Bruce Campbell ST-617 Homework 2

Wed Jul 06 10:43:16 2016

Chapter 4

Problem 7

Suppose that we wish to predict whether a given stock will issue a dividend this year ("Yes" or "No") based on X, last year's percent profit.We examine a large number of companies and discover that the mean value of X for companies that issued a dividend was $\overline{X} = 10$, while the mean for those that didn't was $\overline{x} = 0$. In addition, the variance of X for these two sets of companies was $\hat{\sigma^2} = 36$. Finally, 80% of companies issued dividends. Assuming that X follows a normal distribution, predict the probability that a company will issue a dividend this year given that its percentage profit was X = 4 last year.

Let's denote our 2 classes 0,1 where 0 indicates no dividend and 1 indicates a dividend. Then we will need to calculate the posterior probability $P(Y = 1|X = 4)$. We are given all of the information we need to do this. The prior probabilities are $\pi_1 = 0.8$ and $\pi_0 = 0.2$ and the likelihood of each class is given by $N(\mu_1, \hat{\sigma})(x)$ where $\mu_1 = 10$ and $N(\mu_0, \hat{\sigma})$ where $\mu_0 = 0$ and $\hat{\sigma} = 36$ in both cases. $N(\mu, \sigma)(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-1}{2\sigma^2}(x-\mu)^2}$ is the normal distribution with mean μ and variance $\hat{\sigma^2}$.

Putting this all together into Bayes theorem

$$P(Y = 1|X = 4) = \frac{N(\mu_1, \hat{\sigma})(4)\pi_1}{N(\mu_1, \hat{\sigma})(4)\pi_1 + N(\mu_0, \hat{\sigma})(4)\pi_0}$$

```r
mu_1 = 0.1
mu_0 = 0

pi_1 = 0.8
pi_0 = 0.2

sigma_sq = 36
stdev_est = 6

x = 4

posterior_probability_of_dividend_given_x <- function(x, mu_1, mu_0, stdev_est,
    pi_1, pi_0) {
    probability = (dnorm(x, mean = mu_1, sd = stdev_est) * pi_1)/(dnorm(x, mean = mu_1,
        sd = stdev_est) * pi_1 + dnorm(x, mean = mu_0, sd = stdev_est) * pi_0)
    return(probability)
}

posterior_probability_of_dividend <- posterior_probability_of_dividend_given_x(x,
    mu_1, mu_0, stdev_est, pi_1, pi_0)
```

We have that the probability of a dividend in the event the $X =$ is `0.8017498`.

Chapter 4

Problem 10

This question should be answered using the Weekly data set, which is part of the ISLR package. This data is similar in nature to the Smarket data from this chapter's lab, except that it contains 1,089 weekly returns for 21 years, from the beginning of 1990 to the end of 2010.

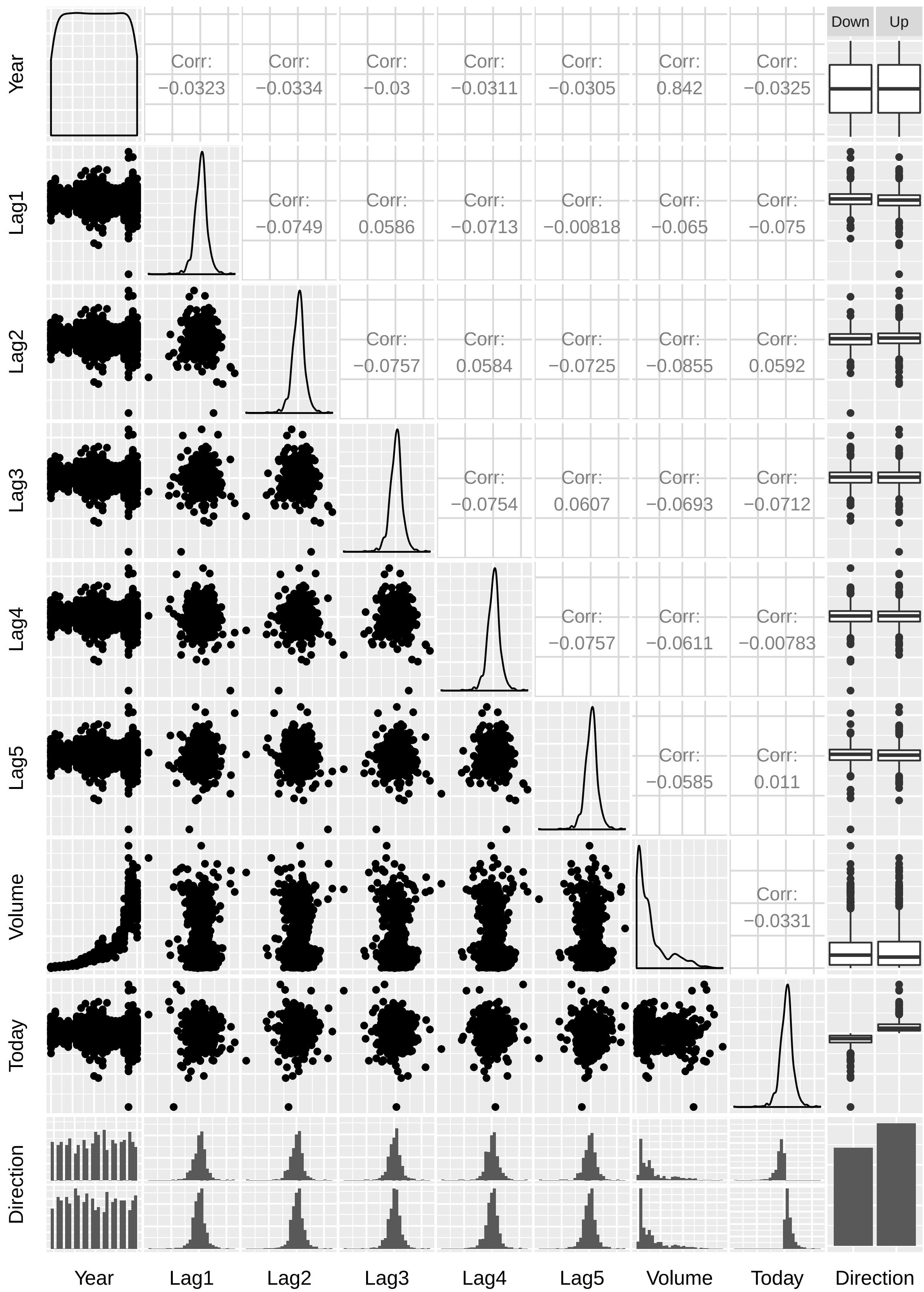
a) Produce some numerical and graphical summaries of the Weekly data. Do there appear to be any patterns?

```r
options(warn = -1)
library(ISLR)
attach(Weekly)
summary(Weekly)
```

```
##       Year           Lag1               Lag2               Lag3
##  Min.   :1990   Min.   :-18.1950   Min.   :-18.1950   Min.   :-18.1950
##  1st Qu.:1995   1st Qu.: -1.1540   1st Qu.: -1.1540   1st Qu.: -1.1580
##  Median :2000   Median :  0.2410   Median :  0.2410   Median :  0.2410
##  Mean   :2000   Mean   :  0.1506   Mean   :  0.1511   Mean   :  0.1472
##  3rd Qu.:2005   3rd Qu.:  1.4050   3rd Qu.:  1.4090   3rd Qu.:  1.4090
##  Max.   :2010   Max.   : 12.0260   Max.   : 12.0260   Max.   : 12.0260
##       Lag4               Lag5              Volume
##  Min.   :-18.1950   Min.   :-18.1950   Min.   :0.08747
##  1st Qu.: -1.1580   1st Qu.: -1.1660   1st Qu.:0.33202
##  Median :  0.2380   Median :  0.2340   Median :1.00268
##  Mean   :  0.1458   Mean   :  0.1399   Mean   :1.57462
##  3rd Qu.:  1.4090   3rd Qu.:  1.4050   3rd Qu.:2.05373
##  Max.   : 12.0260   Max.   : 12.0260   Max.   :9.32821
##      Today          Direction
##  Min.   :-18.1950   Down:484
##  1st Qu.: -1.1540   Up  :605
##  Median :  0.2410
##  Mean   :  0.1499
##  3rd Qu.:  1.4050
##  Max.   : 12.0260
```

```r
library(ggplot2)
require(GGally)
ggpairs(Weekly) + theme(axis.line = element_blank(), axis.text = element_blank(),
    axis.ticks = element_blank())
```

