UTD 2016 Spring

CS6301.5U1 Advanced Computational Methods for Data Science

Assignment 7 – Non-linear Models

(Decision Tree, Boosting, Bagging, Random Forest)

Name: Yu Zhang, Mingyue Sun

NetID: yxz141631, mxs151730

1. Overview
2. Data cleaning up
3. Standard decision tree and pruning
4. Bagging
5. Boosting
6. Random forest
7. Conclusion

1. Overview

In this assignment, we are responsible for analyzing the Northampton Housing Prices dataset using decision tree models. Our solution will be divided into five parts: 1. data cleaning up. 2. build standard decision tree and perform the necessary pruning. 3. use bagging to produce prediction models. 4. use boosting to produce prediction models. 5. use random forest to produce prediction models. Also we will use cross validation on training/test set to compare the models in terms of predicting test MSE and to find the best model.

2. Data cleaning up

This dataset comprises 104 homes in Northampton: MA that were sold in 2007. And it has 30 variables including: housenum, acre, acregroup, adj1998, adj2007, adj2011, bedgroup, bedrooms, bikescore, diff2014, distance, distgroup, garage_spaces, garagegroup, latitude, longitude, no_full_baths, no_half_baths, no_rooms, pctchange, price1998, price2007, price2011, price2014, sfgroup, squarefeet, streetname, streetno, walkscore, zip. The 'pctchange' is the target variable, so we still have 29 predictors left there. Hence we decide to first perform some data cleaning up before buiding the actual models. The steps were performed as below:

```
> #import data
> Housing = read.csv("./Northampton Housing Prices/Northampton Housing Prices/data
set_in_wide_format.csv", sep=",", header = TRUE)
> myData=Housing
> #data clean
> myData=myData[-c(15)]
> myData=myData[-c(1,3:6,8,10,12,13,15,16,21:23,25,27,28)]
> names(myData)
 [1] "acre"          "bedgroup"      "bikescore"     "distance"
 [5] "garagegroup"   "no_full_baths" "no_half_baths" "no_rooms"
 [9] "pctchange"     "price2014"     "squarefeet"    "walkscore"
[13] "zip"
```

There are reasons why we droped out the other 17 predictor variables. As we know neither the price nor the price change of a house will depend on its house number, latitude, longitude, streetname, streetno, hence we first decide to drop them off from the list. Then as we can see that there are dependencies between bedgroup and bedrooms, distance and distgroup, garage_spaces and garagegroup, sfgroup and squarefeet, therefore we just droped off one from each pair. Finally we noticed that the pctchange of a house is express as the diff2014 divided by adj1998 whiel diff2014 could be obtained by substracting price2014 from adj1998, so we think only one variable left will be enough to determine the pctchange and then we kept only the price2014 variable. Also as we think the pctchange between 1999 and 2014 will have nothing to do with the house's price or adjusted price in 2007 or 2011, hence we dropped off these variables as well. That is why after the data cleaning up, we only have 12 predictors left which will make our tree building much more straightforward.

3. Standard decision tree and pruning

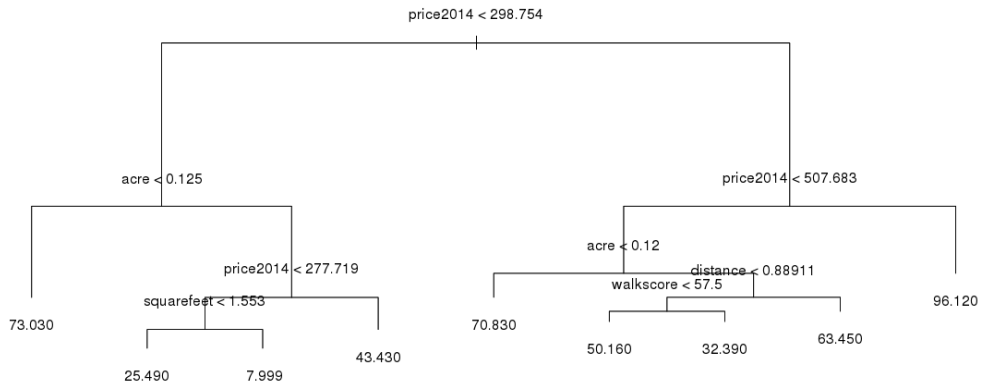
With the data cleaning up finished, then we moved on to the building of decision tree and the prunidng process. To set up the tree model, we first divided the sample into a training set and a testing set with a 0.8:0.2 ratio as shown below:

```
> sub = sample(nrow(myData),floor(nrow(myData)*0.8))
> train = myData[sub,]
> test = myData[-sub,]
> dim(train)
[1] 83 13
> dim(test)
[1] 21 13
> dim(myData)
[1] 104  13
>
```

