I. INTRODUCTION

This report is based on the evaluation of three learning algorithms:

- A "classic" Decision Tree learner.
- A Random Tree learner.
- A Bootstrap Aggregating learner.

All the decision tree learners were implemented with Python, using a matrix data representation (NumPy ndarray). The context of the learners is a regression problem, not classification. Thus, the goal for the learner(a) is to return a continuous numerical result and not a discrete result.

For all experiments, the methodology of training (learning) and generalization (testing) was implemented by partitioning the dataset into 60% Training (Xtrain, Ytrain) and 40% testing (Xtest, Ytest). The learners' utilized the training data to construct regression trees. Generalization of the model was evaluated using the test data by running the Xtest though the regression tree to obtain a predicted Y vector (Ypred). Ypred was evaluated against Yactual using Root Mean Square Error (RMSE):

```
# Root Mean Square Error (rmse) out of sample (os):
rmse_os = math.sqrt(np.mean((Ytest - Y)**2, axis=0))

# Root Mean Square Error (rmse) in sample (is):
rmse_is = math.sqrt(np.mean((Ytrain - Y)**2, axis=0))
```

For this report, ***accuracy*** will be defined by RMSE, i.e., lower RMSE scores equate to higher ***accuracy***.

The dataset used to answer questions 1 and 2, as well as the generalization metric for question 3 was Istanbul.csv, which had a shape (536, 9). That is a dataset with 536 samples, with each sample consisting of 8 features (Xi) and an observation (Yi,) based on the features. Question 3 required at least two quantitative measures. In addition to generalization, I chose computational efficiency. To test computation efficiency, I modified the Istanbul.csv dataset by increasing either the number of rows (samples), or the number of columns (dimensionality) over a sequence of runs. The time it took the learners to learn (create the tree(s)) was measured at each iteration. These experiments will be discussed in more detail in the next section.

II. Explore the Learners

***Question 1**: Does overfitting occur with respect to leaf size? Consider the dataset istanbul.csv with DTLearner. For which values of leaf size does overfitting occur? Use RMSE as your metric for assessing overfitting. Support your assertion with graphs/charts. (Don't use bagging).*

Overfitting does occur with the istanbul.csv dataset with respect to leaf size for the DTLearner model. In general, overfitting in decision trees occurs most frequently from the following conditions:

- Irrelevant attributes can result in overfitting the training data. If the data has many dimensions (large number of attributes), some of this data may obscure the true distinguishing features of the dataset.
- Too little training data, relative to the complexity, i.e., number of dimensions.
- A decision tree's complexity reaches a threshold that includes noise in the tree. In other words, noise in the data makes the decision tree more complex than it should be. The complexity in decision trees is the number of free parameters (the number of nodes), i.e., more nodes equate to greater complexity.

The goal is to achieve the best generalization with minimal number of decisions, i.e., introduce a bias for less complex decision trees. For these experiments, increasing the number of samples per leaf (larger leaf size) establishes this bias for simpler trees. Overfitting can be observed when a model generalizes training (in sample) data better than testing (out of sample) data. Overfitting can also be evaluated by observing RMSE values vs. the bias (leaf size). With a smaller leaf size, the algorithm tries to classify all the training set perfectly. This leads to overfitting, i.e., a model that fails to achieve good prediction (generalization) for novel (test) data.

