Miner Username: Michael_Scarn

Loading and Preparing Data: I've used pandas to read data from the train file using *'pd.read_table'* method. To divide the dataset into train and test samples I have used *'train_test_split'* function from *'sklearn.model_selection'*

Decision Stump: The decision Stump implementation is done by creating a decision tree classifier of depth 1.To find the best split of continuous features Gini index is used. The Gini Index measures the inequality among values of a frequency distribution. First the Gini for sub-nodes is calculated using sum of square of probability for success and failure. The next step is to calculate Gini for the split using weighted Gini Score for each node of that split. $\text{Gini} = 1 - \sum_{i=1}^{n} (p_i)^2$

For building the DecisionTree, Input data is split based on the lowest Gini score of all possible features. After the split at the decisionNode, two datasets are created. This is done for all the possible splits to get the attribute at which we get the lowest Gini score. For predicting the target value, start with the given feature of the test dataset. Starting from the root node, evaluate the sample based on the split criteria.

Adaboost: The main steps in the algorithm are first to assign equal weights for all the records. Second, I created a training dataset by random sampling with replacement from Train dataframe according to the weights. Then I trained the weak classifier stump and applied it to predict the classes of the original data. Then I calculated the weighted error and the importance using the equations from our lecture slides(4and 5 slides of Boosting PPT).In the next step I updated the weights using

New weight = Weight * e(performance) → misclassified records

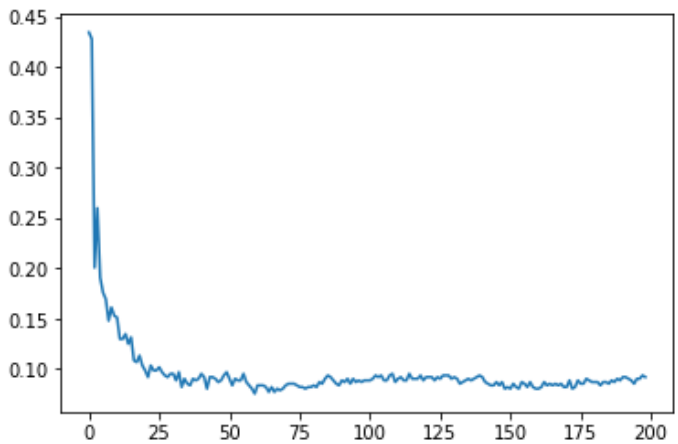
New weight = Weight * e-(performance) → correctly classified records

*new_weight = D['weight']*np.exp(-1*alpha_i*D['Label']*D['pred'])* .To fit our labels into the equation I have taken class 5 as label '1' and class 3 as label '-1'.Because if the 'Label' and 'pred1' are same (i.e., 1 or -1) then substituting the values in the above equation will give the equation corresponding to correctly classified record. Similarly, if the values are different then substituting the values in the above equation will give the equation corresponding to misclassified record.Next I updated the normalized weights for the next iteration. If weighted error is more than 0.5 the weights are reset, and the steps are repeated.

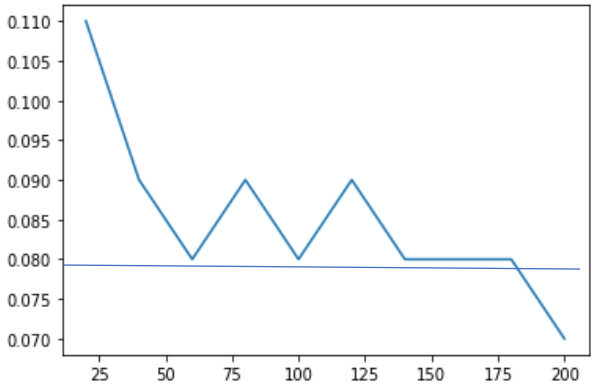
The final prediction is done using , Final prediction/sign (weighted sum) = ∑ (alpha_i* (predicted value at each iteration))

Analysis and interpretation:

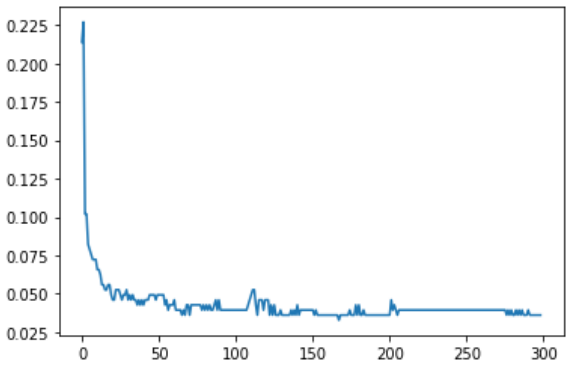
The graph with number of rounds of boosting on the X axis, training set error (unweighted) on the Y axis



Graph with number of rounds of boosting(in order of 20) on the X axis, estimated test set error for AdaBoost on the Y axis



In the above graph the single line is the test error of a single decision tree which was implemented using DecisionTreeClassifier of sklearn.tree package.I have observed that in the starting iteration there was a big boost in the accuracy of the weak classifier as the misclassified records were penalized with an incremental weight. But after 50 iterations there was no significant boost in the training accuracy(or decrease in train error).From this I have interpreted that adaboost has converged quickly but it was robust in improving the weak stump's accuracy. As, the number of misclassifications will NOT go down with each iteration (since each classifier is a week classifier). So, in the next iteration some of the previously misclassified sampled will be correctly classified but this might also result in previously correctly classified samples going wrong (hence training error is not improving). Even though each classifier is weak, since the final output is the weighted sum of all the classifiers the final classification converges to a strong learner. This is to explain the fluctuations in the train/test error for iterations after 50.Also the accuracy of adaboost using sklearn's decision stump was close to accuracy from my adaboost using decision stump with Gini index as criteria. For 200 iteration the adaboost was robust to overfitting. But if number of rounds of boosting are increased to 300-350(see below) then the train error was close to 0.025 but the test error was 0.08 this can be due to overfitting of train data as the test error was not improving with respect to the train error.



References:

[AdaBoost in Python - Machine Learning From Scratch 13 - Python Tutorial - YouTube]

https://github.com/fakemonk1/decision-tree-implementation-from-scratch

[AdaBoost from Scratch. Build your Python implementation of… | by Alvaro Corrales Cano | Towards Data]