Then we imported the tree package and started building the tree model with its tree method as below shown:

```
> tree.house=tree(train$pctchange~.,train,split = "deviance")
> tree.pred=predict(tree.house,test,type="vector")
> mse_1=mean((tree.pred-test$pctchange)^2)
> mse_1
[1] 1049.014
>
> plot(tree.house)
> text(tree.house,pretty = 0)
> tree.house
node), split, n, deviance, yval
      * denotes terminal node

 1) root 83 72370.0 41.940
   2) price2014 < 298.754 51 29170.0 30.230
     4) acre < 0.125 5  6532.0 73.030 *
     5) acre > 0.125 46 12490.0 25.580
      10) price2014 < 277.719 37  8441.0 21.230
        20) squarefeet < 1.553 28  2352.0 25.490 *
        21) squarefeet > 1.553 9  4005.0  7.999 *
      11) price2014 > 277.719 9   481.4 43.430 *
   3) price2014 > 298.754 32 25050.0 60.610
     6) price2014 < 507.683 27 13490.0 54.030
      12) acre < 0.12 7  2718.0 70.830 *
      13) acre > 0.12 20  8109.0 48.150
        26) distance < 0.88911 15  3668.0 43.060
          52) walkscore < 57.5 9  1761.0 50.160 *
          53) walkscore > 57.5 6   770.1 32.390 *
        27) distance > 0.88911 5  2882.0 63.450 *
     7) price2014 > 507.683 5  4089.0 96.120 *
```

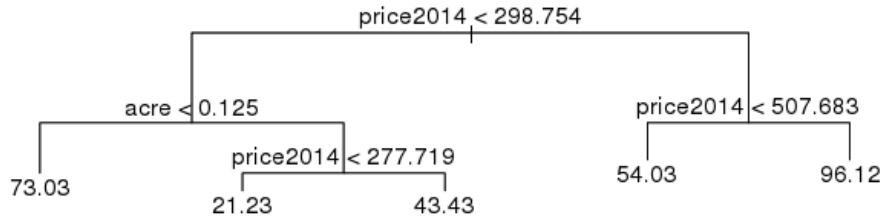
And we will be able to get a decision tree model as below:



In the same time, we found out the MSE for this model to be **1049.014** which was quite huge. Therefore we performed a pruning on the original tree to see whether the MSE on the testing set would be reduced. The procedure is shown as below:

```
> tree.house.cv=cv.tree(tree.house, FUN = prune.tree)
> tree.house.cv
$size
[1] 9 8 7 6 5 4 3 2 1

$dev
[1] 50971.73 49985.12 50860.50 52789.91 54176.53 51461.20 57179.96
[8] 60654.47 75750.75

$k
[1]      -Inf  1136.852  1559.541  2083.593  2665.404  3565.276
[7]  7472.719 10155.183 18145.942

$method
[1] "deviance"

attr(,"class")
[1] "prune"         "tree.sequence"
> plot(tree.house.cv$size,tree.house.cv$dev,type = "b")
> tree.house.prune = prune.tree(tree.house,best = 5)
> plot(tree.house.prune)
> text(tree.house.prune)
>
> tree.prune.pred=predict(tree.house.prune,test)
> test$pctchange
 [1]  42.036518  51.005956  32.017377 -37.775723  46.471579
 [6]  49.381782  33.919240  42.978772  74.904346  52.557787
[11] 117.067738  47.280723  30.837628  29.397031  92.270066
[16]  24.974625  -2.654556  84.166545   9.014561  38.625365
[21]  49.357453
> mse_2=mean((tree.prune.pred-test$pctchange)^2)
> mse_2
[1] 1017.894
```

The decision tree after pruning is shown as below:



We can see that it removed quite a lot of the predictors in the original tree such as squarefeet, distance, walkscore. And to our delight the MSE on the testing set with the pruning was reduced to **1017.894**, although still a little big but was indeed reduced from the original model without pruning. And we could also get a chart as below:

Which gives us an idea that when number of cv was 8, we could get a minimum deviance on the cv model.

