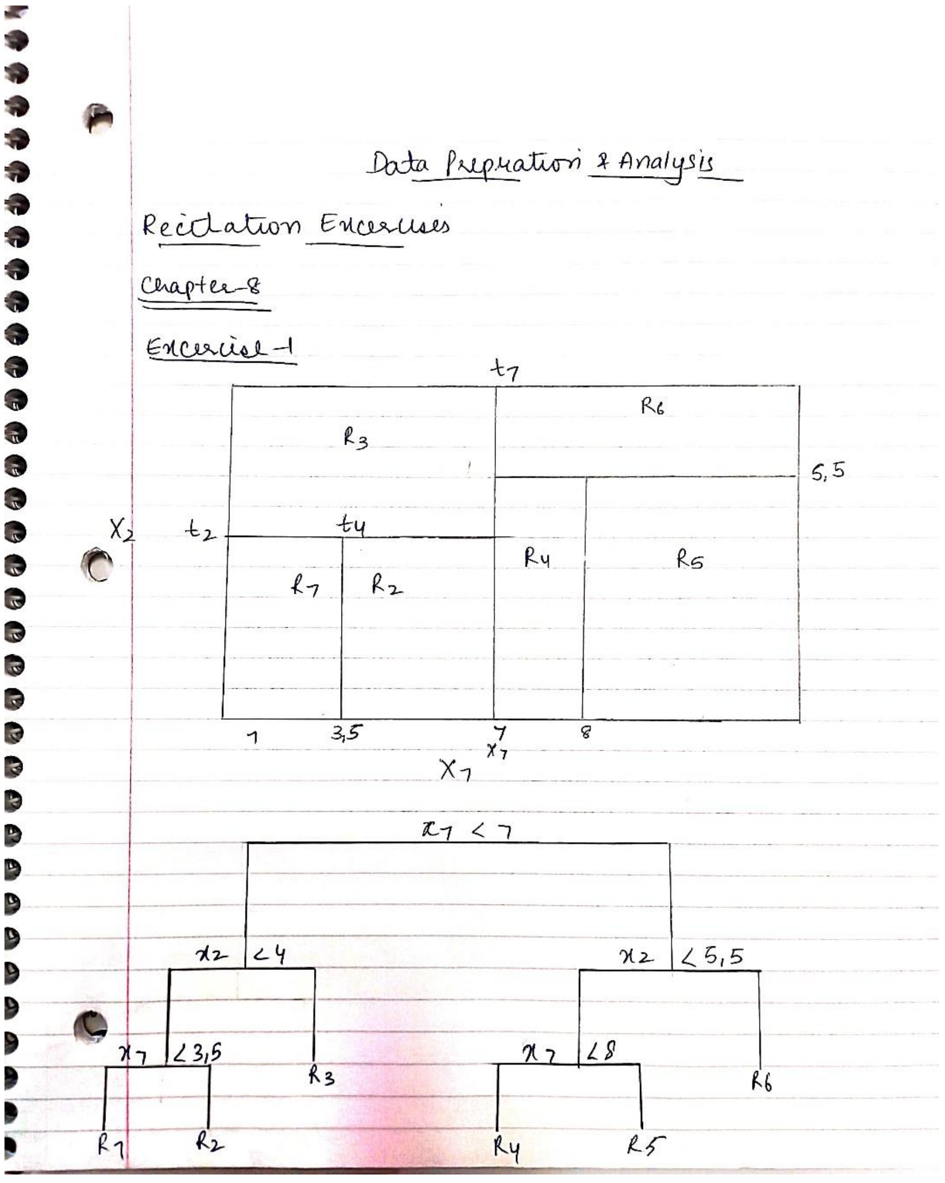
Data Preparation and Analysis (CSP571)

Assignment-4

Recitation Exercises

Chapter-8

1. **Draw an example (of your own invention) of a partition of two-dimensional feature space that could result from recursive binary splitting. Your example should contain at least six regions. Draw a decision tree corresponding to this partition. Be sure to label all aspects of your figures, including the regions R1, R2,..., the cut points t1, t2,..., and so forth.**



Data Preparation & Analysis

Recitation Excercises

Chapter-8

Excercise-1

3. Consider the Gini index, classification error, and entropy in a simple classification setting with two classes. Create a single plot that displays each of these quantities as a function of ˆpm1. The x-axis should display ˆpm1, ranging from 0 to 1, and the y-axis should display the value of the Gini index, classification error, and entropy. Hint: In a setting with two classes, pˆm1 = 1 – pˆm2. You could make this plot by hand, but it will be much easier to make in R.

```
p1 = seq(0 + 1e-06, 1 - 1e-06, length.out = 100)

p2 = 1 - p1

class_error = 1 - apply(rbind(p1, p2), 2, max)
```

gini_index = p1 * (1 - p1) + p2 * (1 - p2)

cross_entropy = -(p1 * log(p1) + p2 * log(p2))

plot(p1, class_error, type = "l", col = "black", xlab = "Pm1", ylab = "Error Metrics",

