Random Forests*†

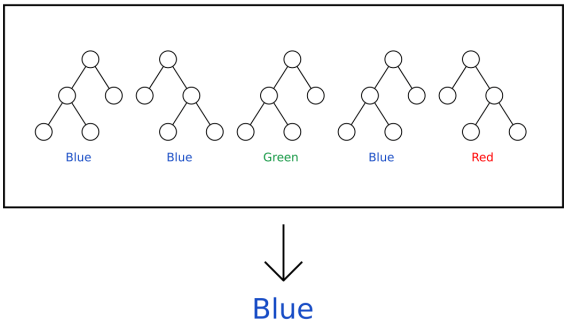
Saahith Janapati

October 2020

1 Overview

The random forest is an ensemble machine learning model based off of decision trees. A random forest is an aggregation of many **unique** decision trees; when given a new input to classify, the random forest hands the input off to all of its decision trees, which all return their own prediction. The random forest will then output what the majority of the decision trees classified the input as. You may think of all the decision trees as voting on the input, and the random forest outputting the majority vote.

Random forests usually outperform individual decision trees, since they are prone to overfitting. This is further discussed in following sections.

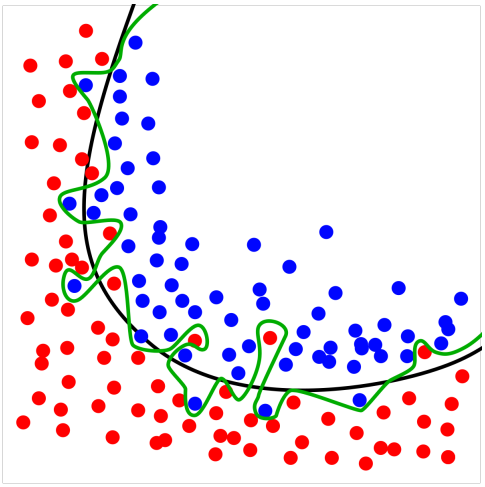


2 Overfitting

Overfitting occurs when a machine learning fits its training dataset too closely, and consequently fails to generalize to other datasets. The overfitting problem is exemplified in the following diagram.

*and Overfitting along with Regression with Decision Trees

†Portions of this lectures were adapted from Sylesh Suresh's and Mihir Patel's Lecture on Random Forrests



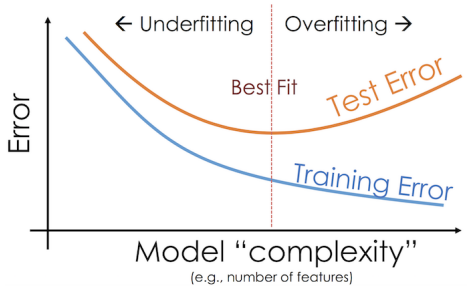
Consider the green and black lines as two seperate machine leaning models. The goal of these two models is to classify the dots as blue or red. The training dataset is displayed above.

The green line is able to separate the training data perfectly. The black line makes some mistakes in several places. If you had the choice, which model would you choose?

The black model would be the better option. The majority of its misclassifications are simply due to random noise in the training data. The green line learns the seperation between the two groups in the training dataset "too well". You may think of the green line as having memorized the training set. It will most likely fail to generalize to new inputs in your testing dataset.

Thus, the general separation between the two groups is more accurately encoded by the black model.

Overfitting can be due to many factors, such as a small dataset, overly complex model, or simply due to training your model for too long. In the following diagram, the x-axis represents model complexity and the y-axis represents error. As you can see, a more complex model is not always better. A model that is too complex for a certain dataset can (as we mentioned before) "memorize" the training dataset and will thus have a higher test error (indicative of the model's failure to generalize to other datasets).



2.1 Overfitting in Decision Trees

Decision trees are prone to the overfitting problem. We can restrict overfitting by limiting maximum number of levels that the decision tree can grow to. In theory, for any dataset, a decision tree can perfectly classify every single input when no bound is provided for the number of levels the tree can grow. To combat this issue, we can provide a bound for the maximum depth of a node. If the depth of the tree branch exceeds this value, we will stop training. This process of providing a bound for the maximum depth is called pruning.

Or you could just try training a random forest instead! As previously mentioned, random forests are an aggregation of many unique decision trees (we will discuss how to train unique decision trees in the next section). Although the decision trees are poor performers on their own, when combined together, they regularly outperform a single, more complex decision tree.

