CS7646 MC3P1 Report

Liangliang Yang (lyang338)

Introduction and Overview

In the project, we are mainly going to implement and evaluate a random tree learner (RTLearner.py) and a Bootstrap Aggregating learner (BagLearner). The main idea for the RTLearner (which is build a random decision tree) is from the paper on Random Trees by Adele Culter. The pseudocode for the algorithm is as below:

```
build_tree(data)
    if data.shape[0] == leaf size: return [leaf, median(data,y), NA, NA]
    if all data.y same: return [leaf, data.y, NA, NA]
    else
        determine random feature i to split on
        SplitVal = (data[random, i] + data[random, i])/2
        lefttree = build_tree(data[data[:, i]<= SplitVal
        righttree = build_tree(data[data[:, i]> SplitVal
        root = [i, SplitVal, 1, lefttree.shape[0] +1]
        return (append(root, lefttree, righttree)
```

By randomly choose the split feature (without calculation of information gain) and randomly choose two data points and use their mean value as split value (no longer calculate the median over all data), this random tree can save a lot of time than the normal decision tree algorithm. However, every coin has sides. Although the algorithm is quite fast, the random feature and data points selection mechanism makes it less accurate. In order to improve the accuracy, here we use Bootstrap Aggregating method. n the real implementation, the steps are like below:

1. *split the data into trainingt (60%) and test set(rest 40%)*
2. *sampling the training data with replacement to create several bags (in each bag the total number of data should still be 60% of original data (some of them can be duplicated)*
3. *create seperate trees from each bag and combine/average the predictions from all the different random trees into one prediction*

In the below sections, we will perform detailed experiment to do analysis about these two learners, and we will see the advantage of Bootstrap Aggregating.

Experiment 1: RTLearner with different leaf size (no bagging)

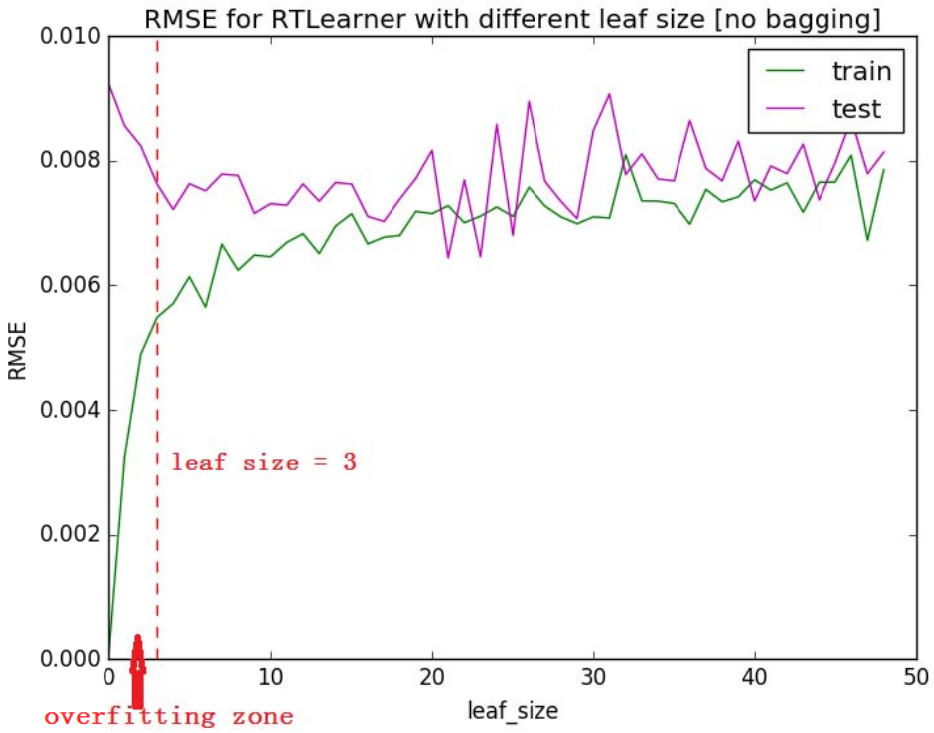
In the first experiment, we need to test how the leaf size will affect the RTLearner, and when the overfitting will occur.

