Random Forests

(https://github.com/learn-co-curriculum/dsc-random-forests) (https://github.com/learn-co-curriculum/dsc-random-forests/issues/new/choose)

Introduction

In this lesson, we'll learn about a powerful and popular ensemble method that makes use of decision trees -- a random forest!

Objectives

You will be able to:

- Describe how the random forest algorithm works
- Describe the subspace sampling method that makes random forests "random"
- Explain the benefits and drawbacks of random forest models

Understanding the Random forest algorithm

The ***Random Forest*** algorithm is a supervised learning algorithm that can be used both for classification and regression tasks. Decision trees are the cornerstone of random forests -- if you don't remember much about decision trees, now may be a good time to go back and review that section until you feel comfortable with the topic.

Put simply, the random forest algorithm is an ensemble of decision trees. However, you may recall that decision trees use a ***greedy algorithm***, meaning that given the same data, the algorithm will make a choice that maximizes information gain at every step. By itself, this presents a problem -- it doesn't matter how many trees we add to our forest if they're all the same tree! Trees trained on the same dataset will come out the exact same way every time -- there is no randomness to this algorithm. It doesn't matter if our forest has a million decision trees; if they are all exactly the same, then our performance will be no better than if we just had a single tree.

Think about this from a business perspective -- would you rather have a team at your disposal where everyone has exactly the same training and skills, or a team where each member has their own individual strengths and weaknesses? The second team will almost always do much better!

As we learned when reading up on ensemble methods, variance is a good thing in any ensemble. So how do we create high variance among all the trees in our random forest? The answer lies in two clever techniques that the algorithm uses to make sure that each tree focuses on different things -- ***Bagging*** and the ***Subspace Sampling Method***.

Bagging

The first way to encourage differences among the trees in our forest is to train them on different samples of data. Although more data is generally better, if we gave every tree the entire dataset, we would end up with each tree being exactly the same. Because of this, we instead use ***Bootstrap Aggregation*** (AKA ***Bagging***) to obtain a portion of our data by sampling with replacement. For each tree, we sample two-thirds of our training data with replacement -- this is the data that will be used to build our tree. The remaining data is used as an internal test set to test each tree -- this remaining one-third is referred to as ***Out-Of-Bag Data***, or ***OOB***. For each new tree created, the algorithm then uses the remaining one-third of data that wasn't sampled to calculate the ***Out-Of-Bag Error***, in order to get a running, unbiased estimate of overall tree performance for each tree in the forest.

Training each tree on its own individual "bag" of data is a great start for getting us some variability between the decision trees in our forest. However, with just bagging, all the trees are still focusing on all the same predictors. This allows for a potential weakness to affect all the trees at once -- if a predictor that usually provides strong signal provides bad information for a given observation, then it's likely that all the trees will fall for this false signal and make the wrong prediction. This is where the second major part of the Random forest algorithm comes in!