3 Random Forests

As previously mentioned, Random Forests are aggregations of many unique decision trees. But how do we ensure that these decision trees are unique? We can accomplish this by using bootstrapped samples and by randomly choosing the features that each tree can split.

3.1 Bootstrapping

Bootstrapping is method of generating small random samples from a population. A bootstrapped sample is simply a sample randomly drawn (with replacement) from a population. Let us walk through a simple example:

$$D_{population} = \big[0.1, 0.3, 0.5, 0.8, 0.2\big]$$

Let's calculate a bootstrapped sample of length 3. When we randomly sample from the population for the first time, we happen to get the value 0.5.

$$D_{population} = \big[0.1, 0.3, 0.5, 0.8, 0.2\big]$$

$$D_{sample} = \big[0.5\big]$$

The next time we randomly sample, we get the value 0.8.

$$D_{population} = \big[0.1, 0.3, 0.5, 0.8, 0.2\big]$$

$$D_{sample} = \big[0.5, 0.8\big]$$

The last time we randomly sample, we again get the value 0.5 (remember, we are sampling with replacement).

$$D_{population} = \big[0.1, 0.3, 0.5, 0.8, 0.2\big]$$

$$D_{sample} = \big[0.5, 0.8, 0.5\big]$$

We now have our bootstrapped sample.

3.2 Building our Random Forest

Prior to constructing the random forest, we have to determine three hyperparameters. Hyperparameters are settings that we, the developers of the machine learning model, can set. These hyperparameters alter various aspects of the algorithm's behaviour. For the random forest, the three hyperparameters are:

1. k = number of decision trees the random forest contains
2. n = size of our bootstrapped samples
3. d = number of distinct features we allow our model to split on

3.2.1 Pseudocode

1. Repeat k times (for k decision trees):
   (a) Draw a bootstrap sample of size n from original training dataset
   (b) Randomly select d features for tree to split on
   (c) Train decision tree (allowing it to split only using the d features you previously selected) on bootstrapped sample
   (d) Save decision tree
2. Given a new input, feed in the new input to all decision trees and output the majority vote

3.3 Example

Let us walk throuh creating a single decision tree for a random forest with the hyperparameters $n = 5$ and $d = 3$. Note that we don't need to worry about the hyperparameter k now, since k controls the number of decision trees and we are just creating one.

id	x1	x2	x3	x4	x5	x6	y
1	0	1	0	0	1	0	-1
2	1	0	0	1	1	1	-1
3	0	1	0	1	1	0	+1
4	0	1	0	1	1	1	+1
5	1	0	0	0	1	1	-1
6	0	0	1	1	0	0	+1
7	0	0	1	1	0	0	+1

Let us first create a bootstrap sample of size n (5). After random sampling with replacement, the bootstrap sample to train our decision trees is the following.:

id	x1	x2	x3	x4	x5	x6	y
1	0	1	0	0	1	0	-1
1	0	1	0	0	1	0	-1
3	0	1	0	1	1	0	+1
5	1	0	0	0	1	1	-1
7	0	0	1	1	0	0	+1

Next, we need to randomly chose d (3) features for our decision tree to split on. Let's say that we randomly chose the features $x2$, $x4$, and $x5$. Our training dataset for our single decision tree will now be:

id	x2	x4	x5	y
1	1	0	1	-1
1	1	0	1	-1
3	1	1	1	+1
5	0	0	1	-1
7	0	1	0	+1

We now train a decision tree on this subset of our training data, splitting on the feature and threshold that yield the highest information gain. We would then repeat this process k times, yielding k unique decision trees.

3.4 Selecting Hyperparameters

Prior to creating a random forest, it is important to first select hyperparameters.

- Selecting k: k is the number of decision trees in the random forest. Although a higher value of k will lead to an increase in performance, it will also cause an increase in required compute power (since you are going to be training more decision trees).
- Selecting n: n is the size of the bootstrap sample. A smaller n will decrease the overfitting of your model (since the individual decision trees will be trained on smaller datasets). However, this may also cause your performance to drop.
- Selecting d: d is the number of features you allow each decision tree to train on. Similar to n, setting d to smaller values will decrease overfitting (since it will cause your trees to have less features to split on), but it may also cause a decrease in the model's performance. A common practice is to set d to the square root of the total number of features or to the base 2 logarithm of the total number features.