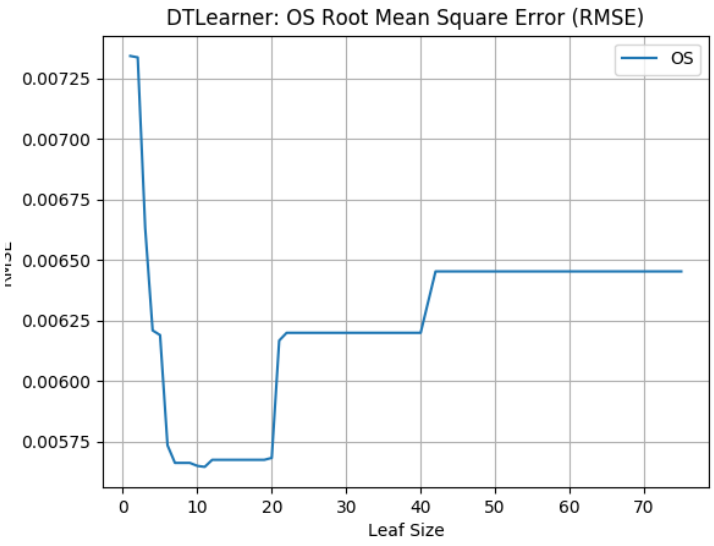


Figure 1: DTLearner out of sample RMSE for Istanbul.csv with leaf sizes [1-75].

RMSE is minimized for leaf sizes between 7 and 20, which is the ***optimal range*** for leaf size given this dataset. ***Notice the RMSE for the model is very high for leaf sizes ~≤ 7. This is where the model overfits***. As the leaf size is increased, RMSE increases. This is not from overfitting, but the result of a larger leaf size allowing too many samples per leaf. Thus, the model will have more opportunity to misinterpret novel data.

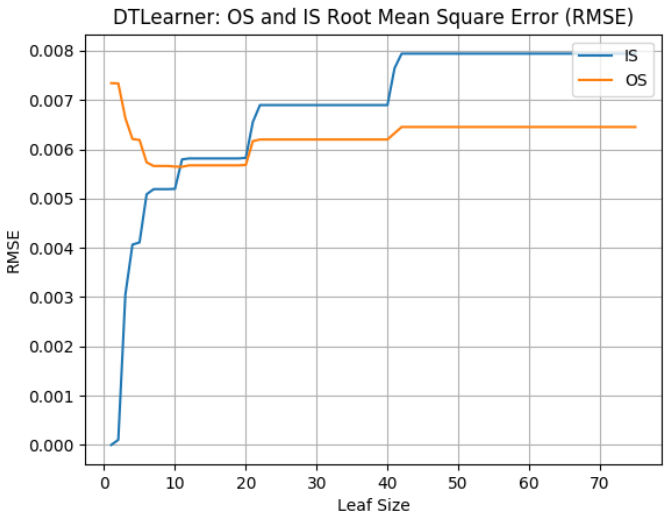


Figure 2: DTLearner out of sample vs. in sample RMSE for Istanbul.csv with leaf sizes [1-75].

This plot corroborates ***overfitting at a leaf size ~≤ 7***. This is an inflection point, where in sample RMSE, which was lower (more accurate) at leaf sizes ~< 7 becomes greater than out of sample RMSE.

Question 2*: Can bagging reduce or eliminate overfitting with respect to leaf size? Again, consider the dataset istanbul.csv with DTLearner. To investigate this, choose a fixed number of bags to use and vary leaf size to evaluate. Provide charts to validate your conclusions. Use RMSE as your metric.*

For this experiment, the number of bags were fixed at 10, while the leaf size was on the interval [1-75] and incremented by one for each run. RMSE was measured and plotted for each leaf size. Bagging, an ensemble method, also known as a random forest, introduces randomness by growing each tree using a bootstrap sample of training data.

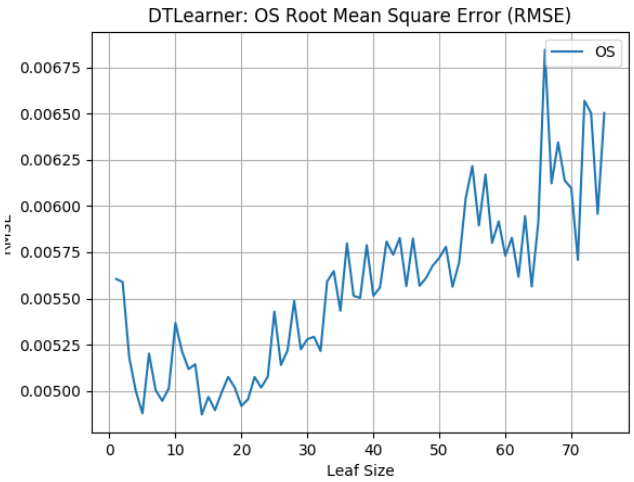


Figure 3: Bag(DTLearner, Bags= 10) out of sample RMSE for Istanbul.csv with leaf sizes [1-75].

