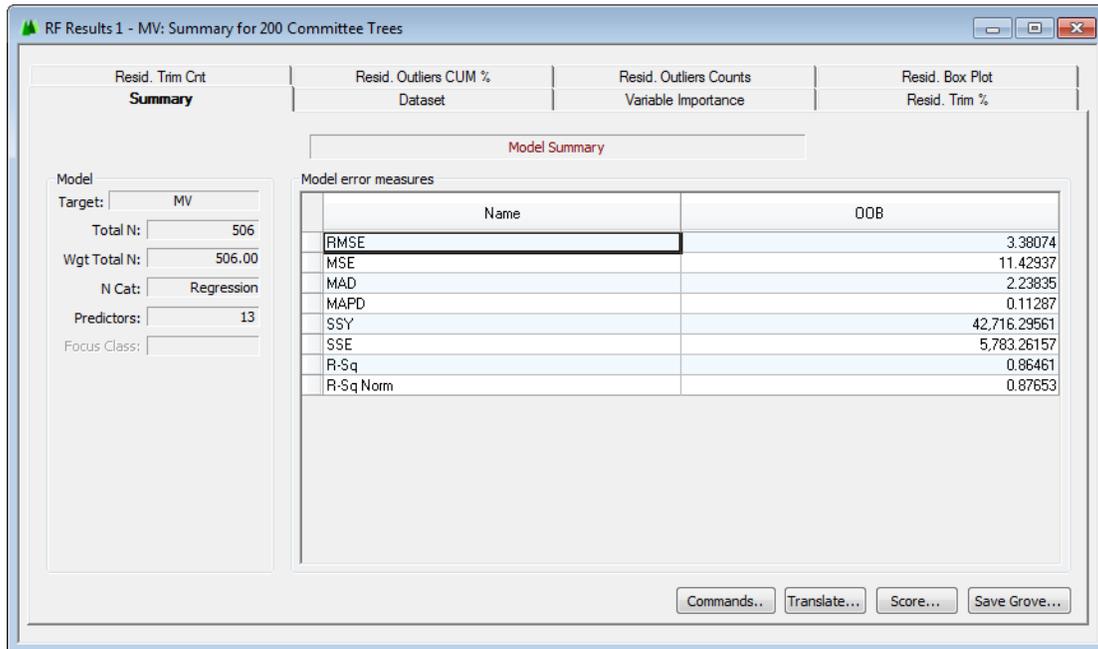
When Leo Breiman first introduced the Bagger in 1996, he observed that the benefits of bootstrap ensembles might be even greater for regression than for classification. In general, a single CART decision tree will rarely be the best performing learning machine for a regression problem. Indeed, Jerry Friedman developed MARS specifically to address the weakness that a single CART tree typically exhibits with regression. Random Forests takes the advances of the Bagger even further and offers potentially remarkable performance.

- ✓ The version of Random Forests regression in SPM 8.0 offers only the Random Forests learning machine and does not include any of the elaborate post-processing available for classification problems.
- ✓ If post-processing is essential for your regression problem, you might consider binning your target variable into say 10 levels and then running a Random Forests classification analysis just to obtain the Random Forests graphics and proximities.

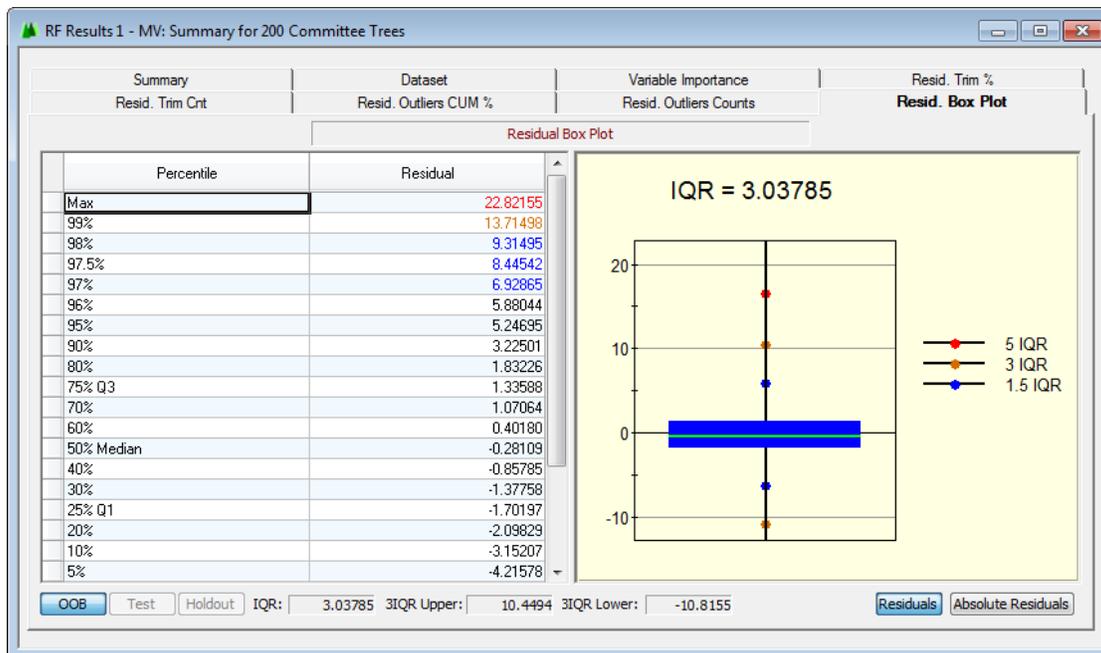
To illustrate Random Forests regression we turn to the classic 1970 Boston housing data set discussed in detail in the original CART monograph. We select MV as the target, all other variables as predictors and go with the Random Forests defaults of 200 trees, and selecting 3 predictors at random at each node as the predictors competing to split that node.



