ST464 - Assignment 3 - Solutions - Version 2

Sean O'Riogain (18145426)

29 March 2019

```
getwd()
```

```
## [1] "C:/Users/oriogain/Dropbox/Maynooth/ST674 - Machine Learning/Assignments"
```

```
suppressPackageStartupMessages(library(tidyverse))    # Needed for deplyr funcs.
```

```
## Warning: package 'tibble' was built under R version 3.5.2
```

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

```
suppressPackageStartupMessages(library(MASS))         # Needed for lda, qda &
                                                      #   stepAIC and Pima data
suppressPackageStartupMessages(library(ISLR))         # Needed for Auto data
```

```
## Warning: package 'ISLR' was built under R version 3.5.2
```

```
suppressPackageStartupMessages(library(class))        # Needed for knn()
suppressPackageStartupMessages(library(leaps))        # Needed for regsubsets()
```

Question 1

The Titanic dataset records for each person on the ship the passenger class, age (child or adult), and sex, and whether they survived or not.

In this assignment you will use logistic regression on a training set (ttrain) to develop a classification rule, and then this rule will be applied to the test set (ttest).

```
ttrain <- read.csv("ttrain.csv", header=T, row.names=1)
ttest <- read.csv("ttest.csv", header=T, row.names=1)
```

a. Use logistic regression to build a model relating Survived to Class Age and Sex for the training data ttrain.

```
str(ttrain)
```

```
## 'data.frame':    1761 obs. of  4 variables:
##  $ Class   : Factor w/ 4 levels "1st","2nd","3rd",..: 3 4 4 1 2 1 4 1 4 4 ...
##  $ Sex     : Factor w/ 2 levels "Female","Male": 2 2 2 1 1 2 2 1 2 2 ...
##  $ Age     : Factor w/ 2 levels "Adult","Child": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Survived: Factor w/ 2 levels "No","Yes": 1 2 1 2 2 1 1 2 1 1 ...
```

```
f<-glm(Survived ~ Class + Age + Sex, data=ttrain, family="binomial")
summary(f)
```

```
##
## Call:
## glm(formula = Survived ~ Class + Age + Sex, family = "binomial",
##     data = ttrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1232  -0.7173  -0.4496   0.6768   2.1642
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   2.1431     0.1922  11.153  < 2e-16 ***
## Class2nd     -1.0136     0.2219  -4.567 4.95e-06 ***
## Class3rd     -1.8467     0.1952  -9.462  < 2e-16 ***
## ClassCrew    -0.8321     0.1779  -4.678 2.90e-06 ***
## AgeChild      1.0606     0.2889   3.671 0.000242 ***
## SexMale      -2.5373     0.1606 -15.795  < 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: 2214.5  on 1760  degrees of freedom
## Residual deviance: 1737.6  on 1755  degrees of freedom
## AIC: 1749.6
##
## Number of Fisher Scoring iterations: 4
```

b. From the fitted model, calculate a vector prob of survival probabilities and a vector pred of predicted classes, for the training data.

```
prob<-predict(f, type="response")                # Probabilities vector
head(prob)
```

```
##        633       1735        900       1941       2067        101
## 0.09614205 0.22683587 0.22683587 0.89502053 0.75575021 0.40270881
```

```
pred<-factor(ifelse(prob < 0.5, "No", "Yes"))   # Predicted classes vector
head(pred)
```

```
##  633 1735  900 1941 2067  101
##   No   No   No  Yes  Yes   No
## Levels: No Yes
```

```
confmat<-table(ttrain$Survived, pred)           # Construct the confusion matrix
confmat
```

```
##      pred
##         No  Yes
##   No  1097   96
##   Yes  284  284
```

What proportion of survivors are missclassified?

```
paste(round(confmat["Yes","No"]/rowSums(confmat)["Yes"]*100, 2),"%",sep="")
```

```
## [1] "50%"
```

What proportion of those who died are missclassified?

```
paste(round(confmat["No","Yes"]/rowSums(confmat)["No"]*100, 2),"%",sep="")
```

```
## [1] "8.05%"
```

What proportion of the predicted survivors actually survived?

```
paste(round(confmat["Yes","Yes"]/colSums(confmat)["Yes"]*100, 2),"%",sep="")
```

```
## [1] "74.74%"
```

What is the overall error rate for the training data?

```
paste(round((confmat["No","Yes"]+confmat["Yes","No"])/sum(confmat)*100, 2),"%",sep="")
```

```
## [1] "21.58%"
```

c. From the fitted model, calculate a vector prob of survival probabilities and a vector pred of predicted classes, for the test data.

```
str(ttest)
```

```
## 'data.frame':    440 obs. of  4 variables:
##  $ Class   : Factor w/ 4 levels "1st","2nd","3rd",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ Sex     : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 1 1 1 1 ...
##  $ Age     : Factor w/ 2 levels "Adult","Child": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Survived: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
```

```
prob<-predict(f, ttest, type="response")            # Probabilities vector
head(prob)
```

```
##         7         8        12        17        26        29
## 0.2350149 0.2350149 0.2350149 0.2350149 0.2350149 0.2350149
```

```
pred<-factor(ifelse(prob < 0.5, "No", "Yes"))       # Predicted classes vector
head(pred)
```

```
##  7  8 12 17 26 29
## No No No No No No
## Levels: No Yes
```

```
confmat<-table(ttest$Survived, pred)                # Construct the confusion matrix
confmat
```

```
##      pred
##        No Yes
##   No  267  30
##   Yes  78  65
```

What proportion of survivors are misclassified?

```
paste(round((confmat["Yes","No"]/rowSums(confmat)["Yes"])*100,2),"%",sep="")
```

```
## [1] "54.55%"
```

What proportion of those who died are misclassified?

```
paste(round((confmat["No","Yes"]/rowSums(confmat)["No"])*100,2),"%",sep="")
```

```
## [1] "10.1%"
```

What proportion of the predicted survivors actually survived?

```
paste(round(((confmat["Yes","Yes"])/colSums(confmat)["Yes"])*100,2),"%",sep="")
```

```
## [1] "68.42%"
```

What is the overall error rate for the test data?

```
paste(round(((confmat["No","Yes"]+confmat["Yes","No"])/sum(confmat))*100,2),"%",sep="")
```

```
## [1] "24.55%"
```

*** Not for Marking ***

Question 2

Suppose we wish to predict whether a given stock will issue a dividend this year (yes or no) based on X, last year's percentage profit.

We examine a large number of companies and discover that the mean value of X for companies that issued a dividend was 10, while the mean for those that didn't was 0.

$$\mu_{Y=Yes} = 10$$

$$\mu_{Y=No} = 0$$

In addition, the variance of X for these two sets of companies was 36.

$$\sigma^2 = 36$$

$$\sigma = 6$$

Finally, 80% of companies issued dividends.

$$P(Y = 1) = 0.8$$

$$P(Y = 0) = 0.2$$

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.

