**# Predicting Stock Movement with Random Forest**

Random Forest is a machine learning ensemble method that is widely used because of its flexibility, simplicity, and often quality results. In this tutorial, we will use the Random Forest algorithm to build a classification model that will help us predict whether a stock will close higher or lower based on range of technical indicators.

**## Background: What is Random Forest?**

Now, up above I gave you a simple definition of what Random Forest is. However, I think it's important to delve in more detail about it before we start coding our model. Now in short, Random Forests is a supervised machine learning algorithm that uses multiple decision trees in aggregate to help make more stable and accurate predictions.

Decision Trees are the building blocks of Random Forest, so to understand Random Forest we first need to understand Decision Trees.

**### Understanding Random Forest: Decision Trees**

Every day of your life, you make decisions. Your decisions can range from what to eat for lunch to what time you leave for work to avoid traffic. However, have you ever taken a step back to understand how you arrive at the decisions you make?

Sometimes you probably do but other times it feels like auto-pilot and you just do it. Well, normally, most decisions are based on a set of rules that you follow to arrive at the decision. For example, if I am hungry, I might have a set of rules I used to determine what I’ll eat. Let’s list my rules:

1. Rule 1: Determine if I am hungry.
2. Rule 2: Determine if I have money.
3. Rule 3: Determine if I want something light or heavy.

If you notice, we flow from one rule to next. In other words, I first must determine if I am even hungry, if I am then I can proceed to rule number 2 and determine if I have money. There is a flow from one rule to the next.

Additionally, I can further specify my rules by adding criteria to them. For example, at rule number 2 I can have multiple possible levels of having money $50.00, $25.00, or $10.00. Then based on that I can then move to rule number 3.

Let’s walk through a simple example. First, I determine I’m hungry, so I then proceed to rule 2. I determine I have $10.00 to my name, so I can move to rule 3. After determining I have $10.00, I decide I want something light, so I decide to go buy soup.

Well if you were able to follow all of that you need have the foundation of a Decision Tree. **Decision Trees is a flowlike chart structure where each node of the tree is used to test an attribute of the object**.

The node in our example up above would represent one of the rules we define. The attribute would be the value we were testing. For example, I tested my net worth (the amount of money I had) and I also tested my hunger level. The object we are testing is me! I’m the one who get’s tested on all the different rules.

**### Understanding Random Forest: Decision Trees Components**

To provide a more formal definition of everything, let’s walk through some points again. The first thing I want to mention is that for Decision Trees there are 3 kinds of nodes and 2 kinds of branches.

***Decision Node:***

Whenever I have to make a choice, we will define this as a “Decision Node”, so, for example, one decision node is testing whether I have money or not. A decision node is a point in the tree where a choice must be made.

***Root Node:***

The starting point of our tree is called the “Root Node”, so in our example up above determing if I am hungry or not is the root node. The root node represents the entire population or sample which is then divided into two or more homogenous sets.

***Leaf/Terminal Node:***

When we reach the end of our tree and we don’t have to make anymore decisions we have reached a “Leaf Node” also called a “Terminal Node”. In my example up above, getting soup would represent a terminal/leaf node. Nodes that do not split are Leaf/Terminal nodes.

***Decision Branches:***

The branches extending from a Decision Node are called “Decision Branches” and represent one of the possible alternative paths we can take. It’s very important to understand that each path must be mutually exclusive, in other words, if I take one path I can’t take the other. Additionally, they are collectively exhaustive, in other words, I’ve provided all possible paths.

***Splitting:***

The process of dividing a node into two or more sub-nodes, for example we split on net-worth.

***Parent and Child Node:***

A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node.

**### Understanding Random Forest: How Decision Trees Work**

Decision Trees work by first “Splitting” the data into subsets. The splits are formed on a feature of the data set.

**#### How Decision Trees Work: How is a Split Determined?**

The goal is to split/partition the data until each node is homogenous in data, or as little "impurity".

**#### How Decision Trees Work: How is Impurity Calculated?**

There are multiple functions you can use to calculate impurity but one of the most commonly used ones is the **Gini Index**. However, you could also use Bayes Error or Cross Entropy Function.

**#### How Decision Trees Work: Calculating Gini Index?**

Gini index or Gini impurity measures the degree or probability of a variable being wrongly classified when it is randomly chosen. But what is meant by ‘impurity’? If all the elements belong to a single class, then it can be called pure. The degree of Gini index varies between 0 and 1, where 0 denotes that all elements belong to a certain class or if there exists only one class, and 1 denotes that the elements are randomly distributed across various classes. A Gini Index of 0.5 denotes equally distributed elements into some classes.

Once the Decision Tree is built by splitting the data across the specified number of features, we then begin the “Pruning” process. Pruning involves reducing the size of the Decision Tree by converting certain branches into leaf nodes and removing leaf nodes.

You may be asking, “Why would we want a smaller decision tree?”. The reason is simple, Decision Trees have an easy time over fitting the data. In fact, the larger the tree the more likely this is to happen. Generally, a simpler decision tree avoids over-fitting the data.

### **Understanding Random Forest: Advantages and Disadvantages of Decision Trees**

Here are some disadvantages and advantages of decision trees:

**Advantages:**

1. Intuitive in nature.
2. Have value even with little hard data.
3. Help determine worst, best and expected values for different scenarios.
4. Use a white box model. If a given result is provided by a model.
5. Can be combined with other decision techniques.
6. The number of hyper-parameters to be tuned is almost null.

