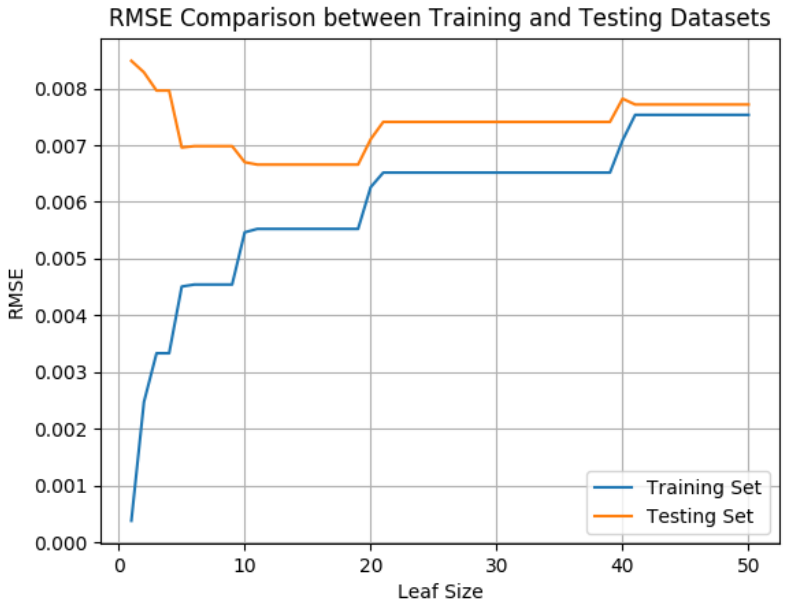
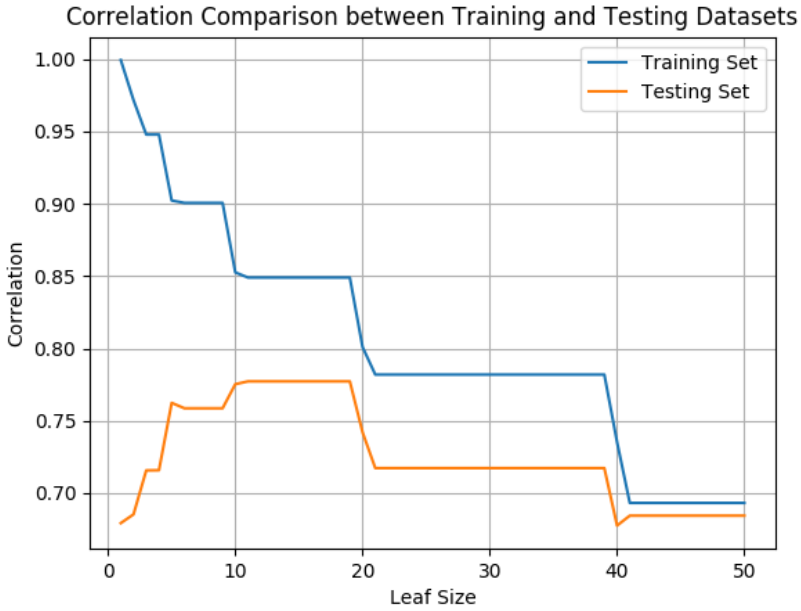
Xicheng Huang
Xhuang343
CS7647 Assess_learner Report

Does overfitting occur with respect to leaf_size? Consider the dataset istanbul.csv with DTLearner. For which values of leaf_size does overfitting occur?

To evaluate this, I took a the RMSE and correlation of the training set and testing set regression results and do a comparison, using leaf sizes from 1 to 50.