$$P(Y = 1|X = 4)$$

```
# Let's simulate some data based on the information provided above

# Simulating data for 8000 companies (80%) for which a dividend is issued...
set.seed(123)
x<-round(rnorm(8000,mean=10,sd=6),2)
y<-rep("Yes",length(x))
data_yes<-data.frame(pct=x, div=y)

# Simulating data for 2000 companies (20%) for which a dividend is not issued...
x<-round(rnorm(2000,mean=0,sd=6),2)
y<-rep("No",length(x))
data_no<-data.frame(pct=x, div=y)

# Combining both of those simulated datasets.....
data<-rbind(data_yes, data_no)
str(data)
```

```
## 'data.frame':    10000 obs. of  2 variables:
##  $ pct: num  6.64 8.62 19.35 10.42 10.78 ...
##  $ div: Factor w/ 2 levels "Yes","No": 1 1 1 1 1 1 1 1 1 1 ...
```

```
head(data)
```

```
##     pct div
## 1  6.64 Yes
## 2  8.62 Yes
## 3 19.35 Yes
## 4 10.42 Yes
## 5 10.78 Yes
## 6 20.29 Yes
```

```
# Fitting a model to the combined dataset.....
f<-glm(div ~ pct, data=data, family="binomial")
summary(f)
```

```
## 
## Call:
## glm(formula = div ~ pct, family = "binomial", data = data)
## 
## Deviance Residuals: 
##     Min       1Q   Median       3Q      Max  
## -2.6980  -0.5077  -0.2681  -0.1008   3.5801  
## 
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)    
## (Intercept) -0.008659   0.037999  -0.228     0.82    
## pct         -0.277105   0.006310 -43.917   <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: 10008.0  on 9999  degrees of freedom
## Residual deviance:  6429.1  on 9998  degrees of freedom
## AIC: 6433.1
## 
## Number of Fisher Scoring iterations: 6
```

```
# Very strange: This model appears to be expecting the probability of a dividend to increase
 as the profit margin decreases (contrary to expectations).

# Predicting if dividend will be issued where a company has a 4% profit margin
prob<-round(predict(f, data.frame(pct=4), type="response"), 2)
paste("Predicted Probability =",prob)
```

```
## [1] "Predicted Probability = 0.25"
```

Question 3

In the Auto data, create a new variable that contains the value 1 for cars with above the median mpg, and 0 for other cars. Name this variable mpg01. Split the data into a test and training sets of size containing 50% and 50% of observations each.

```
m <- median(Auto$mpg)
Auto$mpg01 <- factor(ifelse(Auto$mpg < m, 0, 1))
set.seed(1)
s <- sample(nrow(Auto), round(.5*nrow(Auto)))
Atrain <- Auto[s,]
Atest <- Auto[-s,]
str(Atrain)
```

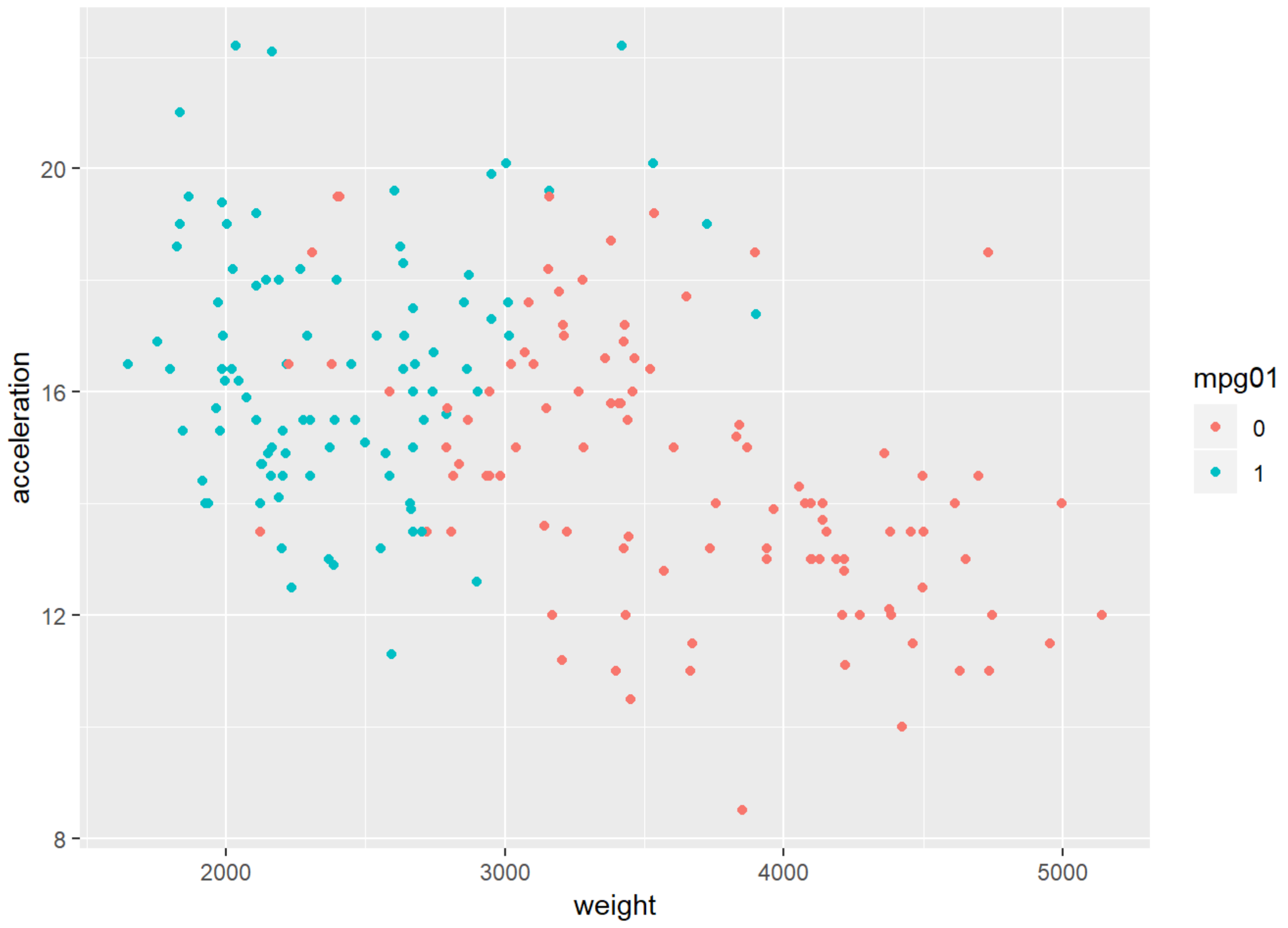
```
## 'data.frame':    196 obs. of  10 variables:
##  $ mpg         : num  13 24 17.5 31.6 26 34.4 34 20.5 21.5 26 ...
##  $ cylinders   : num  8 4 6 4 4 4 4 6 3 4 ...
##  $ displacement: num  360 90 250 120 96 98 112 225 80 121 ...
##  $ horsepower  : num  170 75 110 74 69 65 88 100 110 113 ...
##  $ weight      : num  4654 2108 3520 2635 2189 ...
##  $ acceleration: num  13 15.5 16.4 18.3 18 16.2 18 17.2 13.5 12.5 ...
##  $ year        : num  73 74 77 81 72 81 82 78 77 70 ...
##  $ origin      : num  1 2 1 3 2 1 1 1 3 2 ...
##  $ name        : Factor w/ 304 levels "amc ambassador brougham",..: 219 124 53 175 253 131
46 237 181 22 ...
##  $ mpg01       : Factor w/ 2 levels "0","1": 1 2 1 2 2 2 2 1 1 2 ...
```

```
str(Atest)
```

```
## 'data.frame':    196 obs. of  10 variables:
##  $ mpg         : num  18 15 18 15 14 14 15 14 15 14 ...
##  $ cylinders   : num  8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num  307 350 318 429 454 440 383 340 400 455 ...
##  $ horsepower  : num  130 165 150 198 220 215 170 160 150 225 ...
##  $ weight      : num  3504 3693 3436 4341 4354 ...
##  $ acceleration: num  12 11.5 11 10 9 8.5 10 8 9.5 10 ...
##  $ year        : num  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ name        : Factor w/ 304 levels "amc ambassador brougham",..: 49 36 231 141 54 223 1
01 215 57 30 ...
##  $ mpg01       : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

a. Plot the variables weight and acceleration using colour to show the two levels of mpg01 for the training set.

```
ggplot(data=Atrain, aes(x=weight, y=acceleration, colour=mpg01)) +
  geom_point(stat="identity")
```



This plot indicates that there is a (negative) linear relationship between weight and acceleration.

b. Perform a linear discriminant analysis to predict mpg01, using variables weight and acceleration, on the training set.