Like DTLearner, RMSE is minimized for leaf sizes between 7 and 20. However, notice RMSE does not improve (lower score) as much as DTLearner for leaf sizes ≤ 7, and RMSE variance is smaller for leaf sizes ~≤ 20. This shows that ***bagging reduces overfitting vs. DTLearner***. As the leaf size is increased, RMSE increases. This is not from overfitting, but the result of a larger leaf size allowing too many samples per leaf. Thus, the model will have more opportunity to misinterpret novel data.

Question 3*: Quantitatively compare "classic" decision trees (DTLearner) versus random trees (RTLearner). In which ways is one method better than the other? Provide at least two quantitative measures. Note that for this part of the report you must conduct new experiments, don't use the results of the experiments above for this.*

I chose to compare the DTLearner with RTLearner by evaluating their performance with two quantitative measures:

- **Computational Efficiency**: To evaluate computation efficiency, I created an experiment that would compare the time learners spent in the learning phase, i.e., constructing the model (tree) from the training data. I chose not to compare computational efficiency for generalization to the test set, as, given the same constraints for learning (leaf size), it is expected that the trees created by the learners will share a similar complexity. Thus, query times will not significantly differ.
- **Generalization to the testing data: RMSE**: As per the first two experiments, leaf size was on the interval [1-75] and incremented by one for each run. RMSE scores were obtained and plotted vs. leaf size for each iteration.

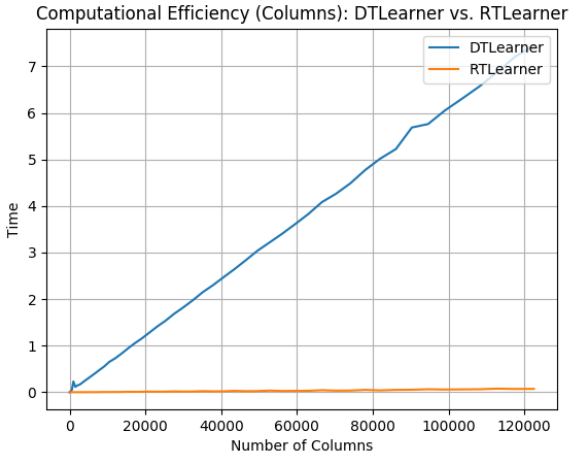
The experiment would validate the following hypothesis:

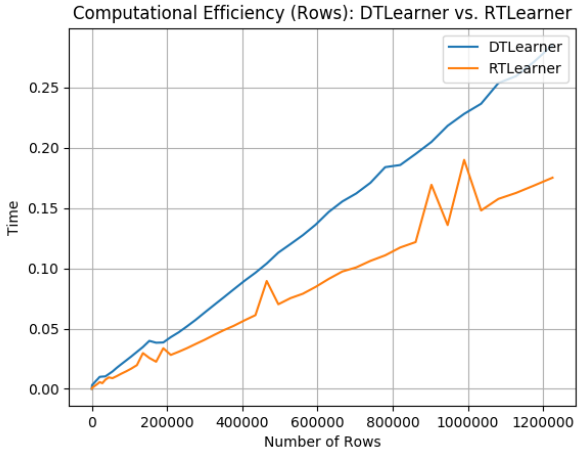
- DTLearners are slower than RTLearners. This has to do with the algorithm that each model employs for learning. A DTLearner will use some form of Information Gain: Entropy, Correlation or Gini Index to determine best feature i to split on. It will then need to find a SplitVal by sorting data to find a median, i.e., ***SplitVal = data[:,i].median()***. RTLearners will randomly determine a feature I and a SplitVal by taking the average of two randomly selected elements in i. i.e., ***SplitVal = (data[random,i] + data[random,i]) / 2***.
- DTLearners generalize (are more accurate) better than RTLearners. Since DTLearners put in more work to create a tree by utilizing information gain for feature selection and median for split value, this single tree might have greater accuracy than a single, randomly constructed tree. Recall, higher accuracy is defined by lower RMSE.

