Econ 7035, Spring 2021, Assignment 5

SIU Chung Pan 20444990

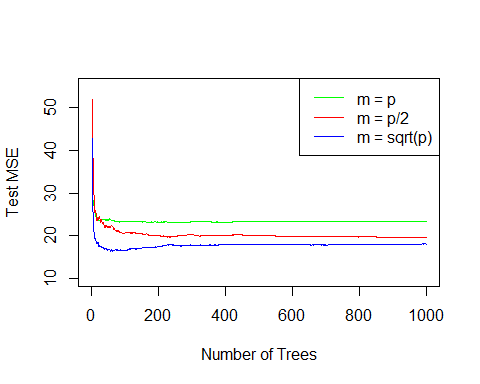
4/27/2021

# Part 1 - Applied Exercises

## Exercise 7 in section 8.4

In the lab, we applied random forests to the Boston data using mtry=6 and using ntree=25 and ntree=500. Create a plot displaying the test error resulting from random forests on this data set for a more comprehensive range of values for mtry and ntree. You can model your plot after Figure 8.10. Describe the results obtained.

library(MASS)  
library(randomForest)  
data(Boston)  
  
set.seed(1)  
train <- sample(1:nrow(Boston), nrow(Boston) / 2)  
Boston.train <- Boston[train, -14]  
Boston.test <- Boston[-train, -14]  
Y.train <- Boston[train, 14]  
Y.test <- Boston[-train, 14]  
rf.boston1 <- randomForest(Boston.train, y = Y.train, xtest = Boston.test, ytest = Y.test, mtry = ncol(Boston) - 1, ntree = 1000)  
rf.boston2 <- randomForest(Boston.train, y = Y.train, xtest = Boston.test, ytest = Y.test, mtry = (ncol(Boston) - 1) / 2, ntree = 1000)  
rf.boston3 <- randomForest(Boston.train, y = Y.train, xtest = Boston.test, ytest = Y.test, mtry = sqrt(ncol(Boston) - 1), ntree = 1000)  
plot(1:1000, rf.boston1$test$mse, col = "green", type = "l", xlab = "Number of Trees", ylab = "Test MSE", ylim = c(10, 55))  
lines(1:1000, rf.boston2$test$mse, col = "red", type = "l")  
lines(1:1000, rf.boston3$test$mse, col = "blue", type = "l")  
legend("topright", c("m = p", "m = p/2", "m = sqrt(p)"), col = c("green", "red", "blue"), cex = 1, lty = 1)



The Test MSE is very large when the number of trees is small. It decreases as the number of trees increases, and become flat when the number of trees is around 100. On the other hand, the Test MSE with all predictors used for training the model is higher comparing with just using half the predictors or the square root of the number of predictors. This is explained by overfitting if all predictors are used to train the model. The minimum Test MSE occurs when m=sqrt(p) is chosen.

## Exercise 8 in section 8.4

In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable. Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable.

(a). Split the data set into a training set and a test set.

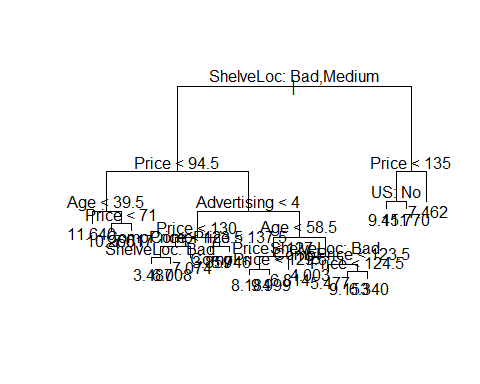
library(ISLR)  
set.seed(1)  
train = sample(1:nrow(Carseats), nrow(Carseats) / 2)  
Car.train = Carseats[train, ]  
Car.test = Carseats[-train,]

(b). Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

library(tree)  
reg.tree <- tree(Sales~.,data = Carseats, subset=train)  
summary(reg.tree)

