FE590. Assignment #4.

Enter Your Name Here, or "Anonymous" if you want to remain anonymous..

2017-05-05

Instructions

When you have completed the assignment, knit the document into a PDF file, and upload *both* the .pdf and .Rmd files to Canvas.

Note that you must have LaTeX installed in order to knit the equations below. If you do not have it installed, simply delete the questions below.

Question 1:

In this assignment, you will be required to find a set of data to run regression on. This data set should be financial in nature, and of a type that will work with the models we have discussed this semester (hint: we didn't look at time series) You may not use any of the data sets in the ISLR package that we have been looking at all semester. Your data set that you choose should have both qualitative and quantitative variables. (or has variables that you can transform)

Provide a description of the data below, where you obtained it, what the variable names are and what it is describing.

```
library(MASS)
library(class)
library(leaps)
library(boot)
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(gbm)
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:boot':
##
##     aml
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:boot':
##
##     melanoma
```

```
## Loading required package: splines
```

```
## Loading required package: parallel

## Loaded gbm 2.1.3
```

```r
library(tree)

setwd("/Users/apple/Desktop/590/H4")
creditcard <- read.csv("creditcard.csv")
head(creditcard)
```

```
##   Time         V1          V2        V3         V4          V5          V6
## 1    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
## 2    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
## 3    1 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
## 4    1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
## 5    2 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
## 6    2 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755
##            V7          V8         V9         V10        V11         V12
## 1  0.23959855  0.09869790  0.3637870  0.09079417 -0.5515995 -0.61780086
## 2 -0.07880298  0.08510165 -0.2554251 -0.16697441  1.6127267  1.06523531
## 3  0.79146096  0.24767579 -1.5146543  0.20764287  0.6245015  0.06608369
## 4  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
## 5  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
## 6  0.47620095  0.26031433 -0.5686714 -0.37140720  1.3412620  0.35989384
##           V13        V14        V15        V16         V17         V18
## 1 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.20797124  0.02579058
## 2  0.4890950 -0.1437723  0.6355581  0.4639170 -0.11480466 -0.18336127
## 3  0.7172927 -0.1659459  2.3458649 -2.8900832  1.10996938 -0.12135931
## 4  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279  1.96577500
## 5  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.23703324 -0.03819479
## 6 -0.3580907 -0.1371337  0.5176168  0.4017259 -0.05813282  0.06865315
##           V19         V20          V21          V22         V23
## 1  0.40399296  0.25141210 -0.018306778  0.277837576 -0.11047391
## 2 -0.14578304 -0.06908314 -0.225775248 -0.638671953  0.10128802
## 3 -2.26185710  0.52497973  0.247998153  0.771679402  0.90941226
## 4 -1.23262197 -0.20803778 -0.108300452  0.005273597 -0.19032052
## 5  0.80348692  0.40854236 -0.009430697  0.798278495 -0.13745808
## 6 -0.03319379  0.08496767 -0.208253515 -0.559824796 -0.02639767
##           V24        V25        V26          V27         V28 Amount Class
## 1  0.06692807  0.1285394 -0.1891148  0.133558377 -0.02105305 149.62     0
## 2 -0.33984648  0.1671704  0.1258945 -0.008983099  0.01472417   2.69     0
## 3 -0.68928096 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66     0
## 4 -1.17557533  0.6473760 -0.2219288  0.062722849  0.06145763 123.50     0
## 5  0.14126698 -0.2060096  0.5022922  0.219422230  0.21515315  69.99     0
## 6 -0.37142658 -0.2327938  0.1059148  0.253844225  0.08108026   3.67     0
```

The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

Variables in this dataset:

Time: How many times this card has been used.

V1 to V28 are principal components obtained with PCA transformation

Amount: Transaction Amount

Class: The actual classification classes. It takes value 1 in case of fraud and 0 otherwise.