Given the hypothesis that a DTLearner is more accurate than a single RTLearner, I decided to explore a DTLearner to an RTLearner and a random forest of RTLearners, i.e., Bagging(RTLearner, Bags=10) by varying leaf size. Generalization was performed using the same dataset (Istanbul.csv) as prior experiments. To be fair, I also considered computational complexity for all three learners. To test computation efficiency, I created a function (***see Appendix for details***) to grow a dataset over a sequence of iterations. I chose the Istanbul.csv dataset and increased either the number of rows (samples), or the number of columns (dimensionality) over a sequencer of runs. I evaluated the time it took for each learner to construct the tree at each iteration. For this experiment, leaf size was held constant at 1.

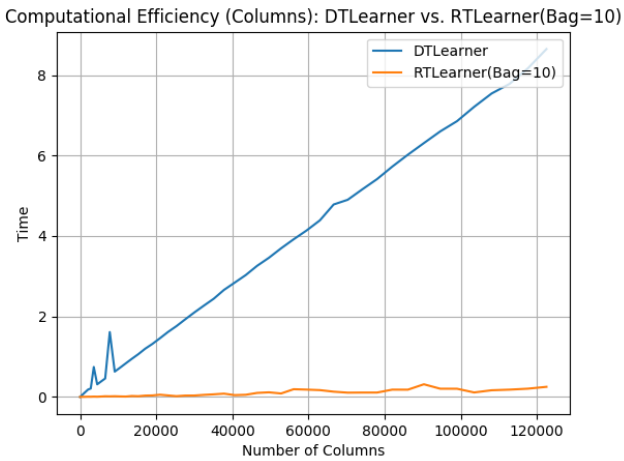
Figures 4-7: Computation efficiency for DTLearner vs. RTLearner vs. Bag(RTLearner, Bags= 10), leaf size = 1 for all learners. Each figure depicts the results of an experiment that evaluates computation efficiency for each learner vs. the size of the dataset (either sample size, or dimensionality) over fifty iterations. The time component is dependent on the dataset size and is measured in seconds.

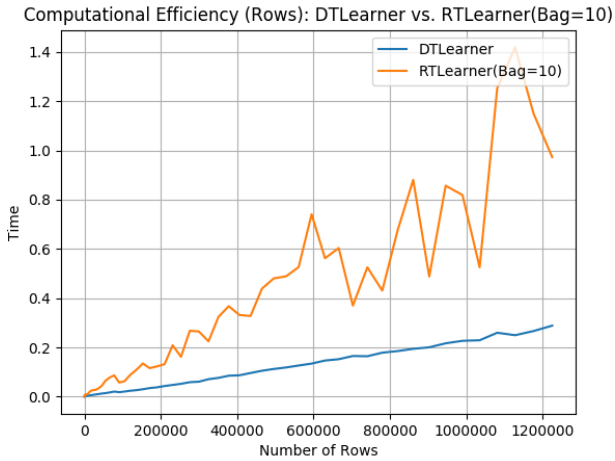
- To evaluate sample size, the number of rows grown by 1000 per iteration.
- To evaluate dimensionality, the number of columns grown by 100 per iteration.