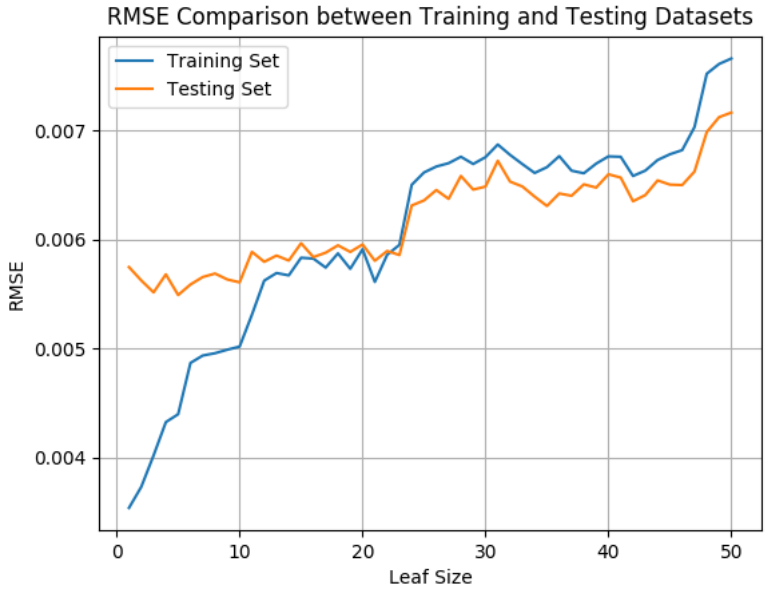


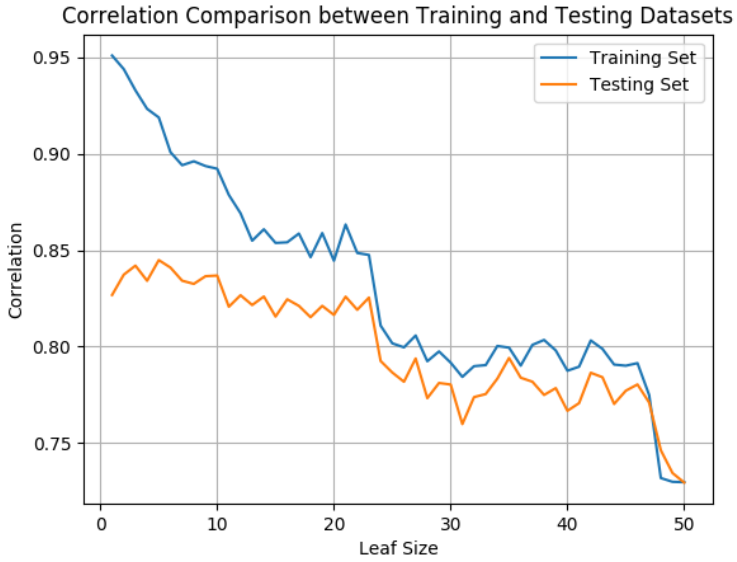


As you can see from the two graphs above that there is a clear inverse relationship between training set and testing set's RMSE and correlation when the leaf size is from 1 to 10. The training set's RMSE is increasing while the testing set's RMSE is decreasing, and the training set's correlation is decreasing while the testing set's correlation is increasing. This inverse relationship indicates that overfitting occurs for leaf size between 1 to 10. For example, when leaf size is at 0, the RMSE for predicting training sample is nearly at 0 and the correlation is at 1. After leaf size becomes greater than 10, the RMSE and correlation relationships between the two samples normalize, and the model doesn't overfit anymore. These two graphs also show that the optimal leaf size for this dataset is 10.

Can bagging reduce or eliminate overfitting with respect to leaf_size?

To evaluate this, I again took RMSE and correlation of training and testing sets using leaf sizes from 1 to 50, setting a fixed number of bags of 20.