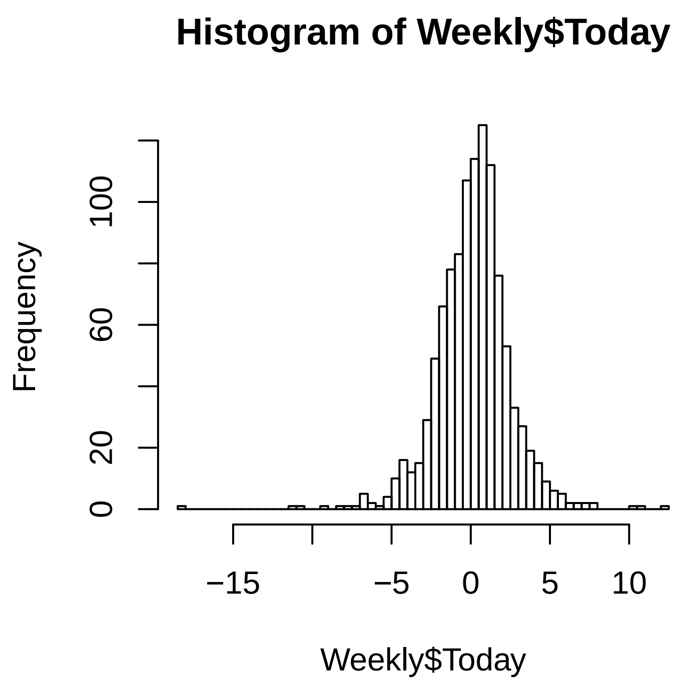
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


Down
Up
Corr: −0.0323
Corr: −0.0334
Corr: −0.03
Corr: −0.0311
Corr: −0.0305
Corr: 0.842
Corr: −0.0325
Corr: −0.0749
Corr: 0.0586
Corr: −0.0713
Corr: −0.00818
Corr: −0.065
Corr: −0.075
Corr: −0.0757
Corr: 0.0584
Corr: −0.0725
Corr: −0.0855
Corr: 0.0592
Corr: −0.0754
Corr: 0.0607
Corr: −0.0693
Corr: −0.0712
Corr: −0.0757
Corr: −0.0611
Corr: −0.00783
Corr: −0.0585
Corr: 0.011
Corr: −0.0331
Year
Lag1
Lag2
Lag3
Lag4
Lag5
Volume
Today
Direction

From the above we note that

- THere are more up than down weeks in the data set
- Volume is increasing over time
- Volume on up days has a longer tail that volumen on down days
- returns may have skew

```
hist(Weekly$Today, 50)
```




```
library(moments)
skewness(Weekly$Today)
```

```
## [1] -0.4805021
```

b)

Use the full data set to perform a logistic regression with Direction as the response and the five lag variables plus Volume as predictors. Use the summary function to print the results. Do any of the predictors appear to be statistically significant? If so, which ones?

```
DFWeekly = Weekly
glm.fit = glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume, data = Weekly,
    family = binomial)
summary(glm.fit)
```

```
##
```

```
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = Weekly)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6949  -1.2565   0.9913   1.0849   1.4579
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.26686    0.08593   3.106   0.0019 **
## Lag1        -0.04127    0.02641  -1.563   0.1181
## Lag2         0.05844    0.02686   2.175   0.0296 *
## Lag3        -0.01606    0.02666  -0.602   0.5469
## Lag4        -0.02779    0.02646  -1.050   0.2937
## Lag5        -0.01447    0.02638  -0.549   0.5833
## Volume      -0.02274    0.03690  -0.616   0.5377
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1486.4  on 1082  degrees of freedom
## AIC: 1500.4
##
## Number of Fisher Scoring iterations: 4
```

The lag2 variable is significant with a p-value of 0.0296

c)

Compute the confusion matrix and overall fraction of correct predictions. Explain what the confusion matrix is telling you about the types of mistakes made by logistic regression.

```r
glm.probs = predict(glm.fit, type = "response")
library(pander)
```

```
##
## Attaching package: 'pander'

## The following object is masked from 'package:GGally':
##
##     wrap
```

```r
contrasts(Direction)
```

```
##      Up
## Down  0
## Up    1
```

```r
glm.pred = rep("Down ", nrow(Weekly))
glm.pred[glm.probs > 0.5] = " Up"
table(glm.pred, Direction)
```

```
##          Direction
## glm.pred Down  Up
##      Up    430 557
##     Down    54  48
```

```r
pi_up = sum(Direction == "Up")
pi_down = sum(Direction == "Down")
```

We see that the accuracy is (557 + 54) / 1089 which is 56% and that the classifier does poorly on the down class where the accuracy is `0.1115702`

d)

Now fit the logistic regression model using a training data period from 1990 to 2008, with Lag2 as the only predictor. Compute the confusion matrix and the overall fraction of correct predictions for the held out data (that is, the data from 2009 and 2010).

```r
invisible(attach(Weekly))
DF <- Weekly
DFTrain <- DF[DF$Year <= 2008, ]
DFTest <- DF[DF$Year > 2008, ]
glm.fit = glm(Direction ~ Lag2, data = DFTrain, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag2, family = binomial, data = DFTrain)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.536  -1.264   1.021   1.091   1.368
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.20326    0.06428   3.162  0.00157 **
## Lag2         0.05810    0.02870   2.024  0.04298 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1354.7  on 984  degrees of freedom
## Residual deviance: 1350.5  on 983  degrees of freedom
## AIC: 1354.5
##
## Number of Fisher Scoring iterations: 4
```

```r
glm.probs = predict(glm.fit, DFTest, type = "response")
library(pander)
glm.pred = rep("Down ", nrow(DFTest))
glm.pred[glm.probs > 0.5] = " Up"
table(glm.pred, DFTest$Direction)
```

```
##
## glm.pred Down Up
##     Up     34 56
##    Down     9  5
```

```r
pi_up = sum(Direction == "Up")
pi_down = sum(Direction == "Down")
```

The accuracy for logistic regerssion on the test set is (9+56)/104 - or 62%.

e) Repeat (d) using LDA.

```r
library(MASS)
lda.fit = lda(Direction ~ Lag2, data = DFTrain)
lda.fit
```

```
## Call:
## lda(Direction ~ Lag2, data = DFTrain)
##
## Prior probabilities of groups:
##      Down        Up
## 0.4477157 0.5522843
##
## Group means:
##             Lag2
## Down -0.03568254
## Up    0.26036581
##
## Coefficients of linear discriminants:
##            LD1
## Lag2 0.4414162
```

```r
lda.pred = predict(lda.fit, DFTest)
names(lda.pred)
```

```
## [1] "class"     "posterior" "x"
```

```r
lda.class = lda.pred$class
table(lda.class, DFTest$Direction)
```

```
##
## lda.class Down Up
##      Down    9  5
##      Up     34 56
```

