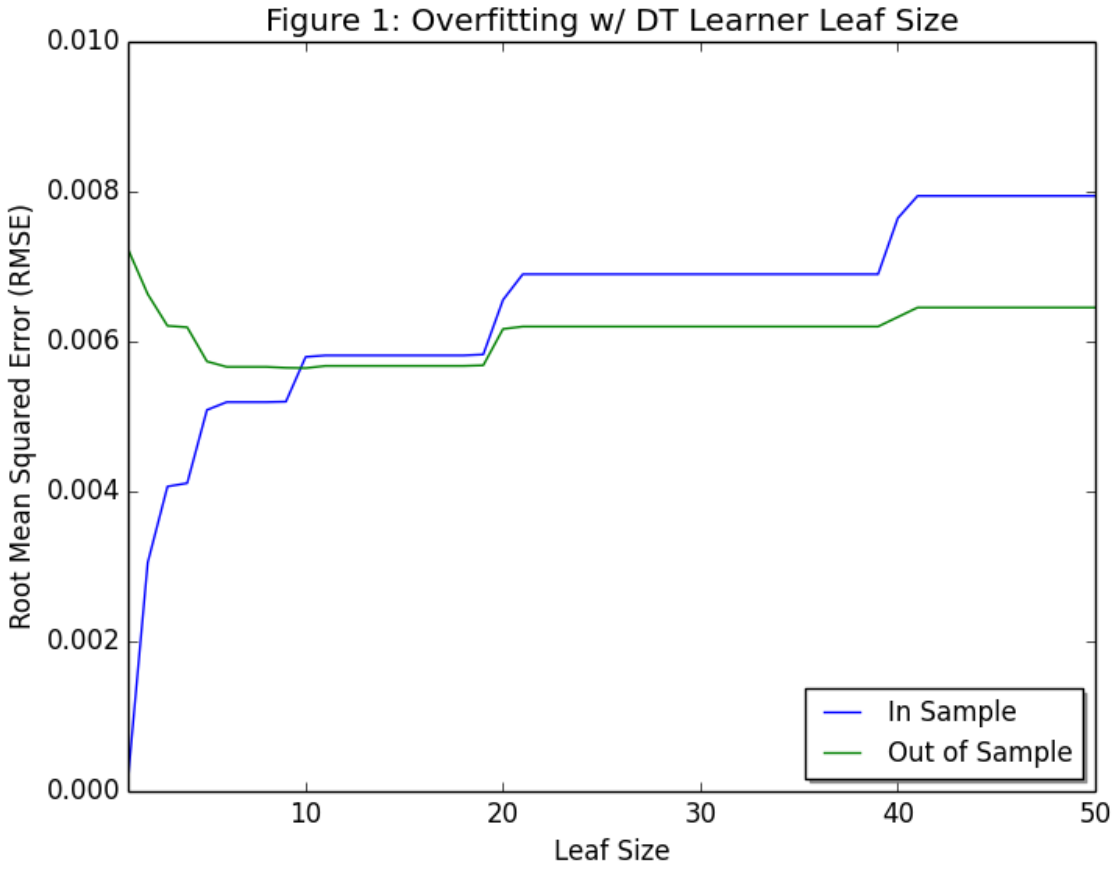
Corey Caskey
CS 7646 – ML for Trading
Homework 3 – Assess Learners