This out of the box performance on OOB data is impressive: observe an R-squared of 0.865 compared to a single tree CART of .781. While we do not get the suite of plots Random Forests produces for classification models, we do get the standard set of Regression reports from the Summary dialog. As always, these performance statistics are OOB, meaning that for any record in the learn sample we generate a prediction using the subset of trees for which that record was “Out of Bag” (not used to grow the tree). Of the 200 trees grown about 74 will be OOB for a given record but exactly which trees those are will differ from record to record.



For a walk through these tabs please refer to the SPM chapter on regression, as the contents and details of these tabs are the same for all regression models. Here we will open just one of those tabs:



These are residuals from the Random Forests regression model predicting the median value of homes in neighborhoods. At the time the data was collected in 1970, the average over all the 506 neighborhood medians was 22.53 (this is thousands of dollars). The residuals are reasonably tightly packed between -1.7 and 1.34. But there is clearly one extreme outlier where the actual neighborhood value is \$22,800 greater than predicted. This outlier would be well worth looking at in detail to determine if there were any errors in recording its data values, but we will not pursue this topic further here.

## Random Forests Regression and Terminal Node Size

For classification models, the best performing forests are those with trees grown out to their maximum possible size. This means that we allow for terminal nodes containing just one learn sample record and that the smallest possible parent node contains two learn sample records (ATOM=2). This finding is not true for regression, however, and the best possible forests might be built on trees with larger terminal nodes.

For the BOSTON data set we ran a series of Random Forests regression models progressively increasing the control set for ATOM. Our results were:

LIMIT ATOM=	R-SQUARED
2	0.86868
3	0.86497
4	0.87150
5	0.86868
6	0.87325
10	0.85839
12	0.85682
15	0.84796
20	0.83653

Here we obtain best results with ATOM=6 and we can see the gradual decline in OOB performance as the terminal nodes become too big. Our point is that in general it is worth experimenting with the ATOM setting when running regression models. In SPM 8, the default setting for regression models is now ATOM=5.

## What is Random Forests Actually Doing?

Random Forests can be thought of as a collection of CART-like trees, using binary node splitting, and growing trees out to the maximum possible size. Each tree differs from a conventional CART tree in several important ways:

Random Forests trees are based on a bootstrap sample drawn from the original learn data. A bootstrap sample typically has the same number of records as the original learn data but it differs in that about 37% of the original learn records are not included in the bootstrap sample. The shortfall of records is made up by allowing some of the 63% of the records selected to appear more than once. This occurs naturally via the “sampling with replacement” mechanism of the bootstrap sample. Every record has an equal chance of being drawn into the bootstrap sample every time we draw a record; if the learn sample has 600 records then we will draw records 600 times, each time allowing every record to possibly be drawn. We do not pay attention to whether a record has already been drawn in this process which is why we will always see some records being drawn more than once and others not at all.

This sampling methodology is critical for Random Forests because it automatically creates a tree-specific holdout sample for every tree and these records are known as OOB (“out of bag”) data. Any one record

will be OOB for about 37% of the trees grown in a forest and if we want any honest estimate of how the forest would perform for this record, we should be looking at just those 37% of the trees (again, which trees we look at will be record-specific). When constructing performance measures and predicted probabilities from a Random Forests model, we always restrict ourselves to OOB trees and measures. Which trees are used to construct these measures will differ from record to record. When these OOB performance measures are aggregated over the entire learn sample we will obtain overall OOB results reported by SPM.