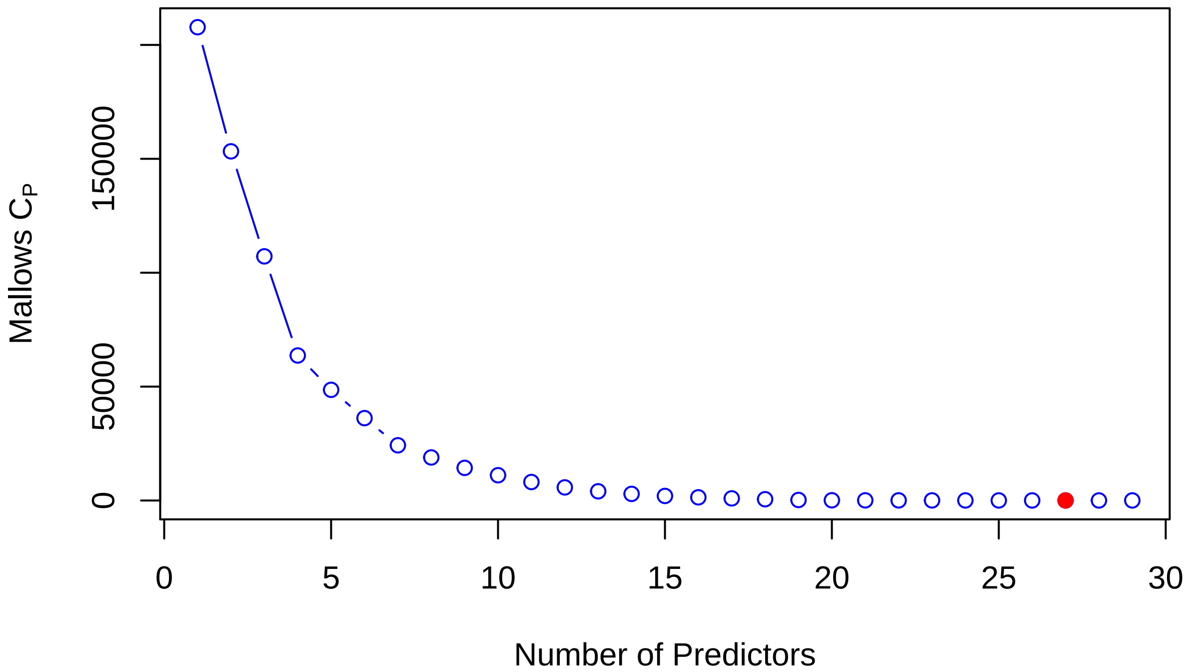
Ref: https://www.kaggle.com/dalpozz/creditcardfraud

Question 2:

Pick a quantitative variable and fit at least four different models in order to predict that variable using the other predictors. Determine which of the models is the best fit. You will need to provide strong reason why the particular model you chose is the best one. You will need to confirm the model you have selected provides the best fit and that you have obtained the best version of that particular model (i.e. subset selection or validation for example). You need to convince the grader that you have chosen the best model.

```r
n <- nrow(creditcard) # there are n records

# Arrange this dataset in a random order
set.seed(1)
d <- 50000 # select 50000 samples
random.order <- sample(seq(1,n), d)
creditcardsub <- creditcard[random.order, ]

# Divide dataset into two group: Training data and test data
training <- creditcardsub[seq(1, d/2), ]
test <- creditcardsub[seq(d/2+1, d), ]
```

```r
# The quantitative variable I chose is Amount
training1 <- training[,-31]
test1 <- test[,-31]

# 1. Best subset selection + logistic regression
m <- regsubsets(Amount~., data = training1, nvmax=30)
regs <- t(summary(m)$which)

cp <- summary(m)$cp
i <- which.min(cp)
plot(cp,type='b',col="blue",
     xlab="Number of Predictors",ylab=expression("Mallows C"[P]))
points(i,cp[i],pch=19,col="red")
```




```r
# Cause the Cps after n=15 are low enough. For simplicity, I chose number of predictors as 18.

regs[, 18]
```