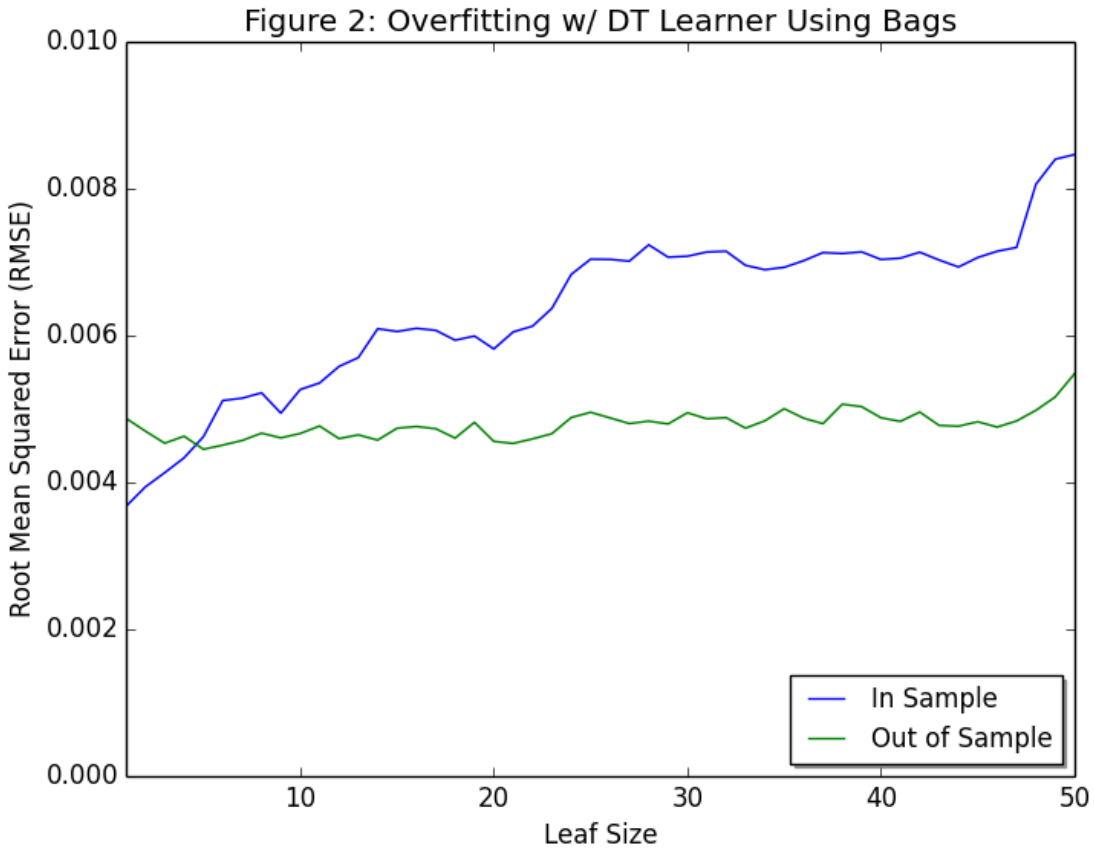
Does overfitting occur with respect to leaf size?

In short, yes, overfitting does occur with respect to leaf size. To answer this question, I ran 50 simulations of my DT Learner in which I passed in the Istanbul.csv data and a different leaf size each time (1 to 50). The root mean squared error (RMSE) is a good indication of how close our learner tree models are at predicting the Y values for the Xi features. In Figure 1 below, I plotted the RMSE values for the training, in-sample data (blue) and testing, out-of-sample data (green) for each distinct leaf size parameter. Based on the plot, I would conclude that the overfitting occurs within the leaf size range of 5 to 1. It is clear that between leaf sizes 50 to 6, the in-sample error is decreasing at a steady rate and the out-of-sample error is fairly stable, albeit slightly decreasing. However, once we hit the leaf size threshold of 5, the RMSE for the in-sample data begins dropping drastically while the out-of-sample RMSE starts increasing. This is a textbook definition of overfitting because we're reaching a point of (almost) perfectly modeling the training data at the cost of imperfectly modeling the testing data.



Can bagging reduce or eliminate overfitting with respect to leaf size?

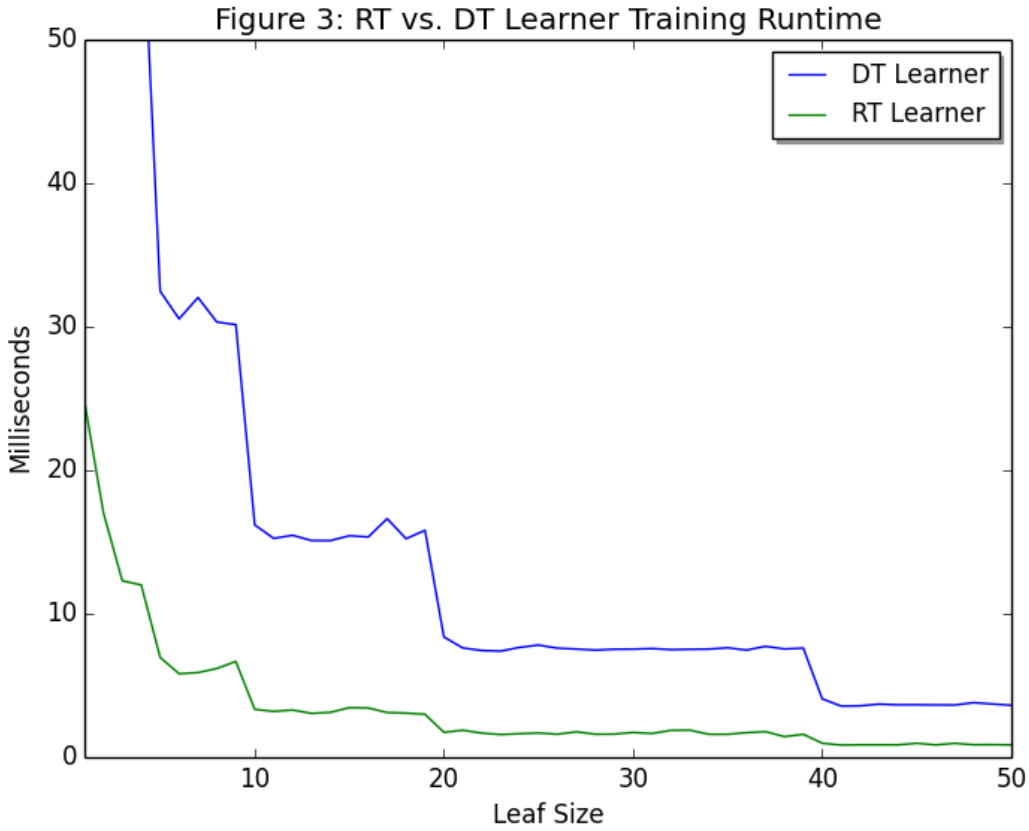
I found that bagging does help reduce overfitting with respect to leaf size, but it does not entirely eliminate it. To answer this question, I once again ran 50 simulations using the Istanbul.csv data and varying leaf sizes between 1 and 50. However, this time I used a Bag Learner that took in my DT Learner as a parameter with a bag size of 20. As mentioned above, the root mean squared error (RMSE) is a good indication of how close our learner tree models are at predicting the Y values for the Xi features, so this was my dependent variable. In Figure 2 below, I plotted the RMSE values for the training, in-sample data (blue) and testing, out-of-sample data (green) for each distinct leaf size. The lines plotted in Figure 2 follow the same general pattern as in Figure 1. Between leaf sizes 50 to 6, the RMSE for in-sample data is decreasing steadily and the out-of-sample RMSE is fairly stable, with a slight decrease. But, again, when we hit the leaf size threshold of 5 (in the direction of leaf size 5 to 1), the RMSE for the out-of-sample data starts increasing and the RMSE for the in-sample data continues to decrease, at a slightly faster rate. Thus, overfitting still persists. However, the out-of-sample error isn't increasing as fast or reaching as high of a value as in Figure 1. Similarly, the in-sample error isn't decreasing as fast or reaching as minimal of a value as in Figure 1. Therefore, bagging has helped reduce the effect of overfitting. This makes sense because bagging allows us to sample with replacement from the training data and make several weak models from randomly selected items that are eventually aggregated. This reduces variance in the data and the likelihood of overfitting.



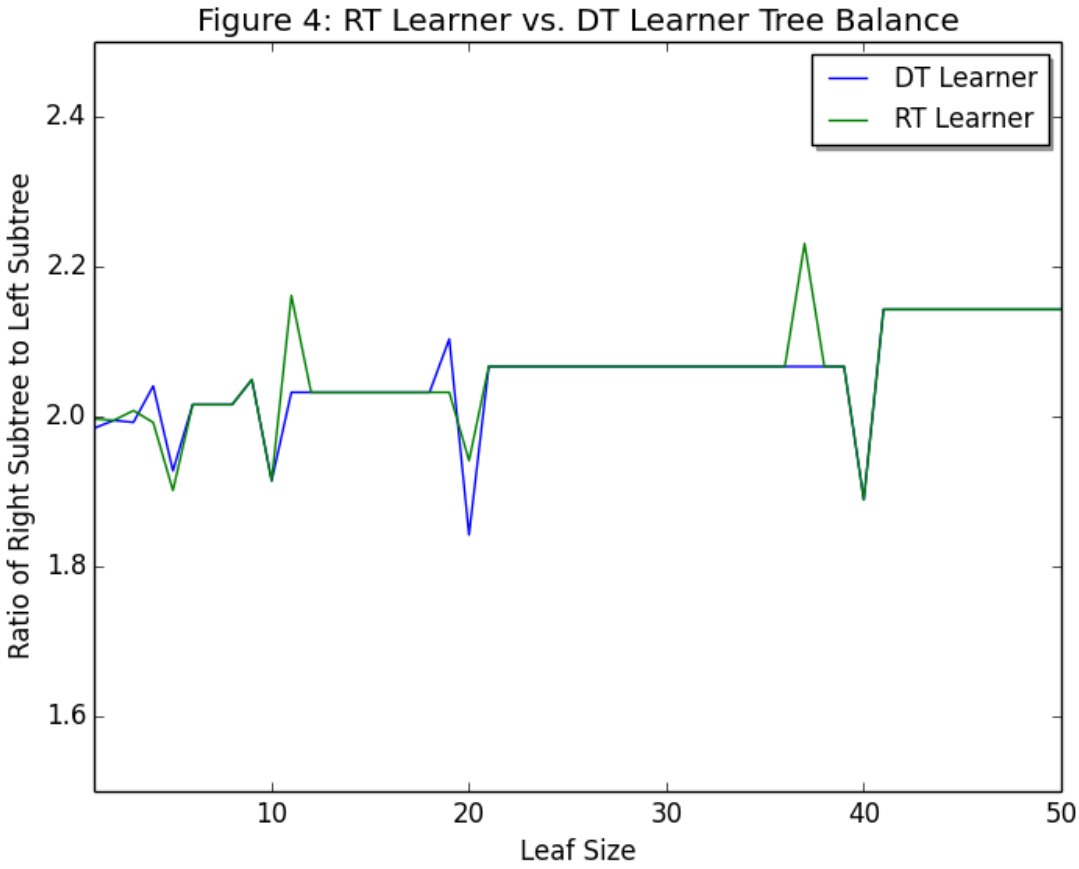
Quantitatively compare "classic" decision trees (DT Learner) versus random trees (RT Learner). In which ways is one method better than the other?

The two quantitative measures I used to compare DT Learner and RT Learner were 1) the time (in milliseconds) it took to train the learner and build the model and 2) the balance ratio of the tree model.

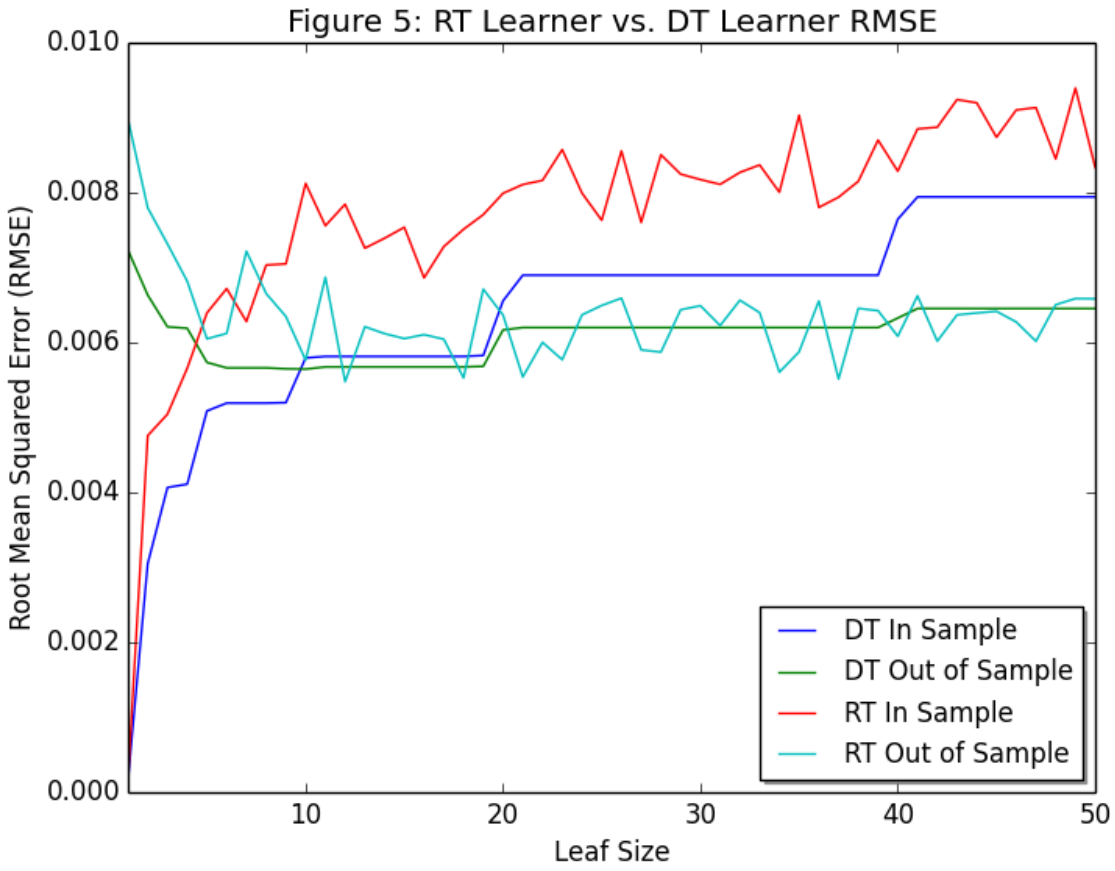
To gather the runtime data, I captured the time immediately before the *addEvidence* method was called as well as right after it finished its execution and then computed the difference between these times. This was done for both the DT Learner and the RT Learner. Overall, I performed this computation process 50 times, in which I varied the leaf size for each simulation run (i.e. iterating over leaf sizes 1 to 50). Figure 3 below depicts the runtime data collected for both learners for each leaf size value, with DT Learner in blue and RT Learner in green. As we can see, the RT Learner trains and builds its model faster than DT Learner for every leaf size. This makes sense from the standpoint of the required code execution because DT Learner has to iterate over every Xi feature when it attempts to pick one to split on for the next subtree. This will take more time to run than RT Learner which relies only on a randomly generated number to select a feature to split on. Furthermore, based on the conclusion from Question 1, we know that DT Learner is susceptible to overfitting at leaf sizes 5 and under. This could account for the drastic spike in runtime we see for the DT Learner at these leaf size values. While RT Learner does have a similar increase, it is not as severe. This could be that RT Learners are less prone to overfitting because of random selection or that the general execution time is not as impacted by overfitting in the model. One other observation is that the difference in execution time between the two learners decreases as the leaf size increases, which tells me that as we aggregate more data into leaves, both learners will reach a point of indistinguishable runtimes.