```
f<-lda(mpg01 ~ weight + acceleration, data=Atrain)       # Fit the model
f
```

```
## Call:
## lda(mpg01 ~ weight + acceleration, data = Atrain)
##
## Prior probabilities of groups:
##         0         1
## 0.5255102 0.4744898
##
## Group means:
##     weight acceleration
## 0 3636.359     14.49806
## 1 2404.151     16.43226
##
## Coefficients of linear discriminants:
##                      LD1
## weight       -0.001635093
## acceleration  0.060260084
```

```
pred<-predict(f)$class                      # Get predicted classes
confmat<-table(Atrain$mpg01, pred)          # Construct the confusion matrix
confmat
```

```
##    pred
##       0  1
##   0 85 18
##   1  6 87
```

```
# Calculate the overall training error rate
paste(round((confmat["0","1"]+confmat["1","0"])/sum(confmat)*100, 2),"%",sep="")
```

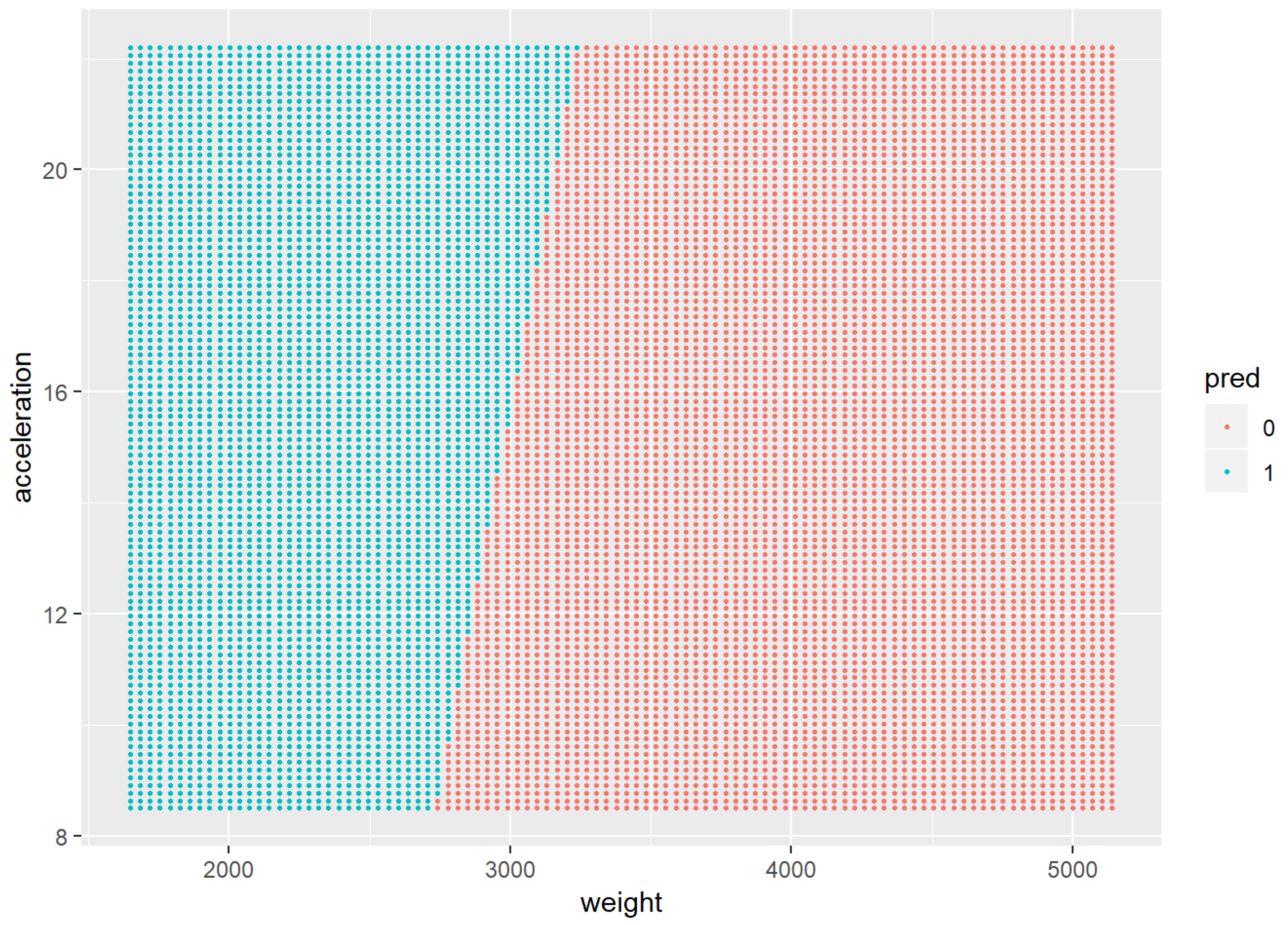
```
## [1] "12.24%"
```

Use a plot to show the discriminant boundaries.

```
# Create a grid representing a range of values for both predictors
grid<-expand.grid(weight=seq(min(Atrain$weight), max(Atrain$weight), length=100),
                  acceleration=seq(min(Atrain$acceleration),
                                   max(Atrain$acceleration),
                                   length=100))

# Use the model to get class predictions for the grid (for 100x100 weight-acceleration value
 combinations) and append results as new column in grid df
grid$pred<-predict(f, grid)$class

str(grid)          # Check the grid data frame
```

```
## 'data.frame':    10000 obs. of  3 variables:
##  $ weight      : num  1649 1684 1720 1755 1790 ...
##  $ acceleration: num  8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 ...
##  $ pred        : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
##  - attr(*, "out.attrs")=List of 2
##   ..$ dim     : Named int  100 100
##   .. ..- attr(*, "names")= chr  "weight" "acceleration"
##   ..$ dimnames:List of 2
##   .. ..$ weight      : chr  "weight=1649.000" "weight=1684.263" "weight=1719.525" "weight=
1754.788" ...
##   .. ..$ acceleration: chr  "acceleration= 8.500000" "acceleration= 8.638384" "acceleratio
n= 8.776768" "acceleration= 8.915152" ...
```

```
# Draw t1he required plot using the data in the grid data frame
ggplot(data=grid, aes(x=weight, y=acceleration, colour=pred)) +
  geom_point(size=0.5)
```



What is the test error of the model obtained?

```
pred<-predict(f, Atest)$class                 # Get predicted classes for test
confmat<-table(Atest$mpg01, pred)             # Construct the confusion matrix
confmat
```

```
##    pred
##      0  1
##   0 72 21
##   1  4 99
```

```
paste(round((confmat["0","1"]+confmat["1","0"])/sum(confmat)*100, 2),"%",sep="")
```

```
## [1] "12.76%"
```

c. Repeat (b) using quadratic discriminant analysis.

```
f<-qda(mpg01 ~ weight + acceleration, data=Atrain)       # Fit the model
f
```

```
## Call:
## qda(mpg01 ~ weight + acceleration, data = Atrain)
##
## Prior probabilities of groups:
##         0         1
## 0.5255102 0.4744898
##
## Group means:
##     weight acceleration
## 0 3636.359     14.49806
## 1 2404.151     16.43226
```

```
pred<-predict(f)$class                        # Get predicted classes
confmat<-table(Atrain$mpg01, pred)            # Construct the confusion matrix
confmat
```

