Pranav Pradeep Rao
prao43
Sept 24th 2018

Project 3: Assess Learners

Introduction:

The aim of the project was to asses the different types of Decision Tree (DT) learners available to use. Decision Tree learners use a tree based data structure to store information. The tree is constructed using training data to evaluate the relationship between different factors and a result. This can further be used to predict results by providing new data about the factors. There are three types learners being experimented here; a generic DT learner, a randomized DT leaner (RT) and a Bootstrap aggregated DT learner (BT).

Methodology:

In this project several experiments were run determine the efficiency of the DT learners. I used several sets of data where multiple factors had a correlation to an end result. The main test data used was the provided file *Istanbul.csv*, this file contains the returns of multiple indexes for a number of days. All the indexes are factors, except for the last one which is MSCI Emerging Markets (EM) index. The DT learners were trained on 60% of the data and the remaining 40% was used to evaluate them. The efficiency here was measured by their root mean square error (RMSE) by comparing the results of the predicted data versus the actual data for the EM index.

The generic DT learner is a recursion based algorithm that generates a tree the learner can use to query for results. The *build_tree* API call can be used to enter the training data. The learner will parse through this data by assessing the highest correlation between the factors and the evaluation value. It will then divide the data up by the median value of the factor with the highest correlation. This continues until we reach the base case where there only two sets of data which become the leaf of that base node. This information is added to create a two - dimensional array which has four columns, the factor under consideration, the split value and the number of rows that need to be traversed to reach the next node on the left or the right. A leaf contains '-1' values except in the split value column where the leaf's value is set. There are a four edge cases that need to be considered here, the first is if multiple factors have an equal correlation, then the first one is always picked. Second edge case is that if the divided data set contains the same result data then a leaf is created since it cannot be further divided. Third edge case if the split value happens to equal the minimum or the max value, we return a leaf value which is the mean of the EM index values. Finally if the leaf size is larger than the data set a leaf is created with the average of the EM index values assigned.

RT learner was designed quite similar to DT learner with one specific change instead of determine the factor with high correlation with the EM index a random one was selected. The remain steps in the RT learner are the same as DT learner resulting in the same solution. BT learner called any learner multiple times(assigned by a parameter called *bag*) generating a forest of data to train against. To ensure that each tree was unique the BT learner would randomize the order of the training data for each learner. When the BT learner is queried it averages the results of each of the learner trees.

Three experiments were carried out on all of these learners. The first experiment was to determine overfitting effect on a DT learner as leaf size was increased. The second experiment was to determine the overfitting effect on a DT learner when using the BT learner with *bag* value assigned to 20. The third experiment was to validate the determine the overfitting effect on a RT learner as leaf size was increased, the condition were kept similar with the first experiment to compare which learner was better. During the first and second experiment additional measurements were taken as leaf sizes were increased the tree sizes for each were recorded along with the time it took for each to generate them. These additional metrics provided useful information in validating which learner was better.

Overfitting with respect to leaf size:

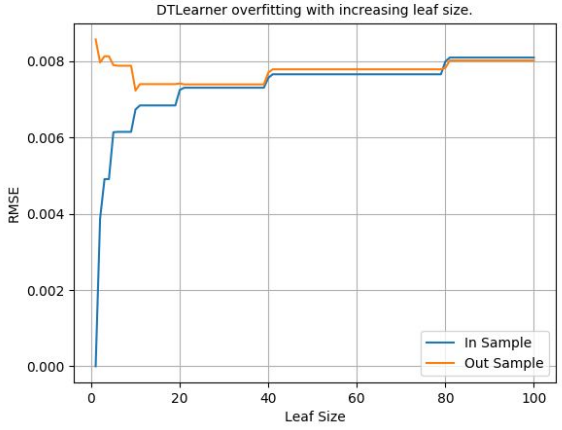
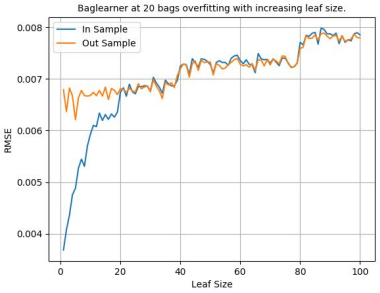