The accuracy for LDA classification on the test set is (9+56)/104 - or 62%. Note this is identical to the logistic regression

f) Repeat (d) using QDA.

```r
invisible(attach(Weekly))
```

```
## The following objects are masked from Weekly (pos = 3):
##
##     Direction, Lag1, Lag2, Lag3, Lag4, Lag5, Today, Volume, Year
```

```r
train = (Year < 2009)
Weekly.2009 = Weekly[!train, ]
Direction.2009 = Weekly$Direction[!train]
qda.fit = qda(Direction ~ Lag2, data = Weekly, subset = train)
qda.class = predict(qda.fit, Weekly.2009)$class
table(qda.class, Direction.2009)
```

```
##          Direction.2009
## qda.class Down Up
##      Down    0  0
##      Up     43 61
```

This classifier did not correctly classify any of the down test points. We diagnose the code a few ways below. First by adding the Lag1 variable and second by reproducing the results on the SMarket dataset.

```r
qda.fit = qda(Direction ~ Lag1 + Lag2, data = DFTrain)
qda.fit
```

```
## Call:
## qda(Direction ~ Lag1 + Lag2, data = DFTrain)
##
## Prior probabilities of groups:
##      Down        Up
## 0.4477157 0.5522843
##
## Group means:
##              Lag1        Lag2
## Down  0.289444444 -0.03568254
## Up   -0.009213235  0.26036581
```

```r
qda.pred = predict(qda.fit, DFTest)
names(qda.pred)
```

```
## [1] "class"     "posterior"
```

```r
qda.class = predict(qda.fit, DFTest)$class

qda.class = qda.pred$class
table(qda.class, DFTest$Direction)
```

```
##
## qda.class Down Up
##      Down    7 10
##      Up     36 51
```

The accuracy for this classifier is (7+51) / 104 - 56%

g) Repeat (d) using KNN with K = 1.

```r
attach(Weekly)
library(class)
train = (Year < 2009)
train.X = data.frame(cbind(Lag2)[train, ])
test.X = data.frame(cbind(Lag2)[!train, ])
train.Direction = Direction[train]
set.seed(1)
knn.pred = knn(train.X, test.X, train.Direction, k = 1)
table(knn.pred, Direction.2009)
```

```
##         Direction.2009
## knn.pred Down Up
##     Down   21 30
##     Up     22 31
```

The accuracy of KNN with k=1 is (32 + 18) / 104 - 48%.

h)

Which of these methods appears to provide the best results on this data? For this data set and model we see that the logistic regression and LDA are the top performers in terms of classification accuracy.

i)

Experiment with different combinations of predictors, including possible transformations and interactions, for each of the methods. Report the variables, method, and associated confusion matrix that appears to provide the best results on the held out data. Note that you should also experiment with values for K in the KNN classifier.

```r
library(class)
train = (Year < 2009)
Weekly.2009 = Weekly[!train, ]
Direction.2009 = Direction[!train]

message("KNN")
```

```
## KNN
```

```
train.X = cbind(Lag1, Lag2)[train, ]
test.X = cbind(Lag1, Lag2)[!train, ]
train.Direction = Direction[train]
set.seed(1)
knn.pred = knn(train.X, test.X, train.Direction, k = 2)
TB <- table(knn.pred, Direction.2009)
ACC_KNN = (TB[1] + TB[4])/length(Direction.2009)
modelsDF <- data.frame(model = "KNN(Direction~Lag1,Lag2) k=2", Accuracy = ACC_KNN)


train.X = cbind(Lag1, Lag2, Volume)[train, ]
test.X = cbind(Lag1, Lag2, Volume)[!train, ]
train.Direction = Direction[train]
set.seed(1)
knn.pred = knn(train.X, test.X, train.Direction, k = 1)
TB <- table(knn.pred, Direction.2009)
ACC_KNN = (TB[1] + TB[4])/length(Direction.2009)
modelsDF <- rbind(modelsDF, data.frame(model = "KNN(Direction~Lag1+Lag2+Volume) k=1",
    Accuracy = ACC_KNN))


knn.pred = knn(train.X, test.X, train.Direction, k = 2)
TB <- table(knn.pred, Direction.2009)
ACC_KNN = (TB[1] + TB[4])/length(Direction.2009)
modelsDF <- rbind(modelsDF, data.frame(model = "KNN(Direction~Lag1+Lag2+Volume) k=2",
    Accuracy = ACC_KNN))


knn.pred = knn(train.X, test.X, train.Direction, k = 4)
TB <- table(knn.pred, Direction.2009)
ACC_KNN = (TB[1] + TB[4])/length(Direction.2009)
modelsDF <- rbind(modelsDF, data.frame(model = "KNN(Direction~Lag1+Lag2+Volume) k=4",
    Accuracy = ACC_KNN))

message("QDA")
```

```
## QDA
```

```
train = (Year < 2009)
Weekly.2009 = Weekly[!train, ]
Direction.2009 = Direction[!train]
qda.fit = qda(Direction ~ Lag1 + Lag2 + Volume, data = Weekly, subset = train)
qda.class = predict(qda.fit, Weekly.2009)$class
TB <- table(qda.class, Direction.2009)
ACC_QDA = (TB[1] + TB[4])/length(Direction.2009)
modelsDF <- rbind(modelsDF, data.frame(model = "QDA(Direction~Lag1+Lag2+Volume)",
    Accuracy = ACC_QDA))

qda.fit = qda(Direction ~ Lag1 + Lag2 + +Volume + Lag1 * Lag2, data = Weekly,
    subset = train)
qda.class = predict(qda.fit, Weekly.2009)$class
TB <- table(qda.class, Direction.2009)
ACC_QDA = (TB[1] + TB[4])/length(Direction.2009)
```

```r
modelsDF <- rbind(modelsDF, data.frame(model = "QDA(Direction~Lag1+Lag2+Volume+Direction + Lag1*Lag2)",
    Accuracy = ACC_QDA))

qda.fit = qda(Direction ~ Lag1 + Lag2 + Lag1 * Lag2, data = Weekly, subset = train)
qda.class = predict(qda.fit, Weekly.2009)$class
TB <- table(qda.class, Direction.2009)
ACC_QDA = (TB[1] + TB[4])/length(Direction.2009)
modelsDF <- rbind(modelsDF, data.frame(model = "QDA(Direction~Lag1+Lag2+Lag1 * Lag1)",
    Accuracy = ACC_QDA))

qda.fit = qda(Direction ~ Lag1 + Lag2, data = Weekly, subset = train)
qda.class = predict(qda.fit, Weekly.2009)$class
TB <- table(qda.class, Direction.2009)
ACC_QDA = (TB[1] + TB[4])/length(Direction.2009)
modelsDF <- rbind(modelsDF, data.frame(model = "QDA(Direction~Lag1+Lag2)", Accuracy = ACC_QDA))

message("LDA")
```

```
## LDA
```

```r
train = (Year < 2009)
Weekly.2009 = Weekly[!train, ]
Direction.2009 = Direction[!train]
lda.fit = lda(Direction ~ Lag1 + Lag2 + Volume, data = Weekly, subset = train)
lda.class = predict(lda.fit, Weekly.2009)$class
TB <- table(lda.class, Direction.2009)
ACC_LDA = (TB[1] + TB[4])/length(Direction.2009)
modelsDF <- rbind(modelsDF, data.frame(model = "LDA(Direction~Lag1+Lag2+Volume)",
    Accuracy = ACC_LDA))

lda.fit = lda(Direction ~ Lag1 + Lag2 + +Volume + Lag1 * Lag2, data = Weekly,
    subset = train)
lda.class = predict(lda.fit, Weekly.2009)$class
TB <- table(lda.class, Direction.2009)
ACC_LDA = (TB[1] + TB[4])/length(Direction.2009)
modelsDF <- rbind(modelsDF, data.frame(model = "LDA(Direction~Lag1+Lag2+Volume+Direction + Lag1*Lag2)",
    Accuracy = ACC_LDA))

lda.fit = lda(Direction ~ Lag1 + Lag2 + Lag1 * Lag2, data = Weekly, subset = train)
lda.class = predict(lda.fit, Weekly.2009)$class
TB <- table(lda.class, Direction.2009)
ACC_LDA = (TB[1] + TB[4])/length(Direction.2009)
modelsDF <- rbind(modelsDF, data.frame(model = "LDA(Direction~Lag1+Lag2+Lag1 * Lag1)",
    Accuracy = ACC_LDA))

lda.fit = lda(Direction ~ Lag1 + Lag2, data = Weekly, subset = train)
lda.class = predict(lda.fit, Weekly.2009)$class
TB <- table(lda.class, Direction.2009)
ACC_LDA = (TB[1] + TB[4])/length(Direction.2009)
modelsDF <- rbind(modelsDF, data.frame(model = "LDA(Direction~Lag1+Lag2)", Accuracy = ACC_LDA))

pander(modelsDF)
```