```
##    pred
##      0  1
##   0 88 15
##   1  5 88
```

```
# Calculate the overall training error rate
paste(round((confmat["0","1"]+confmat["1","0"])/sum(confmat)*100, 2),"%",sep="")
```

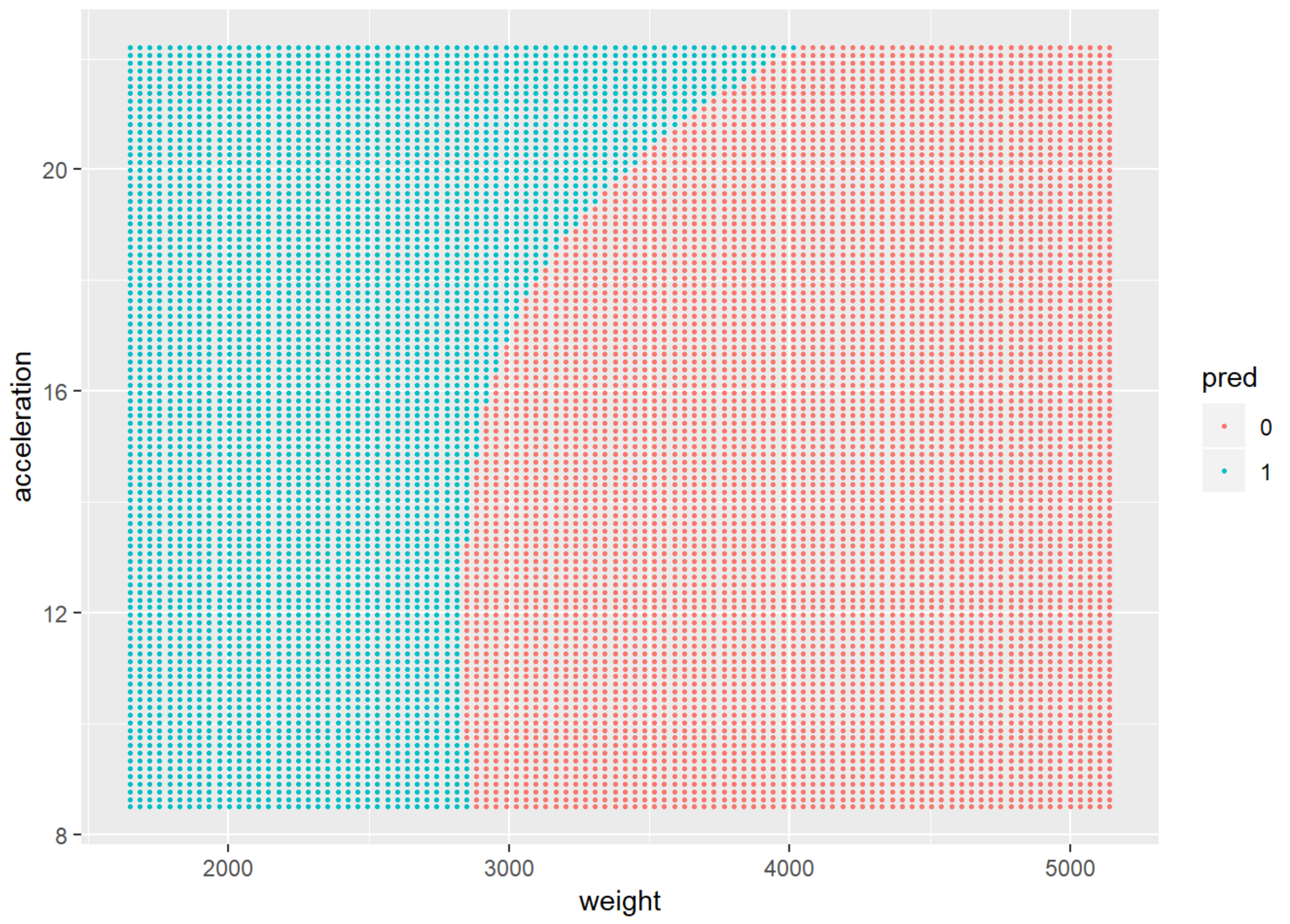
```
## [1] "10.2%"
```

Use a plot to show the discriminant boundaries.

```
# Create a grid representing a range of values for both predictors
grid<-expand.grid(weight=seq(min(Atrain$weight), max(Atrain$weight), length=100),
                  acceleration=seq(min(Atrain$acceleration),
                                   max(Atrain$acceleration),
                                   length=100))

# Use the model to get class predictions for the grid (for 100x100 weight-acceleration value
 combinations) and append results as new column in grid df
grid$pred<-predict(f, grid)$class

str(grid)          # Check the grid data frame
```

```
## 'data.frame':    10000 obs. of  3 variables:
##  $ weight      : num  1649 1684 1720 1755 1790 ...
##  $ acceleration: num  8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 ...
##  $ pred        : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
##  - attr(*, "out.attrs")=List of 2
##   ..$ dim     : Named int  100 100
##   .. ..- attr(*, "names")= chr  "weight" "acceleration"
##   ..$ dimnames:List of 2
##   .. ..$ weight      : chr  "weight=1649.000" "weight=1684.263" "weight=1719.525" "weight=
1754.788" ...
##   .. ..$ acceleration: chr  "acceleration= 8.500000" "acceleration= 8.638384" "acceleratio
n= 8.776768" "acceleration= 8.915152" ...
```

```
# Draw t1he required plot using the data in the grid data frame
ggplot(data=grid, aes(x=weight, y=acceleration, colour=pred)) +
  geom_point(size=0.5)
```



What is the test error of the model obtained?

```
pred<-predict(f, Atest)$class                # Get predicted classes for test
confmat<-table(Atest$mpg01, pred)            # Construct the confusion matrix
confmat
```

```
##    pred
##       0   1
##   0  71  22
##   1   2 101
```

```
paste(round((confmat["0","1"]+confmat["1","0"])/sum(confmat)*100, 2),"%",sep="")
```

```
## [1] "12.24%"
```

Which is better, LDA or QDA?

In this case, QDA performed better than LDA because it achieved the lower error rate on the test set (and on the training set too).

Coincidentally, QDA achieved the same error rate on the test set as LDA achieved on the training set.

d. Perform a linear discriminant analysis to predict mpg01, using variables displacement, horsepower, weight and acceleration on the training set.

```
f<-lda(mpg01 ~ displacement + horsepower + weight + acceleration, data=Atrain)
f                                          # The fitted model
```

```
## Call:
## lda(mpg01 ~ displacement + horsepower + weight + acceleration,
##     data = Atrain)
##
## Prior probabilities of groups:
##         0         1
## 0.5255102 0.4744898
##
## Group means:
##   displacement horsepower   weight acceleration
## 0     272.6214  131.69903 3636.359     14.49806
## 1     123.9140   80.11828 2404.151     16.43226
##
## Coefficients of linear discriminants:
##                        LD1
## displacement -0.0055437546
## horsepower   -0.0039716637
## weight       -0.0009141395
## acceleration  0.0086776698
```

```
pred<-predict(f)$class                     # Get the predicted classes
confmat<-table(Atrain$mpg01, pred)         # Construct the confusion matrix
confmat
```

```
##    pred
##      0  1
##   0 87 16
##   1  5 88
```

```
# Calculate the overall training error rate
paste(round((confmat["0","1"]+confmat["1","0"])/sum(confmat)*100, 2), "%", sep="")
```

```
## [1] "10.71%"
```

What is the test error of the model obtained?

```
pred<-predict(f, Atest)$class             # Get the predicted classes
confmat<-table(Atest$mpg01, pred)         # Construct the confusion matrix
confmat
```

```
##    pred
##       0   1
##   0  73  20
##   1   0 103
```

```
# Calculate the overall test error rate
paste(round((confmat["0","1"]+confmat["1","0"])/sum(confmat)*100, 2), "%", sep="")
```

```
## [1] "10.2%"
```

e. Repeat (d) using quadratic discriminant analysis.

```
f<-qda(mpg01 ~ displacement + horsepower + weight + acceleration, data=Atrain)
f                                          # The fitted model
```

```
## Call:
## qda(mpg01 ~ displacement + horsepower + weight + acceleration,
##     data = Atrain)
##
## Prior probabilities of groups:
##         0         1
## 0.5255102 0.4744898
##
## Group means:
##   displacement horsepower   weight acceleration
## 0     272.6214  131.69903 3636.359     14.49806
## 1     123.9140   80.11828 2404.151     16.43226
```

```
pred<-predict(f)$class                     # Get the predicted classes
confmat<-table(Atrain$mpg01, pred)         # Construct the confusion matrix
confmat
```

```
##    pred
##      0  1
##   0 90 13
##   1  6 87
```

```
# Calculate the overall training error rate
paste(round((confmat["0","1"]+confmat["1","0"])/sum(confmat)*100, 2), "%", sep="")
```

```
## [1] "9.69%"
```

What is the test error of the model obtained?

