Boosting the accuracy of your Machine Learning models

Prashant Gupta Follow
May 30, 2017

Tired of getting low accuracy on your machine learning models? Boosting is here to help. *Boosting is a popular machine learning algorithm that increases accuracy of your model,* something like when racers use nitrous boost to increase the speed of their car.

Boosting uses a base machine learning algorithm to fit the data. This can be any algorithm, but Decision Tree is most widely used. For an answer to why so, just keep reading. ***Also, the boosting algorithm is easily explained using Decision Trees, and this will be focus of this article.*** It builds upon approaches other than boosting, that improve accuracy of Decision Trees. For an introduction to tree based methods, read my other article **here**.

Bootstrapping

I would like to start by explaining an important foundation technique called **Bootstrapping**. Assume that we need to learn a decision tree to predict the price of a house based on 100 inputs. *Prediction accuracy of such a decision tree would be low, given the problem of variance it suffers from*. This means that if we split the training data into two parts at random, and fit a decision tree to both halves, the results that we may get could be quite different. What we really want is a result that has low variance if applied repeatedly to distinct data sets.

> *We can improve the prediction accuracy of Decision Trees using Bootstrapping*

Create many (e.g. 100) random sub-samples of our dataset with replacement (meaning we can select the same value multiple times).

Learn(train) a decision tree on each sample.

Given new dataset, Calculate the prediction for each sub-sample.

Calculate the average of all of our collected predictions(also called bootstrap estimates) and use that as our estimated prediction for the data.

The procedure can be used in similar way for *classification trees*. For example, if we had 5 decision trees that made the following class predictions for an input sample: blue, blue, red, blue and red, we would take the most frequent class and predict blue.

In this approach, trees are grown deep and are not pruned. Thus each individual tree has high variance, but low bias. Averaging these trees reduces the variance dramatically.

Bootstrapping is a powerful statistical method for estimating a quantity from a data sample. Quantity can be a descriptive statistic such as a mean or a standard deviation. ***The application of the Bootstrapping procedure to a high-variance machine learning algorithm, typically decision trees as shown in the above example, is known as Bagging(or bootstrap aggregating).***

Error Estimation

An easy way of estimating the test error of a bagged model, without the need for cross-validation is **Out-of-Bag Error Estimation**. The observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations. We can simply predict the response for the *i*th observation using each of the trees in which that observation was OOB. We average those predicted responses, or take a majority vote, depending on if the response is quantitative or qualitative. *An overall OOB MSE(mean squared error) or classification error rate can be computed*. This is an acceptable test error rate because the predictions are based on only the trees that were not fit using that observation.

Random Forests

Decision trees aspire to minimize the cost, which means they make use of strongest predictors/classifiers for splitting the branches. *So, most of the trees made from bootstrapped samples would use the same strong predictor in different splits. This relates the trees and leads to variance.*

> *We can improve the prediction accuracy of Bagged Trees using Random Forests*

While splitting branches of any tree, a random sampled of m predictors is chosen as split candidates from the full set of p predictors. The split is then allowed to only use one of those m predictors. A fresh sample of $m$predictors is taken at each split. You can try different values and tune it using cross validation.

> For classification a good default is: m = sqrt(p)
>
> For regression a good default is: m = p/3

Thus, on average, $(p—m) / p$ of the splits will not even consider the strong predictor. This is known as ***decorrelating*** *the trees, as we fix the issue of each tree using same strong predictor.*

> ***If* m = p *then random forests is equal to bagging.***

Feature Importance

One problem with computing fully grown trees is that we cannot easily interpret the results. And it is no longer clear which variables are important to the relationship. *Calculating drop in the error function for a variable at each split point gives us an idea of feature importance.* ***It means that we record the total amount that the error is decreased due to splits over a given predictor, averaged over all bagged trees. A large value then indicates an important predictor.*** In regression problems this may be the drop in residual sum of squares and in classification this might be the Gini score.

Boosting

> *The prediction accuracy of decision trees can be further improved by using Boosting algorithms.*

The basic idea behind boosting is converting many weak learners to form a single strong learner. What do we mean by weak learners?

Weak learner is a learner that will always do better than chance, when it tries to label the data, no matter what the distribution over

the training data is. Doing better than chance means we are always going to have an error rate which is less than 1/2. *This means that the learner algorithm is always going to learn something, and will not always be completely accurate i.e., it is weak and poor when it comes to learning the relationships between inputs and target*. It also means a rule formed using a single predictor/classifier is not powerful individually.

We start finding weak learners in the dataset by making some distributions and forming small decision trees from them. The size of the tree is tuned using number of splits it has. Often 1 works well, where each tree consists of a single split. Such trees are known as **Decision Stumps.**

Another parameter boosting takes is the number of iterations or number of trees in this case. Additionally, it assigns weights to the inputs based on whether they were correctly predicted/classified or not. Lets look at the algorithm.

> **First, the inputs are initialized with equal weights**. It uses the first base learning algorithm to do this, which is generally a decision stump. This means, in first stage, it will be a weak learner, that will fit a subsample of the data and make predictions for all the data.
>
> Now we do the following **till maximum number of trees is reached** :
>
> Update the weights of inputs based on previous run, and weights are higher for wrongly predicted/classified inputs
>
> Make another rule(decision stump in this case) and fit it to a subsample of data. Note that this time rule will be formed by keeping the wrongly classified inputs(ones having higher weight) in mind.
>
> Finally we predict/ classify all inputs using this rule.

