**RANDOM FOREST**

**What is a Random Forest?**

Random forest is an ensemble tool which takes a subset of observations and a subset of variables to build a decision trees. It builds multiple such decision tree and amalgamate them together to get a more accurate and stable prediction. This is direct consequence of the fact that by maximum voting from a panel of independent judges, we get the final prediction better than the best judge.

**Illustration**

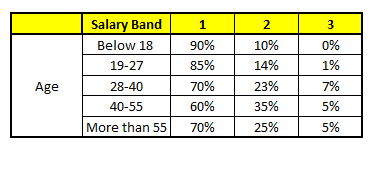
We want to classify in which salary band an individual would fall into, based on certain variables:

Salary bands:

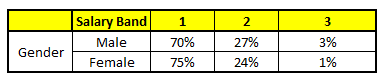
* Band 1: Below $40,000
* Band 2: $40,000 – 150,000
* Band 3: More than $150,000

Random forest builds certain number of trees each having certain number of variables:

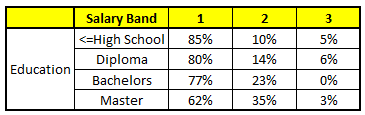
**CART 1: Age**

[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/06/rf1.png)

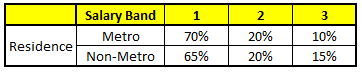
**CART 2: Gender**

[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/06/rf2.png)

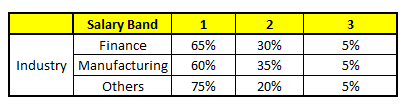
**CART 3: Education**

[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/06/rf3.png)

**CART 4: Residence**

[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/06/rf4.png)

**CART 5: Industry**

[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/06/rf5.png)

Using these 5 CART models, we need to come up with singe set of probability as to which band the person belongs to.

**Parameters / levers to tune Random Forests**

Parameters in random forest are either to increase the ***predictive power*** of the model or to make it ***easier to train*** the model.

**1. Features which make predictions of the model better**

***1.1 max\_features:***

These are the maximum number of features Random Forest is allowed to try in individual tree. There are multiple options available:

* **Auto/None**: This will simply take all the features which make sense in every tree. Here we simply do not put any restrictions on the individual tree.
* **sqrt**: This option will take square root of the total number of features in individual run. For instance, if the total number of variables are 100, we can only take 10 of them in individual tree.”log2″ is another similar type of option for max\_features.
* **0.2**: This option allows the random forest to take 20% of variables in individual run. We can assign and value in a format “0.x” where we want x% of features to be considered.

*How does “max\_features” impact performance and speed?*

*Increasing max\_features generally improves the performance of the model as at each node now we have a higher number of options to be considered. However, this is not necessarily true as this decreases the diversity of individual tree which is the USP of random forest. But, for sure, you decrease the speed of algorithm by increasing the max\_features. Hence, you need to strike the right balance and choose the optimal max\_features.*

***1.2 n\_estimators :***

This is the number of trees you want to build before taking the maximum voting or averages of predictions. Higher number of trees give you better performance but makes your code slower. You should choose as high value as your processor can handle because this makes your predictions stronger and more stable.

***1.3. min\_sample\_leaf:***

If you have built a decision tree before, you can appreciate the importance of minimum sample leaf size. Leaf is the end node of a decision tree. A smaller leaf makes the model more prone to capturing noise in train data. Generally, I prefer a minimum leaf size of more than 50. However, you should try multiple leaf sizes to find the most optimum for your use case.

**2. Features which will make the model training easier**

There are a few attributes which have a direct impact on model training speed. Following are the key parameters which you can tune for model speed:

***2.1. n\_jobs:***

This parameter tells the engine how many processors is it allowed to use. A value of “-1” means there is no restriction whereas a value of “1” means it can only use one processor.

***2.2. random\_state:***

This parameter makes a solution easy to replicate. A definite value of random\_state will always produce same results if given with same parameters and training data. I have personally found an ensemble with multiple models of different random states and all optimum parameters sometime performs better than individual random state.