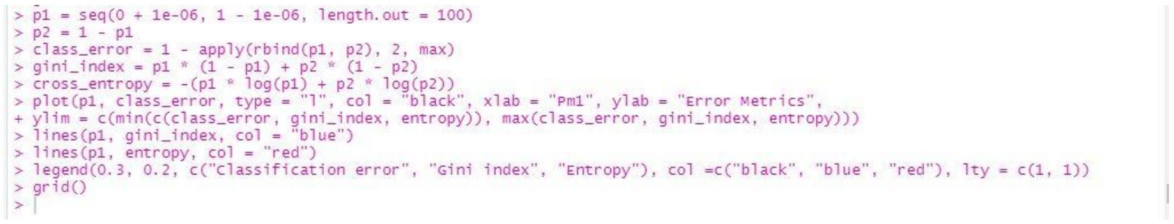
ylim = c(min(c(class_error, gini_index, entropy)), max(class_error, gini_index, entropy)))

lines(p1, gini_index, col = "blue")

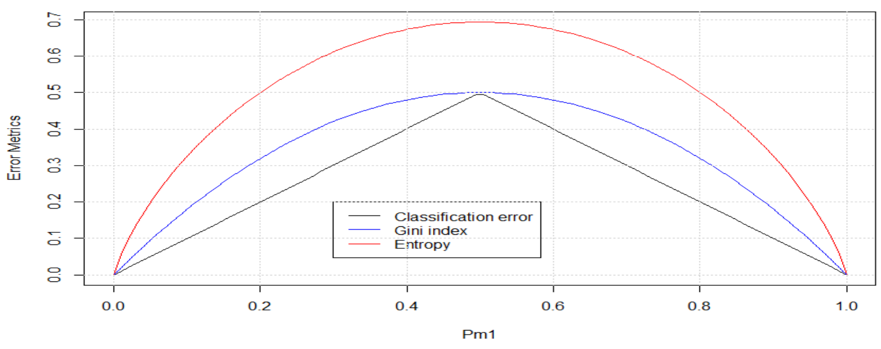
lines(p1, entropy, col = "red")

legend(0.3, 0.2, c("Classification error", "Gini index", "Entropy"), col =c("black", "blue", "red"), lty = c(1, 1))

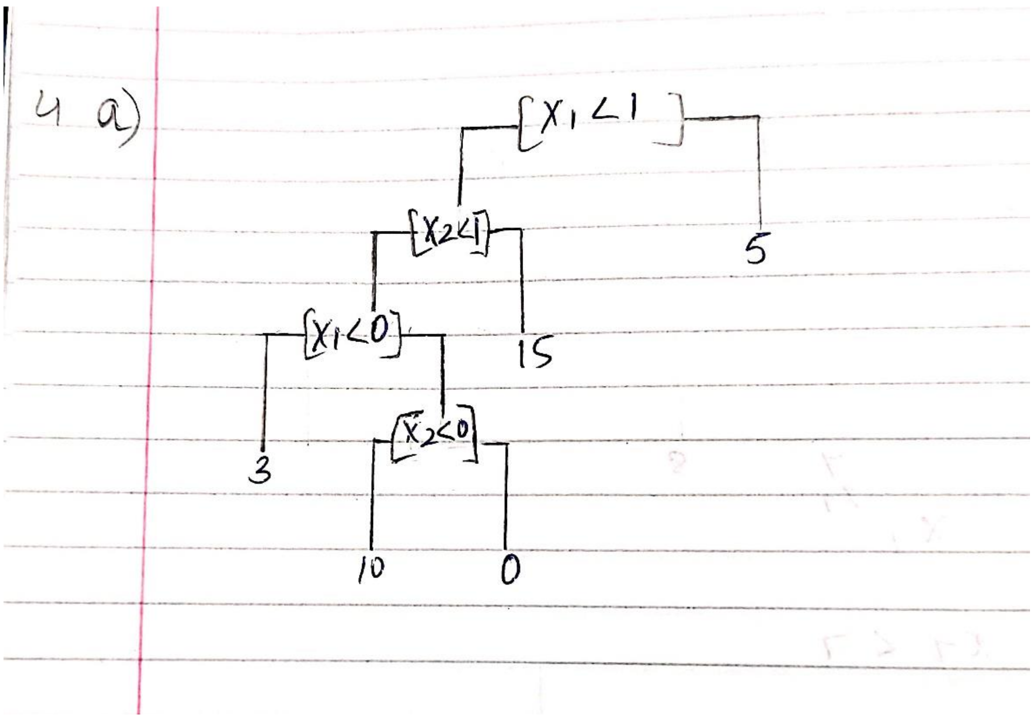
grid()



