Random Forests

Anhthy Ngo

1 Overview

Random forest is an ensemble learning algorithm. The basic premise of the algorithm is that building a small decision-tree with few features is a computationally cheap process. If we can build many small, weak decision trees in parallel, we can then combine the trees to form a single, strong learner by averaging or taking the majority vote. In practice, random forests are often found to be the most accurate learning algorithms to date.

The algorithm works as follows: for each tree in the forest, we select a bootstrap sample from $\mathbf{S}$ where $\mathbf{S}^{(i)}$ denotes the $\mathbf{i}$th bootstrap. We then learn the a decision-tree using a modified decision-tree learning algorithm. The algorithm is modified as follows: at each node of the tree, instead of examining all of the feature-splits, we randomly select some subset of the features $f \subseteq F$, where F is the set of all features. The node splits on the best feature in f rather than F. In practice f is much smaller than F. Deciding on which feature to split is oftentimes the most computationally expensive aspect of decision tree learning. By narrowing the set of features, we drastically speed up the learning of the tree.

2 Bagging

Bagging (also known as Bootstrap AGGregatING) is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It reduces variance and helps to avoid overfitting.

Given a training set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, bagging generates m new training sets $\mathcal{D}_1, \ldots, \mathcal{D}_m$ by sampling from $\mathcal{D}$ uniformly and with replacement. The training set $\mathcal{D}_i$ is expected to have $\approx 63.2\%$ of the unique examples in $\mathcal{D}$, the rest being duplicates. Then, m models are fitted using the above m bootstrap samples to obtain a sequence of m classifiers $h_1(x), \ldots, h_m(x)$.

The final aggregate classifier can be

- for regression

$$\bar{f}(x) = \frac{1}{m} \sum_{i=1}^{m} h_i(x)$$

the average of f_i for $i = 1, \ldots, m$.

- for binary classification where $y \in \{-1, 1\}$

$$\bar{f}(x) = \text{sign}(\sum_{i=1}^{m} h_i(x))$$

essentially a "majority vote", assuming that votes for binary classes are unequal.

2.1 Advantages of Bagging

- Bagging reduces variance by building more bootstrapped classifiers. Combinations of multiple classifiers decrease variance, especially in the case of unstable classifiers, and may produce a more reliable classification than a single classifier.

- Bagging provides an *unbiased* estimate of the test error, which we refer to as the *out-of-bag* (OOB) error. The OOB error estimate is almost identical to that obtained by K-fold cross validation. One can show that on average, each bagged tree makes use of around two-thirds of the observations. The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.

  We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around $\frac{m}{3}$ predictions for the ith observation. In order to obtain a single prediction for the ith observation, we can average these predicted responses (if regression is the goal) or can take a majority vote (if classification is the goal). This leads to a single OOB prediction for the ith observation.

  More formally, for each training point $(x_i, y_i) \in \mathcal{D}$ let $S_i = \{k \mid (x_i, y_i) \notin \mathcal{D}\}$ - in other words, S_i is a set of all training sets $\mathcal{D}_k$ which does not contain (x_k, y_k). Let the averaged classifier over all these datasets be

$$\tilde{h}_i(x) = \frac{1}{|S_i|} \sum_{k \in S_i} h_k(x)$$

  The out-of-bag error simply becomes the average error/loss that all the classifiers yield

$$\epsilon_{OOB} = \frac{1}{n} \sum_{(x_i, y_i) \in \mathcal{D}} \ell(\tilde{h}_i(x_i), y_i)$$

  This is an estimate of the test error, because for each training point we used the subset of classifiers that never saw that training point during training. If m is sufficiently large, the fact that we take out some classifiers has no significant effect and the estimate is pretty reliable.

3 Random Forest Algorithm

One of the most famous and useful bagged algorithms is the Random Forest! A Random Forest is essentially nothing else but bagged decision trees, with a slightly modified splitting criteria.

Each tree is grown as follows:

1. If the number of cases in the training set is m, sample m data sets $\mathcal{D}_1, \ldots, \mathcal{D}_m$ uniformly - but *with replacement*, from the original data $\mathcal{D}$. This sample will be the training set for growing the tree.
2. If there are d features, a number $k \ll d$ is specified such that each node, k variables are selected at random out of the d and the best split on these k is used to split the node. The value of k is held constant during the forest growing.
3. The final aggregate classifier can be
   - for regression
     $$\bar{f}(x) = \frac{1}{m} \sum_{i=1}^{m} h_i(x)$$
     the average of f_i for $i = 1, \ldots, m$.
   - for binary classification where $y \in \{-1, 1\}$
     $$\bar{f}(x) = \text{sign}(\sum_{i=1}^{m} h_i(x))$$
     essentially a "majority vote", assuming that votes for binary classes are unequal.

4 Random Forest Advantages

The Random Forest is one of the best, most popular and easiest to use out-of-the-box classifier. There are two reasons for this:

1. The RF has only two main hyper-parameters, **max_depth** and **n_estimators**. It is extremely *insensitive* to both of these. A good choice for **max_depth** is $\sqrt{d}$ (where d denotes the number of features). You can set **n_estimators** as large as you can afford.

2. Decision trees do not require a lot of preprocessing. For example, the features can be of different scale, magnitude, or slope. This can be highly advantageous in scenarios with heterogeneous data, for example in medical settings where features could be things like *blood pressure, age, gender, ... ,* each of which is recorded in completely different units. Thus, Decision trees are scale-invariant, we can think of the decision as "is feature $x_i \geq$ some_value?" Intuitively, we can see that it really doesn't matter on which scale this feature is (centimeters, Fahrenheit, a standardized scale - it really doesn't matter).