From the discussion at Piazza, there is a tricky point for this experiment. In order to get a

better plot, we need to shuffle the data before splitting the training and test data sets. The core code for this method is as below:

```
for k in range (1, 50):
    np.random.shuffle(data)    # shuffle data
    train_rows = int(round(0.6* data.shape[0]))
    test_rows = data.shape[0] - train_rows
    Xtrain = data[:train_rows,0:-1]
    Ytrain = data[:train_rows,-1]
    Xtest = data[train_rows:,0:-1]
    Ytest = data[train_rows:,-1]
    learner = rt.RTLearner(leaf_size=k)
    learner.addEvidence(Xtrain, Ytrain) # train it
```

The experimental result is in the plot as below:



As we can see from the plot, when we reduce the leaf size, the RMSE of training result gets smaller. However, when the leaf size is too small (which means the tree depth is also very big), the RMSE of the test result gets worse. In other words, the overfitting issue occurs when the leaf size is too small.

Also I have run the code many times, and since the tree is built randomly every time, the plot also varies each time. The typical value of leaf size when overfitting occurs is leafsize <= 3 or 4. In the above plot, from the append time, we can see in this experiment the regions of overfitting is around leaf_size <=3.

One interesting point from the figure (which may not related to the addressing questions), is that we can see, when the leaf size is small (less than 20 in the above experiment case),

the RMSE of training is always smaller than test set. However, when we continue increase the leaf size, there seems to be no difference between the two sets (sometimes the RMSE is bigger for training than test). The reason for this is easy to understand, when the leaf size is two big, the model is too simple. An extreme case is that when the leaf size is the number of all training data, then both training and test are just like blind guess.

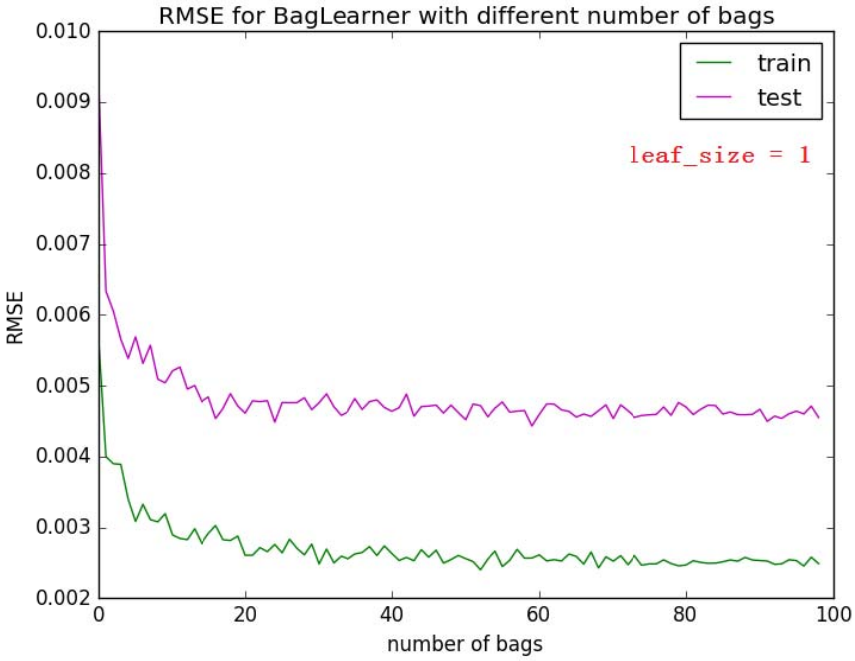
Experiment 2: BagLearner with different bag size (leaf size fixed)

In this experiment we need to explore the effect of Bootstrap Aggregating method. The main idea has been discussed in the above section. In order to study how the number of bags will affect the training/test, we will fix the leaf size separately at 1 (which is in the overfitting region).

The experiment methodology for this experiment is easy. We will call the BagLearner and change the bag number every time. The main code for bagging is as below:

```
for n in range(1, 100):
    learner = bl.BagLearner(learner = rt.RTLearner, kwargs = {"leaf_size":1}, bags =
n,                boost = False, verbose = False)
    learner.addEvidence(Xtrain, Ytrain)    # train it
```

The result plots are as below:



From the above plots, we can see that when we change the bag size, there is no overfitting occurs (no regions that train RMSE decreases and test RMSE increases). Also we can see that both training and test RMSE decrease when we increase the bag size. This can be seen as an evidence that the bagging can reduce the RMSE.