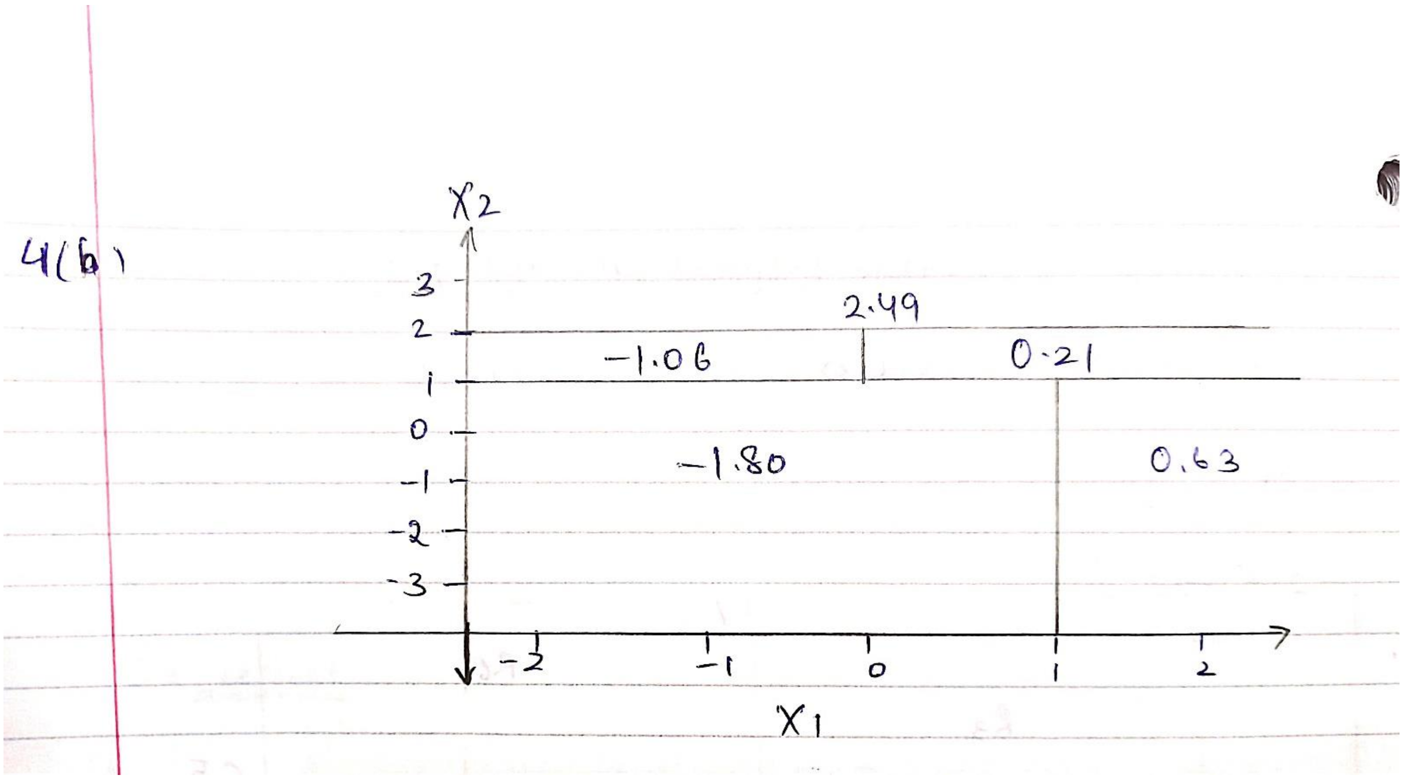
```
> p1 = seq(0 + 1e-06, 1 - 1e-06, length.out = 100)
> p2 = 1 - p1
> class_error = 1 - apply(rbind(p1, p2), 2, max)
> gini_index = p1 * (1 - p1) + p2 * (1 - p2)
> cross_entropy = -(p1 * log(p1) + p2 * log(p2))
> plot(p1, class_error, type = "l", col = "black", xlab = "Pm1", ylab = "Error Metrics",
+ ylim = c(min(c(class_error, gini_index, entropy)), max(class_error, gini_index, entropy)))
> lines(p1, gini_index, col = "blue")
> lines(p1, entropy, col = "red")
> legend(0.3, 0.2, c("Classification error", "Gini index", "Entropy"), col =c("black", "blue", "red"), lty = c(1, 1))
> grid()
> 
```



4(a). This question relates to the plots in Figure 8.14. (a) Sketch the tree corresponding to the partition of the predictor space illustrated in the left-hand panel of Figure 8.14. The numbers inside the boxes indicate the mean of Y within each region.



(b) Create a diagram like the left-hand panel of Figure 8.14, using the tree illustrated in the right-hand panel of the same figure. You should divide up the predictor space into the correct regions, and indicate the mean for each region.