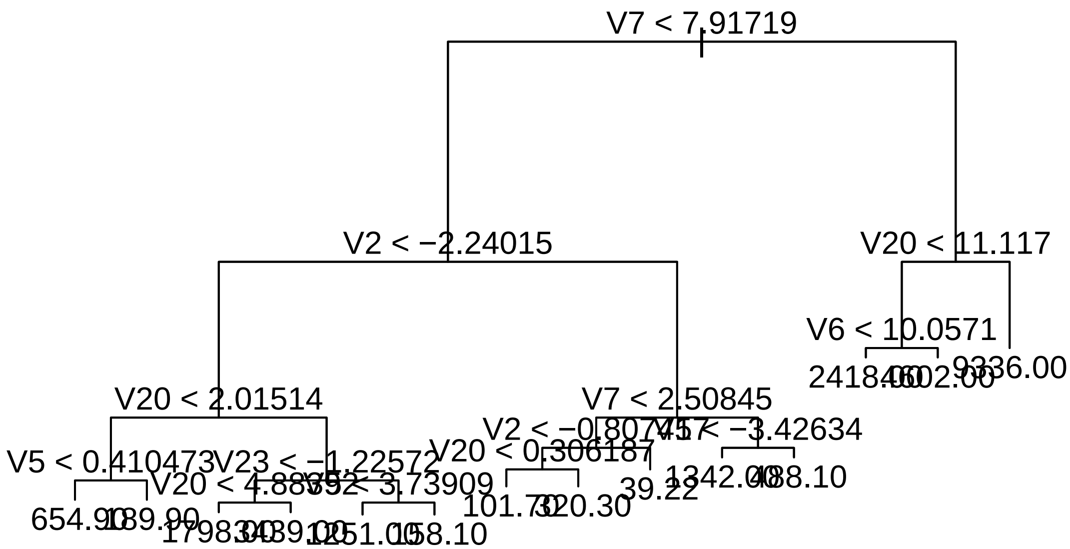
```
## (Intercept)        Time          V1          V2          V3          V4
##        TRUE       FALSE        TRUE        TRUE        TRUE        TRUE
##          V5          V6          V7          V8          V9         V10
##        TRUE        TRUE        TRUE        TRUE        TRUE        TRUE
##         V11         V12         V13         V14         V15         V16
##       FALSE       FALSE       FALSE        TRUE       FALSE       FALSE
##         V17         V18         V19         V20         V21         V22
##       FALSE        TRUE        TRUE        TRUE        TRUE        TRUE
##         V23         V24         V25         V26         V27         V28
##        TRUE       FALSE        TRUE       FALSE       FALSE       FALSE
```

```r
# Select subsets
training2 <- training1[, -c(1, 12, 13, 14, 16, 17, 18, 25, 27, 28, 29)]
test2 <- test1[, -c(1, 12, 13, 14, 16, 17, 18, 25, 27, 28, 29)]
glm.fit <- glm(Amount~., data = training2)
glm.pred <- predict(glm.fit, test2, type ="response")
MSE.glm <- mean((test2$Amount - glm.pred)^2)

# 2. Linear Discriminant Analysis
lda.fit <- lda(Amount~., data=training2)
lda.pred <- predict(lda.fit, test2, type ="response")
lda.x <- lda.pred$x
MSE.lda <- mean((test2$Amount - lda.x)^2)

# 3. Decision tree
tree.credit <- tree(Amount~.,training2)
summary(tree.credit)
```