##   
## Regression tree:  
## tree(formula = Sales ~ ., data = Carseats, subset = train)  
## Variables actually used in tree construction:  
## [1] "ShelveLoc" "Price" "Age" "Advertising" "CompPrice"   
## [6] "US"   
## Number of terminal nodes: 18   
## Residual mean deviance: 2.167 = 394.3 / 182   
## Distribution of residuals:  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -3.88200 -0.88200 -0.08712 0.00000 0.89590 4.09900

plot(reg.tree)  
text(reg.tree ,pretty =0)



yhat <- predict(reg.tree,newdata = Car.test)

The regression tree used six predictors to train the model, and they are “ShelveLoc”, “Price”, “Age”, “Advertising”, “CompPrice”, and “US”. There are 18 terminal nodes, with different values of predicted sales.

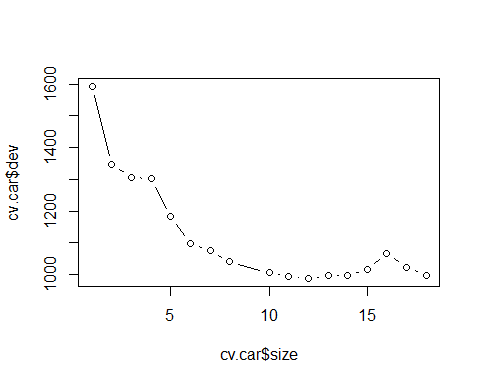
(test.MSE<-mean((yhat - Car.test$Sales)^2))

## [1] 4.922039

The test MSE obtained is 4.92.

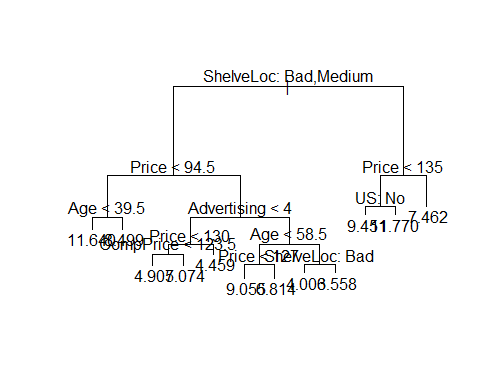
(c). Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

set.seed(10)  
cv.car = cv.tree(reg.tree)  
plot(cv.car$size, cv.car$dev, type = "b")



By using cross-validation, we determined that the optimal number of nodes is 12. Therefore, we prune the tree to obtain a 12-node tree.

prune.car = prune.tree(reg.tree, best = 12)  
plot(prune.car)  
text(prune.car,pretty=0)



yhat=predict(prune.car, newdata= Car.test)  
mean((yhat-Car.test$Sales)^2)

## [1] 4.966929

The test MSE obtained is 4.97. Comparing with the tree without pruning, it has 18-node tree that has a test MSE of 4.92. Therefore, we conclude that, pruning the tree does not improve the test MSE.

(d). Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important.

set.seed(1)  
bag.car = randomForest(Sales~.,data=Car.train,mtry = 10, importance = TRUE)  
yhat.bag = predict(bag.car,newdata=Car.test)  
mean((yhat.bag-Car.test$Sales)^2)

## [1] 2.605253

The test MSE obtained is 2.61.

importance(bag.car)

## %IncMSE IncNodePurity  
## CompPrice 24.8888481 170.182937  
## Income 4.7121131 91.264880  
## Advertising 12.7692401 97.164338  
## Population -1.8074075 58.244596  
## Price 56.3326252 502.903407  
## ShelveLoc 48.8886689 380.032715  
## Age 17.7275460 157.846774  
## Education 0.5962186 44.598731  
## Urban 0.1728373 9.822082  
## US 4.2172102 18.073863

“Price” is the most important variable.