5. Suppose we produce ten bootstrapped samples from a data set containing red and green classes. We then apply a classification tree to each bootstrapped sample and, for a specific value of X, produce 10 estimates of P(Class is Red|X): 0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, and 0.75. There are two common ways to combine these results together into a single class prediction. One is the majority vote approach discussed in this chapter. The second approach is to classify based on the average probability. In this example, what is the final classification under each of these two approaches?

We are given two classes in this question- Red class and Green class

Red[X]: 0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75

Different approaches through which we can combine results are:

(i) **Majority Vote approach:** We can see that out of 10 estimates, 6 estimates have p> 0.5 and 4 estimates have p < 0.5, and this suggests that the majority of estimates classify X as Red.

(ii) **Average Probability Approach:** In this approach, we will calculate the average of all the 10 estimates, and depending upon the result, the p is > 0.5 then, the class for X will be Red otherwise the class will be Green.
P = (0.1+0.15+0.2+0.2+0.55+0.6+0.6+0.65+0.7+0.75)/ 10 = 0.45 Since the p < 0.5, therefore the class for X is Green.

Chapter-9

1(a) This problem involves hyperplanes in two dimensions. (a) Sketch the hyperplane 1 + 3X1 − X2 = 0. Indicate the set of points for which 1 + 3X1 − X2 > 0, as well as the set of points for which 1 + 3X1 − X2 < 0.

(b) On the same plot, sketch the hyperplane −2 + X1 + 2X2 = 0. Indicate the set of points for which −2 + X1 + 2X2 > 0, as well as the set of points for which −2 + X1 + 2X2 < 0.

```
x1 <- -10:10

x2 <- 1 + 3 * x1
```

plot(x1, x2, type = "l", col = "blue")

text(c(0), c(-20), "Greater than 0", col = "blue")

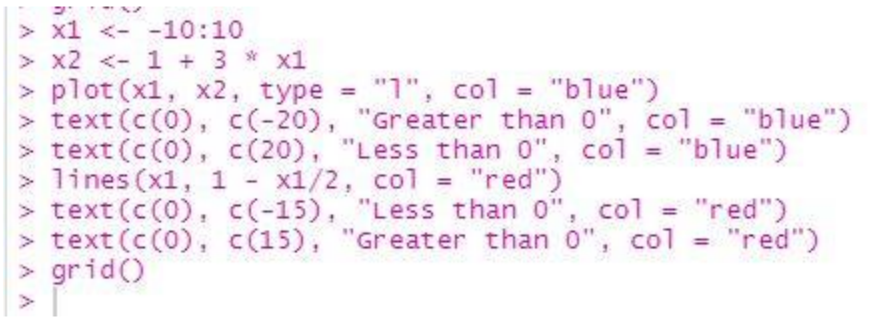
text(c(0), c(20), "Less than 0", col = "blue")

lines(x1, 1 - x1/2, col = "red")

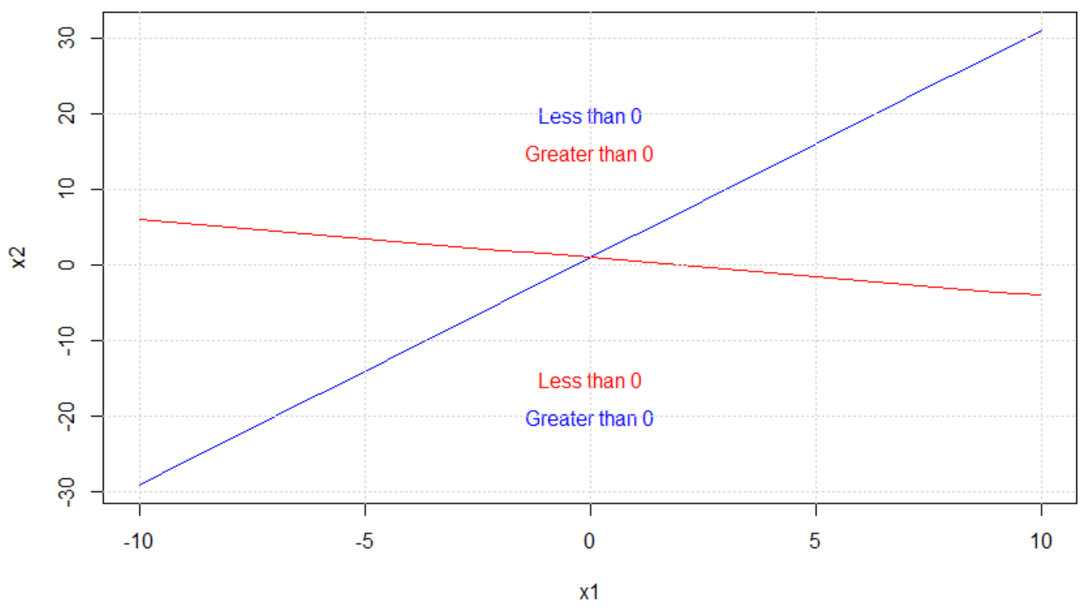
text(c(0), c(-15), "Less than 0", col = "red")