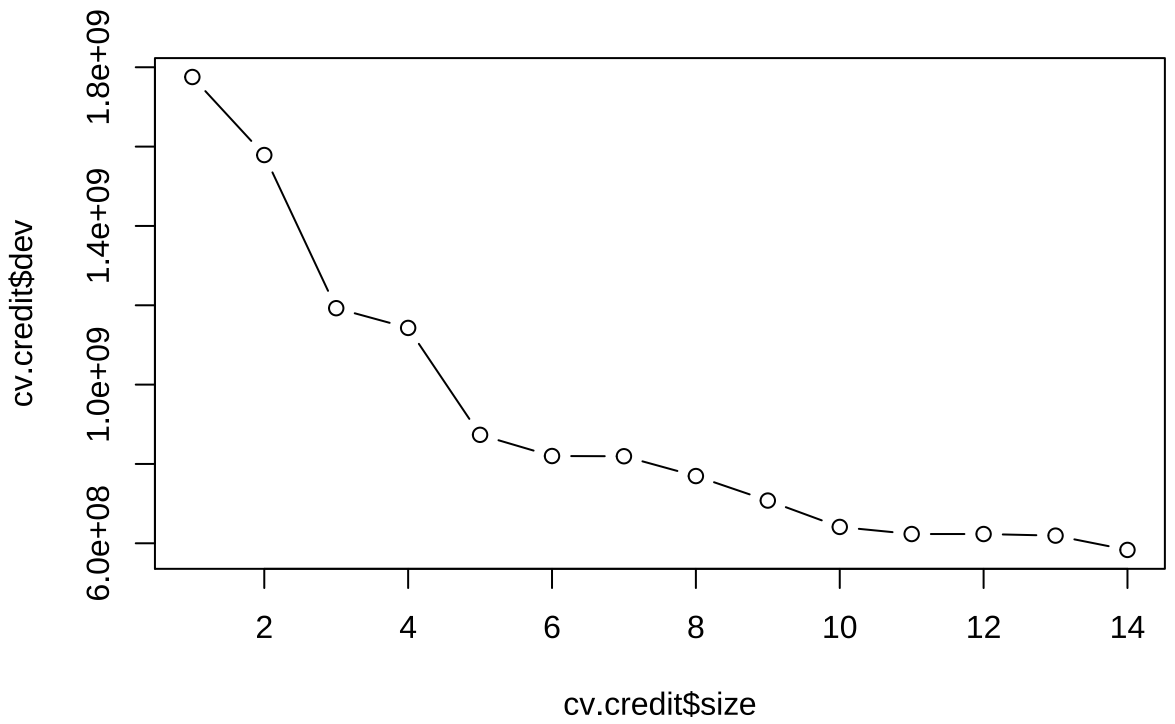
```
##
```

```
## Regression tree:
## tree(formula = Amount ~ ., data = training2)
## Variables actually used in tree construction:
## [1] "V7"  "V2"  "V20" "V5"  "V23" "V6"
## Number of terminal nodes:  14
## Residual mean deviance:  16340 = 408200000 / 24990
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3622.00   -37.24   -24.27     0.00    15.02 10320.00
```

```r
plot(tree.credit)
text(tree.credit ,pretty=0)
```




```r
cv.credit=cv.tree(tree.credit)
plot(cv.credit$size ,cv.credit$dev ,type='b')
```




```r
# Doesn't need to prune
tree.pred <- predict(tree.credit, newdata=test2)
MSE.tree <- mean((test2$Amount - tree.pred)^2)

# 4. Knn
amount <- training1$Amount
train.knn <- as.matrix(training1[, -30])
test.knn <- as.matrix(test1[, -30])

knn.pred1 <- knn(train.knn, test.knn, amount, k = 1)
knn.pred2 <- knn(train.knn, test.knn, amount, k = 2)
knn.pred3 <- knn(train.knn, test.knn, amount, k = 3)
knn.pred1 <- as.numeric(as.vector(knn.pred1))
knn.pred2 <- as.numeric(as.vector(knn.pred2))
knn.pred3 <- as.numeric(as.vector(knn.pred3))
MSE.knn1 <- mean((test2$Amount - knn.pred1)^2)
MSE.knn2 <- mean((test2$Amount - knn.pred2)^2)
MSE.knn3 <- mean((test2$Amount - knn.pred3)^2)

data.frame(MSE.glm, MSE.lda, MSE.tree, MSE.knn1, MSE.knn2, MSE.knn3)
```

```
##    MSE.glm  MSE.lda MSE.tree MSE.knn1 MSE.knn2 MSE.knn3
## 1 4302.039 60380.61 11355.53 61269.39 65225.29 72341.05
```

Among these method, the lowest MSE appears in logistic regression. In fact, amount in every transaction tends to be randomly depending on clients' usage. That may be why the MSE is so large.

Question 3:

Do the same approach as in question 2, but this time for a qualitative variable.

```r
# The qualitative variable I chose is Class
Testclass <- table(test$Class)
Testclass
```

```
##
##     0     1
## 24960    40
```

```r
# Baseline accuracy
Testclass[1]/sum(Testclass)
```

```
##      0
## 0.9984
```

```r
# 1. Logistic Regression
glm.fit <- glm(Class~., data=training, family="binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
summary(glm.fit)
```