Subspace sampling method

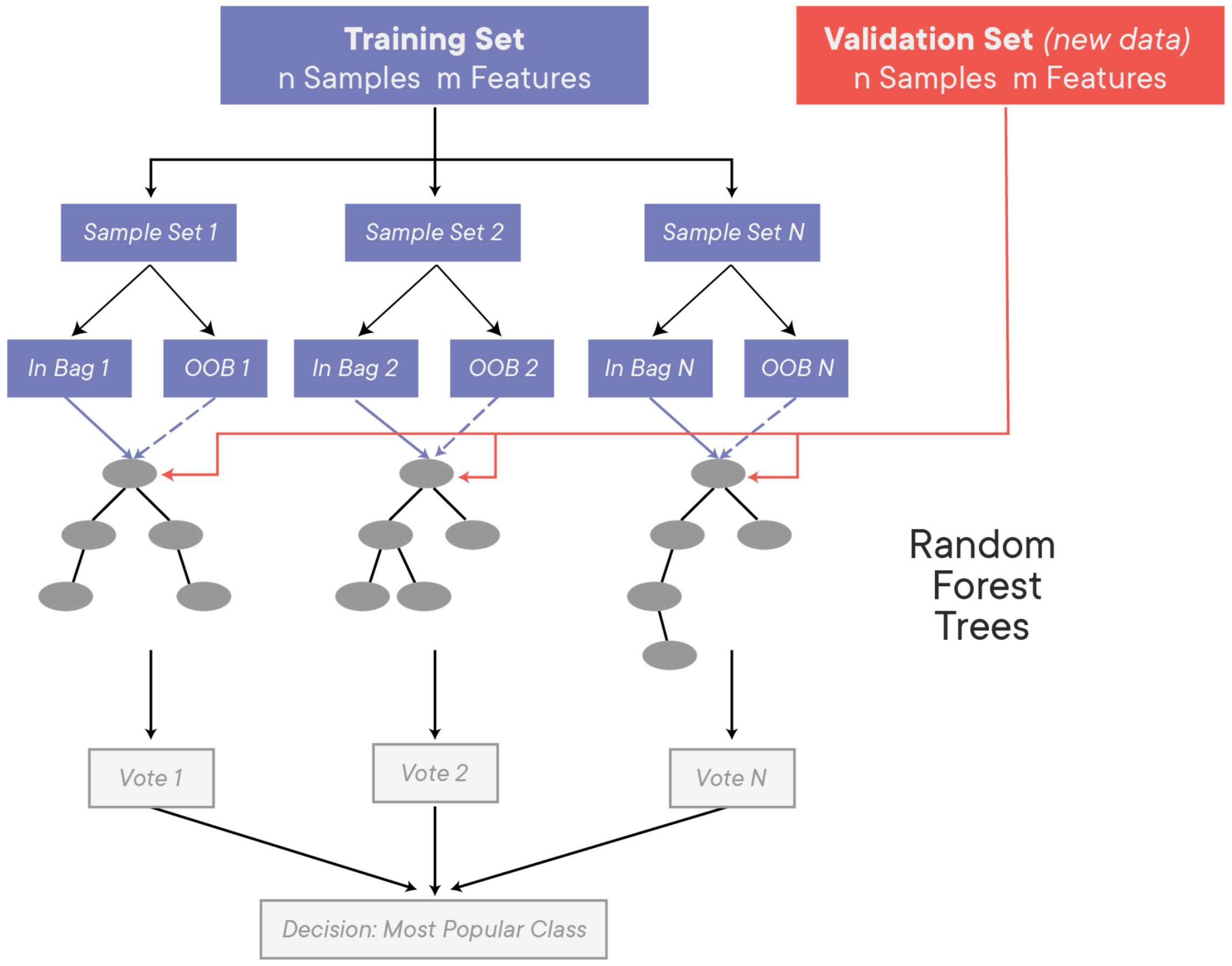
After bagging the data, the random forest uses the ***Subspace sampling method*** to further increase variability between the trees. Although it has a fancy mathematical-sounding name, all this method does is randomly select a subset of features to use as predictors for each node when training a decision tree, instead of using all predictors available at each node.

Let's pretend we're training our random forest on a dataset with 3000 rows and 10 columns. For each given tree, we would randomly "bag" 2000 rows with replacement. Next, we perform a subspace sample by randomly selecting a number of predictors at each node of a decision tree. Exactly how many predictors are used is a tunable parameter for this algorithm -- for simplicity's sake, let's assume we pick 6 predictors in this example.

This brings us to the following pseudocode so far:

For each tree in the dataset:

1. Bag 2/3 of the overall data -- in our example, 2000 rows
2. Randomly select a set number of features to use for training each node within this -- in this example, 6 features
3. Train the tree on the modified dataset, which is now a DataFrame consisting of 2000 rows and 6 columns
4. Drop the unused columns from step 3 from the out-of-bag rows that weren't bagged in step 1, and then use this as an internal testing set to calculate the out-of-bag error for this particular tree



Resiliency to overfitting

Once we've created our target number of trees, we'll be left with a random forest filled with a diverse set of decision trees that are trained on different sets of data, and also look at different subsets of features to make predictions. This amount of diversity among the trees in our forest will make for a model that is extremely resilient to noisy data, thus reducing the chance of overfitting.