text(c(0), c(15), "Greater than 0", col = "red")

grid()



```
> x1 <- -10:10
> x2 <- 1 + 3 * x1
> plot(x1, x2, type = "l", col = "blue")
> text(c(0), c(-20), "Greater than 0", col = "blue")
> text(c(0), c(20), "Less than 0", col = "blue")
> lines(x1, 1 - x1/2, col = "red")
> text(c(0), c(-15), "Less than 0", col = "red")
> text(c(0), c(15), "Greater than 0", col = "red")
> grid()
> |
```

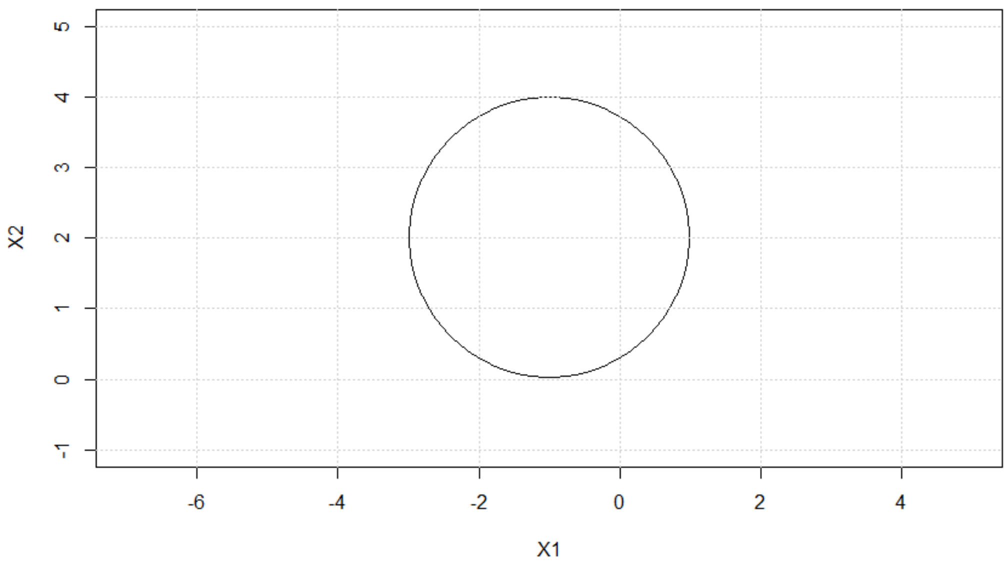


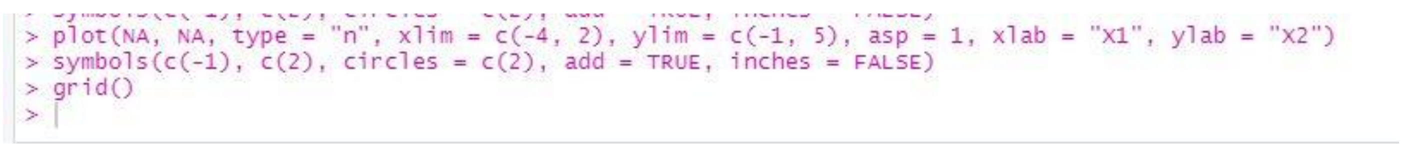
2(a) Sketch the curve (1 + X1) 2 + (2 – X2) 2 = 4

plot(NA, NA, type = "n", xlim = c(-4, 2), ylim = c(-1, 5), asp = 1, xlab = "X1", ylab = "X2")

symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE)

grid()




```
> plot(NA, NA, type = "n", xlim = c(-4, 2), ylim = c(-1, 5), asp = 1, xlab = "X1", ylab = "X2")
> symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE)
> grid()
> 
```

2(b) On your sketch, indicate the set of points for which (1 + X1) 2 + (2 − X2) 2 > 4, as well as the set of points for which (1 + X1) 2 + (2 − X2) 2 ≤ 4.