model	Accuracy
KNN(Direction~Lag1,Lag2) k=2	0.5288
KNN(Direction~Lag1+Lag2+Volume) k=1	0.5
KNN(Direction~Lag1+Lag2+Volume) k=2	0.5096
KNN(Direction~Lag1+Lag2+Volume) k=4	0.4712
QDA(Direction~Lag1+Lag2+Volume)	0.4615
QDA(Direction~Lag1+Lag2+Volume+Direction + Lag1*Lag2)	0.4519
QDA(Direction~Lag1+Lag2+Lag1 * Lag1)	0.4615
QDA(Direction~Lag1+Lag2)	0.5577
LDA(Direction~Lag1+Lag2+Volume)	0.5288
LDA(Direction~Lag1+Lag2+Volume+Direction + Lag1*Lag2)	0.5385
LDA(Direction~Lag1+Lag2+Lag1 * Lag1)	0.5769
LDA(Direction~Lag1+Lag2)	0.5769

We see the best prforming models from this set are QDA(Direction~Lag1+Lag2) and LDA(Direction~Lag1+Lag2) #Chapter 5

Problem 5

In Chapter 4, we used logistic regression to predict the probability of default using income and balance on the Default data set. We will now estimate the test error of this logistic regression model using the validation set approach. Do not forget to set a random seed before beginning your analysis.

a)

Fit a logistic regression model that uses income and balance to predict default.

```r
rm(list = ls())
library(ISLR)
attach(Default)
glm.fit = glm(default ~ income + balance, data = Default, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = binomial,
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

b)

Using the validation set appbroach, estimate the test error of this model. In order to do this, you must perform the following steps: i. Split the sample set into a training set and a validation set. ii. Fit a multiple logistic regression model using only the training observations. iii. Obtain a prediction of default status for each individual in the validation set by computing the posterior probability of default for that individual, and classifying the individual to the default category if the posterior probability is greater than 0.5. iv. Compute the validation set error, which is the fraction of the observations in the validation set that are misclassified.

```r
set.seed(7)
train = sample(nrow(Default), floor(nrow(Default) * 2/3))
DF <- Default
DFTrain <- DF[train, ]
DFTest <- DF[-train, ]

glm.fit = glm(default ~ income + balance, data = DFTrain, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = binomial,
##     data = DFTrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2321  -0.1470  -0.0589  -0.0215   3.7152
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.129e+01  5.255e-01 -21.479   <2e-16 ***
## income       1.430e-05  6.074e-06   2.355   0.0185 *
## balance      5.630e-03  2.792e-04  20.163   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1940.3  on 6665  degrees of freedom
## Residual deviance: 1067.6  on 6663  degrees of freedom
## AIC: 1073.6
##
## Number of Fisher Scoring iterations: 8
```

```r
glm.probs = predict(glm.fit, DFTest, type = "response")

contrasts(DFTest$default)
```

```
##     Yes
## No    0
## Yes   1
```

```r
glm.pred = rep("No ", nrow(DFTest))
glm.pred[glm.probs > 0.5] = " Yes"
TB <- table(glm.pred, DFTest$default)
ACC_Validation = (TB[2] + TB[3])/length(DFTest$default)

modelsDF <- data.frame(iteration = 1, Accuracy = 1 - ACC_Validation)
```

c)

Repeat the process in (b) three times, using three different splits of the observations into a training set and a validation set. Comment on the results obtained.

```r
modelsDF <- data.frame(iteration = numeric(), Accuracy = numeric())

for (i in 1:3) {
    train = sample(nrow(Default), floor(nrow(Default) * 2/3))
    DF <- Default
    DFTrain <- DF[train, ]
    DFTest <- DF[-train, ]

    glm.fit = glm(default ~ income + balance, data = DFTrain, family = binomial)
    summary(glm.fit)

    glm.probs = predict(glm.fit, DFTest, type = "response")

    glm.pred = rep("No ", nrow(DFTest))
    glm.pred[glm.probs > 0.5] = " Yes"
    TB <- table(glm.pred, DFTest$default)
    ACC_Validation = (TB[2] + TB[3])/length(DFTest$default)
    modelsDF <- rbind(modelsDF, data.frame(iteration = i, Accuracy = 1 - ACC_Validation))
}
library(pander)
pander(modelsDF)
```

iteration	Accuracy
1	0.02759
2	0.02789
3	0.02579

The validation test set error rates are all very similar. This indicated the model is stable with respect to the random split into test and training sets and that the validation approach may be vaible in this instance althugh we'd probably want to do more iterations to confirm this.

d)

Now consider a logistic regression model that predicts the probability of default using income, balance, and a dummy variable for student. Estimate the test error for this model using the validation set approach. Comment on whether or not including a dummy variable for student leads to a reduction in the test error rate.

```
modelsDFAug <- data.frame(iteration = numeric(), Accuracy = numeric())

for (i in 1:3) {
    train = sample(nrow(Default), floor(nrow(Default) * 2/3))
    DF <- Default
    DFTrain <- DF[train, ]
    DFTest <- DF[-train, ]

    glm.fit = glm(default ~ income + balance + student, data = DFTrain, family = binomial)
    summary(glm.fit)

    glm.probs = predict(glm.fit, DFTest, type = "response")

    glm.pred = rep("No ", nrow(DFTest))
    glm.pred[glm.probs > 0.5] = " Yes"
    TB <- table(glm.pred, DFTest$default)
    ACC_Validation = (TB[2] + TB[3])/length(DFTest$default)
    modelsDFAug <- rbind(modelsDFAug, data.frame(iteration = i, Accuracy = 1 -
        ACC_Validation))
}
library(pander)
pander(modelsDFAug)
```

iteration	Accuracy
1	0.02729
2	0.02579
3	0.02729

Including the student status as a predictor did not appear to change the validation set error rates - the change is the number of errors for each of the three runs is :

```
diff <- (modelsDFAug$Accuracy - modelsDF$Accuracy) * nrow(DFTest)
pander(diff)
```

-1, *-7* and *5*

To make a more precise statement we'd run more iterations and compare the errors using a statistical test such as a t-test.

Chapter 5

Problem 8

We will now perform cross-validation on a simulated data set. ### a) Generate a simulated data set as follows:

```
set.seed(1)
# y=rnorm (100) #<------------- Not sure why this is necessary
x = rnorm(100)
y = x - 2 * x^2 + rnorm(100)
```

In this data set, what is n and what is p?

$n = 100$ and $p = 2$

Write out the model used to generate the data in equation form.

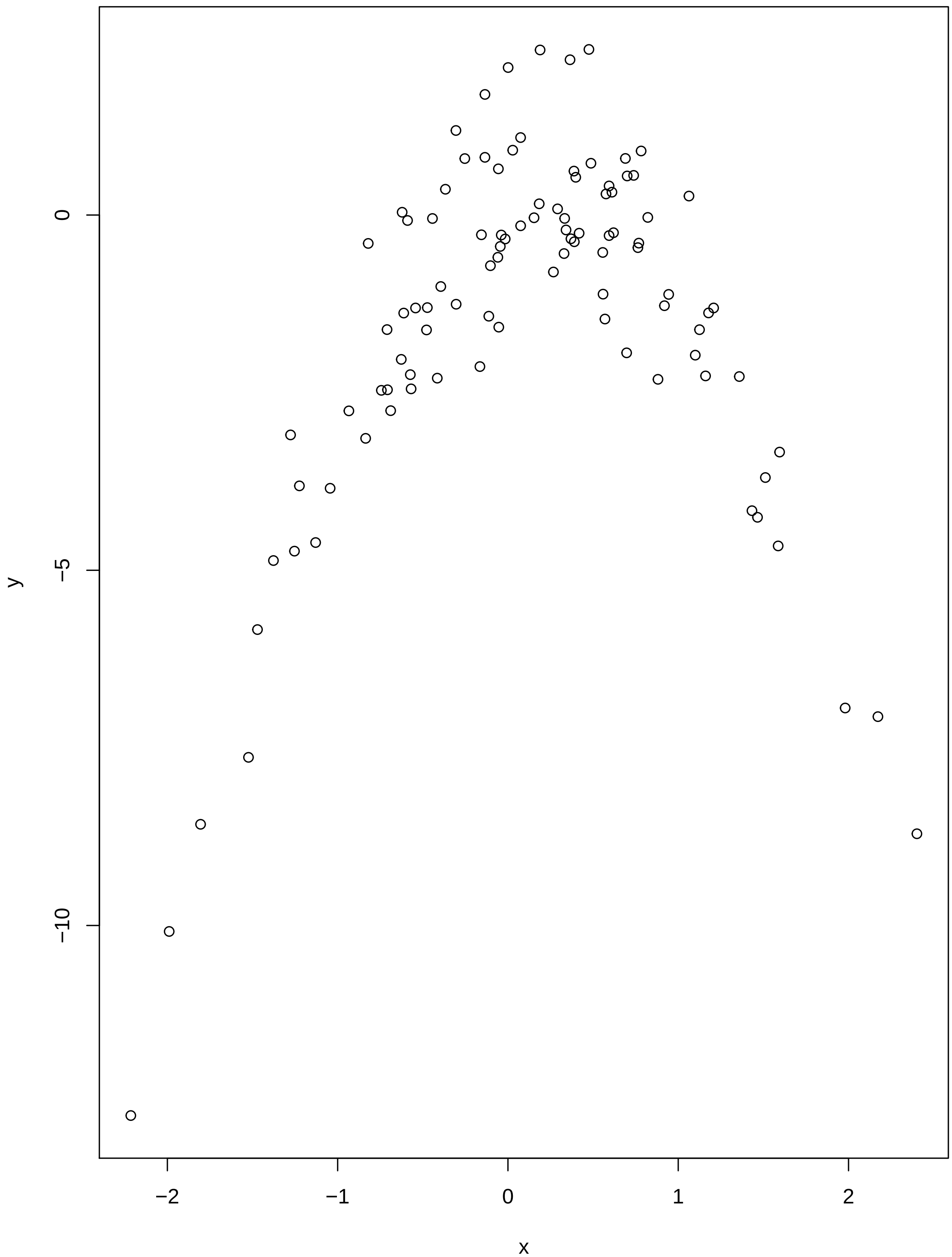
$$Y = \beta_1 X + \beta_2 X^2 + \epsilon$$

Where $\beta_1 = 1$, $\beta_2 = -2$, and $\epsilon = N(0, 1)$

b)

Create a scatterplot of X against Y . Comment on what you find.

```
plot(x, y)
```


y
0
−5
−10
−2
−1
0
1
2
x

We see the quadratic realtionship described in the model corrupted by the noise.

c)

Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares:

i.
$$Y = \beta_0 + \beta_1 X + \epsilon$$

ii.
$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$$

iii.
$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$$

iv.
$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$$

```
set.seed(17)
library(boot)

loocv_rates <- data.frame(model = character(), LOOCV_ERROR_delta1 = numeric(),
    LOOCV_ERROR_delta2 = numeric())
DF <- data.frame(X = x, Y = y)

glm.fit_1 <- glm(Y ~ X, data = DF)
coef(glm.fit_1)
```

```
## (Intercept)           X
##   -1.625427    0.692497
```

```
cv.err_1 <- cv.glm(DF, glm.fit_1)
loocv_rates <- rbind(loocv_rates, data.frame(model = "Y~X", LOOCV_ERROR_delta1 = cv.err_1$delta[1],
    LOOCV_ERROR_delta2 = cv.err_1$delta[2]))

glm.fit_2 <- glm(Y ~ X + I(X^2), data = DF)
coef(glm.fit_2)
```

```
## (Intercept)           X      I(X^2)
##  0.05671501  1.01716087 -2.11892120
```

```
cv.err_2 <- cv.glm(DF, glm.fit_2)
loocv_rates <- rbind(loocv_rates, data.frame(model = "Y~X+X^2", LOOCV_ERROR_delta1 = cv.err_2$delta[1],
    LOOCV_ERROR_delta2 = cv.err_2$delta[2]))



glm.fit_3 = glm(Y ~ X + I(X^2) + I(X^3), data = DF)
cv.err_3 = cv.glm(DF, glm.fit_3)
loocv_rates <- rbind(loocv_rates, data.frame(model = "Y~X+X^2+X^3", LOOCV_ERROR_delta1 = cv.err_3$delta
    LOOCV_ERROR_delta2 = cv.err_3$delta[2]))
```

```
glm.fit_4 = glm(Y ~ X + I(X^2) + I(X^3) + I(X^4), data = DF)
cv.err_4 = cv.glm(DF, glm.fit_4)
loocv_rates <- rbind(loocv_rates, data.frame(model = "Y~X+X^2+X^3+X^4", LOOCV_ERROR_delta1 = cv.err_4$de
    LOOCV_ERROR_delta2 = cv.err_4$delta[2]))

library(pander)
pander(loocv_rates)
```

model	LOOCV_ERROR_delta1	LOOCV_ERROR_delta2
Y~X	7.288	7.285
Y~X+X^2	0.9374	0.9372
Y~X+X^{2+X}3	0.9566	0.9563
Y~X+X^{2+X}3+X^4	0.9539	0.9534

d)

Repeat (c) using another random seed, and report your results. Are your results the same as what you got in (c)? Why?