4. Bagging

With the standard decision tree and it pruned model set up, we then moved on to the bagging part of this assignment. In order to perform bagging on this dataset, we imported the ipred package and started building the bagging model with its bagging method as below shown:

```
> library(ipred)
>
>
> bagging.house = bagging(pctchange~., data=train,nbagg=20 )
> bagging.house

Bagging regression trees with 20 bootstrap replications

Call: bagging.data.frame(formula = pctchange ~ ., data = train, nba
gg = 20)


> summary(bagging.house)
       Length Class      Mode
y      83     -none-     numeric
X      12     data.frame list
mtrees 20     -none-     list
OOB     1     -none-     logical
comb    1     -none-     logical
call    4     -none-     call
> bagging.house.pred = predict(bagging.house, test)
> bagging.house.pred
 [1] 31.28777 28.15526 44.94807 12.19563 58.09146 33.84191
 [7] 24.09110 31.11341 66.69262 68.71356 68.71356 68.71356
[13] 30.21028 27.38018 55.91997 20.04466 20.19544 49.38442
[19] 73.93702 59.31992 48.29321
> test$pctchange
 [1]  42.036518  51.005956  32.017377 -37.775723  46.471579
 [6]  49.381782  33.919240  42.978772  74.904346  52.557787
[11] 117.067738  47.280723  30.837628  29.397031  92.270066
[16]  24.974625  -2.654556  84.166545   9.014561  38.625365
[21]  49.357453
> mse_3 = mean((bagging.house.pred-test$pctchange)^2)
> mse_3
[1] 703.247
```

And we could see here that the MSE with bagging on the testing set was then heavily reduced to **703.247** which showed the power of bagging on the decision tree model building.

Then we performed a 10 fold cv on the bagging models to see whether the MSE could be reduced further as shown below:

```
> k=10
> err=seq(1:10)
> folds=sample(1:k,nrow(myData),replace = TRUE)
> for(i in 1:k)
+ {
+   train=myData[folds!=i,]
+   test=myData[folds==i,]
+   bagging.house = bagging(pctchange~., data=train,nbagg=20 )
+   pred=predict(bagging.house, test)
+   mse.temp=mean((pred-test$pctchange)^2)
+   err[i]=mse.temp
+ }
> err
 [1]  338.8645  282.4244  123.0997  244.1782 1083.2689  343.4160
 [7]  372.0020  530.8675  928.5136  882.3374
> mse_3=mean(err)
> mse_3
[1] 512.8972
```

And again to our delight, we found out that the MSE of testing set with CV on bagging was then heavily reduced to **512.8972** which was less than half of the original decision tree model (**1049.014**).

5. Boosting

In this part we will use boosting method to do this regression work. The functions we used are provided in R "gbm" package. First, we will split the data into training (80%) and testing (20%) datasets. After training models, we will observe the out comes to see the characteristics of the outcomes of boosting. In the last we will use 10-folds cross validation to calculate MSE in order to compare with other models.

```
> set.seed(1)
> boosting.house = gbm(pctchange~.,data=train, distribution = "gaussian", n.trees = 500, shrinkage
 = 0.03,cv.folds = 10)
> boosting.house.pred = predict(boosting.house,test)
Using 221 trees...
> test$pctchange
 [1]  42.03652  51.00596  32.01738  37.05055  34.15015  32.17845  34.49017  52.93909  46.99423
[10]  45.87919  71.81250 108.41434 112.36714  37.92389  42.17620  39.51595  30.31341  31.29420
[19]  12.62848  51.72926  28.12611
> boosting.house.pred
 [1] 34.51606 31.95186 40.46345 33.02418 27.44324 28.75653 54.45824 48.68686 68.33069 67.74257
[11] 57.08699 73.56492 54.67101 40.78559 37.86647 60.07312 33.71284 23.45898 37.54005 67.38746
[21] 56.40885
> mean((boosting.house.pred-test$pctchange)^2)
[1] 422.0032
```

As the figure showing above, we built 221 "weak models" using the learning rate of 0.03. The MSE in this case is 422.0032. Comparing the prediction and the testing data, we can find that when using Boosting doing regression, most data fit well. However, some data points can be in a bad result. In this case the number 12 and 13 prediction performs really bad and thus contributes a great deal to the MSE.