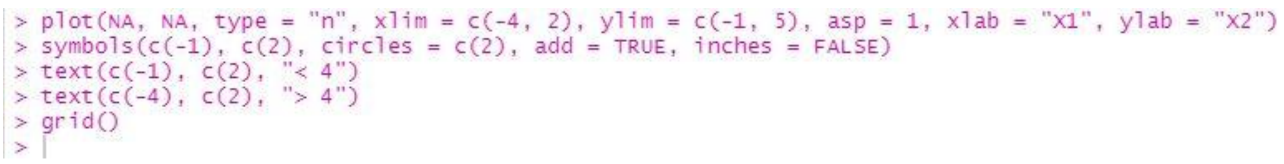
plot(NA, NA, type = "n", xlim = c(-4, 2), ylim = c(-1, 5), asp = 1, xlab = "X1", ylab = "X2")

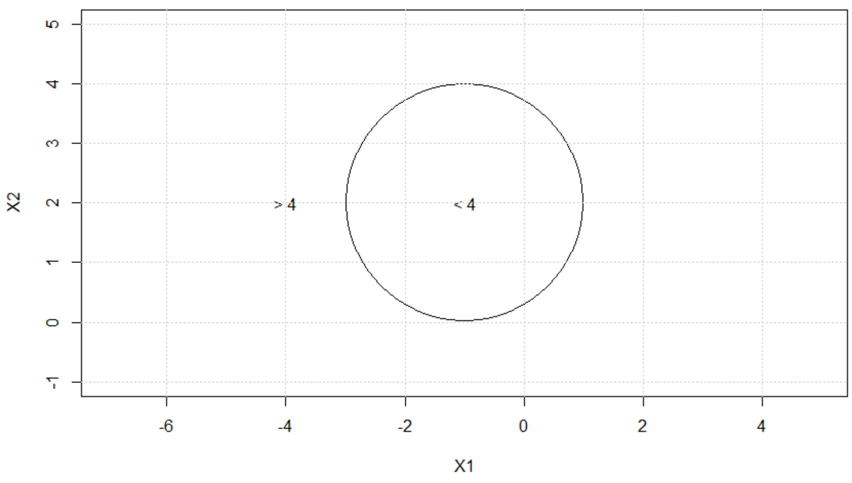
symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE)

text(c(-1), c(2), "< 4")

text(c(-4), c(2), "> 4")

grid()


```
> plot(NA, NA, type = "n", xlim = c(-4, 2), ylim = c(-1, 5), asp = 1, xlab = "X1", ylab = "X2")
> symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE)
> text(c(-1), c(2), "< 4")
> text(c(-4), c(2), "> 4")
> grid()
> 
```



(c) Suppose that a classifier assigns an observation to the blue class if (1 + X1) 2 + (2 − X2) 2 > 4, and to the red class otherwise. To what class is the observation (0, 0) classified? (−1, 1)? (2, 2)? (3, 8)?

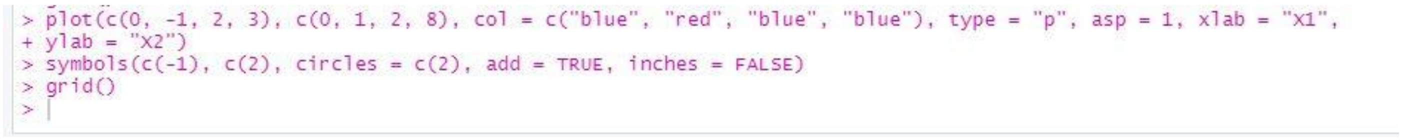
It replaces x1 and x2 by the coordinates of the points in the equation and check if the result is greater than or less than 4. For point(0,0) we have 5>4 (blue class), for point(-1,1) we have 1<4(red class), for (2,2) we have class blue with 9>4 and for (3,8) we have blue class with 52 >4.

plot(c(0, -1, 2, 3), c(0, 1, 2, 8), col = c("blue", "red", "blue", "blue"), type = "p", asp = 1, xlab = "X1",

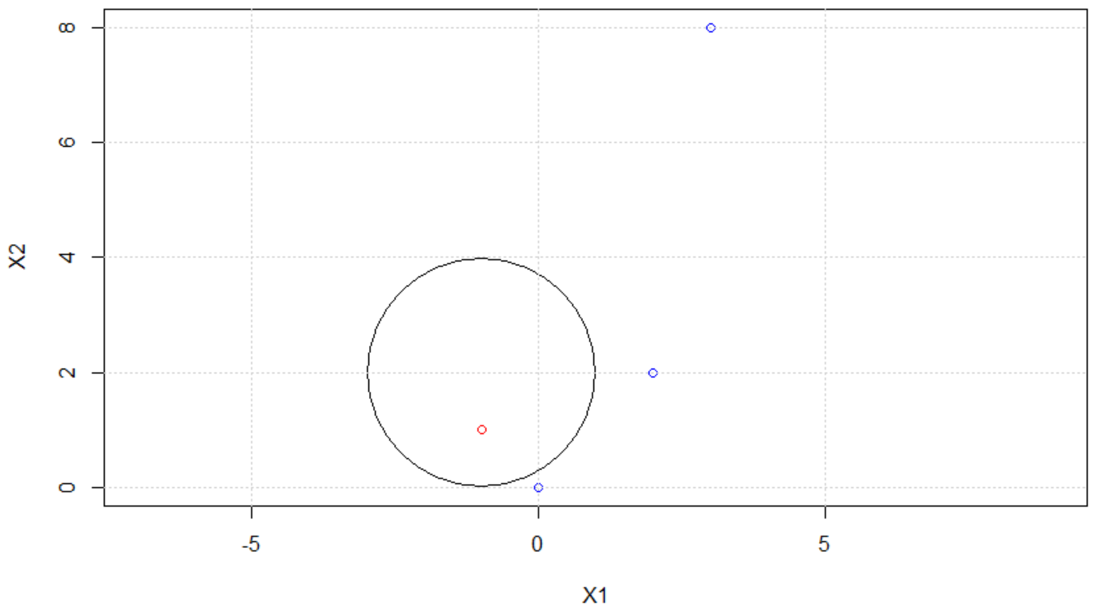
ylab = "X2")

symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE)

grid()



```
> plot(c(0, -1, 2, 3), c(0, 1, 2, 8), col = c("blue", "red", "blue", "blue"), type = "p", asp = 1, xlab = "X1",
+ ylab = "X2")
> symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE)
> grid()
>
```



(d) Argue that while the decision boundary in (c) is not linear in terms of X1 and X2, it is linear in terms of X1, X2 1, X2, and X2 ^2.

It is obvious that we can expand the equation of decision boundary.

(1+X1)^2+(2−X2)2=4(1+X1)^2+(2−X2)2=4

by

X1^2+X2^2+2X1−4X2+1=0

which is linear in terms of X1, X1^2, and X2^2.

3. Here we explore the maximal margin classifier on a toy data set.

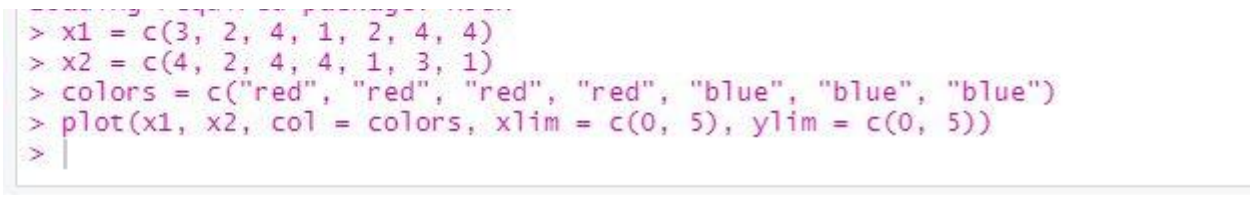
(a) We are given n = 7 observations in p = 2 dimensions. For each observation, there is an associated class label.

x1 = c(3, 2, 4, 1, 2, 4, 4)

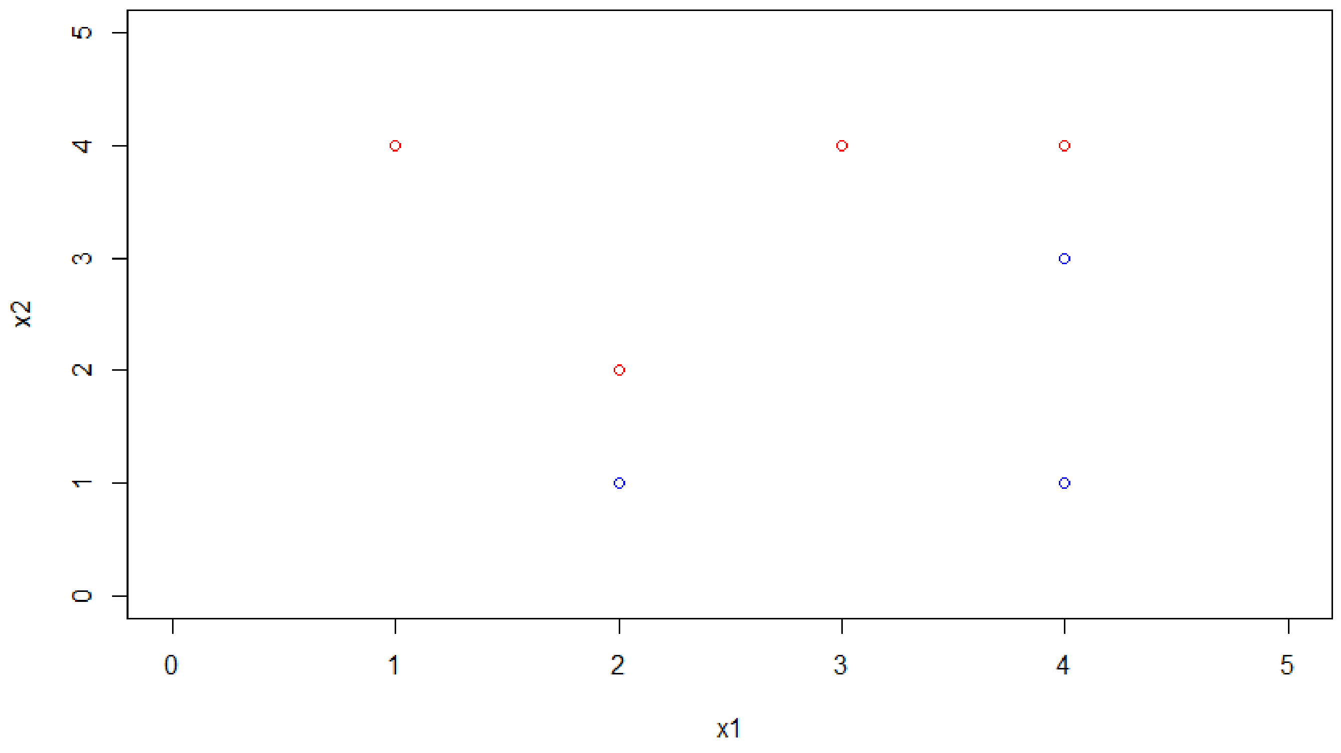
x2 = c(4, 2, 4, 4, 1, 3, 1)

