CS 7646 Project 3 Report: Assess Learners

Teng Xue
txue34@gatech.edu

Abstract—This report involves 1). A classic Decision Tree learner based on JR Quinlan algorithm; 2). A Random Tree learner based on A Cutler algorithm; 3). A Bootstrap Aggregating (Bagging) learner ensembled different learners; 4). An Insane leaner used specific use-case of the Bagging learner. Given the same data set, the differences and performance of these learners will be compared, thoroughly discussed, and evaluated by detailed analysis.

1 INTRODUCTION

Decision trees are built with nodes and leaves in which data are split up based on a best feature, and hence different algorithms have been developed to determine this optimal split criterion. Among them, a classic **decision tree algorithm** was firstly described by JR Quinlan: beginning with the root node in the tree, data will be split up into two parts by taking the highest absolute correlation of the data as the optimal split criterion, and then recursively sent back through the tree building algorithm. Eventually, all the data will be grouped into most similar groups. However, for a **random tree algorithm**, developed by A Cutler, only focusing on randomly select split criterion to group data. By combining with **Bootstrap Aggregating (Bagging) algorithm**, training by the same model but randomly pick data into different bags and eventually yield a satisfied result. Hence a **hypothesis** here is the random tree (RT) will be built faster than the decision tree (DT) but produce similar results with the decision tree (DT).

2 METHODS

All the tests will be run in '*testlearner.py*' file that includes three learners (algorithms) with APIs to implement: **DTLearner** (decision tree algorithm), **RTLearner** (random tree algorithm), and **BagLearner** (Bootstrap Aggregating (Bagging) algorithm). Using the following command to run:

“PYTHONPATH=../:. python testlearner.py Data/Istanbul.csv”

All learner **APIs** will be implemented by three steps in Python scripts: 1). Add training data to learner via *add_evidence()* method; 2). Build and save the tree via *build_tree()* method in which the model will be trained with a specific algorithm by cross validation method: 60% In-Sample training data with 40% Out-of-Sample testing data; 3). Estimate the predictions given the tree we built via *query()* method. Importantly, all the input needs to be Numpy-objected nd-arrays, and the output will be Numpy-objected 1d-arrays as the predicted results.

The only dataset that all the learners and tests could use is ‘*Istanbul.csv*’ file, a **hypothesis** here is to treat the dataset as a non-time series dataset, so the date column should be ignored.

3 DISCUSSION

3.1 Experiment 1: Does the overfitting occur with respect to leaf_size? For which values of leaf_size does overfitting occur?

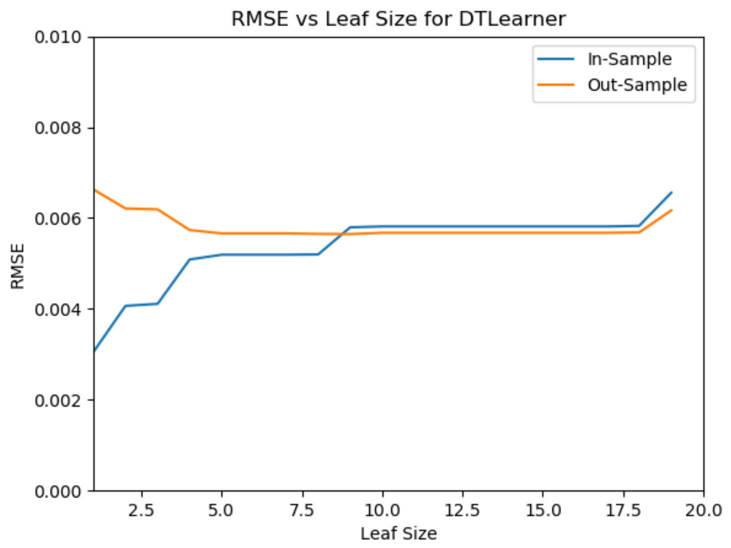

Figure 1— RMSE value change of In-Sample and Out-of-Sample data with varying Leaf Size for the DTLeaner.

Yes, overfitting does occur gradually from a leaf size of 8 and overfitting mostly from a leaf size of 5, the direction of overfitting is with the leaf size decrease (from right to left). As shown in Figure 1 above, starting from a leaf size of 8 with the leaf size decreases (from right to left): In-Sample error (RMSE value) gradually decreases, and Out-of-Sample error (RMSE value) gradually increases that matches the definition of overfitting. The most clearly overfitting occurs from a leaf size of 5 as that point: In-Sample error (RMSE value) starting dramatically decreases, and Out-of-Sample error (RMSE value) starting dramatically increases.

3.2 Experiment 2: Can bagging reduce overfitting with respect to leaf_size? Can bagging eliminate overfitting with respect to leaf_size?

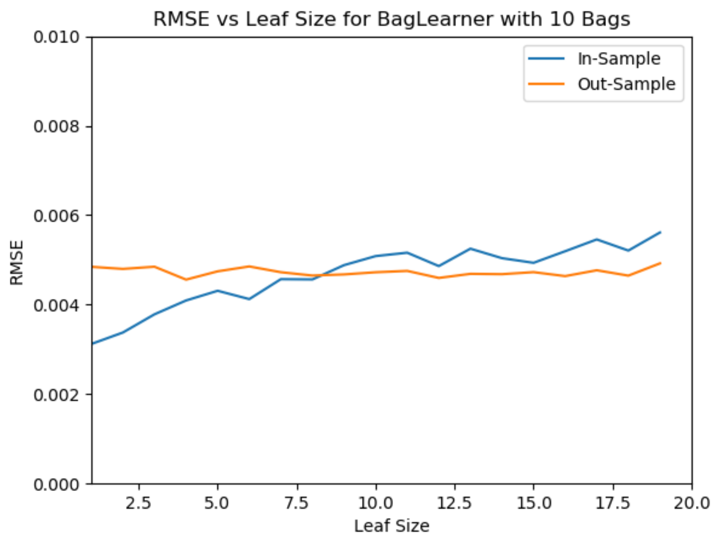

Figure 2 — RMSE value change of In-Sample and Out-of-Sample data with varying Leaf Size for the BagLearner that has 10 Bags and each with DTLearner inside.