(e). Use random forests to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

set.seed(1)  
mse.vec <- NA  
for (a in 1:10){  
 rf.car <- randomForest(Sales ~ . , data=Car.train,   
 mtry=a, ntree=500, importance=TRUE)  
 rf.pred <- predict(rf.car, Car.test)  
 mse.vec[a] <- mean((Car.test$Sales - rf.pred)^2)  
}  
  
# best model  
which.min(mse.vec)

## [1] 9

# Minimum test MSE  
mse.vec[which.min(mse.vec)]

## [1] 2.550463

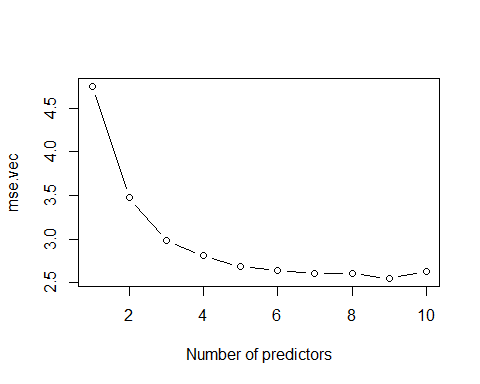
The test MSE obtained is 2.55. We used 3 predictors to train the model, since we have 10 predictors, and square root of 10 is close to 3.

# best model  
rf.car <- randomForest(Sales ~ . , data = Carseats,   
 mtry=9, ntree=500, importance=TRUE)  
importance(rf.car)

## %IncMSE IncNodePurity  
## CompPrice 40.82228195 331.37582  
## Income 11.91755389 168.73021  
## Advertising 26.09963528 244.71972  
## Population -0.04414755 100.71006  
## Price 83.35133230 904.54575  
## ShelveLoc 86.62416976 968.90541  
## Age 25.07352014 281.85675  
## Education 2.98566014 81.90778  
## Urban -0.73121981 14.06407  
## US 5.09506455 15.85731

“ShelveLoc” is the most important variable.

# plot the test MSE with different number of predictors  
plot(mse.vec, xlab = "Number of predictors", type = "b")



The test MSE is large if we only use 1 predictor. It decreases with increasing number of predictors and reaches its minimum value when the number of predictors is 9.

## Exercise 9 in section 8.4

This problem involves the OJ data set which is part of the ISLR package.

1. Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

set.seed(1000)  
train <- sample(dim(OJ)[1],800)  
OJ.train <- OJ[train,]  
OJ.test <- OJ[-train,]

1. Fit a tree to the training data, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

OJ.tree <- tree(Purchase~., data=OJ.train)  
summary(OJ.tree)

##   
## Classification tree:  
## tree(formula = Purchase ~ ., data = OJ.train)  
## Variables actually used in tree construction:  
## [1] "LoyalCH" "PriceDiff" "SalePriceMM"  
## Number of terminal nodes: 8   
## Residual mean deviance: 0.7486 = 592.9 / 792   
## Misclassification error rate: 0.16 = 128 / 800

The training error rate is 16%. The tree has 8 terminal nodes.

1. Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

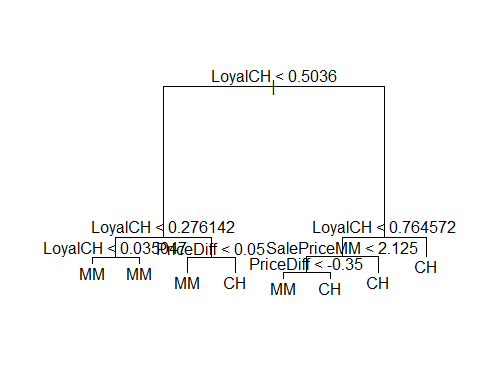
OJ.tree

## node), split, n, deviance, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 800 1066.00 CH ( 0.61500 0.38500 )   
## 2) LoyalCH < 0.5036 353 422.60 MM ( 0.28612 0.71388 )   
## 4) LoyalCH < 0.276142 170 131.00 MM ( 0.12941 0.87059 )   
## 8) LoyalCH < 0.035047 57 10.07 MM ( 0.01754 0.98246 ) \*  
## 9) LoyalCH > 0.035047 113 108.50 MM ( 0.18584 0.81416 ) \*  
## 5) LoyalCH > 0.276142 183 250.30 MM ( 0.43169 0.56831 )   
## 10) PriceDiff < 0.05 78 79.16 MM ( 0.20513 0.79487 ) \*  
## 11) PriceDiff > 0.05 105 141.30 CH ( 0.60000 0.40000 ) \*  
## 3) LoyalCH > 0.5036 447 337.30 CH ( 0.87472 0.12528 )   
## 6) LoyalCH < 0.764572 187 206.40 CH ( 0.75936 0.24064 )   
## 12) SalePriceMM < 2.125 120 156.60 CH ( 0.64167 0.35833 )   
## 24) PriceDiff < -0.35 16 17.99 MM ( 0.25000 0.75000 ) \*  
## 25) PriceDiff > -0.35 104 126.70 CH ( 0.70192 0.29808 ) \*  
## 13) SalePriceMM > 2.125 67 17.99 CH ( 0.97015 0.02985 ) \*  
## 7) LoyalCH > 0.764572 260 91.11 CH ( 0.95769 0.04231 ) \*