5 Random Forest Hyperparameters

1. **max_depth**: is defined as the longest path between the root node and the leaf node. We can limit the depth of every tree in the forest with this hyperparameter. As the **max_depth** of the decision trees increase, the performance of the model over the training set will increase continuously. Conversely, as **max_depth** increases, the performance of the test set will increase initially, but will start to decrease rapidly at a certain point.

2. **min_samples_split**: represents the minimum number of samples required to split an internal node. This can vary between considering at least one sample at each node to considering all of the samples at each node. When we increase this parameter, each tree in the forest becomes more constrained as it has to consider more samples at each node. Having a low hyperparameter value such as 2 poses the issue that a tree often keeps on splitting until the nodes are completely pure. As a result, the tree grows in size and therefore overfits the data in this case. By increasing the value of the **min_sample_split**, we can reduce the number of splits that happen in the decision tree and therefore prevent the model from overfitting.

3. **max_terminal_nodes**: This hyperparameter sets a condition on the splitting of the nodes in the three and hence restricts the growth of the tree. If after splitting we have more terminal nodes than the specified number of terminal nodes, it will stop the splitting and the tree will not grow further. This hyperparameter helps control overfitting.

4. **min_samples_leaf**: Specifies the minimum number of samples that should be present in the leaf node after splitting a node. This hyperparameter allows us to control the growth of the tree by setting a minimum sample criterion for terminal nodes and helps prevent overfitting. Generally, when the hyperparameter value is very low, the model will overfit.

5. **n_estimators**: Represents the number of trees to grow. Generally, as the number of trees increases, the better the model learns the data. However, adding a lot of trees

can slow down the training process considerably, therefore we do a hyperparameter search to find the sweet spot.

6. **max_samples**: Determines what fraction of the original dataset is given to any individual tree. More data is always better, however at times we can use a lesser fraction of the bootstrap data at times and still get high performance. For example, the hyperparameter value could be 0.2, and we could still generate high performance. This means training time will be reduced considerably but still garner high performance.

7. **max_features**: Represents the number of features to consider when looking for the best split. Empirical good default values are max_features = n_features for regression problems and max_features = $\sqrt{\text{n\_features}}$ for classification tasks.

6 Random Forest Variable Importance

Random forests are among the most popular machine learning methods thanks to their relatively good accuracy, robustness and ease of use. They also provide two straightforward methods for feature selection: mean decrease in gini impurity and mean decrease in accuracy.

6.1 Mean Decrease in Accuracy (MDA)

The mean decrease in accuracy (MDA) for a feature is determined during the out of bag error calculation phase. The more the accuracy of the random forest decreases due to the exclusion (or permutation) of a single variable, the more important that variable is deemed, and therefore variables with a large mean decrease in accuracy are more important for classification of the data.

Steps for MDA:

1. Train Random Forest
2. Out-of-Bag CV accuracy is calculated (OOB_acc_base)
3. Permute feature i
4. Out-of-Bag CV accuracy is calculated (OOB_acc_perm_i)
5. Feature Importance of $i = -$(OOB_acc_perm_i - OOB_acc_base)

6.2 Mean Decrease In Impurity (MDI)

In order to understand Mean Decrease in Impurity, it is important first to understand Gini Impurity, which is a metric used in Decision Trees to determine how (using which feature, and at what threshold) to split the data into smaller groups. Gini Impurity is the probability of incorrectly classifying a randomly chosen element in the dataset if it were randomly labeled according to the class distribution in the dataset. (e.g., if half of the records in a group are "A" and the other half of the records are "B", a record randomly labeled based on the composition of that group has a 50% chance of being labeled incorrectly).

It's calculated as

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i))$$

where C is the number of classes and $p(i)$ is the probability of randomly picking an element of class i.

Gini Impurity reaches zero when all records in a group fall into a single category (i.e., if there is only one possible label in a group, a record will be given that label 100% of the time). This measure is essentially the probability of a new record being incorrectly classified at a given node in a Decision Tree, based on the training data.

Because Random Forests are an ensemble of individual Decision Trees, Gini Importance can be leveraged to calculate Mean Decrease in Impurity, which is a measure of feature importance for estimating a target variable. Mean Decrease in Gini is the average (mean) of a features's total decrease in node impurity, weighted by the proportion of samples reaching that node in each individual decision tree in the random forest. This is effectively a measure of how important a feature is for estimating the value of the target variable across all of the trees that make up the forest. A higher Mean Decrease in Impurity indicates higher feature importance. The most important features to the model have the largest Mean Decrease in Impurity Values, conversely, the least important variable will have the smallest Mean Decrease in Impurity values.

6.3 Pros and Cons of Random Forest

6.3.1 Pros

1. It overcomes the problem of overfitting by averaging or combining the results of different decision trees.

2. Random forest has less variance then single decision tree.

3. Scaling of data does not require in random forest algorithm. It maintains good accuracy even after providing data without scaling.

4. Random Forest algorithms maintains good accuracy even a large proportion of the data is missing.

6.3.2 Cons

1. Complexity is the main disadvantage of Random forest algorithms.

2. More computational resources are required to implement Random Forest algorithm.

3. It is less intuitive in case when we have a large collection of decision trees.