Lab 7: Tree-Based Methods

First Name: Siyuan **Last Name**: Yin **NetID**: sy652

Lab 7 is due May 5 by 4:30pm in the homework box at 2nd floor of Rhodes Hall. For in-session questions, you only need to submit your answers. For take-home questions, submit both your code and your answers to the problems.

In this lab, we will learn decision trees, bagging and random forests, and apply those tree-based methods on regression and classification problems. Make sure random seeds are always set before doing experiments.

Fitting Classification Trees

The `tree` library is used to construct classification and regression trees. We first use classification trees to analyze the `Carseats` data set. In these data, `Sales` is a continuous variable, so we begin by converting it to a binary variable.

```
> library(tree)
> library(ISLR)
> attach(Carseats)
> High=ifelse(Sales<=8,"No","Yes")
> Carseats=data.frame(Carseats,High)
```

We now use the `tree()` function to fit a classification tree in order to predict `High` using all variables but `Sales`. We can use the `summary()` function to look at the fitted tree. This function displays the variables that are used as internal nodes in the tree, the number of terminal nodes, as well as the (training) error rate.

```
> tree.carseats=tree(High∼.-Sales ,Carseats )
> summary(tree.carseats)
```

Among the 10 variables included in our formula, how many of them are actually used in tree construction? Which variables are not used? Why?

Variable "ShelveLoc", "Price", "Income", "CompPrice", "Population", "Advertising", "Age", and "US" are used, while variable "Urban", and "Education" are not used since they do not have significant influences on the dependent variable.

One of the most attractive properties of trees is that they can be graphically displayed. We use the `plot()` function to display the tree structure, and the `text()` function to display the node labels.

```
> plot(tree.carseats)
> text(tree.carseats,pretty=0)
```

Which indicator appears to be the most important one? Why?

"ShelveLoc" shelving location appears to be the most important indicator since at the first branch it differentiates Good locations from Bad and Medium locations.

In order to estimate the test error, we split the observations into a training set and a test set, build the tree using the training set, and evaluate its performance on the test data.

```
> set.seed(2)
> train=sample(1:nrow(Carseats), 200)
> Carseats.test=Carseats[-train,]
> High.test=High[-train]
> tree.carseats=tree(High~.-Sales,Carseats,subset=train)
> tree.pred=predict(tree.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
```

What percentage of the predictions are correct in the test data?

(86 + 57)/200 * 100% = 71.5%, 71.5% of the predictions are correct.

Next, we consider whether pruning the tree might lead to improved results. The function `cv.tree()` performs cross-validation in order to determine the optimal level of tree complexity. The `cv.tree()` function records the number of terminal nodes of each tree considered (`size`) as well as the corresponding error rate and the value of the cost-complexity parameter used. Note that here by specifying `FUN=prune.misclass`, we use the misclassification rate as the error metric in pruning, rather than the default metric (which is purity).

```
> set.seed(3)
> cv.carseats=cv.tree(tree.carseats,FUN=prune.misclass)
> cv.carseats
```

The tree with 9 terminal nodes results in the lowest cross-validation error rate, with 50 cross-validation errors. We now apply the `prune.misclass()` function in order to prune the tree to obtain the nine-node tree. Then we evaluate its performance on the test data.

```
> prune.carseats=prune.misclass(tree.carseats,best=9)
> plot(prune.carseats)
> text(prune.carseats,pretty=0)
> tree.pred=predict(prune.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
```

Now what percentage of the predictions are correct? Is the classification accuracy improved?

(94 + 60)/ 200 * 100% = 77%, 77% of the predictions are correct and the accuracy is improved by 5.5%

Bagging and Random Forests

Here we apply bagging and random forests to the `Boston` data for a a regression task. The `randomForest()` function from the `randomForest` package can be used to perform both random forests and bagging. Why we do not need different functions for random forests and bagging?

Bagging is a special case of a random forest with m = p. Thus randomForest() can be used to perform both random forests and bagging.

We perform bagging as follows.

```
> library(MASS)
> library(randomForest)
> library(MASS)
> set.seed(1)
> train = sample(1:nrow(Boston), nrow(Boston)/2)
> bag.boston=randomForest(medv~.,data=Boston,subset=train,mtry=13,
        importance=TRUE)
> bag.boston
```

How well does this bagged model perform on the test set?

```
> set.seed(1)
> yhat.bag = predict(bag.boston,newdata=Boston[-train,])
> boston.test=Boston[-train,"medv"]
> plot(yhat.bag, boston.test)
> abline(0,1)
```

From the plot we can see that the our predictions match the true responses pretty well. What is the test set MSE associated with the bagged regression tree? What is the code you use to compute that?

The MSE is 13.34 and the code is mean((yhat.bag-boston.test)^2)

We could change the number of trees grown by randomForest() using the ntree argument:

```
> set.seed(1)
> bag.boston=randomForest(medv~.,data=Boston,subset=train,mtry=13,ntree=25)
> yhat.bag = predict(bag.boston,newdata=Boston[-train,])
```

Report the test MSE.

The MSE this time is 14.05

Growing a random forest proceeds in exactly the same way, except that we use a smaller value of the `mtry` argument. By default, `randomForest()` uses $p/3$ variables when building a forest of regression trees, and $\sqrt{p}$ variables when building a forest of classification trees. Here we use `mtry = 6`.

```
> set.seed(1)
> rf.boston=randomForest(medv~.,data=Boston,subset=train,mtry=6,
        importance=TRUE)
> yhat.rf = predict(rf.boston,newdata=Boston[-train,])
```

Report the test MSE. Did the random forests yield an improvement over bagging in this case?

The MSE in this case is still 14.05 which means random forest does not yield improvement

Using the `importance()` function, we can view the importance of each variable. Two measures of variable importance are reported. The former is based upon the mean decrease of accuracy in predictions on the out of bag samples when a given variable is excluded from the model. The latter is a measure of the total decrease in node impurity that results from splits over that variable, averaged over all trees (this was plotted in ISLR Figure 8.9). In the case of regression trees, the node impurity is measured by the training RSS, and for classification trees by the deviance. Plots of these importance measures can be produced using the `varImpPlot()` function.

```
> importance(rf.boston)
> varImpPlot(rf.boston)
```

Which two variables are most important?

"rm" and "lstat" are the two most important variables.

Take-Home Questions

In the take-home part, we will apply bagging approach and random forests to classification problem. We will us the the `Carseats` data set from the first part of the lab. Follow the instructions, submit the code as well as your answers. Remember to **always** set random seed to be 2 with `set.seed(2)` before **each** step.

1. As in the first part of lab, create a qualitative response "High", and split the observations into a training set and a test set of equal size.

2. Use the bagging approach with 10 trees to fit a model on the training set. (Remember to exclude the `Sales` variable from the predictors). Report the classification error rate on the test data. Is it better or worse than the pruned tree you obtained in the first part of the lab? Use the importance() function to determine the two most important variables for decreasing Gini index.

   The classification error rate is the same as the error rate of pruned trees that both are 23%. The two most important variables in this case are "Price" and "ShelveLoc"

3. Use the bagging approach with 500 trees. Report the test classification error rate.

   The error rate in this case is 19.5%

4. Use the random forest approach with $m = 3$ random features (there are $p = 10$ features in total) and 500 trees. Report the test classification error rate.

   The error rate in this case is 18.5%.

5. **(Optional)** Use the boosting approach with 5000 trees of depth 4 and the default shrinkage parameter. Report the test classification error rate. Use function `gbm()` from package `gbm` to construct boosted trees.

6. **(Optional)** Use logistic regression for the same classification task. Compare its performance with the previous tree-based methods.