```
pred<-predict(f, Atest)$class             # Get the predicted classes
confmat<-table(Atest$mpg01, pred)         # Construct the confusion matrix
confmat
```

```
##    pred
##      0  1
##   0 75 18
##   1  6 97
```

```
# Calculate the overall test error rate
paste(round((confmat["0","1"]+confmat["1","0"])/sum(confmat)*100, 2), "%", sep="")
```

```
## [1] "12.24%"
```

Which is better, LDA or QDA?

In this case, LDA performed better than QDA because it achieved the lower error rate on the test set.

In fact, the test error rate by QDA in this instance is the same as the one it achieved in (d) which had fewer predictors (although the confusion matrix shows that the errors are distributed slightly differently).

f. Perform KNN with response of mpg01, and the four predictors displacement, horsepower, weight and acceleration.

Remember to scale the predictors. Use k = 5 and k = 30.

```
set.seed(123)    # Setting seed value to ensure that knn gives consistent results

# Scale the predictors for both the training and test sets
xdata<-scale(Atrain[,3:6])
means<-attr(xdata,"scaled:center")
sds<-attr(xdata,"scaled:scale")
xdataTest<-scale(Atest[,3:6], center=means, scale=sds)

# Create function to perform KNN
my_knn<-function(train, test, resp, k){
  pred<-knn(train, test, cl=resp, k=k)
  confmat<-table(resp, pred)
  return(confmat)
}

# Create a function to extract the various error rates from the specified confusion matrix
my_err<-function(confmat, pos_first=F){
  err<-(confmat[1,2]+confmat[2,1])/sum(confmat)       # Overall error rate

  if(pos_first){                                      # 1st row is the positive one
    tpr<-confmat[1,1]/sum(confmat[1,])                # True positive rate
    fpr<-confmat[2,1]/sum(confmat[2,])                # False positive rate
    fnr<-confmat[1,2]/sum(confmat[1,])                # False negative rate
    tnr<-confmat[2,2]/sum(confmat[2,])                # True negative rate
  } else{
    tpr<-confmat[2,2]/sum(confmat[2,])                # True positive rate
    fpr<-confmat[1,2]/sum(confmat[1,])                # False? positive rate
    fnr<-confmat[2,1]/sum(confmat[2,])                # False negative rate
    tnr<-confmat[1,1]/sum(confmat[1,])                # True negative rate
  }

  return(data.frame(ERR=err,FPR=fpr,TPR=tpr,FNR=fnr,TNR=tnr))   # Return error rates
}

# For k = 5
k<-5

# Perform KNN on the training set
confmat<-my_knn(xdata, xdata, Atrain$mpg01, k)
confmat
```

```
##     pred
## resp  0  1
##    0 96  7
##    1  7 86
```

```
paste("k = ", k, ": ", "Training Error Rate = ",
      round(my_err(confmat)["ERR"]*100,2),"%",sep="")
```

```
## [1] "k = 5: Training Error Rate = 7.14%"
```

```
# Perform KNN on the test set
confmat<-my_knn(xdata, xdataTest, Atrain$mpg01, k)
confmat
```

```
##     pred
## resp  0  1
##    0 43 60
##    1 42 51
```

```
paste("k = ", k, ": ", "Test Error Rate = ",
      round(my_err(confmat)["ERR"]*100,2),"%",sep="")
```

```
## [1] "k = 5: Test Error Rate = 52.04%"
```

```
# For k = 30
k<-30

# Perform KNN on the training set
confmat<-my_knn(xdata, xdata, Atrain$mpg01, k)
confmat
```

```
##     pred
## resp  0  1
##    0 87 16
##    1  7 86
```

```
paste("k = ", k, ": ", "Training Error Rate = ",
      round(my_err(confmat)["ERR"]*100,2),"%",sep="")
```

```
## [1] "k = 30: Training Error Rate = 11.73%"
```

```
# Perform KNN on the test set
confmat<-my_knn(xdata, xdataTest, Atrain$mpg01, k)
confmat
```

```
##     pred
## resp  0  1
##    0 39 64
##    1 40 53
```

```
paste("k = ", k, ": ", "Test Error Rate = ",
      round(my_err(confmat)["ERR"]*100,2),"%",sep="")
```

```
## [1] "k = 30: Test Error Rate = 53.06%"
```

Which value of k gives the best result on the test set?

k=5 gives a better result for the test set because it achieved the lower overall error rate (and for the training set too).

*** Not for Marking ***

Question 4

4. A classifier gives the following result. In the table below, Group gives the true class, and Prob gives the estimated probability of Group A (positive) using the classifier.

```
Group<-c(rep("A",6), rep("B",4))
Prob<-c(0.486,0.560,0.701,0.936,0.441,0.593,0.623,0.436,0.415,0.041)
Group_Other<-c(rep("B",6), rep("A",4))
data<-data.frame(Group, Prob, Group_Other)
data
```

```
##    Group  Prob Group_Other
## 1      A 0.486           B
## 2      A 0.560           B
## 3      A 0.701           B
## 4      A 0.936           B
## 5      A 0.441           B
## 6      A 0.593           B
## 7      B 0.623           A
## 8      B 0.436           A
## 9      B 0.415           A
## 10     B 0.041           A
```

(You can do this question in R or by hand)

a. What are the predicted classes? Use a threshold of 0.5.

```
pred<-data$Group                            # Assume the classifier is perfect
misses<-which(data$Prob < 0.5)             # Identify the classifier misses
pred[misses]<-data$Group_Other[misses]      # Select the other group
pred
```

```
##  [1] B A A A B A B A A A
## Levels: A B
```

b. What is the error rate?

```
confmat<-table(data$Group, pred)                  # Create the confusion matrix
confmat                                           # Display the confusion matrix
```

```
##    pred
##     A B
##   A 4 2
##   B 3 1
```

```
paste("Error Rate = ",round(my_err(confmat,T)["ERR"]*100,2),"%",sep="")
```

```
## [1] "Error Rate = 50%"
```

What is the false positive rate?

```
paste("False Positive Rate = ",round(my_err(confmat,T)["FPR"]*100,2),"%",sep="")
```

```
## [1] "False Positive Rate = 75%"
```

The true positive rate?

```
paste("True Positive Rate = ",round(my_err(confmat, T)["TPR"]*100,2),"%",sep="")
```

