**Understanding Random Forest**

**How the Algorithm Works and Why it Is So Effective**

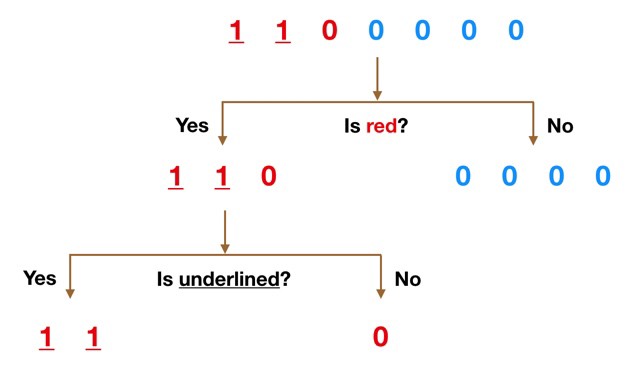
A big part of machine learning is classification — we want to know what class (a.k.a. group) an observation belongs to. The ability to precisely classify observations is extremely valuable for various business applications like predicting whether a particular user will buy a product or forecasting whether a given loan will default or not.

Data science provides a plethora of classification algorithms such as [logistic regression](https://towardsdatascience.com/understanding-logistic-regression-using-a-simple-example-163de52ea900), support vector machine, naive Bayes classifier, and decision trees. But near the top of the classifier hierarchy is the random forest classifier (there is also the random forest regressor but that is a topic for another day).

In this post, we will examine how basic decision trees work, how individual decisions trees are combined to make a random forest, and ultimately discover why random forests are so good at what they do.

**Decision Trees**

Let’s quickly go over decision trees as they are the building blocks of the random forest model. Fortunately, they are pretty intuitive. I’d be willing to bet that most people have used a decision tree, knowingly or not, at some point in their lives.



Simple Decision Tree Example

It’s probably much easier to understand how a decision tree works through an example.

Imagine that our dataset consists of the numbers at the top of the figure to the left. We have two 1s and five 0s (1s and 0s are our classes) and desire to separate the classes using their features. The features are color (red vs. blue) and whether the observation is underlined or not. So how can we do this?

Color seems like a pretty obvious feature to split by as all but one of the 0s are blue. So we can use the question, “Is it red?” to split our first node. You can think of a node in a tree as the point where the path splits into two — observations that meet the criteria go down the Yes branch and ones that don’t go down the No branch.

The No branch (the blues) is all 0s now so we are done there, but our Yes branch can still be split further. Now we can use the second feature and ask, “Is it underlined?” to make a second split.

The two 1s that are underlined go down the Yes subbranch and the 0 that is not underlined goes down the right subbranch and we are all done. Our decision tree was able to use the two features to split up the data perfectly. Victory!

Obviously in real life our data will not be this clean but the logic that a decision tree employs remains the same. At each node, it will ask —

What feature will allow me to split the observations at hand in a way that the resulting groups are as different from each other as possible (and the members of each resulting subgroup are as similar to each other as possible)?

**The Random Forest Classifier**

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an [ensemble](https://en.wikipedia.org/wiki/Ensemble_learning). Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model’s prediction (see figure below).



Visualization of a Random Forest Model Making a Prediction

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:

**A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.**

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. **The reason for this wonderful effect is that the trees protect each other from their individual errors** (as long as they don’t constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.
2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.