```
set.seed(173)
library(boot)

loocv_rates <- data.frame(model = character(), LOOCV_ERROR_delta1 = numeric(),
    LOOCV_ERROR_delta2 = numeric())
DF <- data.frame(X = x, Y = y)

glm.fit_1 <- glm(Y ~ X, data = DF)
coef(glm.fit_1)
```

```
## (Intercept)           X
##   -1.625427    0.692497
```

```
cv.err_1 <- cv.glm(DF, glm.fit_1)
loocv_rates <- rbind(loocv_rates, data.frame(model = "Y~X", LOOCV_ERROR_delta1 = cv.err_1$delta[1],
    LOOCV_ERROR_delta2 = cv.err_1$delta[2]))

glm.fit_2 <- glm(Y ~ X + I(X^2), data = DF)
coef(glm.fit_2)
```

```
## (Intercept)           X      I(X^2)
##  0.05671501  1.01716087 -2.11892120
```

```
cv.err_2 <- cv.glm(DF, glm.fit_2)
loocv_rates <- rbind(loocv_rates, data.frame(model = "Y~X+X^2", LOOCV_ERROR_delta1 = cv.err_2$delta[1],
    LOOCV_ERROR_delta2 = cv.err_2$delta[2]))


glm.fit_3 = glm(Y ~ X + I(X^2) + I(X^3), data = DF)
cv.err_3 = cv.glm(DF, glm.fit_3)
```

```r
loocv_rates <- rbind(loocv_rates, data.frame(model = "Y~X+X^2+X^3", LOOCV_ERROR_delta1 = cv.err_3$delta
    LOOCV_ERROR_delta2 = cv.err_3$delta[2]))


glm.fit_4 = glm(Y ~ X + I(X^2) + I(X^3) + I(X^4), data = DF)
cv.err_4 = cv.glm(DF, glm.fit_4)
summary(glm.fit_4)
```

```
##
## Call:
## glm(formula = Y ~ X + I(X^2) + I(X^3) + I(X^4), data = DF)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0550  -0.6212  -0.1567   0.5952   2.2267
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.156703   0.139462   1.124    0.264
## X            1.030826   0.191337   5.387 5.17e-07 ***
## I(X^2)      -2.409898   0.234855 -10.261  < 2e-16 ***
## I(X^3)      -0.009133   0.067229  -0.136    0.892
## I(X^4)       0.069785   0.053240   1.311    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9197797)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  87.379  on 95  degrees of freedom
## AIC: 282.3
##
## Number of Fisher Scoring iterations: 2
```

```r
loocv_rates <- rbind(loocv_rates, data.frame(model = "Y~X+X^2+X^3+X^4", LOOCV_ERROR_delta1 = cv.err_4$de
    LOOCV_ERROR_delta2 = cv.err_4$delta[2]))

library(pander)
pander(loocv_rates)
```

model	LOOCV_ERROR_delta1	LOOCV_ERROR_delta2
Y~X	7.288	7.285
Y~X+X^2	0.9374	0.9372
Y~X+X^{2+X}3	0.9566	0.9563
Y~X+X^{2+X}3+X^4	0.9539	0.9534

These are the same results. The reason for this is that the LOOCV algorithm is deterministic. It trains n models with n-1 training points reserving the nth point as a test point. There is no random splitting of the training and test data.

e)

Which of the models in (c) had the smallest LOOCV error? Is this what you expected? Explain your answer.

The model with the lowest LOOCV error rate is the quadratic model. This is as expected since the data was generated via a quadratic relationship.

f)

Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (c) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results?

The p-values for the third and fourth coefficient are not significant. This is consistent with the cross validation results where the quadratic model had the lowest error.

Chapter 5

Problem 9

We will now consider the Boston housing data set, from the MASS library. ### a) Based on this data set, provide an estimate for the population mean of medv. Call this estimate $\hat{\mu}$

```
library(MASS)
attach(Boston)
mu_hat <- mean(Boston$medv)
```

$\hat{\mu} =$ `22.5328063`

b)

Provide an estimate of the standard error of $\hat{\mu}$. Interpret this result.

Hint: We can compute the standard error of the sample mean by dividing the sample standard deviation by the square root of the number of observations.

```
se_sm <- sd(Boston$medv)/nrow(Boston)
```

Our estimate of the standard error of the sample mean based on the standard deviation is `0.0181761`

c)

Now estimate the standard error of $\hat{\mu}$ using the bootstrap. How does this compare to your answer from (b)?

```
library(boot)
alpha.fn <- function(data, index) {
    D = data[index, ]
    result <- mean(D$medv)
    return(result)
}
```

```
se_bootstrap <- boot(Boston, alpha.fn, 100)

se_bootstrap
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = alpha.fn, R = 100)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 22.53281 0.02247233   0.4180892
```

d)

Based on your bootstrap estimate from (c), provide a 95% confidence interval for the mean of medv. Compare it to the results obtained using t.test(Boston$medv).

Hint: You can approximate a 95% confidence interval using the formula $[\hat{\mu} - 2SE(\hat{\mu}), \hat{\mu} + 2SE(\hat{\mu})]$.

```
left_ci <- mu_hat - 2 * sd(se_bootstrap$t)
right_ci <- mu_hat + 2 * sd(se_bootstrap$t)
```

Our 95% CI as estimated from the boostrap calculation is [`21.6966279` , `23.3689847`]

The results of the t-test are comparable to the bootstrap estimate, but the one sample t-test provides a smaller estimate of the 95% CI.

```
t.test(Boston$medv)
```

```
##
##  One Sample t-test
##
## data:  Boston$medv
## t = 55.111, df = 505, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  21.72953 23.33608
## sample estimates:
## mean of x
##  22.53281
```

e)

Based on this data set, provide an estimate, med, for the median value of medv in the population.

We can use the sample median for an estimate of the population median which is :

```r
median(Boston$medv)
```

```
## [1] 21.2
```

f)

We now would like to estimate the standard error of $\hat{\mu_med}$. Unfortunately, there is no simple formula for computing the standard error of the median. Instead, estimate the standard error of the median using the bootstrap. Comment on your findings.

```r
alpha.fn <- function(data, index) {
    D = data[index, ]
    result <- median(D$medv)
    return(result)
}
se_median_bootstrap <- boot(Boston, alpha.fn, 100)

se_median_bootstrap
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = alpha.fn, R = 100)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*     21.2   0.015   0.3877219
```

The estimate of standard error of the median as calculated by the bootstrap is

```r
sd(se_median_bootstrap$t)
```

```
## [1] 0.3877219
```

```r
left_ci <- mu_hat - 2 * sd(se_median_bootstrap$t)
right_ci <- mu_hat + 2 * sd(se_median_bootstrap$t)
```

Our 95% CI as estimated from the boostrap calculation is [`21.7573625` , `23.3082502`]

We're not sure if this estimate is valid for the median so let's calculate the 5th and 95th quantile from the bootstrap sample

```r
left_ci <- quantile(se_median_bootstrap$t, 0.05)
right_ci <- quantile(se_median_bootstrap$t, 0.95)
```

[`20.6` , `21.7`]

This is a much tighter CI, so we wonder if the estimate above applied to the median.

g)

Based on this data set, provide an estimate for the tenth percentile of medv in Boston suburbs. Call this quantity $\hat{\mu_0}.1$. (You can use the quantile() function.)

The sample tenth percentile of medv is

```
quantile(Boston$medv, 0.1)
```

```
##   10%
## 12.75
```

h)

Use the bootstrap to estimate the standard error of $\hat{\mu_0}.1$. Comment on your findings.

```
alpha.fn <- function(data, index) {
    D = data[index, ]
    result <- quantile(D$medv, 0.1)
    return(result)
}
se_quantile_bootstrap <- boot(Boston, alpha.fn, 100)

se_quantile_bootstrap
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = alpha.fn, R = 100)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    12.75 -0.0495   0.4884781
```

The estimate of standard error of the tenth percentile as calculated by the bootstrap is

```
sd(se_quantile_bootstrap$t)
```

```
## [1] 0.4884781
```

Chapter 6

Problem 10

We have seen that as the number of features used in a model increases, the training error will necessarily decrease, but the test error may not. We will now explore this in a simulated data set. ### a) Generate a data set with p = 20 features, n = 1,000 observations, and an associated quantitative response vector generated according to the model $Y = X\beta + \epsilon$, where β has some elements that are exactly equal to zero.