:





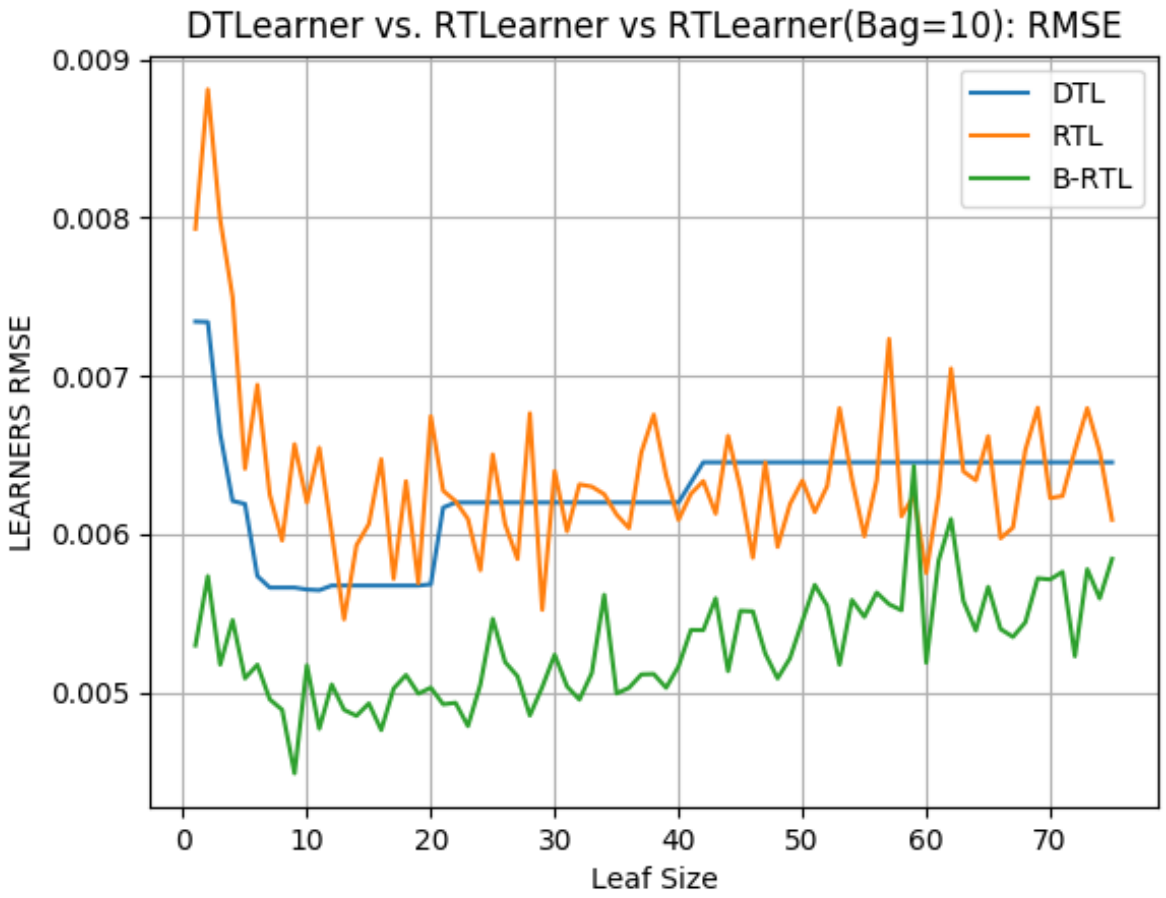
The experiments present some interesting, and perhaps surprising observations into each learners' computation efficiency.

- DTLearners are less efficient than a single RTLearner. This is not surprising, however, it's interesting to observe the source of the greatest inefficiency. Increasing the number of samples did not have a large impact on efficiency. However, as the dimensionality (number of column) is increased, the RTLearners efficiency is very noticeable.
- RTLearners with Bagging is significantly more efficient that a DTLearner as dimensionality is increased. As sample (row) size is increased, RTLearners with Bagging become slightly slightly less efficient that a DTLearner.

Figure 8: Generalization (RMSE) for DTLearner vs. RTLearner vs. Bag(RTLearner, Bags= 10), leaf size = 1 for all learners. Each figure depicts the results of an experiment that evaluates RMSE for each learner vs. the leaf size of the dataset over 75 iterations. For each iteration, the leaf size is incremented by 1.