```
## [1] "True Positive Rate = 66.67%"
```

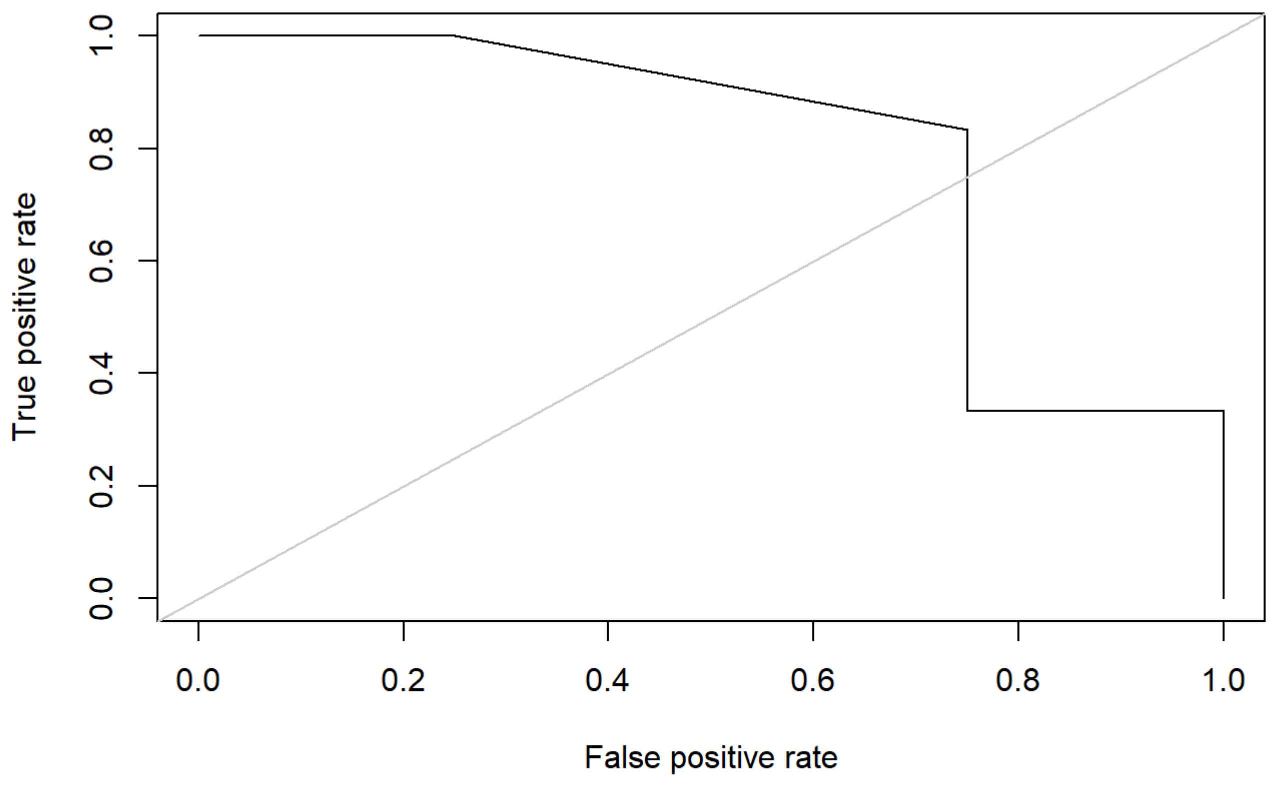
c. Now let the threshold take values 0, .2, .4,.6,.8,1. For each threshold calculate the false positive rate, and the true positive rate. (If doing this in R use more thresholds.)

```
# Create a function to calculate all error rates for the specified threshold level
t_err<-function(data, t){
  pred<-data$Group                              # Assume the classifier is perfect
  misses<-which(data$Prob < t)                  # Identify the classifier misses
  pred[misses]<-data$Group_Other[misses]        # Select the other group

  confmat<-table(data$Group, pred)              # Create the confusion matrix
  return(data.frame(Threshold=t, my_err(confmat, T)))
}

t<-seq(0, 1, 0.05)                  # Generate a sequence of threshold values

df<-t_err(data, t[1])               # Initialise the error rate details data frame

for(i in 2:length(t)){              # Loop for all threshold values
    df<-rbind(df,                   # Retrieve and append the other error details
              t_err(data, t[i]))
}

df                                  # Display the error rate details
```

```
##    Threshold ERR  FPR       TPR       FNR  TNR
## 1       0.00 0.0 0.00 1.0000000 0.0000000 1.00
## 2       0.05 0.1 0.25 1.0000000 0.0000000 0.75
## 3       0.10 0.1 0.25 1.0000000 0.0000000 0.75
## 4       0.15 0.1 0.25 1.0000000 0.0000000 0.75
## 5       0.20 0.1 0.25 1.0000000 0.0000000 0.75
## 6       0.25 0.1 0.25 1.0000000 0.0000000 0.75
## 7       0.30 0.1 0.25 1.0000000 0.0000000 0.75
## 8       0.35 0.1 0.25 1.0000000 0.0000000 0.75
## 9       0.40 0.1 0.25 1.0000000 0.0000000 0.75
## 10      0.45 0.4 0.75 0.8333333 0.1666667 0.25
## 11      0.50 0.5 0.75 0.6666667 0.3333333 0.25
## 12      0.55 0.5 0.75 0.6666667 0.3333333 0.25
## 13      0.60 0.7 0.75 0.3333333 0.6666667 0.25
## 14      0.65 0.8 1.00 0.3333333 0.6666667 0.00
## 15      0.70 0.8 1.00 0.3333333 0.6666667 0.00
## 16      0.75 0.9 1.00 0.1666667 0.8333333 0.00
## 17      0.80 0.9 1.00 0.1666667 0.8333333 0.00
## 18      0.85 0.9 1.00 0.1666667 0.8333333 0.00
## 19      0.90 0.9 1.00 0.1666667 0.8333333 0.00
## 20      0.95 1.0 1.00 0.0000000 1.0000000 0.00
## 21      1.00 1.0 1.00 0.0000000 1.0000000 0.00
```

d. Plot the true positive rate versus the false positive rate. This is the ROC curve.

```
plot(df$FPR, df$TPR, col="black", type="l", ylim=c(0,1),
     xlab="False positive rate", ylab="True positive rate")
abline(0,1,col="grey80")
```



e. (Optional, if doing in R) Another classifier just assigns class probabilities randomly, ie the estimated probabilities are:

```
Group<-c(rep("A",6), rep("B",4))
Prob<-c(0.206,0.177,0.687,0.384,0.770,0.498,0.718,0.992,0.380,0.777)
Group_Other<-c(rep("B",6), rep("A",4))
data_e<-data.frame(Group, Prob, Group_Other)
data_e
```

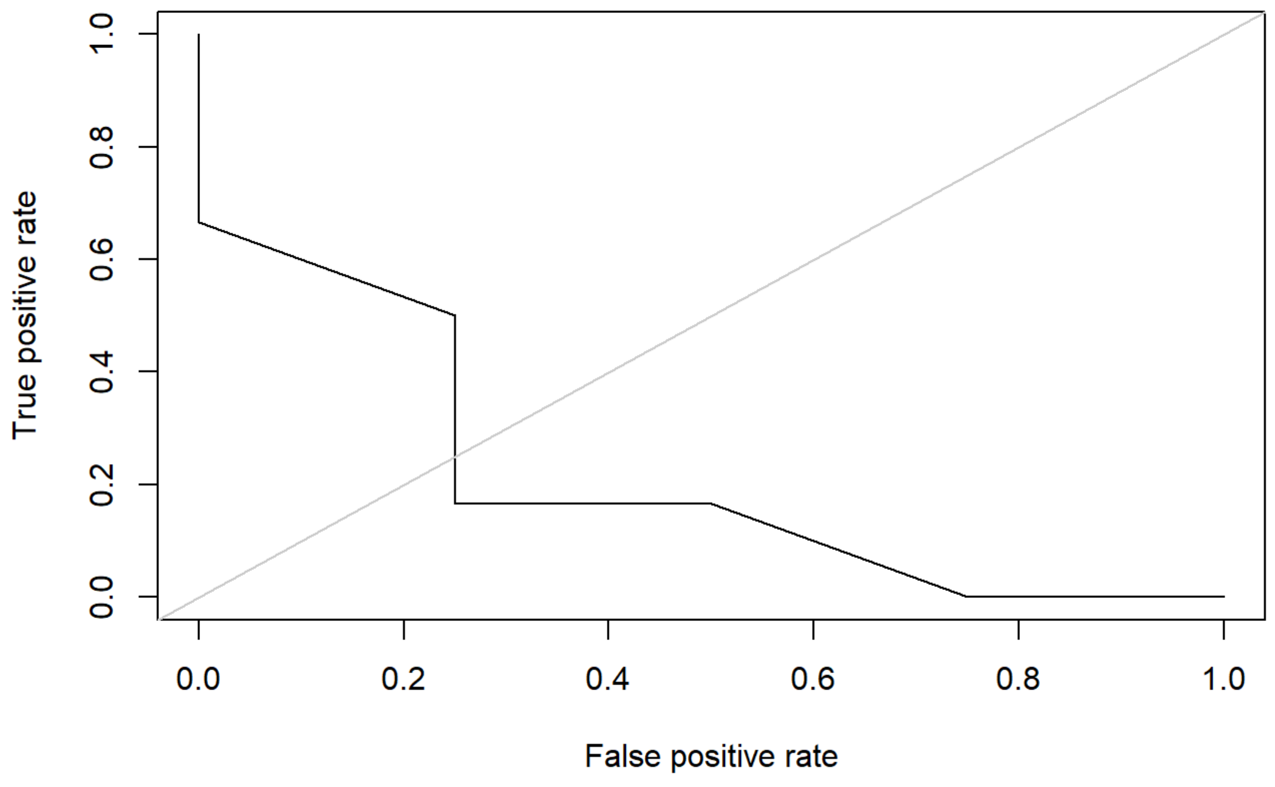
```
##    Group  Prob Group_Other
## 1      A 0.206           B
## 2      A 0.177           B
## 3      A 0.687           B
## 4      A 0.384           B
## 5      A 0.770           B
## 6      A 0.498           B
## 7      B 0.718           A
## 8      B 0.992           A
## 9      B 0.380           A
## 10     B 0.777           A
```

Plot the ROC curve for this classifier.