I pick the node labelled 7, which is a terminal node because of the asterisk. The split criterion is LoyalCH > 0.764572, the number of observations in that branch is 260 with a deviance of 91.11. In this node, there are more observations are predicted as the branch of CH. 95.769% of the observations in that branch take the value of CH, and the remaining 4.231% take the value of MM.

1. Create a plot of the tree, and interpret the results.

plot(OJ.tree)  
text(OJ.tree,pretty=TRUE)



The predictor “LoyaclCH” is the most important to predict the target variable, “Purchase”. Since the first branch use that feature, and it appears most frequently in the tree.

1. Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

tree.pred = predict(OJ.tree, newdata = OJ.test, type = "class")  
table(tree.pred,OJ.test$Purchase)

##   
## tree.pred CH MM  
## CH 150 38  
## MM 11 71

The test error rate = (38+11)/(150+38+11+71) = 18.15%

1. Apply the cv.tree() function to the training set in order to determine the optimal tree size.

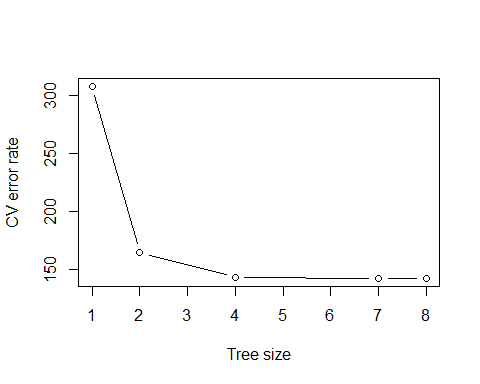
(cv.OJ = cv.tree(OJ.tree, FUN = prune.misclass))

## $size  
## [1] 8 7 4 2 1  
##   
## $dev  
## [1] 142 142 143 164 308  
##   
## $k  
## [1] -Inf 0.000000 2.666667 10.500000 151.000000  
##   
## $method  
## [1] "misclass"  
##   
## attr(,"class")  
## [1] "prune" "tree.sequence"

The optimal tree size is 4, which has a cross validation error very close to the lowest cross validation error. I prefer the tree size to be 4 because the improvement in error rate is very small with increasing tree size.

1. Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

plot(cv.OJ$size,cv.OJ$dev,type='b', xlab = "Tree size", ylab = "CV error rate")

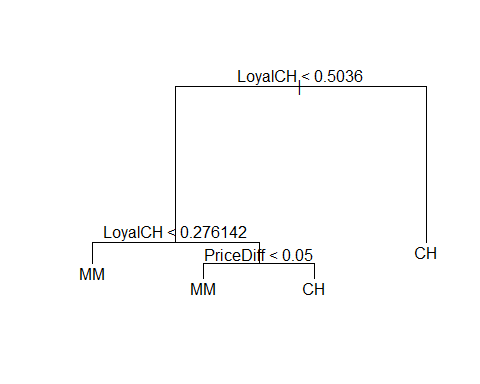


1. Which tree size corresponds to the lowest cross-validated classification error rate?

The lowest cross-validated classification error rate appears when the tree size is 7 or 8.

1. Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

prune.OJ = prune.misclass(OJ.tree, best=4)  
plot(prune.OJ)  
text(prune.OJ,pretty=0)



1. Compare the training error rates between the pruned and unpruned trees. Which is higher?

tree.pred <- predict(prune.OJ, newdata = OJ.train, type = "class")  
table(tree.pred,OJ.train$Purchase)

##   
## tree.pred CH MM  
## CH 454 98  
## MM 38 210

The training error rate of pruned tree = (98+38)/(454+98+38+210) = 17%. It is higher than that of unpruned tree which has the training error rate of 16%.

1. Compare the test error rates between the pruned and unpruned trees. Which is higher?

tree.pred <- predict(prune.OJ, newdata = OJ.test, type = "class")  
table(tree.pred,OJ.test$Purchase)

##   
## tree.pred CH MM  
## CH 150 44  
## MM 11 65

The test error rate = (44+11)/(150+44+11+65) = 20.37%. It is higher than that of unpruned tree which has the testing error rate of 18.15%.

## Exercise 10 in section 8.4

We now use boosting to predict Salary in the Hitters data set.

1. Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

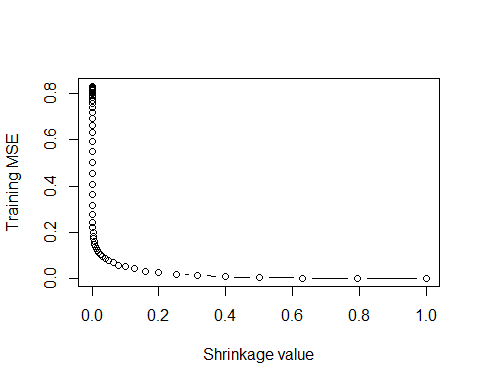
Hitters <- na.omit(Hitters)  
Hitters$Salary <- log(Hitters$Salary)

1. Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

train<- 1:200  
hitters.train <- Hitters[train,]  
hitters.test <- Hitters[-train,]

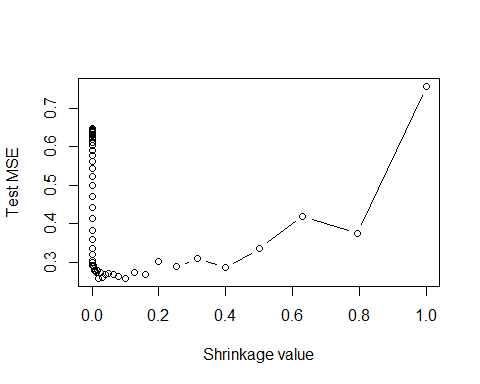
1. Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter λ. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

library(gbm)  
  
set.seed(100)  
pows <- seq(-10, 0, by = 0.1)  
lambdas <- 10^pows  
train.err <- rep(NA, length(lambdas))  
  
for (i in 1:length(lambdas)) {  
 boost.hitters <- gbm(Salary ~ ., data = hitters.train, distribution = "gaussian", n.trees = 1000, shrinkage = lambdas[i])  
 pred.train <- predict(boost.hitters, hitters.train, n.trees = 1000)  
 train.err[i] <- mean((pred.train - hitters.train$Salary)^2)  
}  
  
plot(lambdas, train.err, type = "b", xlab = "Shrinkage value", ylab = "Training MSE")



1. Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

set.seed(100)  
test.err <- rep(NA, length(lambdas))  
  
for (i in 1:length(lambdas)) {  
 boost.hitters <- gbm(Salary ~ ., data = hitters.train, distribution = "gaussian", n.trees = 1000, shrinkage = lambdas[i])  
 yhat <- predict(boost.hitters, hitters.test, n.trees = 1000)  
 test.err[i] <- mean((yhat - hitters.test$Salary)^2)  
}  
  
plot(lambdas, test.err, type = "b", xlab = "Shrinkage value", ylab = "Test MSE")



1. Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.

min(test.err)

## [1] 0.2587138

The minimum test MSE of boosting is 0.2587.

# linear regression  
library(glmnet)  
fit1 <- lm(Salary ~ ., data = hitters.train)  
pred1 <- predict(fit1, hitters.test)  
mean((pred1 - hitters.test$Salary)^2)

## [1] 0.4917959

The test MSE of linear regression is 0.4918.