Experiment 3: BagLearner with different leaf size (bag size fixed)

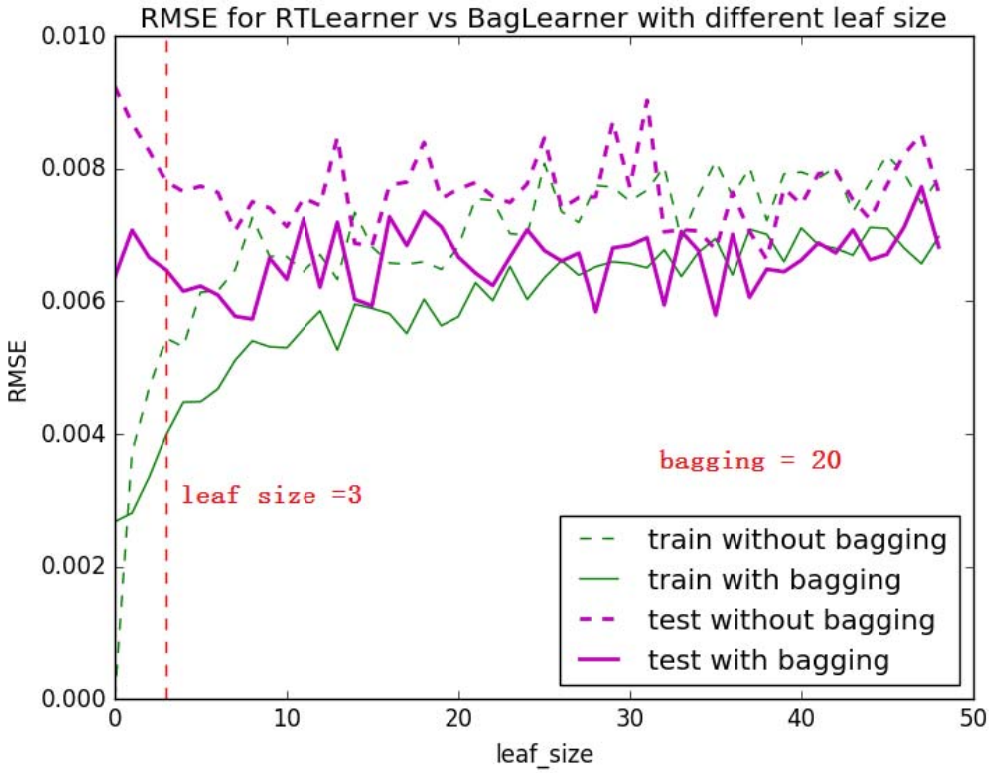
In this experiment we need to continue exploring the effect of Bootstrap Aggregating method, and this time we will fix the bag size, and by varying the leaf size (from overfitting region to non-overfitting region), we would like to see whether the bagging will help reduce or eliminate the effect of overfitting.

The experiment methodology for this experiment is also a little tricky. We will fix the bag size (which is set to be 20), this is easy. However, in order to compare the results of bagging and non-bagging, we need to keep the training data set same for them during the process when we change the leaf size. However, as described in experiment 1, in order to get a better plot, we will shuffle the data every time, so we need to be careful here. The idea is after shuffling the data each time, we will use the same data to train the two learners. The main code under this idea is as below:

```
for k in range (1, 50):
    np.random.shuffle(data) # shuffle data
    train_rows = int(round(0.6* data.shape[0]))
    test_rows = data.shape[0] - train_rows
    Xtrain = data[:train_rows,0:-1]
    Ytrain = data[:train_rows,-1]
    Xtest = data[train_rows:,0:-1]
    Ytest = data[train_rows:,-1]

    #learner no bagging
    learner0 = rt.RTLearner(leaf_size=k)
    learner0.addEvidence(Xtrain, Ytrain) # train it
    #learner with bagging
    learner1 = bl.BagLearner(learner = rt.RTLearner, kwargs = {"leaf_size":k}, bags
=           20, boost = False, verbose = False)
    learner1.addEvidence(Xtrain, Ytrain) # train it
```

The experimental result is as below in the plot:



From the plot, we can see that when there is no bagging, there is still clear overfitting occurs in the region leaf size <= 3 (see dashed train and test result). However, when we apply the Bootstrap Aggregating learner (BagLearner), there is no clear overfitting region anymore. In other words, we can say that by using bagging, we can help reduce or eliminate the effect of overfitting. Also, we can see from the above plot, after applying bagging, the test RMSE under almost all leaf size will become smaller/better.

● Summary

We have successfully implement the random decision tree learner and bootstrap aggregating learner. During the experiment, we have seen that the overfitting issue will occur when the leaf size of the random tree becomes too small (the model is over complicated). Also we can see that by using Bootstrap Aggregating (bagging) method, we will be able to reduce or eliminate the effect of overfitting.

This is a very interesting project, and I have express my thanks for many useful posts in the Piazza. I have gained a much better understanding for the knowledge of Ensemble learners, bagging and boosting.