3.5 Extra-Trees

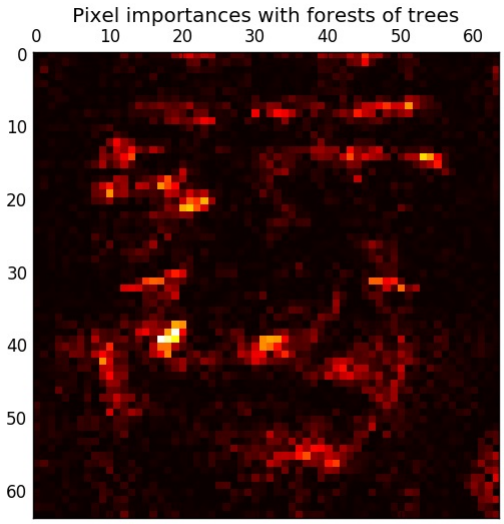
Another measure that may be taken to prevent overfitting when building a random forest is to randomly select a threshold value for each feature and select the feature-threshold pair that has the highest information gain. Random forests

built with such decision trees are termed Extra-Trees (Extra Randomized Trees). This technique is used to decrease over fitting, but it may also lead to a drop in performance.

3.6 Feature Importance

A neat property of decision trees (and consequently random forests) is that "important" features are used to split objects higher up the tree (this property arises from our use of information gain to calculate the best features and thresholds to split on). Thus, by finding the depth of a feature in a tree, we can calculate its importance. In a random forest, we can take the average depth of a certain feature across all the trees to rank it with other features.

This property is exemplified in the image below. The image is a graphical representation of the importance of features (in this case, the features are pixels) to a random forest, which classifies whether or not a image is a face. Brightly colored pixels are of greater importance.



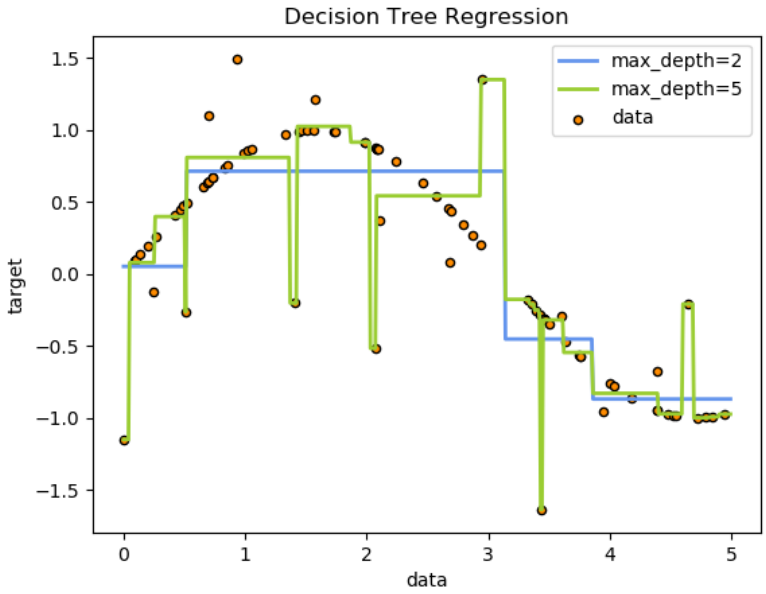
4 Regression with Decision Trees and Random Forests

Up till now, we have only discussed using decision trees and random forests for classification tasks; that is seperating objects into discrete classes. However, decision trees and random forests can also be used for regression tasks (in which we need to output a continuous variable). An example of a regression task is stock prediction.

In order to train a decision tree for regression, we use a criterion such as MSE (mean squared error) instead of gini impurity to determine the quality of a split. We continue to use the same information gain formula. Trees are grown until they reach a max depth, which is preset by the user (setting a max depth is quite important for regression tasks, since trees will normally grow until all their leaves are pure).

When given a new input, a decision tree will eventually find the leaf which that input falls into. The tree will then output the mean of all the labels of the training samples in that leaf as the final output.

Below is a graphical representation of a decision tree trained to perform regression. Notice the effect of max depth on the complexity of the models.



A random forest for regression will simply output the average all the outputs of each of its decision trees.

5 Bibliography

- Sylesh Suresh and Mihir Patel's Lecture on Random Forests
- Victor Zhou's Random Forests Blog Post
- Principles and Techniques of Data Science
- Scikit-Learn Blog: Pixel importances with a parallel forest of trees
- Scikit-Learn Blog: Decision Tree Regression