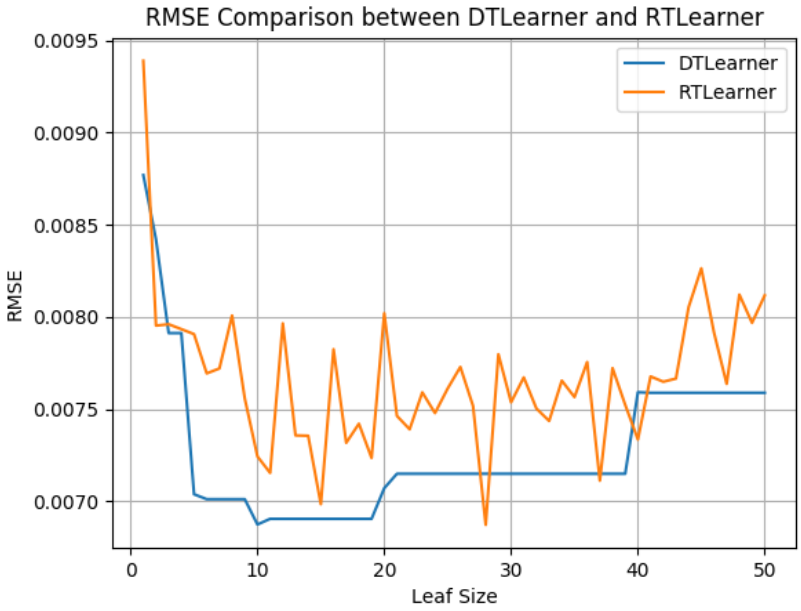


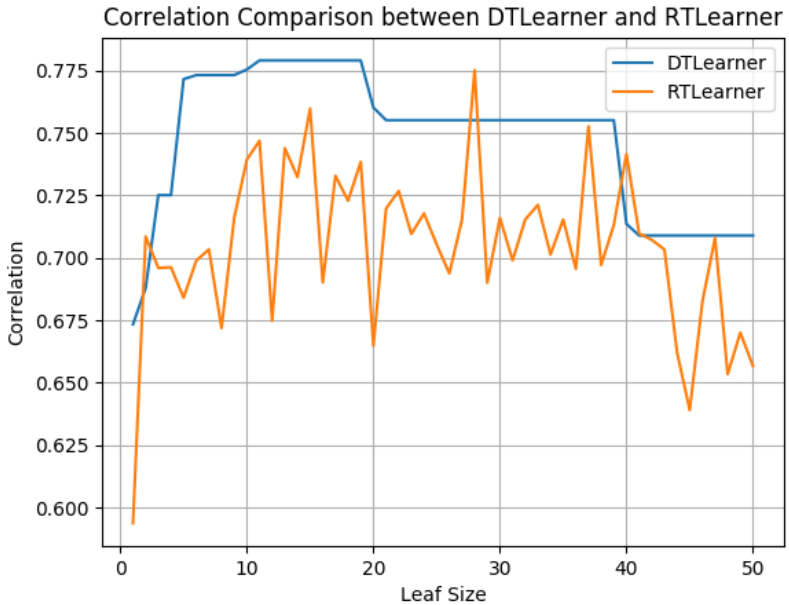


Comparing these two graphs with the graphs above, it is clear that bagging reduces overfitting where the relationships of RMSE and correlation between testing and training set is not as inversely strong as the results without bagging. However, this bagging sample does not eliminate overfitting completely in which there are still minor inverse relationships for RMSE and correlation for leaf sizes less than 5. Once the leaf size becomes greater than 5, the relationships is generally direct and overfitting disappears.