## Are Random Forests Trees Overfit?

The topic of overfitting is especially interesting in the context of Random Forests. An individual Random Forests tree is massively overfit to the bootstrap sample on which it is grown; if we were to succeed in growing the tree out far enough to allow every record to have its own terminal node, we would both fit that data perfectly and have a model that could only reproduce the data used to construct it. But we are not at all interested in the performance of the Random Forests trees on their bootstrap training samples; it is only performance of previously unseen data that matters. What we see here is that a single Random Forests tree is typically a poor (but not hopeless) model, but that as progressively more of these trees are grown, the ensemble performs progressively better on previously unseen data. Interestingly, the overfitting on the training data is actually desirable and was originally discovered by Breiman in his research during 1995 on bootstrap aggregation models (the “bagger”, a simplified predecessor to Random Forests). When Breiman experimented with ensembles in which each tree was pruned back to try to eliminate the overfitting, the ensemble actually performed worse when applied to new data! Breiman’s explanation was this: overly large trees turn out to be variations of dynamically constructed nearest neighbor classifiers and these are low bias, high variance predictors. By averaging many such predictors we retain the low bias and eliminate the high variance (this is what averaging always accomplishes). Pruning the trees would increase the bias and this cannot be repaired by averaging; therefore, unpruned trees are preferred.

## Principal Random Forests Settings

### How many trees?

The first decision to make is how many trees to grow. If all we are interested in is accuracy, we can often get away with a relatively small number of trees. You will see this again and again when observing the performance profile of a forest: relatively rapid decline of an error measure over the first few dozen trees and then a slow improvement in performance over the next several hundred. Our default of 200 trees is based on such experience but it is important to experiment.

Having more trees than needed to achieve almost optimal predictive performance is critical when it comes to some other aspects of Random Forests. Variable importance rankings and proximity measures are best derived from large forests and we will discuss these topics subsequently.

### How many variables to consider at every node?

This control is ultra-important and to get the best out of Random Forests we advise you to experiment. In Breiman's earliest work on Random Forests (in his 1999 paper) he suggested that picking one variable at random to be the splitter every time we need to split a node would yield acceptable results. It should not take much experimentation on real data to convince yourself this is not correct. The number of predictors to consider for splitting at each node should almost always be greater than 1 and might even be equal to the number of predictors available (but see our comments below). In later research Breiman variously suggested the square root of  $K$  (where  $K$  is the number of predictors), as well as  $1/2*\sqrt{K}$ ,  $2*\sqrt{K}$ , and  $\log\text{-base-}2(K)+1$ .

For PRO, PRO EX, and ULTRA versions, SPM contains an "AUTOMATE RFNPRED" that will automatically rerun the analysis using a sequence of different values for this control. Additionally, the Random Forests tab in Model Setup now includes a drop-down menu with these options for the N Predictors control.

An important note about the number of variables considered at every node:

In our example data set we have 13 available predictors. So what happens if we set this control to 13? You would think that this would essentially be the "bagger" where we split nodes in the conventional CART way by considering all available predictors but taking bootstrap samples for growing each tree. In Random Forests, things work a little differently because the selection of the variables at random also follows the bootstrap model. This means that when we select variables at random, we select with replacement, allowing the same variable to be potentially selected more than once. As we discussed earlier if we select a bootstrap sample of 13 variables from a set of 13 candidates we will probably leave out 4 or 5 of the predictors. In other words, as Random Forests is currently organized we cannot make it deliver a pure bagger – the model will always be a form of true Random Forests.

### How large should one allow the Random Forests trees to grow?

The maximum size of the Random Forests trees is controlled by "parent node minimum cases" which defaults to 2 for classification models and 5 for regression models. This states that a node with as few as two (or five) data records is still eligible to be split. You might be tempted to increase this value especially for large data sets. You should keep in mind, however, that Random Forests delivers superior performance with the largest possible trees.

## Random Forests in TreeNet

While the Random Forests engine provides speed and accuracy, digging deeper into predictor relationships isn't available through summary results or post-processed plots. TreeNet gradient boosting machine offers partial dependency plots that better illustrate how individual variables are affecting response or interacting with each other. Utilizing the TreeNet engine to build a Random Forests model preserves the advantages of RF while providing the benefits of TreeNet reporting. The major tweaks are instructing TreeNet to model the original target variable at each stage (versus residuals) and removing the restriction on tree size. For full details and examples, consult the TreeNet manual.