# Random Forest Classification

## Introduction

If you have played the child's game \*20 Questions\* [^71] where the answers are \*Yes/No\* (true/false), then this may help you understand Random Forest classification? Consider for a moment all the questions that could be asked are all branches of a large tree. This tree of all possible items is akin to a decision tree. Now imagine a person plays this game tens, hundreds, or even thousands of times. After many games, one would learn the \*best\* questions to ask and the quickest way to win. Random Forests are very much like the child's game \*20 Questions\*, which is played over and over in the hope of finding the best questions such that you may win as quickly as possible.

[^71]:https://en.wikipedia.org/wiki/Twenty\_Questions

A Random Forest classifier is a meta classifier built on nothing more than a large number of Decision Trees \*grown\* under specific conditions. Similar to a meta-analysis of clinical data, the Random Forest uses a broad set of previously constructed Decision Trees processed into an ensemble learning tool.

Let us start with the more fundamental explanation of a Decision Tree learning system. A Decision Tree is a set of nested \*If-Then\* logical questions designed to provide an efficient way of steering a course through a set of narrowing branches to come to an answer or a definition of a class. The simplest decision tree has a root node, which is the first \*If-Then\* question, which splits to only two offspring, daughter leaves. A leaf is simply the terminal point for the branching tree, which can contain one or more members.

It is common to discuss decision trees in terms of how deep it may be built. Depending on the application, it may be advantageous to construct a tree with a minimal number of nodes. A one-node tree is referred to as a stump. Alternatively, a fully developed tree with every item on a leaf allows a decision-maker to find every unique object or observation. For instance, a tree that is only one node may be produced in the hope of finding the best first question or a \*strong classifier\*. Conversely, fully defined trees with one item per leaf may be inefficient and may need to be \*pruned\* like a tree out of control. As you see the tree analogy has its roots in everyday objects

Since this work uses R, the algorithm I choose utilizes the \*ranger\* software package. "Ranger is a fast implementation of Random Forests, particularly suited for high dimensional data." [^72]

[^72]:https://cran.r-project.org/web/packages/ranger/index.html

## Outline of Ranger Random Forest Algorithm

As mentioned earlier, a Random Forest Classifier is a Supervised learning technique. We, therefore, require a labeled pair of observations.

\*\*Data set\*\*: $(X\_1, y\_1), (X\_2 , y\_2), ~. . ., ~(X\_N , y\_N); ~~~y \in \{1, ..., ~C\}$

Where $C$ is the number of classes, $X$ is a matrix of $m$ observations (rows) and $n$ features (columns), such that $x\_{ij} \in \Re$.

The Random Forest process starts by choosing a sample from the original data set for a complete run. This process is called Bootstrapping. Bootstrapping uses a random number generator to sample from the set of initial observations producing a new subset, i.e., drawing a sample of size N with replacement. The subset is then partitioned into two further sets. One set is approximately 66% the size of N and is the training set. The second set, which is 33% of N in size, is used as a testing set later on. Although it is possible to change this proportion in other software packages, this value is 'factory set' in Ranger.

Once the training set is produced, the splitting process may start. The first step in this process is to determine the number of features which will be used for optimum splitting. Using R, the hyperparameter is called the `mtry`. Using a simple grid type search, one can find an optimum value of `mtry` is the square root of the number of features in the experiment, i.e., `mtry` = $\sqrt{M}$, $M$ is the number of features.

The splitting process has two additional hyperparameters, which can be set by the researcher. These hyperparameters are called `split-rule` and `min.node.size`. The `split-rule` determines which calculation will be used to measure the split purity. In general, there are two methods for calculating the purity of splits for decision trees. One method is the \*Gini coefficient\*, while the second common method is the \*entropy coefficient\*.

\*\*Gini coefficient\*\*:

$$G\_1 ~=~ 1 - \sum\_{k=1}^{n} (X\_k - X\_{k-1}) (Y\_k + Y\_{k-1})$$