**RANDOM FOREST FOR TEXT CLASSIFICATION**

The “forest” it builds, is an **ensemble of Decision Trees**, most of the time trained with the “**bagging**” method. The general idea of the bagging method is that a combination of learning models increases the overall result (*more accurate and stable prediction.*).

Random Forests grows many classification trees. 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).

Random forest reduces variance of a large number of "complex" models with low bias.

**Features of Random Forests**

* It runs efficiently on **large data bases**.
* It can **handle thousands of input variables** without variable deletion.
* It gives estimates of **what variables are important** in the classification.
* It generates an internal unbiased estimate of the generalization error as the forest building progresses.
* It has an **effective method for estimating missing data** and **maintains accuracy when a large proportion of the data are missing.**
* It has methods for **balancing error in class population unbalanced data sets**.
* It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.
* It offers an experimental method for detecting variable interactions.

**Why Random forest algorithm**

To address why random forest algorithm. I am giving you the below advantages.

* The same **random forest algorithm** or the random forest classifier can use for both classification and the regression task.
* Random forest classifier will **handle the missing** values.
* When we have more trees in the forest, random forest classifier **won’t** **overfit** the model.
* Can model the random forest classifier for **categorical values** also.
* Instead of building a bagging-classifier and passing it into a decision-tree-classifier, you can just use the random-forest classifier class, which is more convenient and optimized for decision trees (bring extra randomness).
* Instead of searching for the best feature while splitting a node, it searches for the best feature among a random subset of features. This process creates a wide diversity, which generally results in a better model.
* Therefore, when you are growing a tree in random forest, only a random subset of the features is considered for splitting a node. You can even make trees more random, by using random thresholds on top of it, for each feature rather than searching for the best possible thresholds (like a normal decision tree does).
* The study of error estimates for bagged classifiers in Breiman [1996b], gives empirical evidence to show that **the out-of-bag estimate is as accurate as using a test set of the same size as the training set**. Therefore, using the out-of-bag error estimate removes the need for a set aside test set.

In general, the **more trees in the forest** the more robust the forest looks like. In the same way in the random forest classifier, the **higher the number** of trees in the forest gives **the high accuracy** results.

To model more number of decision trees to create the forest you are not going to use the same apache of constructing the decision with information gain or Gini index approach.

**Drawback**

The main limitation of Random Forest is that a large number of trees can make the algorithm to slow and ineffective for real-time predictions. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained. A more accurate prediction requires more trees, which results in a slower model.

## How random forests work

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](https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm" \l "ooberr)** 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 of the data are run down the tree, and [**proximities**](https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#prox) 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.

### The out-of-bag (oob) error estimate

After creating the classifiers (S trees), for each (Xi,yi) in the original training set i.e. T, select all Tk which does not include (Xi,yi). This subset, pay attention, is a set of boostrap datasets which does not contain a particular record from the original dataset. This set is called out-of-bag examples. There are n such subsets (one for each data record in original dataset T). OOB classifier is the aggregation of votes ONLY over Tk such that it does not contain (xi,yi).   
  
Out-of-bag estimate for the generalization error is the error rate of the out-of-bag classifier on the training set (compare it with known yi's).

**ALGORITHM**

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

#### Random Forest pseudocode:

1. Randomly select **“k”** features from total **“m”** features.
   1. Where **k << m**
2. Among the**“k”** features, calculate the node **“d”** using the best split point.
3. Split the node into **daughter nodes** using the **best split**.
4. Repeat **1 to 3** steps until “l” number of nodes has been reached.
5. Build forest by repeating steps **1 to 4** for “n” number times to create **“n” number of trees**.

Will the first 3 stages until we form the tree with a root node and having the target as the leaf node.

#### Random forest prediction pseudocode:

To perform prediction using the trained random forest algorithm uses the below pseudocode.

1. Takes the **test features** and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)
2. Calculate the **votes** for each predicted target.
3. Consider the **high voted** predicted target as the **final prediction** from the random forest algorithm.