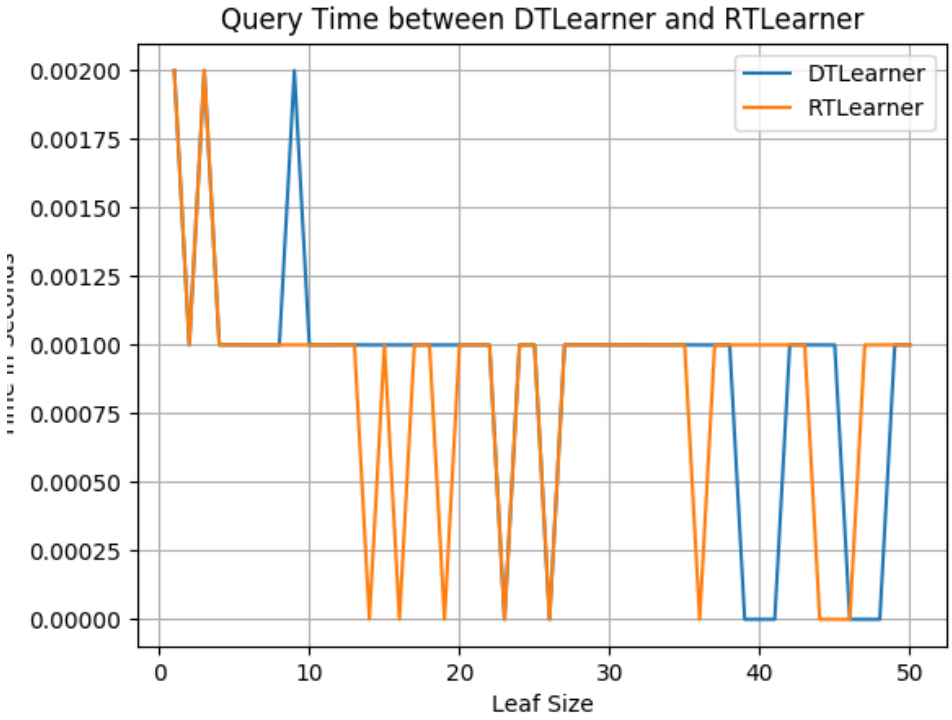
Quantitatively compare "classic" decision trees (DTLearner) versus random trees (RTLearner). In which ways is one method better than the other?

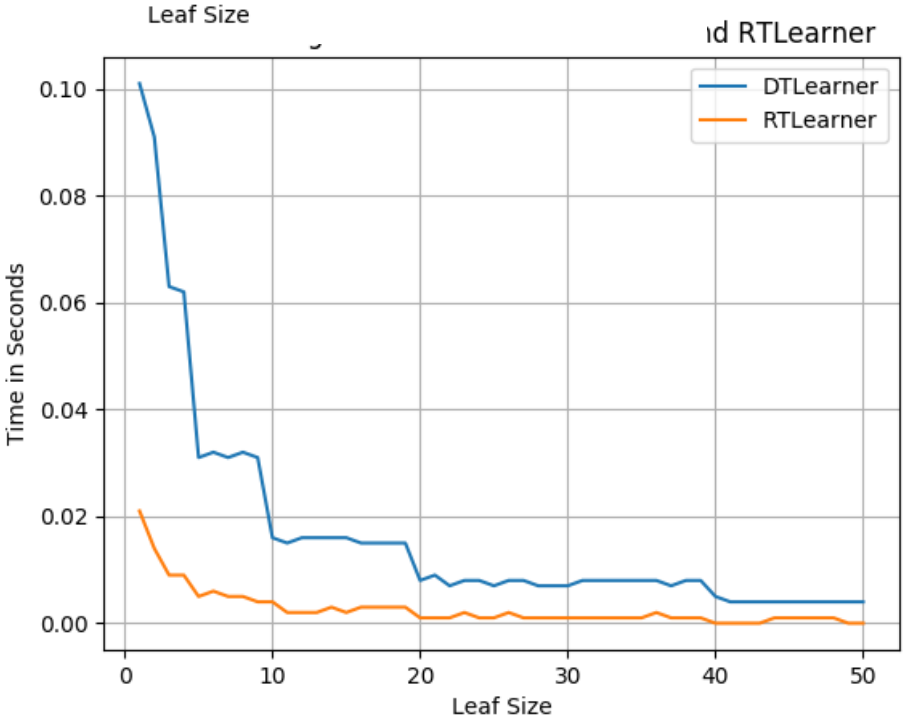
To answer this question, I evaluated the RMSE and correlation of testing sets between the two learners and looked at the time it takes to build and query the two different learners.