To understand why this is the case, let's put it in practical terms. Let's assume that of the 10 columns that we mentioned in our hypothetical dataset, column 2 correlates heavily with our target. However, there is still some noise in this dataset, and this column doesn't correlate *perfectly* with our target -- there will be times where it suggests one class or another, but this isn't actually the case -- let's call these rows "false signals". In the case of a single decision tree, or even a forest where all trees focus on all the same predictors, we can expect to get the model to almost always get these false signal examples wrong. Why? Because the model will have learned to treat column 2 as a "star player" of sorts. When column 2 provides a false signal, our model will fall for it and get the prediction wrong.

Now, let's assume that we have a random forest complete with subspace sampling. If we randomly use 6 out of 10 predictors when creating each node of each tree, then this means that ~40% of the nodes of the trees in our forest won't even know column 2 exists! In the cases where column 2 provides a "false signal", the nodes of trees that use column 2 will likely make an incorrect prediction -- but that only matters to the ~60% that look at column 2. Our forest still contains another 40% of nodes within trees that are essentially "immune" to the false signal in column 2, because they don't use that predictor. In this way, the "wisdom of the crowd" buffers the performance of every constituent in that crowd. Although for any given example, some trees may draw the wrong conclusion from a particular predictor, the odds that *every tree* makes the same mistake because they looked at the same predictor is infinitesimally small!

Making predictions with random forests

Once we have trained all the trees in our random forest, we can effectively use it to make predictions! When given data to make predictions on, the algorithm provides only the appropriate features to each tree in the forest, gets that tree's individual prediction, and then aggregates all predictions together to determine the overall prediction that the algorithm will make for said data. In essence, each tree "votes" for the prediction that the forest will make, with the majority winning.

Benefits and drawbacks

Like any algorithm, the random forest comes with its own benefits and drawbacks.

Benefits

- ***Strong performance***: The random forest algorithm usually has very strong performance on most problems, when compared with other classification algorithms. Because this is an ensemble algorithm, the model is naturally resistant to noise and variance in the data, and generally tends to perform quite well.
- ***Interpretability***: Conveniently, since each tree in the random forest is a ***Glass-Box Model*** (meaning that the model is interpretable, allowing us to see how it arrived at a certain decision), the overall random forest is, as well! You'll demonstrate this yourself in the upcoming lab, by inspecting feature importances for both individual trees and the entire random forest.

Drawbacks

- ***Computational complexity***: Like any ensemble method, training multiple models means paying the computational cost of training each model. On large datasets, the runtime can be quite slow compared to other algorithms.
- ***Memory usage***: Another side effect of the ensembled nature of this algorithm, having multiple models means storing each in memory. Random forests tend to have a larger memory footprint

than other models. Whereas a parametric model like a logistic regression just needs to store each of the coefficients, a random forest has to remember every aspect of every tree! It's not uncommon to see random forests that were trained on large datasets have memory footprints in the tens or even hundreds of MB. For data scientists working on modern computers, this isn't typically a problem -- however, there are special cases where the memory footprint can make this an untenable choice -- for instance, an app on a smartphone that uses machine learning may not be able to afford to spend that much disk space on a random forest model!

(Optional) Random forests White paper

This algorithm was not invented all at once -- there were several iterations by different researchers that built upon each previous idea. However, the version used today is the one created by Leo Breiman and Adele Cutler, who also own the trademark for the name "random forest".

Although not strictly necessary for understanding how to use random forests, we highly recommend taking a look at the following resources from Breiman and Cutler if you're interested in really digging into how random forests work:

- **Random forests paper (https://www.stat.berkeley.edu/%7Ebreiman/randomforest2001.pdf)**
- **Random forests website (https://www.stat.berkeley.edu/%7Ebreiman/RandomForests/cc_home.htm)**

Summary

In this lesson, you learned about a random forest, which is a powerful and popular ensemble method that uses decision trees!

How do you feel about this lesson?

Have specific feedback?

Tell us here! (https://github.com/learn-co-curriculum/dsc-random-forests/issues/new/choose)