```r
rm(list = ls())
set.seed(7)
beta_v <- rnorm(20, 0, 1)
clamped_coeff = (beta_v < 0.5 & beta_v > -0.5)
beta_v[clamped_coeff] <- 0

X <- matrix(NA, nrow = 1000, ncol = 20)
Y <- matrix(NA, nrow = 1000, ncol = 1)
for (i in 1:1000) {
    x_i = rnorm(20, 0, 1)
    err = 0.1 * rnorm(1, 0, 1)
    Y_i = beta_v %*% x_i + err

    X[i, ] <- x_i

    Y[i] <- Y_i

}

DF <- as.data.frame(X)
DF <- cbind(DF, Y)
names(DF) = c("X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10", "X11",
    "X12", "X13", "X14", "X15", "X16", "X17", "X18", "X19", "X20", "Y")
```

b)

Split your data set into a training set containing 100 observations and a test set containing 900 observations.

```r
train = sample(nrow(DF), 900)
DFTrain <- DF[train, ]
DFTest <- DF[-train, ]
```

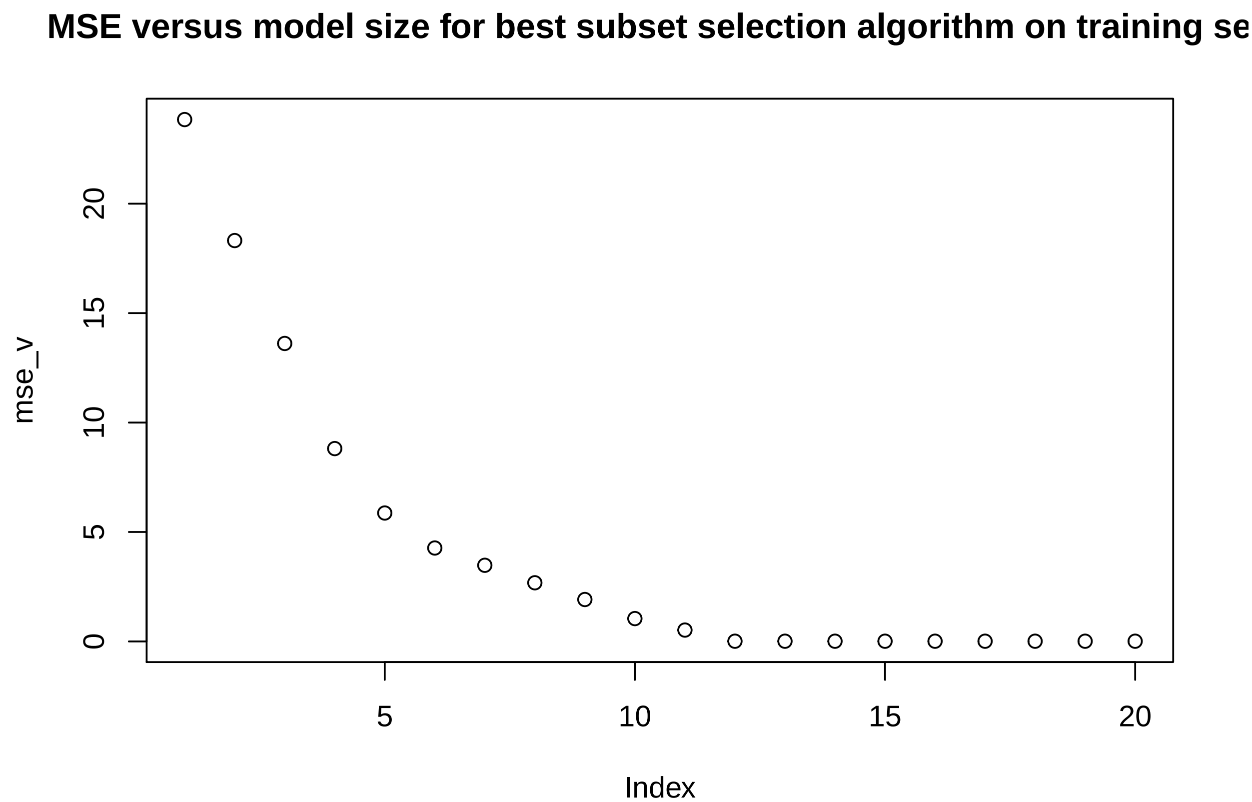
c)

Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size.

```r
library(leaps)
attach(DFTrain)
```

```
## The following object is masked _by_ .GlobalEnv:
##
##     Y
```

```r
regfit.full <- regsubsets(Y ~ ., data = DFTrain, nvmax = 20)
reg.summary <- summary(regfit.full)

mse_v <- reg.summary$rss/nrow(DFTrain)

plot(mse_v)
title("MSE versus model size for best subset selection algorithm on training set.")
```

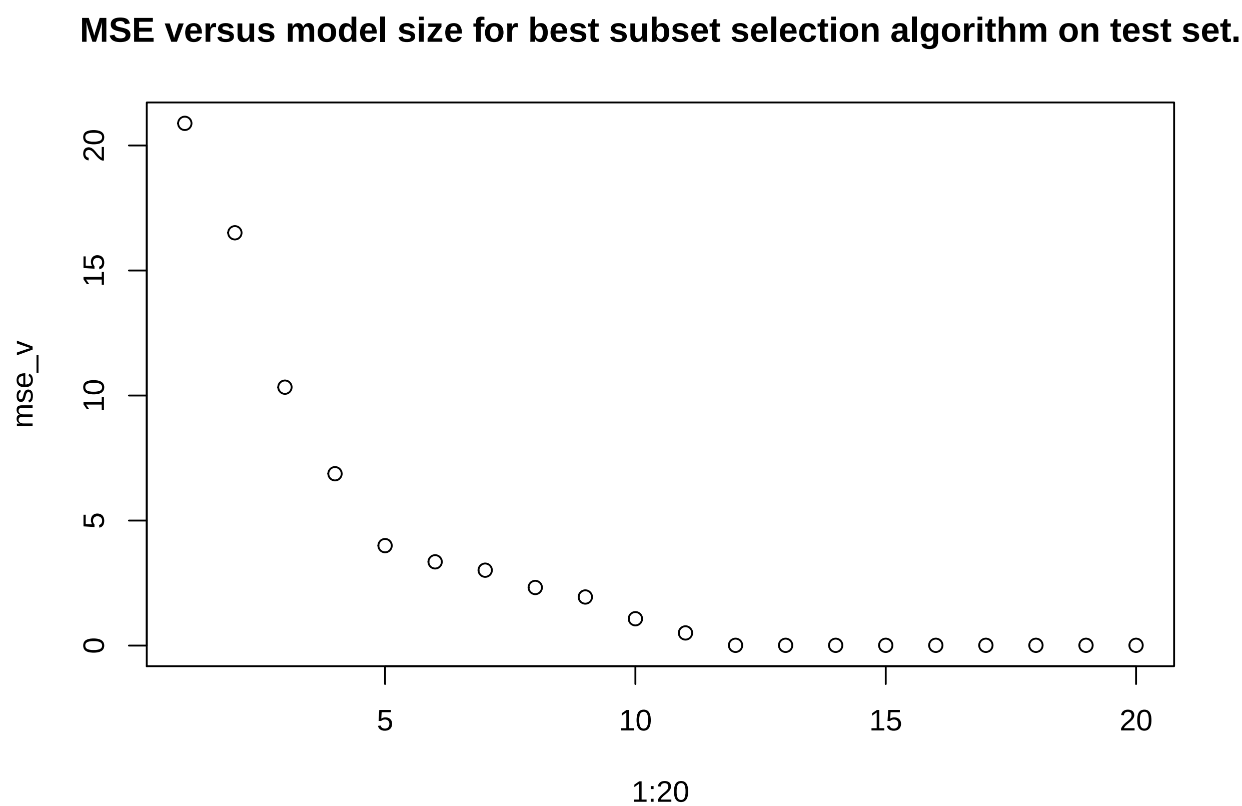


d)

Plot the test set MSE associated with the best model of each size.