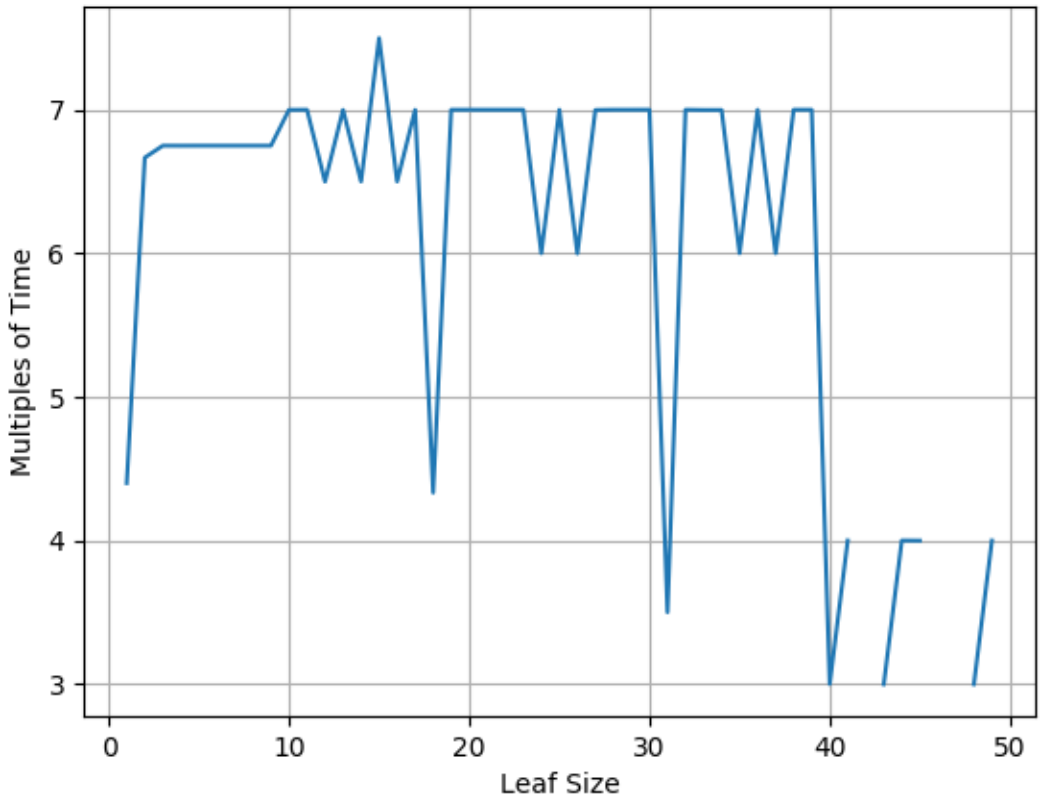


As you can see from the two graphs above, a classic decision trees learner's model is more accurate when predicting where the RMSE is generally lower and correlation is generally higher than a model created by random trees learner. Also, it shows that the accuracy of the random tree learner's model fluctuates more. If bagging is used for random tree learner to create a random forest learner, fluctuation problem will be reduced. This indicates that a classic decision tree learner is has better prediction accuracy than a random tree learner. Now please see below for model building and query time comparison between the two learners.





According to the Query Time between DTLearner and RTLearner graph, there isn't much difference between the two so query time is negligible when comparing the two. Looking at the model building time between the two learners, we can see that a classic decision tree learner takes much longer. The chart below shows time multiples using DTLearner's model building time dividing by RTLearner's. It ranges from 3 times up to around 7 times more.



In conclusion, a classic decision tree learner has better accuracy while a random decision tree learner has better model building performance. To decide which learner is better, we must look at the problem we are trying to solve. If accuracy is the most important factor, a classic decision tree model is a better option. However, if the training dataset is big and grows rapidly which is often the case nowadays and accuracy difference can be negligible, a random decision tree learner is better.