```
t<-seq(0, 1, 0.05)                  # Generate a sequence of threshold values

df<-t_err(data_e, t[1])             # Initialise the error rate details data frame

for(i in 2:length(t)){              # Loop for all other threshold values
    df<-rbind(df,                   # Retrieve and append the other error details
              t_err(data_e, t[i]))
}

df                                  # Display the error rate details
```

```
##    Threshold ERR  FPR       TPR       FNR  TNR
## 1       0.00 0.0 0.00 1.0000000 0.0000000 1.00
## 2       0.05 0.0 0.00 1.0000000 0.0000000 1.00
## 3       0.10 0.0 0.00 1.0000000 0.0000000 1.00
## 4       0.15 0.0 0.00 1.0000000 0.0000000 1.00
## 5       0.20 0.1 0.00 0.8333333 0.1666667 1.00
## 6       0.25 0.2 0.00 0.6666667 0.3333333 1.00
## 7       0.30 0.2 0.00 0.6666667 0.3333333 1.00
## 8       0.35 0.2 0.00 0.6666667 0.3333333 1.00
## 9       0.40 0.4 0.25 0.5000000 0.5000000 0.75
## 10      0.45 0.4 0.25 0.5000000 0.5000000 0.75
## 11      0.50 0.5 0.25 0.3333333 0.6666667 0.75
## 12      0.55 0.5 0.25 0.3333333 0.6666667 0.75
## 13      0.60 0.5 0.25 0.3333333 0.6666667 0.75
## 14      0.65 0.5 0.25 0.3333333 0.6666667 0.75
## 15      0.70 0.6 0.25 0.1666667 0.8333333 0.75
## 16      0.75 0.7 0.50 0.1666667 0.8333333 0.50
## 17      0.80 0.9 0.75 0.0000000 1.0000000 0.25
## 18      0.85 0.9 0.75 0.0000000 1.0000000 0.25
## 19      0.90 0.9 0.75 0.0000000 1.0000000 0.25
## 20      0.95 0.9 0.75 0.0000000 1.0000000 0.25
## 21      1.00 1.0 1.00 0.0000000 1.0000000 0.00
```

```
plot(df$FPR, df$TPR, col="black", type="l", ylim=c(0,1),
     xlab="False positive rate", ylab="True positive rate")
abline(0,1,col="grey80")
```



Question 5

Dataset on diabetes in Pima Indian Women in library(MASS). For a description of the data see ?Pima.tr.

```
str(Pima.tr)
```

```
## 'data.frame':    200 obs. of  8 variables:
##  $ npreg: int  5 7 5 0 0 5 3 1 3 2 ...
##  $ glu  : int  86 195 77 165 107 97 83 193 142 128 ...
##  $ bp   : int  68 70 82 76 60 76 58 50 80 78 ...
##  $ skin : int  28 33 41 43 25 27 31 16 15 37 ...
##  $ bmi  : num  30.2 25.1 35.8 47.9 26.4 35.6 34.3 25.9 32.4 43.3 ...
##  $ ped  : num  0.364 0.163 0.156 0.259 0.133 ...
##  $ age  : int  24 55 35 26 23 52 25 24 63 31 ...
##  $ type : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 1 1 2 ...
```

```
str(Pima.te)
```

```
## 'data.frame':    332 obs. of  8 variables:
##  $ npreg: int  6 1 1 3 2 5 0 1 3 9 ...
##  $ glu  : int  148 85 89 78 197 166 118 103 126 119 ...
##  $ bp   : int  72 66 66 50 70 72 84 30 88 80 ...
##  $ skin : int  35 29 23 32 45 19 47 38 41 35 ...
##  $ bmi  : num  33.6 26.6 28.1 31 30.5 25.8 45.8 43.3 39.3 29 ...
##  $ ped  : num  0.627 0.351 0.167 0.248 0.158 0.587 0.551 0.183 0.704 0.263 ...
##  $ age  : int  50 31 21 26 53 51 31 33 27 29 ...
##  $ type : Factor w/ 2 levels "No","Yes": 2 1 1 2 2 2 2 1 1 2 ...
```

Use any supervised classification technique to predict diabetes from the 7 available features.

Train your algorithms on Pima.tr

Will use logistic regression to build the prediction model.

Using the stepwise logistic regression (step function) to recommend the ‘optimal’ predictor selection…..

```
# Fit a model with all (7) predictors
f <- glm(type ~ ., family = binomial, data = Pima.tr)
summary(f)
```

```
##
## Call:
## glm(formula = type ~ ., family = binomial, data = Pima.tr)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9830  -0.6773  -0.3681   0.6439   2.3154
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.773062   1.770386  -5.520 3.38e-08 ***
## npreg        0.103183   0.064694   1.595  0.11073
## glu          0.032117   0.006787   4.732 2.22e-06 ***
## bp          -0.004768   0.018541  -0.257  0.79707
## skin        -0.001917   0.022500  -0.085  0.93211
## bmi          0.083624   0.042827   1.953  0.05087 .
## ped          1.820410   0.665514   2.735  0.00623 **
## age          0.041184   0.022091   1.864  0.06228 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 256.41  on 199  degrees of freedom
## Residual deviance: 178.39  on 192  degrees of freedom
## AIC: 194.39
##
## Number of Fisher Scoring iterations: 5
```

```
# Perform stepwise model fitting (direction=backward by default)
steps<-step(f, trace=F)

# Summarise the predictors that are recommended to be dropped
steps$anova
```

```
##     Step Df    Deviance Resid. Df Resid. Dev      AIC
## 1        NA          NA       192   178.3907 194.3907
## 2 - skin  1 0.007227569       193   178.3979 192.3979
## 3   - bp  1 0.072624742       194   178.4705 190.4705
```

```
# Doublechecking using the stepAIC function in the MASS library....
stepAICs<-stepAIC(f, trace=F)

# Summarise the predictors that are recommended to be dropped
stepAICs$anova
```

```
## Stepwise Model Path
## Analysis of Deviance Table
##
## Initial Model:
## type ~ npreg + glu + bp + skin + bmi + ped + age
##
## Final Model:
## type ~ npreg + glu + bmi + ped + age
##
##
##     Step Df    Deviance Resid. Df Resid. Dev      AIC
## 1                             192   178.3907 194.3907
## 2 - skin  1 0.007227569       193   178.3979 192.3979
## 3   - bp  1 0.072624742       194   178.4705 190.4705
```

Both the step and stepAIC functions recommend omission of the skin and bp predictors - which reduces the AIC value from 194.39 to 190.47 (because omitting other predictors failed to reduce the AIC value any further).

As it happens, those are also the two variables with the highest p-values in the summary(f) output above.

```
# Create function to create the confusion matrix for the specified fit
my_confmat<-function(f, data, resp){
  prob<-predict(f, data, type="response")             # Get the predicted probs.
  pred<-factor(ifelse(prob < 0.5, "No", "Yes"))      # Translate probs. to preds.

  confmat<-table(resp, pred)                          # Create the confusion matrix
  return(confmat)                                     # Return the confusion matrix
}

f<-glm(type ~ npreg + glu + bmi + ped + age, family=binomial, data=Pima.tr)

confmat<-my_confmat(f, Pima.tr, Pima.tr$type)
confmat
```

```
##      pred
## resp   No Yes
##   No  117  15
##   Yes  29  39
```

```
paste("Error Rate = ",round(my_err(confmat)["ERR"]*100,2),"%",sep="")
```

```
## [1] "Error Rate = 22%"
```

and present the overall error rate for the test data Pima.te.