Bagging can definitely reduce overfitting with respect to leaf_size, but cannot eliminate it. As shown in Figure 2, the BagLearner runs with a bag size of 10 with varying leaf size from 0 to 20, each bag uses the same DTLearner as Experiment 1 did. While overfitting still occurs from a leaf size of 8, the direction of overfitting is with the leaf size decrease (from right to left), the final RMSE difference of in-sample error and out-of-sample error has been significantly reduced. In Experiment 1 (DTLearner only), the final difference between two errors is around 0.004. However, in this experiment (BagLearner), the final difference between two errors is around 0.002. Hence, bagging can definitely reduce overfitting with respect to leaf_size, but it is unlikely to eliminate it as this algorithm always use the same model (DTLearner, in this case) to train the data.

3.3 Experiment 3

3.3.1 In which ways is one method better than the other? Which learner had better performance (based on your selected measures) and why do you think that was the case? Is one learner likely to always be superior to another (why or why not)?

In the two experiments of performance comparing DTLearner and RTLearner, **Training Time** and **Mean Absolute Error (MAE)** are implemented as the key metrics to compare.

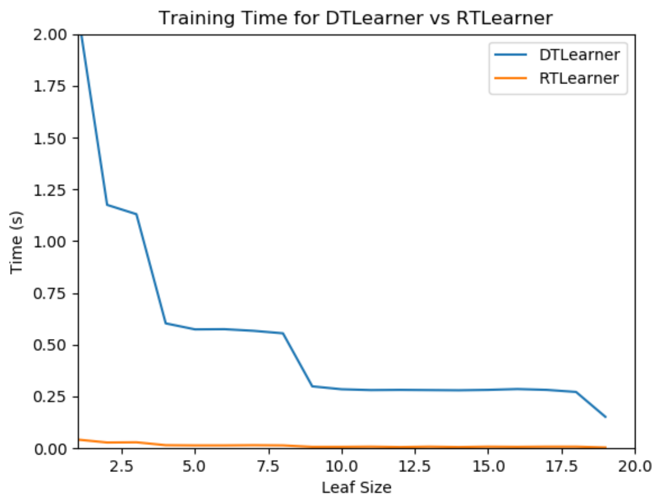

Figure 3— Training time measurement for DTLearner and RTLearner with varying Leaf Size.

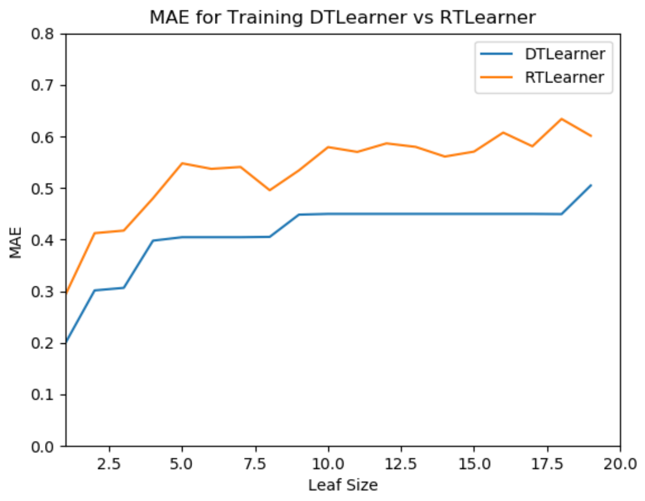

Figure 4— MAE value change of DTLearner and RTLearner with varying Leaf Size.

As shown in Figure 3 and 4, **RTLearner outperformed the DTLearner during the training time measurement**, since Figure 3 clearly shows RTLearner always need far less time to build the tree and train the model than DTLearner. No doubly, comparing to DTLearner's calculation of the highest absolute correlation, RTLearner's randomly pick does not cost that much time.

On the other hand, however, **DTLearner outperformed the RTLearner from the MAE experiment**, since Figure 4 clearly shows DTLearner always has lower MAE value than RTLearner. And a lower Mean Absolute Error (MAE) indicates DTLearner has a higher prediction accuracy than RTLearner. Indeed, DTLearner's calculation of the highest absolute correlation should give a more accurate result than RTLearner's randomly pick, albeit need more time to trade off. Therefore, **one learner will never likely to always be superior to another as their performance depend on different key metrics a lot.**

4 SUMMARY

As the same with all algorithms, building a decision tree also need to consider **trade-offs**. If accuracy is the priority, bagging algorithm that uses decision tree algorithm for each bag is the best choice as it gives the highest accuracy with the least overfitting. While bagging can definitely reduce overfitting, it cannot eliminate overfitting. Conversely, if speed is the priority, random tree algorithm is the best choice as it need shortest time to build the tree and train the data. In terms of leaf size, it does correspond to the overfitting change. From this perspective, random tree algorithm could be a better option for large datasets as it is the fastest for predictions.