**Parameters on which random forest model depends:**

**Increasing the Predictive Power**

**n\_estimators(total number of trees)**

**max\_features :** maximum number of features Random Forest is allowed to try in an individual tree.

**min\_sample\_leaf:**the number of leafs.

Random forest reduces variance of a large number of "complex" models with low bias(Decision Tree). We can see the composition elements are not "weak" models but too complex models. If you read about the algorithm, the underlying trees are planted "somewhat" as large as "possible". The underlying trees are independent parallel models. And additional **random variable selection** is introduced into them to make them even more independent, which makes it **perform better than ordinary bagging** and entitle the name "random".

* Logistic Regression
  + Pros
    - low variance
    - provides probabilities for outcomes
    - works well with diagonal (feature) decision boundaries
    - NOTE: logistic regression can also be used with kernel methods
  + Cons
    - high bias
* Decision Trees
  + Regular (not bagged or boosted)
    - Pros
      * easy to interpret visually when the trees only contain several levels
      * Can easily handle qualitative (categorical) features
      * Works well with decision boundaries parellel to the feature axis
    - Cons
      * prone to overfitting
      * possible issues with diagonal decision boundaries
  + Bagged Trees: train multiple trees using bootstrapped data to reduce variance and prevent overfitting
    - Pros
      * reduces variance in comparison to regular decision trees
      * Can provide variable importance measures
        + classification: Gini index
        + regression: RSS
      * Can easily handle qualitative (categorical) features
      * Out of bag (OOB) estimates can be used for model validation
    - Cons
      * Not as easy to visually interpret
      * Does not reduce variance if the features are correlated
* Random Forest (Successor of bagged tress more random)
  + Pros
    - Decorrelates trees (relative to bagged trees)
      * important when dealing with multiple features which may be correlated
    - reduced variance (relative to regular trees)
  + Cons
    - Not as easy to visually interpret

**For models with millions of sparse features, logistic regression will be much faster to train and execute, and is less prone to overfitting (as are all linear models.) Google’s ad prediction system**[**[1]**](https://www.quora.com/When-should-random-forest-be-used-over-logistic-regression-for-classification-and-vice-versa#CTZhS)**used an advanced form of logistic regression (I don’t know if that is still the case.)**

All that said, RF is a versatile algorithm (it can also do regression), and can be expected to outperform LR on many medium-sized tasks. It can handle categorical and real-valued features with ease—little to no pre-processing required. With proper cross-validation technique, they are readily tuned.

You'll want to keep in mind though that a logistic regression model is searching for a single linear decision boundary in your feature space, whereas a decision tree is essentially partitioning your feature space into half-spaces using *axis-aligned* linear decision boundaries. The net effect is that you have a non-linear decision boundary, possibly more than one.

**This is nice when your data points aren't easily separated by a single hyperplane, but on the other hand, decisions trees are so flexible that they can be prone to overfitting(Solution Random Forest).Logistic regression tends to be less susceptible (but not immune!) to overfitting**.

Lastly, another thing to consider is that decision trees can automatically take into account **interactions between variables**, e.g. xyxy if you have two independent features xx and yy. With logistic regression, you'll have to manually add those interaction terms yourself.

So, you have to ask yourself:

* what kind of decision boundary makes more sense in your particular problem?
* how do you want to balance bias and variance?
* are there interactions between my features?

Conclusion

Random forest solves the overfitting problem of decision tress. It uses bagging technique to randomly make sub tress and in addition to It, it also selects features randomly for every subtree. (very useful for multiple features)

Future Work

Logistic vs Random Forest (Multiclass Text Classification)

<https://www.kaggle.com/umeshnarayanappa/exploring-the-data/data>

<https://www.kaggle.com/c/spooky-author-identification/>

**References**

<https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#overview>

<http://davidsiroky.faculty.asu.edu/PA2016.pdf>

https://github.com/ctufts/Cheat\_Sheets/wiki/Classification-Model-Pros-and-Cons

<https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd>