Assess Learners Report

Experimental Setup

To generate the graphs for all sections, I simply ran a modified version of Professor Balch's grade_learners.py file. I modified the code to collect RMSE values for an array of leaf sizes and plot the data. I have included an example of the modified code file as an appendix if you would like to reproduce one of my charts (this assumes you have access to the data folder and learner files).

Discussion of Overfitting vs Leaf Size

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).

Yes, overfitting does occur at lower leaf sizes. At the extreme end with a leaf size of 1, the terminal node only has identical entries in it (1 observation). This means that if you ran a prediction with training data for that node into the tree, you predict the Y-value very accurately (low training error). But if you put an observation into the tree that hasn't been trained with, your tree will simply provide the closest estimate it has seen in the training data. Overfitting occurs when *test error* increases even as you reduce the *training* error through additional tuning.

In the Istanbul data, our model shows signs of overfitting between Leaf Sizes of 1-45.

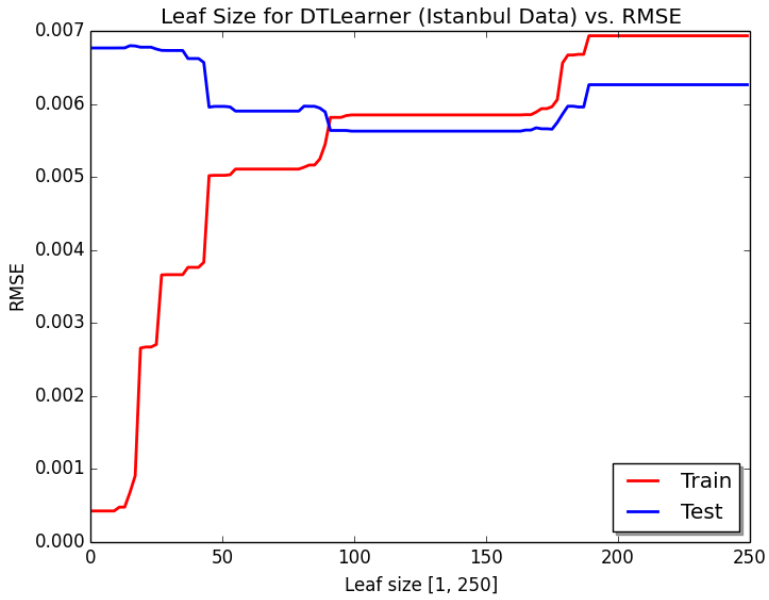

Figure 1: You can see that Train error is lowest and Test error is highest when leaf size is small (1-10) and then we see rapid change as we increase leaf size. Test and Train RMSE is approximately equal at Leaf Size of 90. Performance on test data ceases to decrease appreciably by Leaf Size 100, and eventually as leaf size continues to increase our model becomes too general and test and train error increase together.

Discussion of Bagging and Overfitting Resistance

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 and/or tables to validate your conclusions.

Yes, bagging significantly reduces overfitting with respect to leaf size. I ran an experiment with a 20-bag and 3-bag learner where the base model was DTLearner.

As you can see in Figures 2 and 3, the number of bags you use determines the resistance to overfitting. There are some signs of overfitting between Leaf Sizes 1 and 45 (just as before), but the error is smaller than in our prior experiment. The 3-bag experiment still shows a higher degree of overfitting at Leaf Size 1-45 (Figure 3), but the 20-bag experiment shows

minimal overfitting on the same range (Figure 2). Both learners are also quite resistant to over-generalization, as the Test performance remains relatively flat while the Train error jumps in the 200-250 Leaf Size range.

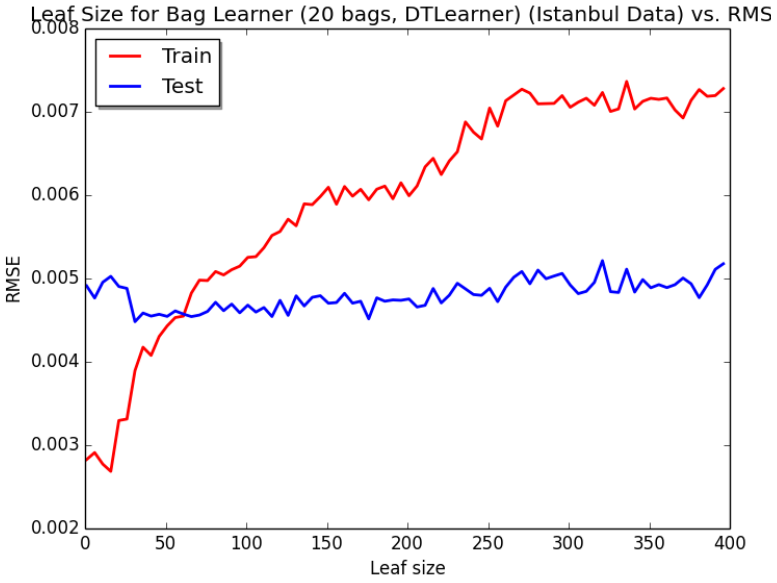

Figure 2

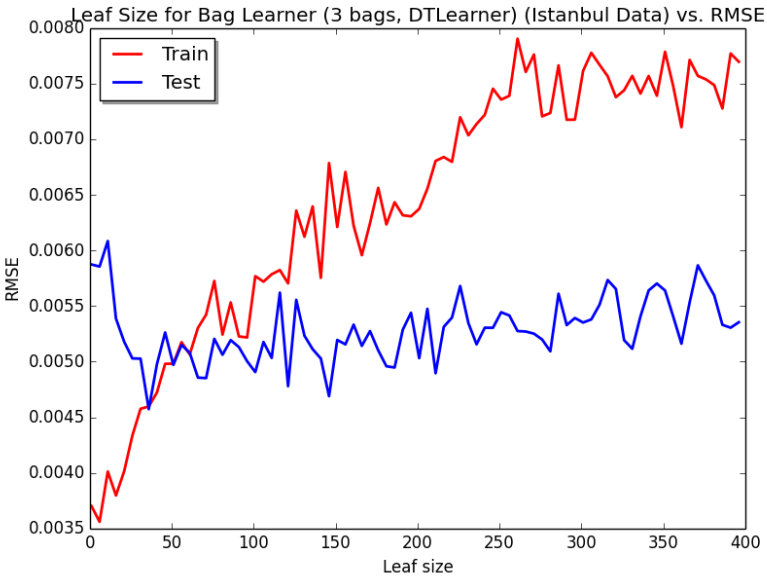

Figure 3

Comment on Figure 2: Notice that as we saw before bagging, the training error is low when leaf size is small (Leaf Sizes 1-25). But unlike before, we see that Test error shows a lower value of 0.0048 at Leaf Size 1 (versus 0.0068 for a single DTLearner). The bag learner shows a gradual decrease in Test Error until it reaches a minimum of approximately 0.0045 at leaf size ~100. Even as Train Error continues to grow as we expand Leaf Size, test performance remains resilient with only a small increase in Test RMSE back to starting levels. At Leaf Size 1, Test RMSE for the 20-bag learner was 30% lower than the DTLearner Test RMSE!

3. <u>Quantitatively compare "classic" decision trees (DTLearner) versus random trees (RTLearner). In which ways is one method better than the other?</u>

Figure 4

In certain cases where there may be unclear (e.g. nonlinear) relationships between features and the output value, a random RTLearner can outperform the classic DTLearner at smaller leaf sizes.

At small leaf sizes, RTLearner shows better performance on specific datasets.

Figure 4: Notice that the RTLearner test performance is better than DTLearner test performance at small leaf sizes (1-100), and the advantage continues until Leaf Size reaches 125. At Leaf Size 125, the advantage dissipates and Test RMSE is roughly equal for DTLearner and RTLearner for a short time (Leaf Size 125 – 160). As leaf size grows, RTLearner looses the advantage, and eventually DTLearner test error achieves a minimum (although RTLearner is not far off).

The RTLearner advantage is dependent on the dataset. It shows a smaller advantage in Red Wine data and no clear advantage on the Istanbul data –it is possible that features in the Red Wine and Istanbul datasets are more clearly related to the output variable through a linear relationship (thus DTLearner's feature selection heuristic is better than random guessing).

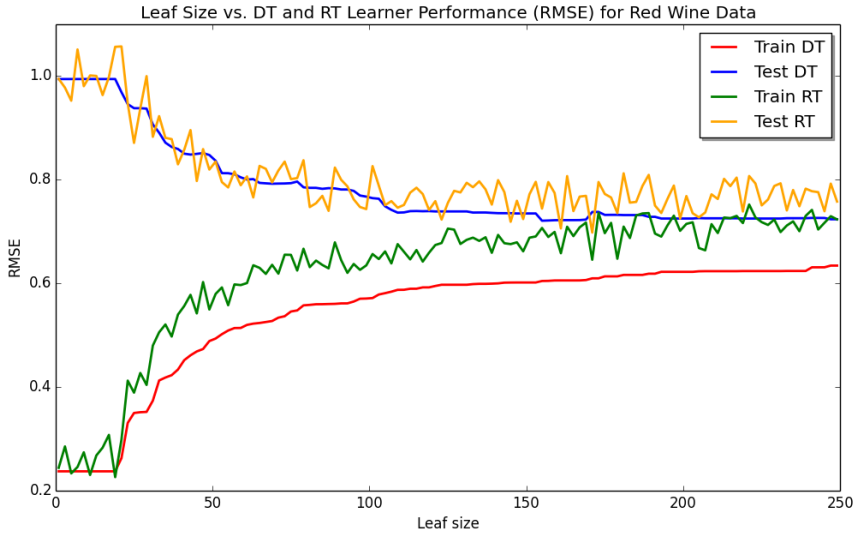

Figure 5

Initially, I wondered why someone would choose features to split on at random rather than using a correlation coefficient or some other decision-making heuristic. Here is a comparison of DTLearner and RTLearner for the Istanbul dataset:

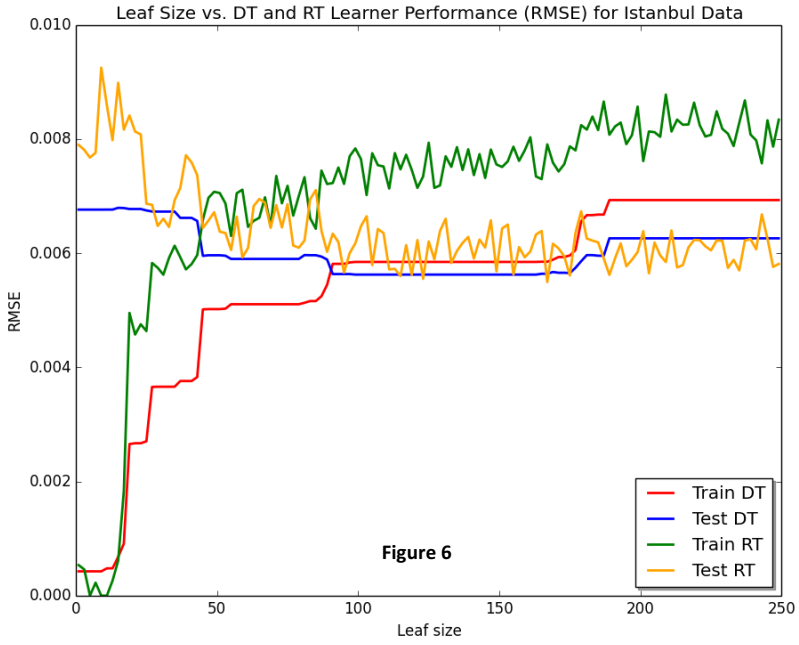

Figure 6

It appears that DTLearner is superior to RTLearner over Leaf Sizes 1-175 (Figure 6). RTLearner shows resilience to overgeneralization past Leaf Size 175 (DT Learner Test RMSE goes up because the model becomes too generalized). DTLearner generally shows a lower Test RMSE than RTLearner, although RTLearner does perform better at random times when its Test RMSE sporadically drops below the DTLearner Test RMSE. There is a point at Leaf Size ~180 where the RTLearner consistently performs better than DTLearner's best Test RMSE, and it continues to perform better than DTLearner as we grow leaf size to 250.

APPENDIX

Code to Reproduce Results for RT and DT comparison on wine data:

`Command: python testDTRT.py ./Data/winequality-white.csv`

Code for testDTRT.py below:

```
"""
Test a learner.  (c) 2015 Tucker Balch
"""

import numpy as np
import math
import DTLearner as dt
import RTLearner as rt
import sys
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

if __name__=="__main__":
    if len(sys.argv) != 2:
        print "Usage: python testlearner.py <filename>"
        sys.exit(1)
    inf = open(sys.argv[1])
    data = np.array([map(float,s.strip().split(',')) for s in inf.readlines()])

    # compute how much of the data is training and testing
    train_rows = int(0.6* data.shape[0])
    test_rows = data.shape[0] - train_rows

    # separate out training and testing data
    trainX = data[:train_rows,0:-1]
    trainY = data[:train_rows,-1]
    testX = data[train_rows:,0:-1]
    testY = data[train_rows:,-1]

    print testX.shape
    print testY.shape

    # create a learner and train it
    leaf_sizes = np.arange(1, 250, 2)
    train_RMSE_DT = []
    test_RMSE_DT = []
    train_RMSE_RT = []
    test_RMSE_RT = []

    for leaf_size in leaf_sizes:
        learner = dt.DTLearner(leaf_size=leaf_size, verbose = True) # create a LinRegLearner
        learner.addEvidence(trainX, trainY) # train it
        print learner.author()

        # evaluate in sample for DT
        predY = learner.query(trainX) # get the predictions
        rmse = math.sqrt(((trainY - predY) ** 2).sum()/trainY.shape[0])
        print
        print "Leaf size is ", leaf_size
        print "In sample results"
        print "RMSE: ", rmse
        train_RMSE_DT.append(rmse)
        c = np.corrcoef(predY, y=trainY)
        print "corr: ", c[0,1]

        # evaluate out of sample
        predY = learner.query(testX) # get the predictions
        rmse = math.sqrt(((testY - predY) ** 2).sum()/testY.shape[0])
```

```
        print
        print "Out of sample results"
        print "RMSE: ", rmse
        test_RMSE_DT.append(rmse)
        c = np.corrcoef(predY, y=testY)
        print "corr: ", c[0,1]

        learner = rt.RTLearner(leaf_size=leaf_size, verbose = True) # create a LinRegLearner
        learner.addEvidence(trainX, trainY) # train it
        print learner.author()

        # evaluate in sample for RT
        predY = learner.query(trainX)  # get the predictions
        rmse = math.sqrt(((trainY - predY) ** 2).sum() / trainY.shape[0])
        print
        print "Leaf size is ", leaf_size
        print "In sample results"
        print "RMSE: ", rmse
        train_RMSE_RT.append(rmse)
        c = np.corrcoef(predY, y=trainY)
        print "corr: ", c[0, 1]

        # evaluate out of sample
        predY = learner.query(testX)  # get the predictions
        rmse = math.sqrt(((testY - predY) ** 2).sum() / testY.shape[0])
        print
        print "Out of sample results"
        print "RMSE: ", rmse
        test_RMSE_RT.append(rmse)
        c = np.corrcoef(predY, y=testY)
        print "corr: ", c[0, 1]

        print "TERMINATE ROUND"

    #plot code

    #DT Plot Code
    train_line = plt.plot(leaf_sizes, train_RMSE_DT, label="Train DT")
    test_line = plt.plot(leaf_sizes, test_RMSE_DT, label="Test DT")
    # use keyword args
    plt.setp(train_line, color='r', linewidth=2.0)
    plt.setp(test_line, color = 'blue', linewidth = 2.0)

    train_line2 = plt.plot(leaf_sizes, train_RMSE_RT, label="Train RT")
    test_line2 = plt.plot(leaf_sizes, test_RMSE_RT, label="Test RT")
    # use keyword args
    plt.setp(train_line2, color='green', linewidth=2.0)
    plt.setp(test_line2, color = 'orange', linewidth = 2.0)

    # Now add the legend with some customizations.
    legend = plt.legend(loc='best', shadow=True)

    plt.xlabel("Leaf size")
    plt.ylabel(("RMSE"))
    plt.title("Leaf Size vs. DT and RT Learner Performance (RMSE) for White Wine Data")
    plt.show()
```