**Disadvantages:**

1. Instability: Even small changes to the input data can have dramatic changes to the overall structure of the decision tree.
2. They are often relatively inaccurate. Many other predictors perform better with similar data.
3. For data including categorical variables with different number of levels, information gain in decision trees is biased in favor of those attributes with more levels.
4. Calculations can get very complex, particularly if many values are uncertain and/or if many outcomes are linked.

**### Understanding Random Forest: Bootstrap Aggregation**

Sometimes it can be hard to model a problem with a single decision tree. However, sometimes when we use multiple decision trees to help make predictions, we can more easily model a problem.

Bagging or bootstrap aggregation is a technique for reducing the variance of an estimated prediction function. Bagging seems to work especially well for high-variance, low-bias procedures, such as trees. For

regression, we simply fit the same regression tree many times to bootstrapsampled versions of the training data and average the result. For classification, a committee of trees each cast a vote for the predicted class.

Bagging reduces the variance of the base learner but has limited effect on the bias. The old idea of 2 minds are better than one, in my opinion, perfectly sums up bagging.

**### Understanding Random Forest: Bootstrap Aggregation Flaws**

1. The problem with Bagging algorithm is it's using Decision Trees.
2. Decision Trees uses Gini-Index, a greedy algorithm to find the best split.
3. We end up with trees that are structurally similar to each other. The trees are highly correlated among the predictions.
4. We will see that Random Forest addresses this problem

**### Understanding Random Forest: Why Use it?**

After understanding what Decision Trees are and some of their disadvantages and advantages are, we probably want to know how these decision trees fit into Random Forest. Additionally, we can obviously see that with a multiple decision trees we can better model the data we have through bagging. However, bagging can also have its own issues because of trees that are highly correlated with each other.

What exactly does Random Forest do to fix these issues? **Well, it uses bagging but it what it does instead is to create trees that have no correlation or weak correlation**. To keep it simple at this point, it is averaging many noisy but approximately unbiased models (Decision Trees) to reduce the overall variance of the model. What Random Forest tries to do is avoid over-fitting of the data and improve the stability and accuracy of predictions.

**### Understanding Random Forest: How it works?**

**To classify a new object from an input vector, put the input vector down each of the trees in the forest.** Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

In simple terms, we do the following:

1. Build the forest.
2. Make a “prediction” by taking the highest vote of the decision trees.

By using the votes of multiple decision trees, we are leveraging a concept called “Ensemble Learning”. I’ll get into what “Ensemble Learning” is later.

In more complicated terms we do the following:

1. Take a random sample of size N with replacement from the data.
2. Take a random sample **without replacement** of the predictors.
3. Construct the first CART partition of the data.
4. Repeat Step 2 for each subsequent split until the tree is as large as desired. Do not prune.
5. Repeat Steps 1–4 many times (e.g., 500).

It’s important to understand that step 2 is how Random Forest deals with the highly correlated trees generated by bagging. Also, it’s important to realize that there is no magic number for “many times” it depends on your data. Best advice is to evaluate the model and use cross validation.

**### Understanding Random Forest: How it builds the Forest?**

Each tree is grown as follows:

1. If the number of cases in the training set is N, sample N cases at random - but with replacement, from the original data. This sample will be the training set for growing the tree.
2. If there are M input variables, a number **m<<M** is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. **There is no pruning**.

The forest error rate depends on two things:

1. The correlation between any two trees in the forest. Increasing the correlation increases the forest error rate.
2. The strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

**Reducing m reduces both the correlation and the strength. Increasing it increases both**. Somewhere in between is an "optimal" range of m - usually quite wide.

Using the Out-Of-Bag error rate a value of m in the range can quickly be found. This is the only adjustable parameter to which random forests is somewhat sensitive.

To understand and use the various options, further information about how they are computed is useful. Most of the options depend on two data objects generated by random forests.

When the training set for the current tree is drawn by sampling with replacement, about one-third of the cases are left out of the sample. This oob (out-of-bag) data is used to get a running unbiased estimate of the classification error as trees are added to the forest. It is also used to get estimates of variable importance.

After each tree is built, all the data are run down the tree, and proximities are computed for each pair of cases. If two cases occupy the same terminal node, their proximity is increased by one. At the end of the run, the proximities are normalized by dividing by the number of trees. Proximities are used in replacing missing data, locating outliers, and producing illuminating low-dimensional views of the data.

**### Understanding Random Forest: What is Ensemble Learning?**

An Ensemble Learning model is a model in which decisions are used from MULTIPLE MODELS to improve the overall performance of the model. The old idea that 2 minds are better than 1, perfectly summarizes ensemble learning. We use the results of multiple models to get a better idea of what the true answer is. Our Random Forest algorithm uses Bagging to help improve performance.

**### Understanding Random Forest: What is Supervised Learning?**

In machine learning, we have two categories of learning. Supervised learning and unsupervised learning. With unsupervised learning, we don't supervise the model and instead allow it to discover information on its own. We do this by providing an "UNLABELED" data set that doesn't tell the model what category or value is the "correct" answer.

With supervised learning, we provide the model with a "LABELED" data set which tells the model what the "correct" value it should be. Random Forest is an example of a supervised learning algorithm because we provide the model a labeled data set.