```
confmat<-my_confmat(f, Pima.te, Pima.te$type)
confmat
```

```
##      pred
## resp   No Yes
##   No  199  24
##   Yes  42  67
```

```
paste("Error Rate = ",round(my_err(confmat)["ERR"]*100,2),"%",sep="")
```

```
## [1] "Error Rate = 19.88%"
```

Question 6

Generate some fake data using the following code:

```
set.seed(1)
x <- rnorm(100)
y <- 1 + .2*x+3*x^2+.6*x^3 + rnorm(100)
d <- data.frame(x=x,y=y)

str(d)
```

```
## 'data.frame':    100 obs. of  2 variables:
##  $ x: num  -0.626 0.184 -0.836 1.595 0.33 ...
##  $ y: num  1.284 1.184 1.667 11.548 0.759 ...
```

a. Use best subset selection to choose the best model containing predictors X, X^2,. . .X^{10}.

```
for(i in 2:10){
  d<-cbind(d, d$x^i)
  colnames(d)[i+1]<-paste("x",i,sep="")
}

str(d)
```

```
## 'data.frame':    100 obs. of  11 variables:
##  $ x  : num  -0.626 0.184 -0.836 1.595 0.33 ...
##  $ y  : num  1.284 1.184 1.667 11.548 0.759 ...
##  $ x2 : num  0.3924 0.0337 0.6983 2.5449 0.1086 ...
##  $ x3 : num  -0.24585 0.00619 -0.5835 4.05986 0.03578 ...
##  $ x4 : num  0.15401 0.00114 0.48759 6.47662 0.01179 ...
##  $ x5 : num  -0.096482 0.000209 -0.407443 10.332031 0.003884 ...
##  $ x6 : num  6.04e-02 3.84e-05 3.40e-01 1.65e+01 1.28e-03 ...
##  $ x7 : num  -3.79e-02 7.04e-06 -2.85e-01 2.63e+01 4.22e-04 ...
##  $ x8 : num  2.37e-02 1.29e-06 2.38e-01 4.19e+01 1.39e-04 ...
##  $ x9 : num  -1.49e-02 2.38e-07 -1.99e-01 6.69e+01 4.58e-05 ...
##  $ x10: num  9.31e-03 4.36e-08 1.66e-01 1.07e+02 1.51e-05 ...
```

```
allfits<-regsubsets(y ~ ., data=d)

allfits_s<-summary(allfits)
```

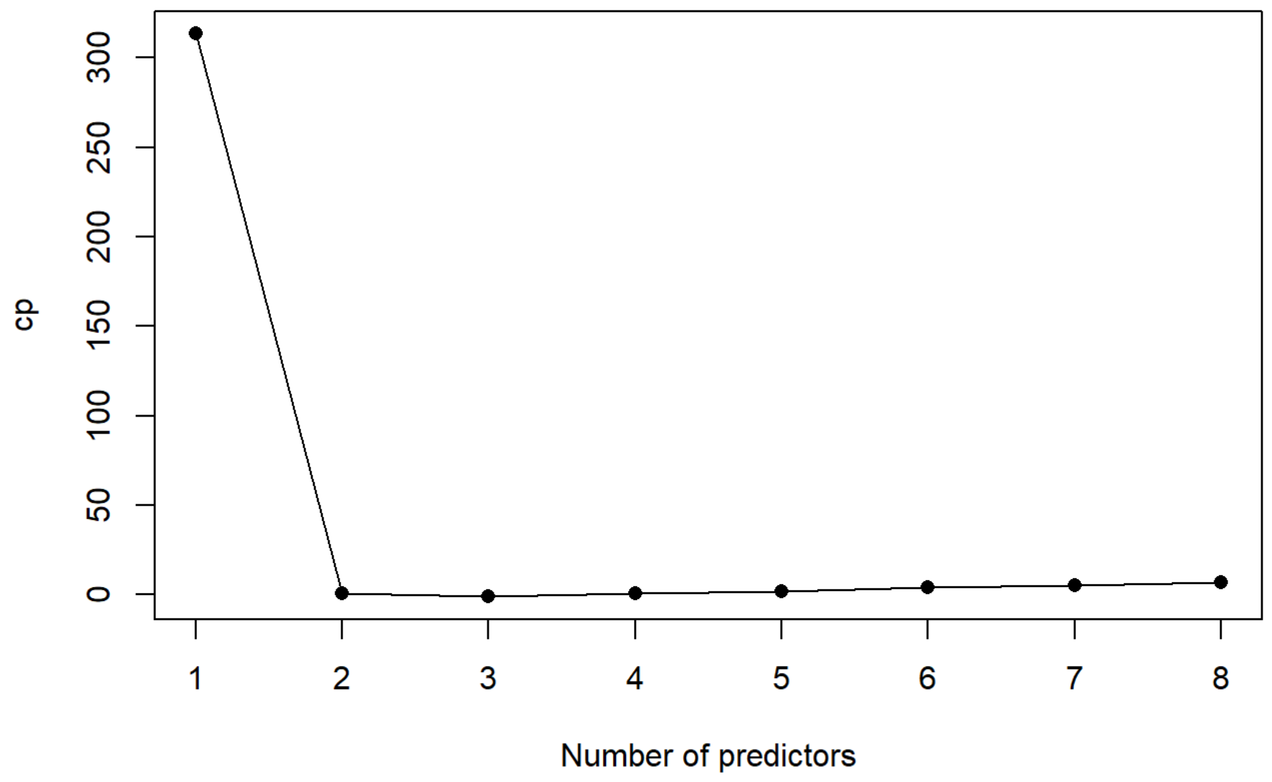
Which terms are included in the best 3 variable model?

```
terms<-attr(which(summary(allfits)$which[3,]),"names")
terms[2:length(terms)]
```

```
## [1] "x"  "x2" "x5"
```

b. Make a plot of C_p versus number of predictors for the models in allfits.

```
cp<-allfits_s$cp                  # Extract the Cp values for all stepwise models
np<-rowSums(allfits_s$which)-1    # Extract the number of predictors for each too

# Draw the required plot
plot(np, cp, pch=16, xlab="Number of predictors")
lines(np, cp)
```



Which model has the lowest C_p?

```
min_cp<-which(cp==min(cp))
min_cp
```

```
## [1] 3
```

What are its predictors?

```
names(which(allfits_s$which[min_cp,]))[-1]
```

```
## [1] "x"  "x2" "x5"
```

c. Reconstruct allfits with option method = “forward”.

```
allfits<-regsubsets(y ~ ., data=d, method="forward")

allfits_s<-summary(allfits)

cp<-allfits_s$cp                  # Extract the Cp values for all stepwise models
np<-rowSums(allfits_s$which)-1   # Extract the number of predictors for each too
```

Which model has the lowest C_p?

```
min_cp<-which(allfits_s$cp==min(cp))
min_cp
```

```
## [1] 2
```

What are its predictors?

```
names(which(allfits_s$which[min_cp,]))[-1]
```

```
## [1] "x2" "x3"
```

d. Reconstruct allfits with option method = “backward”.

```
allfits<-regsubsets(y ~ ., data=d, method="backward")

allfits_s<-summary(allfits)

cp<-allfits_s$cp                  # Extract the Cp values for all stepwise models
np<-rowSums(allfits_s$which)-1   # Extract the number of predictors for each too
```

Which model has the lowest C_p?

```
min_cp<-which(allfits_s$cp==min(cp))
min_cp
```

```
## [1] 3
```

What are its predictors?

```
names(which(allfits_s$which[min_cp,]))[-1]
```

```
## [1] "x"  "x2" "x5"
```