***2.3. oob\_score:***

This is a random forest cross validation method. It is very similar to leave one out validation technique, however, this is so much faster. This method simply tags every observation used in different tress. And then it finds out a maximum vote score for every observation based on only trees which did not use this particular observation to train itself.

An ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model.

Decision trees suffer from high variance. A slightly different sample might give you entirely different splits; using decision trees for data interpretation or feature selection is an art at best (and for some data sets uselessly unreliable).

Decision trees are weak learners.

This is untrue. They're only used as weak learners in boosting because the tree depth is limited to some small constant.

Decision trees run fast even with lots of observations and variables

I don't know all the decision tree learning algorithms, but at least some of the common ones run in O(features \* samples \* splits). That's not terrible, but you can handle much larger data sets optimizing w/ stochastic gradient descent or coordinate descent.

Decision trees can easily handle unbalanced datasets.

This links to a post about bagging, which is not really specific to decision trees (but can be done with any learning algorithm)

Decision tree is sensitive to where it splits and how it splits. Therefore, even small changes in input variable values might result in very different tree structure.

**Decision trees intuition:**

Intuitively, it can be understood in this way. When there are too many decision nodes to go through before arriving at the result i.e number of nodes to traverse before reaching the leaf nodes is high, the conditions that you are checking against becomes multiplicative. That is, the computation becomes (condition 1)&&(condition 2)&&(condition 3)&&(condition 4)&&(condition5).

Only if all the conditions are satisfied, a decision is reached. As you can see, this will work very well for the training set as you are continuously narrowing down on the data. The tree becomes highly tuned to the data present in the training set.

But when a new data point is fed, even if one of the parameters deviates slightly, the condition will not be met and it will take the wrong branch.

**Bootstrap Method**

Before we get to Bagging, let’s take a quick look at an important foundation technique called the bootstrap.

The bootstrap is a powerful statistical method for estimating a quantity from a data sample. This is easiest to understand if the quantity is a descriptive statistic such as a mean or a standard deviation.

Let’s assume we have a sample of 100 values (x) and we’d like to get an estimate of the mean of the sample.

We can calculate the mean directly from the sample as:

***mean(x) = 1/100 \* sum(x)***

We know that our sample is small and that our mean has error in it. We can improve the estimate of our mean using the bootstrap procedure:

Create many (e.g. 1000) random sub-samples of our dataset with replacement (meaning we can select the same value multiple times).

Calculate the mean of each sub-sample.

Calculate the average of all of our collected means and use that as our estimated mean for the data.

For example, let’s say we used 3 resamples and got the mean values 2.3, 4.5 and 3.3. Taking the average of these we could take the estimated mean of the data to be 3.367.

This process can be used to estimate other quantities like the standard deviation and even quantities used in machine learning algorithms, like learned coefficients.

**Bootstrap Aggregation Algorithm (Bagging)**

A bootstrap is a sample of a dataset with replacement.

This means that a new dataset is created from a random sample of an existing dataset where a given row may be selected and added more than once to the sample.

It is a useful approach to use when estimating values such as the mean for a broader dataset, when you only have a limited dataset available. By creating samples of your dataset and estimating the mean from those samples, you can take the average of those estimates and get a better idea of the true mean of the underlying problem.

This same approach can be used with machine learning algorithms that have a high variance, such as decision trees. A separate model is trained on each bootstrap sample of data and the average output of those models used to make predictions. This technique is called bootstrap aggregation or bagging for short.

Variance means that an algorithm’s performance is sensitive to the training data, with high variance suggesting that the more the training data is changed, the more the performance of the algorithm will vary.

The performance of high variance machine learning algorithms like unpruned decision trees can be improved by training many trees and taking the average of their predictions. Results are often better than a single decision tree.

Another benefit of bagging in addition to improved performance is that the bagged decision trees cannot overfit the problem. Trees can continue to be added until a maximum in performance is achieved.

**LINK:**

[1]. <https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest-simplified/>

[2]. <https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/>

[3]. <http://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/>

[4]. <http://machinelearningmastery.com/implement-bagging-scratch-python/>

[5].