# ridge regression  
x <- model.matrix(Salary ~ ., data = hitters.train)  
x.test <- model.matrix(Salary ~ ., data = hitters.test)  
y <- hitters.train$Salary  
fit2 <- glmnet(x, y, alpha = 0)  
pred2 <- predict(fit2, s = 0.01, newx = x.test)  
mean((pred2 - hitters.test$Salary)^2)

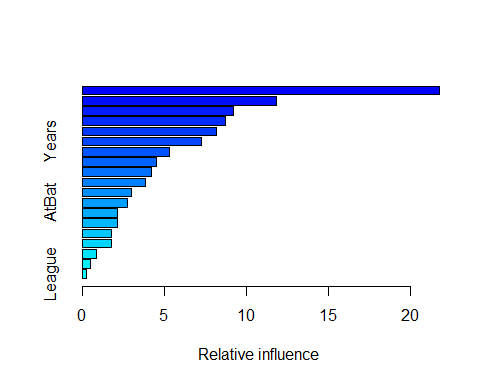
## [1] 0.4570283

The test MSE of ridge regression is 0.4570.

To conclude, the test MSE using boosting is the lowest among the three models, with the value of 0.2587.

1. Which variables appear to be the most important predictors in the boosted model?

boost.hitters <- gbm(Salary ~ ., data = hitters.train, distribution = "gaussian", n.trees = 1000, shrinkage = lambdas[which.min(test.err)])  
summary(boost.hitters)



## var rel.inf  
## CAtBat CAtBat 21.7541461  
## CRuns CRuns 11.8347873  
## CHits CHits 9.2235745  
## CRBI CRBI 8.7180562  
## CWalks CWalks 8.1897796  
## Years Years 7.2859920  
## CHmRun CHmRun 5.3360700  
## Walks Walks 4.5058572  
## PutOuts PutOuts 4.1834434  
## Hits Hits 3.8241861  
## RBI RBI 3.0006265  
## AtBat AtBat 2.7743672  
## HmRun HmRun 2.1216796  
## Runs Runs 2.1160499  
## Errors Errors 1.7879030  
## Assists Assists 1.7827784  
## Division Division 0.8399423  
## NewLeague NewLeague 0.4888717  
## League League 0.2318890

From the summary result, the most important predictor in the boosted model is “CAtBat”.

1. Now apply bagging to the training set. What is the test set MSE for this approach?

set.seed(100)  
bag.hitters <- randomForest(Salary ~ ., data = hitters.train, mtry = 19, ntree = 1000)  
yhat.bag <- predict(bag.hitters, newdata = hitters.test)  
mean((yhat.bag - hitters.test$Salary)^2)

## [1] 0.2304126

The test MSE using bagging is 0.2304. The bagging approach gives the lowest test MSE among all the models tried.

## Exercise 11 in section 8.4

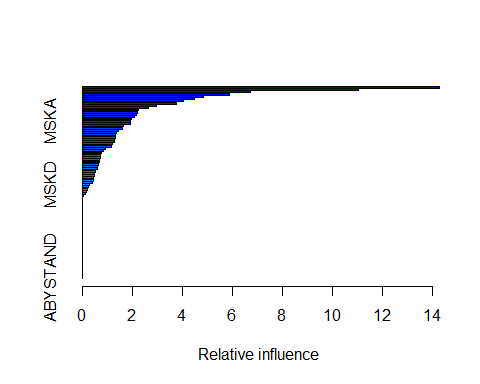
This question uses the Caravan data set.

1. Create a training set consisting of the first 1,000 observations, and a test set consisting of the remaining observations.

train = 1:1000  
Caravan.train <- Caravan[train,]  
Caravan.test <- Caravan[-train,]

1. Fit a boosting model to the training set with Purchase as the response and the other variables as predictors. Use 1,000 trees, and a shrinkage value of 0.01. Which predictors appear to be the most important?

set.seed(1000)  
Caravan$Purchase <- ifelse(Caravan$Purchase == "Yes", 1, 0)  
boost.model <- gbm(Purchase ~ ., data = Caravan.train, distribution = "gaussian", n.trees = 1000, shrinkage = 0.01)  
  
summary(boost.model)