Appendix 1 Python code used to generate graphs:

```
import numpy as np
import pandas as pd
import math
import LinRegLearner as lrl
from DTLearner import DTLearner
import sys
from RTLearner import RTLearner
from BagLearner import BagLearner
import random
from InsaneLearner import InsaneLearner
import time
import matplotlib.pyplot as plt

def fake_seed(*args):
    pass
def fake_rseed(*args):
    pass

if __name__=="__main__":
    if len(sys.argv) != 2:
        print "Usage: python GraphLearner.py <filename>"
        sys.exit(1)
    inf = open(sys.argv[1])
    data = np.genfromtxt(inf,delimiter=',')
    if 'Istanbul.csv' in sys.argv[1]:
        alldata = data[1:,1:]

    datasize = alldata.shape[0]
    cutoff = int(datasize*0.6)
    permutation = np.random.permutation(alldata.shape[0])
    col_permutation = np.random.permutation(alldata.shape[1]-1)
    train_data = alldata[permutation[:cutoff],:]
    # trainX = train_data[:,:-1]
    trainX = train_data[:,col_permutation]
    trainY = train_data[:,-1]
    test_data = alldata[permutation[cutoff:],:]
    # testX = test_data[:,:-1]
    testX = test_data[:,col_permutation]
    testY = test_data[:,-1]

    create a learner and train it
    result=[]
    leafSizes=[]
    for i in range(50):
        learner = DTLearner(leaf_size=i+1, verbose = True) # create a LinRegLearner
        learner.addEvidence(trainX, trainY) # train it
        predY = learner.query(trainX) # get the predictions
        rmseTrain = math.sqrt(((trainY - predY) ** 2).sum()/trainY.shape[0])
        cTrain = np.corrcoef(predY, y=trainY)
```

```
        predY = learner.query(testX) # get the predictions
        rmseTest = math.sqrt(((testY - predY) ** 2).sum()/testY.shape[0])
        cTest = np.corrcoef(predY, y=testY)
        result.append([rmseTrain, cTrain[0,1], rmseTest, cTest[0,1]])
        leafSizes.append(i+1)
    result = np.array(result)
    leafSizes = np.array(leafSizes)
    print result
    print leafSizes
    plt.plot(leafSizes, result[:,0])
    plt.plot(leafSizes, result[:,2])
    plt.legend(['Training Set', 'Testing Set'])
    plt.title('RMSE Comparison between Training and Testing Datasets')
    plt.xlabel('Leaf Size')
    plt.grid(True)
    plt.ylabel('RMSE')
    plt.savefig('plot1.png')
    plt.clf()
    plt.plot(leafSizes, result[:,1])
    plt.plot(leafSizes, result[:,3])
    plt.legend(['Training Set', 'Testing Set'])
    plt.grid(True)
    plt.title('Correlation Comparison between Training and Testing Datasets')
    plt.xlabel('Leaf Size')
    plt.ylabel('Correlation')
    plt.savefig('plot2.png')

    result=[]
    leafSizes=[]
    for i in range(50):
        learner =
BagLearner(learner=DTLearner,kwargs={"leaf_size":i+1},bags=20,boost=False,verbose=False)
        learner.addEvidence(trainX,trainY)
        predY = learner.query(trainX) # get the predictions
        rmseTrain = math.sqrt(((trainY - predY) ** 2).sum()/trainY.shape[0])
        cTrain = np.corrcoef(predY, y=trainY)
        predY = learner.query(testX) # get the predictions
        rmseTest = math.sqrt(((testY - predY) ** 2).sum()/testY.shape[0])
        cTest = np.corrcoef(predY, y=testY)
