Lab #4

Purpose: In this lab, we implement CART, Bagging and Random Forest.

1. Classification Tree with data iris.csv. Load Data iris.csv to object iris and load package tree.

```
rm(list = ls())

iris<-read.csv(file.choose(),header=T)

library(tree)

names(iris)

attach(iris)

summary(iris)
```

It contains 3 classes of a total 150 instances. Each class refers to a type of iris plant. We first create a training set, and fit the tree using the training set.

```
set.seed(1)

train=sample(nrow(iris),nrow(iris)*0.8)

tree.model=tree(Species~.,iris,subset =train)

iris.test=iris[-train,]

Species.test=Species[-train]
```

We use the cv.tree() function to perform 10-fold cross validation to find the best subtree or the optimal way to prune the tree.

```
cv.model=cv.tree(tree.model,K=10,FUN=prune.misclass)

cv.model
```

It shows that the tree with either 3 or 5 terminal nodes results in the lowest cross-validation error rate. Here we prune tree to size 3 using the prune.tree() function and plot the tree. We use the argument FUN=prune.misclass since it is the classification tree. It indicates that we want the classification error rate to guide the cross-validation and pruning process, rather than the default---deviance.

```
prune.model=prune.tree(tree.model,best=3)

plot(prune.model)

text(prune.model,pretty=0)
```

Then we use the pruned tree to make predictions in the testing set and calculate the confusion matrix.

```
prunetree.pred=predict(prune.model,iris.test,type="class")

table(prunetree.pred,Species.test)
```

2. Regression Tree with data Boston.csv. Load Data Boston.csv to object Boston load package tree again.

```
rm(list = ls())

library(tree)

Boston=read.csv(file.choose(),header=T)

head(Boston)
```

There are 506 records with one continuous response "medv" (median house value) and 13 predictors. We again create a training set

```
set.seed(1)

train = sample(1:nrow(Boston), nrow(Boston)/2)

tree.boston=tree(medv~.,Boston,subset=train)
```

We use the cv.tree() function to perform 10-fold cross validation to find the best subtree. We do not need to specify the argument FUN since it is the regression tree.

```
cv.boston=cv.tree(tree.boston,K=10)

cv.boston
```

8 is the optimal size. Thus, we prune tree to size 8 using the prune.tree() function and plot the tree.

```
prune.boston=prune.tree(tree.boston,best=8)

plot(prune.boston)

text(prune.boston,pretty=0)
```

Then we use the pruned tree to make predictions in the testing set and calculate the mean square error.

```
boston.test=Boston[-train,"medv"]

tree.pred=predict(prune.boston,newdata=Boston[-train,])

mean((tree.pred-boston.test)^2)
```

3. Bagging and Random Forest with data Boston.csv.

We perform random forest and illustrate a case for bagging using the randomForest() function, which is part of the randomForest package. Recall that random forest is using bagging for decision tree without using all p predictors in the original dataset. Thus, if we use all p predictors, it is equal to bagging. The randomForest() function can be used to perform both bagging and random forest.

We first load the randomForest package.

```
library(randomForest)
```

In the randomForest() function, the argument mtry=13 indicates that all 13 predictors should be used, that is bagging. subset=train indicates that we train this model only using the training dataset. importance=TRUE indicates that the importance of predictors is assessed. bag.boston stores the bagging model.

```
set.seed(1)

bag.boston=randomForest(medv~.,data=Boston,subset=train,mtry=13,importance=TRUE)

bag.boston
```

We next evaluate the performance of bagging by fitting it to the testing dataset Boston[-train,]. Then we calculate the MSE.

```
yhat.bag = predict(bag.boston,newdata=Boston[-train,])

mean((yhat.bag-boston.test)^2)
```

Now let's use the function to implement random forest. The difference is that random forest does not use all input variables in each tree. It usually uses p/3 variables for regression trees and $\sqrt{p}$for classification trees. Now let's use mtry=5. rf.boston stores the random forest model. We test the MSE of this model by comparing the predicted values with the true values.

```
set.seed(1)

rf.boston=randomForest(medv~.,data=Boston,subset=train,mtry=5,importance=TRUE)

yhat.rf = predict(rf.boston,newdata=Boston[-train,])

mean((yhat.rf-boston.test)^2)
```

The importance of each variable can be evaluated using the importance() function. The function varImpPlot() plots the important measures. Two measures of variable importance are reported. One is based on the mean decrease of accuracy in predictions on the out of bag samples when a given variable is excluded from the model. The second is a measure of the total decrease in node impurity that results from splits over that variable, averaged over all trees.

```
importance(rf.boston)

varImpPlot(rf.boston)
```

This concludes lab #4.