## var rel.inf  
## PPERSAUT PPERSAUT 14.28361020  
## MKOOPKLA MKOOPKLA 11.02798173  
## MOPLHOOG MOPLHOOG 6.73403936  
## MBERMIDD MBERMIDD 5.88964130  
## PBRAND PBRAND 4.84946593  
## MGODGE MGODGE 4.46714301  
## ABRAND ABRAND 4.05234497  
## MINK3045 MINK3045 3.74792913  
## PWAPART PWAPART 2.95269274  
## MOSTYPE MOSTYPE 2.64617251  
## MAUT1 MAUT1 2.24969862  
## MGODPR MGODPR 2.21648456  
## MAUT2 MAUT2 2.18103411  
## MSKC MSKC 2.09993937  
## MBERARBG MBERARBG 1.95336166  
## MSKA MSKA 1.92703304  
## PBYSTAND PBYSTAND 1.91079633  
## MGODOV MGODOV 1.64995661  
## MBERHOOG MBERHOOG 1.59987956  
## MRELGE MRELGE 1.42797389  
## MINKGEM MINKGEM 1.36085238  
## MSKB1 MSKB1 1.34236868  
## MFGEKIND MFGEKIND 1.33981999  
## MRELOV MRELOV 1.29338382  
## MINK7512 MINK7512 1.28502087  
## MINKM30 MINKM30 1.21187090  
## MFWEKIND MFWEKIND 1.15111693  
## MINK4575 MINK4575 0.93057361  
## MOSHOOFD MOSHOOFD 0.86668291  
## MHHUUR MHHUUR 0.78668999  
## MHKOOP MHKOOP 0.73419830  
## MOPLMIDD MOPLMIDD 0.71612470  
## MBERARBO MBERARBO 0.70323836  
## MZFONDS MZFONDS 0.67927049  
## MGODRK MGODRK 0.66496827  
## MGEMLEEF MGEMLEEF 0.62241977  
## MINK123M MINK123M 0.62221616  
## MZPART MZPART 0.50892513  
## MAUT0 MAUT0 0.48396636  
## MBERBOER MBERBOER 0.46841618  
## MSKB2 MSKB2 0.45288837  
## PMOTSCO PMOTSCO 0.43413529  
## APERSAUT APERSAUT 0.41705962  
## MSKD MSKD 0.28814618  
## MOPLLAAG MOPLLAAG 0.22940635  
## MGEMOMV MGEMOMV 0.18971449  
## MFALLEEN MFALLEEN 0.17790932  
## PLEVEN PLEVEN 0.10922701  
## MBERZELF MBERZELF 0.06221096  
## MAANTHUI MAANTHUI 0.00000000  
## MRELSA MRELSA 0.00000000  
## PWABEDR PWABEDR 0.00000000  
## PWALAND PWALAND 0.00000000  
## PBESAUT PBESAUT 0.00000000  
## PVRAAUT PVRAAUT 0.00000000  
## PAANHANG PAANHANG 0.00000000  
## PTRACTOR PTRACTOR 0.00000000  
## PWERKT PWERKT 0.00000000  
## PBROM PBROM 0.00000000  
## PPERSONG PPERSONG 0.00000000  
## PGEZONG PGEZONG 0.00000000  
## PWAOREG PWAOREG 0.00000000  
## PZEILPL PZEILPL 0.00000000  
## PPLEZIER PPLEZIER 0.00000000  
## PFIETS PFIETS 0.00000000  
## PINBOED PINBOED 0.00000000  
## AWAPART AWAPART 0.00000000  
## AWABEDR AWABEDR 0.00000000  
## AWALAND AWALAND 0.00000000  
## ABESAUT ABESAUT 0.00000000  
## AMOTSCO AMOTSCO 0.00000000  
## AVRAAUT AVRAAUT 0.00000000  
## AAANHANG AAANHANG 0.00000000  
## ATRACTOR ATRACTOR 0.00000000  
## AWERKT AWERKT 0.00000000  
## ABROM ABROM 0.00000000  
## ALEVEN ALEVEN 0.00000000  
## APERSONG APERSONG 0.00000000  
## AGEZONG AGEZONG 0.00000000  
## AWAOREG AWAOREG 0.00000000  
## AZEILPL AZEILPL 0.00000000  
## APLEZIER APLEZIER 0.00000000  
## AFIETS AFIETS 0.00000000  
## AINBOED AINBOED 0.00000000  
## ABYSTAND ABYSTAND 0.00000000

By the summary result, the variable ‘PPERSAUT’ is the most important.

