Assignment 4

11/05/2020

Name: Yusuf Elnady

Q1.a

Having many linear regressions in boosted ensembles can't do better than a single linear regression. When gradient boosting is done along with linear regression, it is nothing more than another linear model over the existing linear model. This can intuitively be understood as adding something to the already found coefficients, and if the linear regression has already found the best coefficients it will be of no use. Gradient Boost works by having a model that is fitted on data, then the next model is build on residuals of previous model. But usually residuals of linear models can't be fitted with another linear model.

So, in a nutshell, GB involves fitting a model, and then fitting a model to the residuals from the first model, etc. But in a linear regression, the residuals are orthogonal to the predictors. And in the second-stage, the coefficients will all be zero.

The Math: The least squares projection matrix is given by $X(X^TX)^{-1}X^T$. So in the first iteration, we can get the predicted values using the following formula

$$\hat{y}_1 = X(X^TX)^{-1}X^Ty$$

Then, we can fit a regression and calculate the residuals, we will have

$$r = y - \hat{y} = y - X(X^TX)^{-1}X^Ty$$

In the second iteration, we will use the residuals we get as a new dependent variable for the regression in this iteration. By using the projection matrix to calculate the predictions in the second regression, we will get the following predictions:

$$\begin{aligned}\hat{y}_2 &= X(X^TX)^{-1}X^Tr \\ &= X(X^TX)^{-1}X^Ty - X(X^TX)^{-1}X^TX(X^TX)^{-1}X^Ty \\ &= X(X^TX)^{-1}X^Ty - X(X^TX)^{-1}X^Ty \\ &= 0\end{aligned} \tag{1}$$

A reason for this is that by construction the residual vector r from the initial regression is orthogonal to the X Space i. e. $\hat{y}$ is a orthogonal projection from y onto the X space. We actually could prove it easier since

$$X^T r = 0$$

then $\hat{y}_2$ is directly 0. So, we find that the result will still be a linear function and won't have any of the beneficial properties of a multivariate linear regression, because after the first mlr, the residuals don't have any patterns (IID, homoscedastic, etc.). Then r is orthogonal to the X space, and all subsequent models won't have anything to fit to.
Another way to prove that I found on stack exchange: We know that gradient boosting and linear regression are both trying to solve $\hat{\beta} = \text{argmin}_\beta (y - X\beta)^t (y - X\beta)$. For linear regression we can directly find the linear solution for $X^t X = X^t y$ and automatically gives the best β. But in Boosting, whether we have a weak classifier that is a one variable or multi variable regression, we will end up with a sequence of coefficient vectors Betas. Therefore, the final model prediction will have the same form as a linear regression: $X\beta_1 + X\beta_2 + \cdots + X\beta_n = X(\beta_1 + \beta_2 + \cdots + \beta_n)$ and again this could have been found from the beginning just by having a full linear regression.

Q1.c

Any realistic model of learning from samples have the the issue of noisy data and AdaBoost is an effective method for improving the performance of base classifiersm although Adaboost can have over-fitting in noisy domains. According to the paper "Edited AdaBoost by weighted KNN", they made an algorithm called "AdaBoost by weighted KNN" (EAdaBoost). It is designed in a way that AdaBoost and KNN naturally complement each other. First, AdaBoost is run on the training data to capitalize on some statistical regularity in the data. Then, a weighted KNN algorithm is run on the feature space composed of classifiers produced by AdaBoost to achieve competitive results.
They used AdaBoost to enhance the classification accuracy and avoid over-fitting by editing the data sets using the weighted KNN algorithm for improving the quality of training data. The results of this experiment found that the new Boosting algorithm achieves considerably better classification accuracy than AdaBoost alone. And they made another experiments on data with artificially controlled noise indicate and indicated that the new Boosting algorithm is robust to noise. So, Using Adaboost with KNN increases the performance and decreases the variance.

In general, ensemble methods rely on the instability of the classifiers to improve their performance, as KNN is fairly stable with respect to re-sampling, these methods fail in their attempt to improve the performance of KNN classifier. Also, it's worth mentioning that KNN falls under lazy learning, which means that there is no explicit training phase before classification. Instead any attempts to generalize or abstract the data is made upon classification. AdaBoost builds an ensemble classification model during training, instead KNN builds no such classification model. Instead, it just stores the labeled training data. and when new unlabeled data comes it starts looking for the nearest k neighbors and

take the majority vote. So, I found Another paper which is "A direct boosting algorithm for the k-nearest neighbor classifier via local warping of the distance metric" where they mentioned that to increase the accuracy of these condensed models, they presented a direct boosting algorithm for the KNN classifier that creates an ensemble of models with locally modified distance weighting. They made an empirical study conducted on 10 standard databases from the UCI repository and the results showed that this new Boosted KNN algorithm has increased generalization accuracy in the majority of the data-sets and never performs worse than standard KNN. As KNN is fairly stable with respect to instance selection, so they proposed two methods that take advantage of the instability of KNN with regard to input space modifications. The input space is modified, by input selection or projection, to favor difficult instances to be accurately classified. In this way, boosting of KNN is achieved.

Q2.a

There are many ways we can use to ensemble the output of the K predictors/hypotheses that we have obtained.

If our problem is a classification problem, we can consider the outputs to be (+1) and (-1) for a binary classification, and if it's a multi-class classification, then we can focus on one class using one-vs-all, and we will again have only two outputs (+1) and (-1). To get the final prediction and ensemble the output of the hypotheses, where each hypothesis has a weight $w_i = \frac{1}{K}$, then we can use the following formula :

$$Sign(\sum_{i=1}^{K} w_i h_i(x))$$

For example: If we have 3 hypotheses, then $w_i = 1/3$, and we are given $h_1 = +1, h_2 = -1, h3 = +1$ then the result will be $Sign(+1/3) = +1$. So, the predicted class is (+1). This is method is very similar to taking the majority vote (weights are equal).

If our problem is a regression one, then we can just simply get the output of each hypothesis and average them using this formula:

$$\frac{1}{K} \sum_{i=1}^{K} h_i(x)$$

Q2.b.iii

The highest accuracy I got before in Assignment 1 when I was using KNN without any bagging was 82.5%, but here we were able to increase the accuracy to 86% which is a large increase in the performance. However, bagging is rarely used in conjunction with KNN classifiers, as the decision surfaces are typically too stable and any multiples of datapoints in the bootstrap sample do not shift the 'weight' like in many other models. Moreover, if we just used 1-NN, then it doesn't have any effect and perfectly equivalent to the bagged 1-NN. But ensemble subsets of KNN outperform the KNN based methods.

The reason for that is that we are not stuck to using one model and based on the nearest neighbors for it we decide which are the correct class for it. In bagging we do bootstrap sampling with replacement, which means that not all samples will not be existing in each base learner so we are expected to see some points are classified from the previous iteration. But again we use the majority vote to decide on classifying each point, so herein we improve the accuracy of our model. . A test datapoint can change classification only if its nearest neighbors in the learning set is not in at least half of the N bootstrap samples. The probability for this to occur is the same as the probability of flipping a weighted coin with a 0.632 probability for heads N times and getting less than 0.5N heads. As N gets larger this probability gets smaller and smaller. The Similar logic holds for multiclass problems and KNN.