4.2 10-folds Cross Validation for Boosting

```
> k=10
> err=seq(1:10)
> folds=sample(1:k,nrow(myData),replace = TRUE)
> for(i in 1:k)
+ {
+   train=myData[folds!=i,]
+   test=myData[folds==i,]
+   set.seed(1)
+   boosting.house = gbm(pctchange~.,data=train, distribution = "gaussian", n.trees =700, shrinkag
e = 0.01, cv.folds = 10)
+   pred=predict(boosting.house, test)
+   mse.temp=mean((pred-test$pctchange)^2)
+   err[i]=mse.temp
+ }
Using 647 trees...
Using 686 trees...
Using 690 trees...
Using 700 trees...
Using 688 trees...
Using 697 trees...
Using 575 trees...
Using 699 trees...
Using 657 trees...
Using 700 trees...
> err
 [1]  934.72452 1146.33019  267.23044 1071.50232  764.71030  501.96413  193.22906   88.87927
 [9]  759.51388  319.87473
> mse_boosting=mean(err)
> mse_boosting
[1] 604.7959
```

According to the running result above, we can see the MSE for 10-folds cross validation result is 604.7959. And by observing the details of the MSE in each run, it can be easily find that different training and testing dataset do influence the MSE greatly, the smallest MSE is only 88.87927, the largest, however, can be 1146.33019.

6. Random Forest

In this part we will use random forest to do this regression work. The functions we used are provided in R “randomForest” package. First, we will split the data into training (80%) and testing (20%) datasets. After training models, we will observe the out comes to see the characteristics of the outcomes of boosting. In the last we will use 10-folds cross validation to calculate MSE in order to compare with other models.

```
> library(randomForest)
> rf.house = randomForest(pctchange~., data = train, ntree = 1000, mtry = 3)
> rf.house.pred = predict(rf.house,test)
> test$pctchange
 [1]  42.03652  51.00596  32.01738  37.05055  34.15015  32.17845  34.49017  52.93909  46.99423
[10]  45.87919  71.81250 108.41434 112.36714  37.92389  42.17620  39.51595  30.31341  31.29420
[19]  12.62848  51.72926  28.12611
> rf.house.pred
       1        2        3       10       12       16       30       34       39       40
28.06393 22.49225 42.33297 30.13578 28.28711 20.62973 37.50078 48.41762 57.20173 52.71503
      44       46       49       57       63       64       80       81       86       87
57.53083 59.96885 65.42605 50.08810 37.49684 46.00897 30.04413 21.14990 36.32198 50.34126
      95
54.32400
> mean((rf.house.pred-test$pctchange)^2)
[1] 372.8432
```

According to the results provided above, we can see that the MSE in this case is 377.8432. Which is relatively low in this task. Observing the comparison between the test data and predict result, we can find that usually the prediction is ok. However, when the number of the test data is much larger than the other data, the prediction of them may also go bad. In the following, we will use 10-folds cross validation to see the overall performance of random forest in this case.

```
> k=10
> err=seq(1:10)
> folds=sample(1:k,nrow(myData),replace = TRUE)
> set.seed(1)
> for(i in 1:k)
+ {
+   train=myData[folds!=i,]
+   test=myData[folds==i,]
+
+   rf.house = randomForest(pctchange~., data = train, ntree = 1000, mtry = 3)
+   pred=predict(rf.house,test)
+   mse.temp=mean((pred-test$pctchange)^2)
+   err[i]=mse.temp
+ }
> err
 [1]  444.10505  296.41766  422.93934  564.87785   43.01741 1013.02243 1231.59707  333.64792
 [9] 1388.69878  591.45513
> mse.rf=mean(err)
> mse.rf
[1] 632.9779
```

From the result, the overall MSE of 10-folds cross validation for random forest model in this case is 632.9779. Like boosting, the MSEs in each case vary a lot. The smallest MSE is 43.01741, the largest, however, can be 1388.69878. This mainly because of the distribution of the training and testing data.

7. Conclusion

So finally in terms of predicting test MSE, we could tell that if we use bagging with 10-folds cross validation on the dataset. According to the results shown below, the best model in this case is “Bagging”, which has a lowest MSE of 512.8972. The worst model may be Decision Tree, the MSE of which is 1017.894. Using the tree structure model to do the regression work, the volatility of data set is significant for the final result. Bagging using the strategy of “rebuild” samples can reduce the volatility of the dataset, thus it greatly reduced the MSE. Boosting try to train “harder” on the miscalculated results, so, to a certain extent it reduced some of the “side-effect” of the volatility of the dataset. Random Forest used a similar strategy as Bagging does. Therefore, in fact, Bagging, Boosting and Random Forest all improved the performance of predictions. The difference of the results may come from the difference of the distribution of 10-folds samples.

Model	Decision Tree	Bagging	Boosting	Random Forest
CV MSE	**1017.894**	**512.8972**	**604.7959**	**632.9779**