1. Use the boosting model to predict the response on the test data. Predict that a person will make a purchase if the estimated probability of purchase is greater than 20%. Form a confusion matrix. What fraction of the people predicted to make a purchase do in fact make one? How does this compare with the results obtained from applying KNN or logistic regression to this data set?

# boosting  
probs.test <- predict(boost.model, Caravan.test, n.trees = 1000, type = "response")  
pred.test <- ifelse(probs.test > 0.2, 1, 0)  
table(Caravan.test$Purchase, pred.test)

## pred.test  
## 1  
## No 4533  
## Yes 289

The fraction of the people predicted to make a purchase do in fact make one = 13/(13+38) = 25.49%

# logistic regression  
logit.caravan <- glm(Purchase ~ ., data = Caravan.train, family = "binomial")  
probs.test2 <- predict(logit.caravan, Caravan.test, type = "response")  
pred.test2 <- ifelse(probs.test > 0.2, 1, 0)  
table(Caravan.test$Purchase, pred.test2)

## pred.test2  
## 1  
## No 4533  
## Yes 289

The fraction of the people predicted to make a purchase do in fact make one = 13/(13+38) = 25.49%

The fraction using logistic regression is the same with using boosting.

# Part 2 - Caogle Competition

library(caret)  
  
setwd('C:/Users/Jacob/Desktop/Master\'s studies/ECON7035 Artificial Intelligence for Business')  
  
# read training data  
x <- read.csv("train\_x.csv")  
y <- read.csv("train\_y.csv")  
  
# drop the first column  
x <- data.frame(x[,-1])  
y <- data.frame(y[,-1])  
  
# rename the column name  
colnames(y) <- 'y'  
  
# merge data  
df<-data.frame(y,x)  
  
# convert y to factor variable  
df$y <- as.factor(df$y)  
  
#train/test split based on 80/20  
set.seed(100)  
train <- sample(nrow(df),800)  
df.train <- df[train,]  
df.test <- df[-train,]

# random forest  
set.seed(1)  
accuracy\_vec <- NA  
  
# included training the model with 28 predictors, since sqrt(784)=28  
  
for (i in 20:40){  
 rf.cao <- randomForest(y ~ . , data=df.train,   
 mtry=i, ntree=500, importance=TRUE)  
 rf.pred <- predict(rf.cao, df.test, method='class')  
 accuracy\_vec[i-19]<-(confusionMatrix(rf.pred,df.test$y))$overall['Accuracy']  
}

# the best model, need to add back 19 to get the most appropriate number of predictors  
which.max(accuracy\_vec)+19

## [1] 36

# best model using random forest, with 36 predictors  
rf\_caogle <- randomForest(y ~ . , data=df.train,   
 mtry=36, ntree=500, importance=TRUE)  
  
save(rf\_caogle, file = "caogle.RData")

# read testing data  
# the testing data is a direct copy of the training data  
x <- read.csv("test\_x.csv")  
y <- read.csv("test\_y.csv")  
  
# drop the first column  
x <- data.frame(x[,-1])  
y <- data.frame(y[,-1])  
  
# rename the column name  
colnames(y) <- 'y'  
  
# merge data  
df<-data.frame(y,x)  
  
# convert y to factor variable  
df$y <- as.factor(df$y)  
  
library(randomForest)  
(load(file = "caogle.RData"))

## [1] "rf\_caogle"

library(caret)  
pred <- predict(rf\_caogle, x, method='class')  
(accuracy<-(confusionMatrix(pred,df$y))$overall['Accuracy'])

## Accuracy   
## 0.986

I am willing to share my code to the class.