        result.append([rmseTrain, cTrain[0,1], rmseTest, cTest[0,1]])
        leafSizes.append(i+1)
    result = np.array(result)
    leafSizes = np.array(leafSizes)
    plt.plot(leafSizes, result[:,0])
    plt.plot(leafSizes, result[:,2])
    plt.legend(['Training Set', 'Testing Set'])
    plt.title('RMSE Comparison between Training and Testing Datasets')
    plt.xlabel('Leaf Size')
    plt.grid(True)
    plt.ylabel('RMSE')
    plt.savefig('plot3.png')
```

```
plt.clf()
plt.plot(leafSizes, result[:,1])
plt.plot(leafSizes, result[:,3])
plt.legend(['Training Set', 'Testing Set'])
plt.grid(True)
plt.title('Correlation Comparison between Training and Testing Datasets')
plt.xlabel('Leaf Size')
plt.ylabel('Correlation')
plt.savefig('plot4.png')

resultDTL=[]
resultRTL=[]
leafSizes=[]
for i in range(50):
    learner = DTLearner(leaf_size=i+1, verbose = True)
    start = time.time()
    learner.addEvidence(trainX,trainY)
    end = time.time()
    buildTreeTime = end-start
    start = time.time()
    predY = learner.query(testX) # get the predictions
    end = time.time()
    queryTime = end-start
    rmseTest = math.sqrt(((testY - predY) ** 2).sum()/testY.shape[0])
    cTest = np.corrcoef(predY, y=testY)
    resultDTL.append([rmseTest, cTest[0,1], buildTreeTime, queryTime])

    learner = RTLearner(leaf_size=i+1, verbose = True)
    start = time.time()
    learner.addEvidence(trainX,trainY)
    end = time.time()
    buildTreeTime = end-start
    start = time.time()
    predY = learner.query(testX) # get the predictions
    end = time.time()
    queryTime = end-start
    rmseTest = math.sqrt(((testY - predY) ** 2).sum()/testY.shape[0])
    cTest = np.corrcoef(predY, y=testY)
    resultRTL.append([rmseTest, cTest[0,1], buildTreeTime, queryTime])
    leafSizes.append(i+1)
resultDTL = np.array(resultDTL)
resultRTL = np.array(resultRTL)
leafSizes = np.array(leafSizes)
plt.plot(leafSizes, resultDTL[:,0])
plt.plot(leafSizes, resultRTL[:,0])
plt.legend(['DTLearner', 'RTLearner'])
plt.title('RMSE Comparison between DTLearner and RTLearner')
plt.xlabel('Leaf Size')
plt.grid(True)
plt.ylabel('RMSE')
plt.savefig('plot5.png')
```

```
plt.clf()
plt.plot(leafSizes, resultDTL[:,1])
plt.plot(leafSizes, resultRTL[:,1])
plt.legend(['DTLearner', 'RTLearner'])
plt.grid(True)
plt.title('Correlation Comparison between DTLearner and RTLearner')
plt.xlabel('Leaf Size')
plt.ylabel('Correlation')
plt.savefig('plot6.png')
plt.clf()
plt.plot(leafSizes, resultDTL[:,2])
plt.plot(leafSizes, resultRTL[:,2])
plt.legend(['DTLearner', 'RTLearner'])
plt.title('Model Building Time between DTLearner and RTLearner')
plt.xlabel('Leaf Size')
plt.grid(True)
plt.ylabel('Time in Seconds')
plt.savefig('plot7.png')
plt.clf()
plt.plot(leafSizes, resultDTL[:,3])
plt.plot(leafSizes, resultRTL[:,3])
plt.legend(['DTLearner', 'RTLearner'])
plt.grid(True)
plt.title('Query Time between DTLearner and RTLearner')
plt.xlabel('Leaf Size')
plt.ylabel('Time in Seconds')
plt.savefig('plot8.png')
print resultDTL[:,2]/resultRTL[:,2]
plt.clf()
plt.plot(leafSizes, resultDTL[:,2]/resultRTL[:,2])
plt.grid(True)
plt.xlabel('Leaf Size')
plt.ylabel('Multiples of Time')
plt.savefig('plot9.png')
```