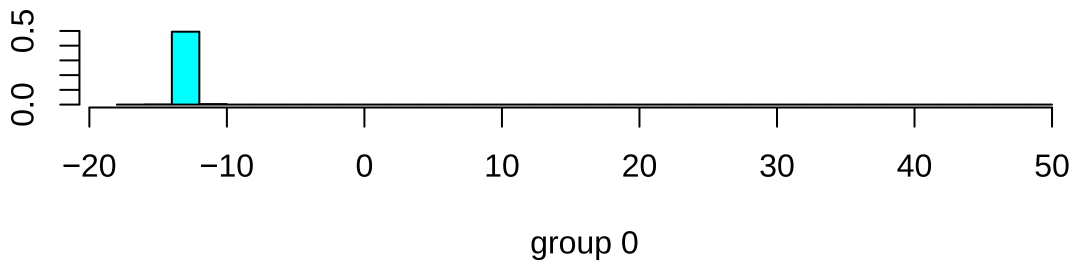
```
##
## Call:
## glm(formula = Class ~ ., family = "binomial", data = training)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.3421  -0.0191  -0.0105  -0.0039   4.1401
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.257e+01  5.633e+00  -2.231   0.0256 *
## Time        -2.488e-06  1.054e-05  -0.236   0.8134
## V1           4.457e-01  7.908e-01   0.564   0.5730
## V2           9.046e-01  2.315e+00   0.391   0.6960
## V3           3.570e-01  3.917e-01   0.911   0.3620
## V4           2.635e+00  3.856e+00   0.683   0.4943
## V5           1.436e+00  2.071e+00   0.694   0.4879
## V6          -8.073e-02  4.044e-01  -0.200   0.8418
## V7           5.669e-01  2.359e+00   0.240   0.8101
## V8          -2.266e-01  3.777e-01  -0.600   0.5486
## V9           1.392e+00  4.630e+00   0.301   0.7636
## V10         -2.604e+00  3.613e+00  -0.721   0.4710
## V11          1.549e-01  3.787e-01   0.409   0.6826
## V12          1.051e+00  3.507e+00   0.300   0.7644
## V13         -2.972e-01  7.455e-01  -0.399   0.6902
## V14          3.055e-01  1.767e+00   0.173   0.8627
## V15          4.236e-01  1.603e+00   0.264   0.7915
## V16         -2.531e+00  3.829e+00  -0.661   0.5087
## V17         -1.190e+00  2.668e+00  -0.446   0.6554
## V18          1.888e+00  4.361e+00   0.433   0.6650
## V19         -1.568e+00  2.590e+00  -0.606   0.5448
## V20         -1.646e+00  9.877e-01  -1.666   0.0957 .
## V21         -2.773e-01  1.183e+00  -0.234   0.8148
## V22         -9.605e-01  2.828e+00  -0.340   0.7342
## V23          7.577e-01  7.015e-01   1.080   0.2801
```