```
test_set_size <- 100
mse_v = matrix(NA, 1, 20)
for (i in 1:20) {
    beta_subset <- data.frame(coef(regfit.full, i))

    mse_subset <- 0
    for (j in 1:test_set_size) {
        X_j = DFTest[, !(colnames(DFTest) %in% c("Y"))]

        X_j = X_j[j, ]

        Y_j = DFTest$Y[j]

        intercept = beta_subset["(Intercept)", ]

        coeff_names <- rownames(beta_subset)

        coeff_names <- coeff_names[-1]

        coeff_x <- as.matrix(beta_subset[-1, ])
```

```
        mse_v
        X_red <- X_j[, colnames(X_j) %in% coeff_names]

        X_red <- as.matrix(X_red)

        V1 = matrix(NA, 1, length(coeff_names))
        V2 = matrix(NA, length(coeff_names), 1)
        V1 = coeff_x
        V2 = X_red

        Yhat_j = intercept + V2 %*% V1

        mse_subset <- mse_subset + (Yhat_j - Y_j)^2
    }
    mse_subset <- mse_subset/test_set_size
    mse_v[i] = mse_subset
}
plot(1:20, mse_v)
title("MSE versus model size for best subset selection algorithm on test set.")
```



e)

For which model size does the test set MSE take on its minimum value? Comment on your results. If it takes on its minimum value for a model containing only an intercept or a model containing all of the features, then play around with the way that you are generating the data in (a) until you come up with a scenario in which the test set MSE is minimized for an intermediate model size.

```r
min_mse_model_index <- which.min(mse_v)
library(pander)
coeff_min <- coef(regfit.full, min_mse_model_index)
pander(coeff_min)
```

Table 6: Table continues below

(Intercept)	X1	X2	X3	X5	X6	X7	X10	X11	X12
-0.0006906	2.289	-1.19	-0.6916	-0.9697	-0.9397	0.748	2.194	-0.001764	2.718

X13	X14	X15	X17	X18	X19	X20
2.28	-0.001799	1.894	-0.8938	-0.005165	0.006309	0.994

The 12th subset is the one with the minimum MSE.

f)

How does the model at which the test set MSE is minimized compare to the true model used to generate the data? Comment on the coefficient values.

```r
V <- beta_v
V <- as.data.frame(V)
rownames(V) = c("X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10",
    "X11", "X12", "X13", "X14", "X15", "X16", "X17", "X18", "X19", "X20")
pander(t(V))
```

Table 8: Table continues below

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
V	2.287	-1.197	-0.6943	0	-0.9707	-0.9473	0.7481	0	0	2.19	0

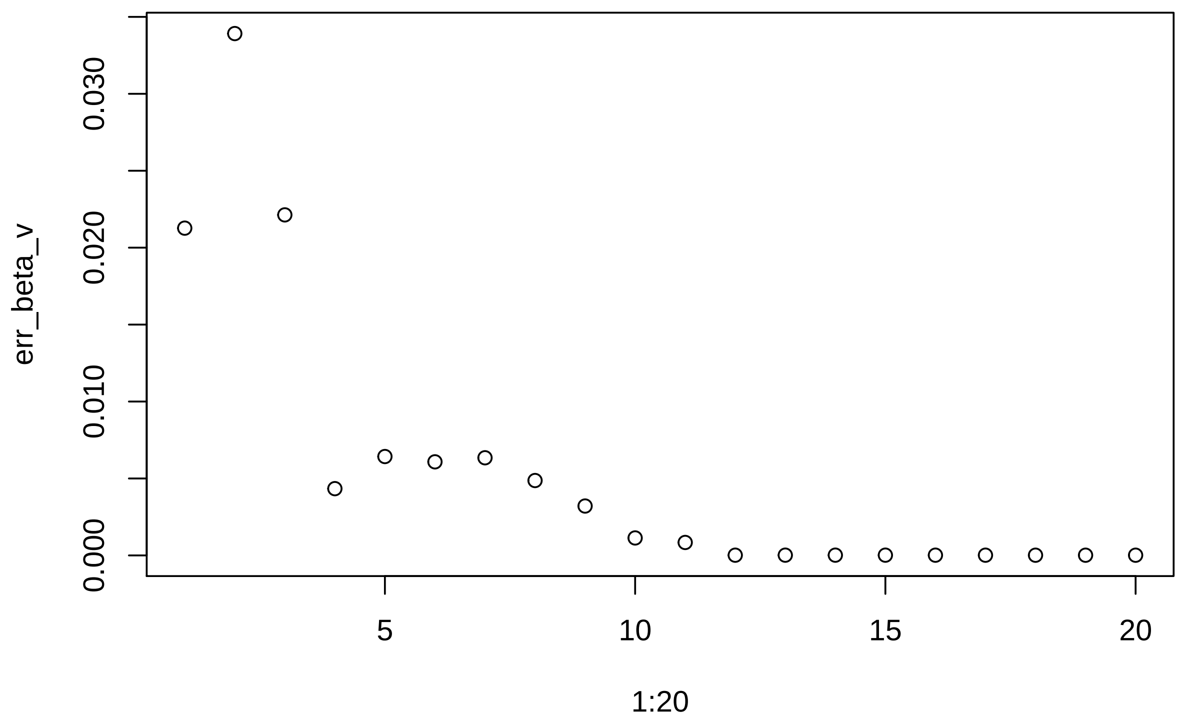
	X12	X13	X14	X15	X16	X17	X18	X19	X20
V	2.717	2.281	0	1.896	0	-0.8938	0	0	0.9882

Interestingly, we see that the best subset with minimum MSE has exactly the same non-zero features we used to generate the data.

g)

Create a plot displaying $\sqrt{\sum(\beta_j - \hat{\beta}_j^r)^2}$ for a range of values of r, where $\hat{\beta}_j^r$ is the jth coefficient estimate for the best model containing r coefficients. Comment on what you observe. How does this compare to the test MSE plot from (d)?

```
err_beta_v = matrix(NA, 1, 20)
betaDF <- as.data.frame(V)
rownames(betaDF) = c("X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10",
    "X11", "X12", "X13", "X14", "X15", "X16", "X17", "X18", "X19", "X20")
betaDF <- t(betaDF)

for (i in 1:20) {
    beta_subset <- data.frame(coef(regfit.full, i))

    err_beta_subset <- 0

    intercept = beta_subset["(Intercept)", ]

    coeff_names <- rownames(beta_subset)

    coeff_names <- coeff_names[-1]

    coeff_x <- as.matrix(beta_subset[-1, ])

    beta_red <- betaDF[, colnames(betaDF) %in% coeff_names]

    for (j in 1:length(coeff_names)) {
        Beta_j = betaDF[, coeff_names[j]]

        BetaHat_j = beta_subset[coeff_names[j], ]
        err_beta_subset <- err_beta_subset + (Beta_j - BetaHat_j)^2
    }
    err_beta_subset <- err_beta_subset/length(coeff_names)
    err_beta_v[i] = err_beta_subset
}
plot(1:20, err_beta_v)
```



The error in the coefficients decreases as model size increases, but we see that it is not a monotonic decrease.

```
which.min(err_beta_v)
```

```
## [1] 20
```

The plots look similar. The minimum coefficient error is achieved on the full model.