Figure 1: DT Leaner RMSE vs Leaf Size

The data file *Istanbul.csv* was used to train the DT learner and the RMSE was calculated this was done for every leaf size till 100. The results are shown in Figure 1. Overfitting is defined as where the in-sample error is increasing while the out-sample error is decreasing. In this graph as we increase the leaf size we generalize the solution so the RMSE values merge, but when the leaf size is reduced our expectation is that solution is more specialized and the error rate rate drops for the in-sampling testing while the out-sampling testing stays high. Yes there is an overfitting here and it starts from leaf of size 1 to approximately 10 after which the RMSE values for both converge. This makes intuitive sense since smaller leaf sizes generate a more precise tree so the Insample data will have fewer errors.

The effect of Bagging on overfitting:



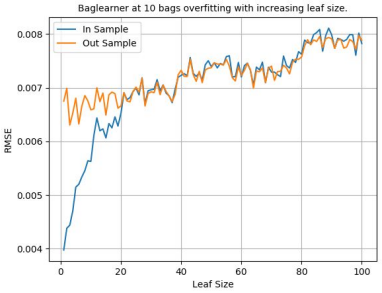


Figure 2(a): Bag Learner RMSE vs Leaf Size for bag at 20 **Figure 2(b): Bag Learner RMSE vs Leaf Size for bag at 10**

In the second experiment the data file *Istanbul.csv* was used to train the BT learner when the base was set to a DT learner. The BT learner was set to a bag of 20. The results

are shown in Figure 2(a). The BT learner was set to a bag of 10 for Figure 2(b). In both cases there is no significant overfitting as the out-sample plot does not drop for significant period of time while the in-sample plot increases. This is an improvement from the previous experiment. This change can be explained by the fact that bagging allows for multiple decision trees to be used to create a more accurate representation of the expected result.

Quantitatively compare DTLearver vs RTLearner:

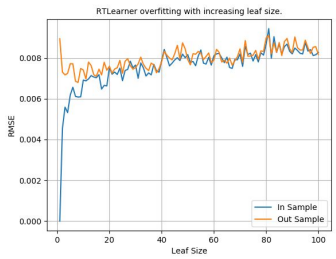


Figure 3 (a) : RT Leaner RMSE vs Leaf Size

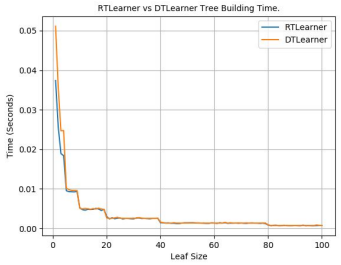


Figure 3 (b) : RT vs DT Tree building time

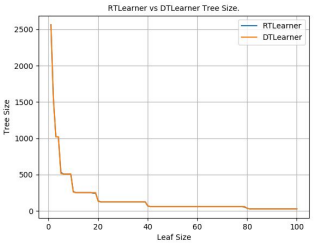


Figure 3 (c) : RT vs DT Tree size

Randomization is powerful tool in computer science it has been used repeatedly to improve the efficiency of algorithms. This is the case here as well, the difference between RT and DT learners as explained in the methodology section of this report has two implications that the tree generation time should be faster for RT learner since the correlation calculations step is not required. This is demonstrated in Figure 3(b) where the leaf size is reduced RT learner is faster than DT learner they appear to converge after leaf size of 20. There is no impact of the tree size as explained in methodology since no changes were made to the base case logic of the algorithm. This is shown in Figure 3(c).

Another important fact in assessing the two learners is Figure 3(a) which shows the RMSE values of RT learner this can be compared to Figure 1 for DT learner. The overfitting effect here is much less starting at one leaf and ending around three. This shows the RT learner is able to create a much more accurate tree with the same contratinst by using randomization. Given these three metrics I can conclude that RT leaner is the better algorithm than DT learner.

Notes on generating graphs:

To run *testlearner.py* you do not need to pass an input as it will attempt to find the *istanbul.csv* data file using the provided *util* library. If the file has been moved you will need to provide the directory path with the *istanbul.csv* file name to correctly load it using the *util* library. You can pass another file along with file path that the *util* library can read from.