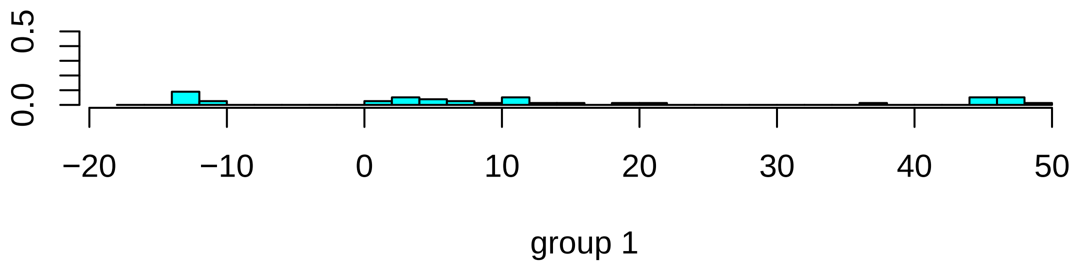
```
## V24          5.802e-01  6.638e-01   0.874   0.3821
## V25         -1.237e+00  1.240e+00  -0.998   0.3185
## V26          2.386e+00  5.001e+00   0.477   0.6333
## V27         -1.154e+00  1.069e+00  -1.079   0.2805
## V28         -1.401e-02  6.271e-01  -0.022   0.9822
## Amount       4.386e-03  6.822e-03   0.643   0.5203
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 582.06  on 24999  degrees of freedom
## Residual deviance: 128.95  on 24969  degrees of freedom
## AIC: 190.95
##
## Number of Fisher Scoring iterations: 18
```

```
glm.probs <- predict(glm.fit, test, type ="response")

glm.pred <- rep(0, dim(test)[1])
glm.pred[glm.probs >.5] <- 1
table(test$Class, glm.pred)
```

```
##    glm.pred
##         0     1
##   0 24957     3
##   1    18    22
```

```
Ac.glm <- mean(test$Class == glm.pred)
# 0.99916

# 2. Linear Discriminant Analysis
lda.fit <- lda(Class~., data=training)
plot(lda.fit)
```






```
lda.pred <- predict(lda.fit, test)
lda.class <- lda.pred$class
table(test$Class, lda.class)
```

```
##    lda.class
##         0     1
##   0 24955     5
##   1    11    29
```

```
Ac.lda <- mean(test$Class == lda.class)
# 0.99936

# 3. Quadratic Discriminant Analysis
qda.fit <- qda(Class~., data=training)

qda.pred <- predict(qda.fit, test)
qda.class <- qda.pred$class
table(test$Class, qda.class)
```

```
##    qda.class
##         0     1
##   0 24835   125
##   1     8    32
```

```
Ac.qda <- mean(test$Class == qda.class)
# 0.99468

# 4. K-Nearest Neighbors
class <- training$Class
train.knn <- as.matrix(training[, seq(1,30)])
test.knn <- as.matrix(test[, seq(1,30)])

knn.pred1 <- knn(train.knn, test.knn, class, k = 1)
```

```r
knn.pred2 <- knn(train.knn, test.knn, class, k = 2)
knn.pred3 <- knn(train.knn, test.knn, class, k = 3)
table(test$Class, knn.pred1)
```

```
##    knn.pred1
##         0     1
##   0 24933    27
##   1    40     0
```

```r
Ac.knn1 <- mean(test$Class == knn.pred1)
# 0.99732
table(test$Class, knn.pred2)
```

```
##    knn.pred2
##         0     1
##   0 24929    31
##   1    40     0
```

```r
Ac.knn2 <- mean(test$Class == knn.pred2)
# 0.99716
table(test$Class, knn.pred3)
```

```
##    knn.pred3
##         0     1
##   0 24960     0
##   1    40     0
```

```r
Ac.knn3 <- mean(test$Class == knn.pred3)
# 0.9984
```

Accuracy of predictions:

```r
data.frame(Ac.glm, Ac.lda, Ac.qda, Ac.knn1, Ac.knn2, Ac.knn3)
```

```
##    Ac.glm  Ac.lda  Ac.qda Ac.knn1 Ac.knn2 Ac.knn3
## 1 0.99916 0.99936 0.99468 0.99732 0.99716  0.9984
```

Since the dataset is highly unbalanced, all the accuracy is very high. But only logestic regression and LDA's accuracy surpass the baseline accuracy. It turns out that LDA has the highest accuracy rate.

Question 4:

For the Boston data set, fit a tree trying to predict crime (crim) based on all of the other variables. This should be the best tree that you can fit (you should try bumping, bagging, and boosting to ensure this).

```r
# bag tree function
bag_function <- function(Boston,index){
    bag.boston <- randomForest(crim~., data = Boston, subset=index,
                              mtry=13,importance=TRUE)
    boston.bag <- predict(bag.boston,newdata=Boston[-index,])
    boston.test <- Boston[-index,"crim"]
    return(mean(boston.bag))
}

#boost tree function
boost_function <- function(Boston,train){
```

```r
    boost.boston <-gbm(crim~., data=Boston[train,],distribution = "gaussian",
                       n.trees = 5000,interaction.depth = 4)
    boston.boost <-predict(boost.boston,newdata=Boston[-train,],n.trees=5000)
    boston.test <- Boston[-train,"crim"]
    return(mean(boston.test))
}
```

Determine your error rate

```r
result_bag  <- boot(data=Boston,statistic = bag_function,R=100)
result_boost <- boot(data=Boston,statistic = boost_function,R=100)
result_bag
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = bag_function, R = 100)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*      NaN     NaN   0.4822616
```

```r
result_boost
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = boost_function, R = 100)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*      NaN     NaN   0.4987222
```

The standard error of bagging (0.4822616) is a little lower than boosting (0.4987222).