3. **After the iterations have been completed, we combine weak rules to form a single strong rule,** which will then be used as our model.

The above algorithm is better explained with help of a diagram. Lets assume we have 10 input observations that we want to classify as “+” or “-”.

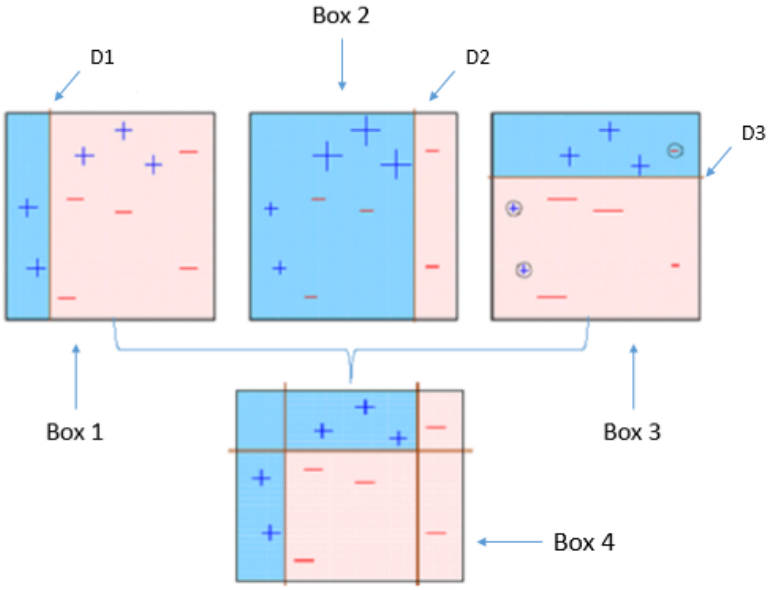

Image source : Analytics Vidhya

The boosting algorithm will start with box 1 as shown above. It assigns equal weights(denoted by size of the signs) to all inputs and predicts "+" for inputs in blue region and "-" for inputs in the reddish region, using decision stump D1.

In next iteration, Box 2, you can see weights of wrongly classified plus signs are greater than other inputs. So a decision stump D2 is chosen such that, these observations are now classified correctly.

In the final iteration, Box 3, it has 3 misclassified negatives from the previous run. So a decision stump D3 is chosen to correct that.

Finally, the output strong learner or Box 3, has a strong rule that is made by combining individual weak decision stumps. You can see how we boosted the classification power of our model.

In regression setting, the prediction error(usually calculated using least squares) is used to adjust weights of inputs, and consequent learners focus more on inputs with large error.

This type of boosting approach is known as Adaptive Boosting or AdaBoost. As with trees, boosting approach also minimizes a loss function. *In case of Adaboost, it is the exponential loss function.*

Another popular version of boosting is Gradient Boosting Algorithm. The basic concept remains the same, *except here we don't play with the weights, but* ***fit the model on residuals***(measurement of the difference in prediction and original outcome) rather than original outcomes. ***Adaboost is implemented using iteratively refined sample weights while Gradient Boosting uses an internal regression model trained iteratively on the residuals.*** This means that the new weak learners are formed keeping in mind the inputs that have high residuals.

In both algorithms, a tuning parameter ***lambda* or *shrinkage*** slows the processes even further by allowing more and different shaped trees to attack the residuals. This is also known as **learning rate** as it controls the magnitude by which each tree contributes to the model. As you can see, ***Boosting also does not involve bootstrapping***, instead each tree is fit on a modified version of the original data. Instead of fitting a single large decision tree, which results in hard fitting the data, and potentially overfitting. ***The boosting approach learns slowly.***

As you can see that the algorithm is explained clearly using Decision Trees, but there are other reasons that it's mostly used with trees.

> ***Decision trees are non-linear. Boosting with linear models simply doesn't work well.***
>
> ***The weak learner needs to be consistently better than random guessing. You don't normally need to do any parameter tuning to a decision tree to get that behavior.*** Training an SVM, for instance, really does need a parameter search. Since the data is re-weighted on each iteration, you likely need to do another parameter search on each iteration. So you are increasing the amount of work you have to do by a large margin.
>
> ***Decision trees are reasonably fast to train.*** Since we are going to be building 100's or 1000's of them, that's a good property. They are also fast to classify, which is again important when you need 100's or 1000's to run before you can output your decision.
>
> By changing the depth ***you have a simple and easy control over the bias/variance trade off,*** knowing that boosting can reduce bias but also significantly reduces variance.

This is an extremely simplified (probably naive) explanation of boosting, but will help you understand the very basics. A popular library for implementing this algorithm is **Scikit-Learn**. It has a wonderful api that can get your model up an running with **just a few lines of code in python**.

If you liked this article, be sure to click ♥ below to recommend it and if you have any questions, **leave a comment** and I will do my best to answer.

I'll soon be writing more on how to implement different boosting algorithms. So, for being more aware of the world of machine learning, **follow me**. It's the best way to find out when I write more articles like this.

You can also follow me on **Twitter at @Prashant_1722**, **email me directly** or **find me on linkedin**. I'd love to hear from you.

That's all folks, Have a nice day :)