The second measure I used for comparison was the balance ratio. To gather this data, I captured the models for the DT Learner and RT Learner after they were trained. I started at the root node of each tree and traversed down the rightmost subtree until I reached a leaf node, at which point I marked its index value (this will be the largest index value for the tree). Next, I went to the root node of the left subtree from the original root node. I likewise traversed down its rightmost subtree until I reached a leaf node. This index represents the largest value of the left subtree. Lastly, I added 1 to the rightmost index and then divided by the left subtree index value. If the ratio was equal to 2, that told me the tree was perfectly balanced. Any value below 2 and any value above 2 would inform me that the tree was heavy on the left side and right side, respectively. Overall, I performed this tree traversal process 50 times, in which I varied the leaf size for each simulation run (i.e. iterating over leaf sizes 1 to 50). Figure 4 below depicts the ratio for both learners for each leaf size, with DT Learner in blue and RT Learner in green. Based on how tightly fit the two trees are to one another for large ranges of leaf size values, it appears that, albeit different implementations, both the DT Learner and RT Learner will generate similarly balanced models. Furthermore, the DT Learner has smaller variations in this ratio value than the RT Learner, which makes sense because we are purposely picking the best feature to split on rather than randomly selecting one. One final thing I observed was that for each leaf size value $a(n) = 2 * a(n - 1)$ where $a(0) = 5$, there is a drop in the ratio value for both the RT Learner and DT Learner. I'm not entirely sure why this happens, but it was an interesting pattern to see.



To get a little more clarity on how DT Learner and RT Learner can differ from one another, I plotted their RMSE values for both their in-sample and out-of-sample data. I'm not including this as a quantitative measure. This is simply to help me draw a better conclusion about these learners. To do this, I performed the same process as in Question 1, but with the addition of the RT Learner. The results are depicted in Figure 5 below. As we can see, the RT Learner has very sporadic and unpredictable RMSE values over small intervals (i.e. a lot of value fluctuation), but, over time, it follows the same general direction as the DT Learner. It also appears that, just like DT Learner, the RT Learner suffers from overfitting in the range of leaf sizes from 5 to 1, which provides evidence against the prediction I made earlier using Figure 3.



From the results in Figures 3 through 5, I have reached the following three conclusions:

1) RT Learner is better than DT Learner in training runtime.
2) There is no *extreme* difference in the tree balance of the two learners.
3) Building off conclusion 2, while these two learners are similarly balanced, DT Learner is ultimately a better modeling option than RT Learner because it's more predictable in the short-term (large portions of stable error) and, overall, has RMSE values that are generally lower than that of the RT Learner.