The experiments show that:

- Bagging(RTLearners) consistently has the lowest RMSE score. For leaf size on the interval [7-20], DTLearners has lower RMSE than RTLearner. For leaf size ~≥ 20 the mean RMSE for RTLearners and RMSE for DTLearners RMSE are in agreement. Both DTLearners and RTLearners exhibit overfitting when leaf size ~≤ 7. Bagging(RTLearners) minimizes overfitting, which agrees with the observation from the Bagging(DTLearners) experiment.



DTLearners generalize better than RTLearners for leaf sizes in the experimental optimal range [7-20]. Both learners' exhibit overfitting when leaf size ~≤ 7. Bagging(RTLearners) greatly reduces overfitting, while also demonstrating the best generalization. Accounting for computational efficiency, bagging(RTLearners) also had comparable or even better (for high dimensional) performance than a single DTLearner.

III. Conclusion

DTLearners and RTLearners overfit when the when there is no bias for simpler trees. Introducing a bias by increasing the number of samples per leaf (larger leaf size) reduces overfitting. Another approach to reduce overfitting is to utilize ensembles of different trees (random forest). For these experiments, random forests were created through bagging learners, which introduces randomness by growing each tree using a bootstrap sample of training data.

RTLearners with bagging proved to be more accurate that a classic DTLearner using information gain. Bagging(RTLearners) also compares very well and even exceeds the computational efficiency for DTLearners for complex (high dimensionality) datasets. This can be explained from the experiments that demonstrated dimensionality having a greater effect on computational efficiency than sample size. Given that a classic DTLearner relies information gain and median to make decisions, it can be expected that applying these computations over a high dimension dataset would negatively impact computation efficiency vs. the random approach for feature selection/splitting employed by RTLearners.

Some areas to explore in future work would consider bag size, and how it affects both generalization and computation efficiency.

IV. Appendix

Function to grow a dataset by row, column, or both. This function uses NumPy slicing of 2-dimensional arrays.

```
learner_dtl = dt.DTLearner(leaf_size = 1, verbose = False)  # constructor
learner_rtl = rt.RTLearner(leaf_size = 1, verbose = False)  # constructor

num_runs = 50
# scalar_row = 1000
scalar_row = 0
# scalar_col = 0
scalar_col = 100

total_runs = np.zeros(num_runs, dtype=float)
num_rows = np.zeros(num_runs, dtype=float)
num_cols = np.zeros(num_runs, dtype=float)
run_time_dtl = np.zeros(num_runs, dtype=float)
run_time_rtl = np.zeros(num_runs, dtype=float)

      for x in range(1, num_runs):
            grow_data = np.zeros((data.shape[0]+scalar_row*x,
                        data.shape[1]+scalar_col*x), dtype=float)
            grow_data[scalar_row*x:,scalar_col*x:] = data       # lower right
            grow_data[scalar_row*x:,:data.shape[1]] = data[:,:] # lower left
            grow_data[:data.shape[0],:data.shape[1]] = data[:,:]# upper left
            grow_data[:data.shape[0],scalar_col*x:] = data[:,:] # upper right

            data = grow_data
            Ytrain = data[:,-1]           # Last col, Yi
            Xtrain = data[:,:-1]    # All data, excludeing last col, Xi

            start_dtl = time.time()
            learner_dtl.addEvidence(Xtrain, Ytrain)
            run_time_dtl[x] = time.time() - start_dtl

                  start_rtl = time.time()
                  learner_rtl.addEvidence(Xtrain, Ytrain)
                  run_time_rtl[x] = time.time() - start_rtl

                  total_runs[x] = x
                  num_rows[x] = Xtrain.shape[0]
                  num_cols[x] = Xtrain.shape[1]
```

To generate the graphs for the three questions, I used Boolean values to select the graph based on the question.

```
verify_learners = False
questions_1and2 = False
question_3_1 = False
question_3_1_BRTL = False
question_3_2 = True
```