colors = c("red", "red", "red", "red", "blue", "blue", "blue")

plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))



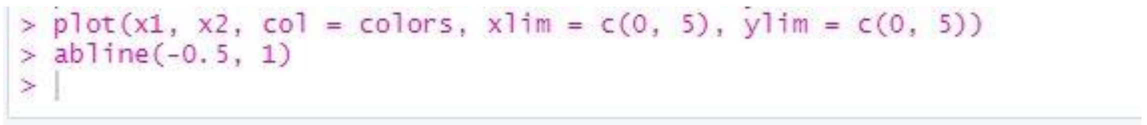
```
> x1 = c(3, 2, 4, 1, 2, 4, 4)
> x2 = c(4, 2, 4, 4, 1, 3, 1)
> colors = c("red", "red", "red", "red", "blue", "blue", "blue")
> plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
> 
```



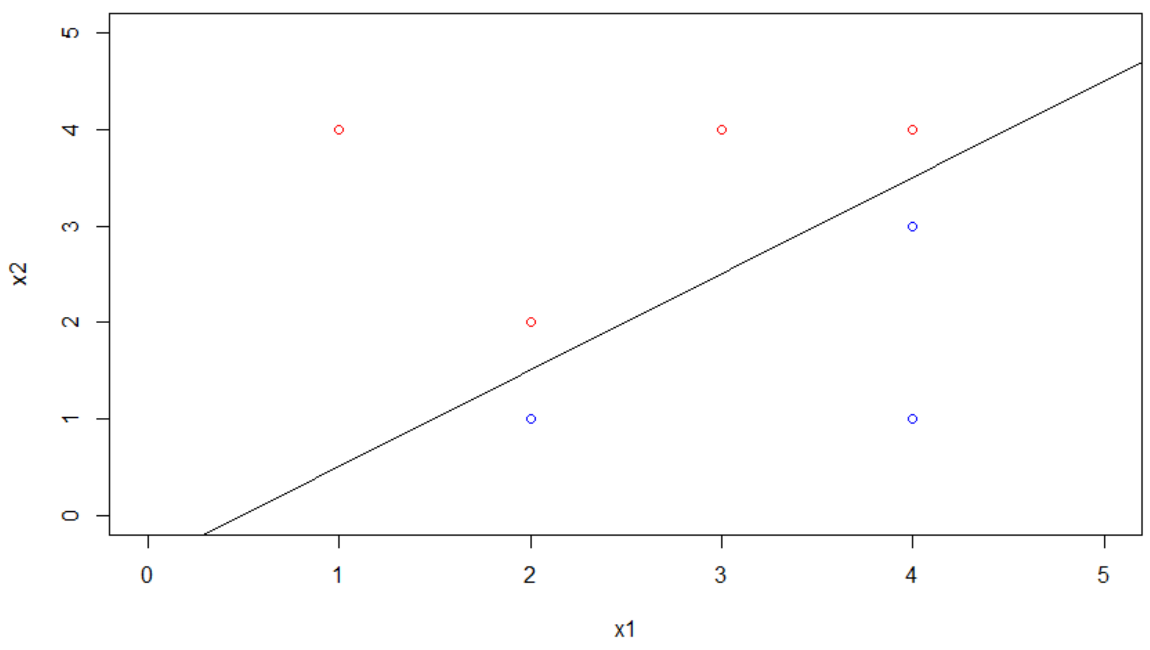
(b) Sketch the optimal separating hyperplane and provide the equation for this hyperplane (of the form (9.1)).

plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))

abline(-0.5, 1)



```
> plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
> abline(-0.5, 1)
> 
```



As shown, the optimal separating hyperplane has to be between the observations (2,1) (2,1) and (2,2) (2,2) and also between the observations(4,3) (4,3) and (4,4) (4,4). So, a line passes through the points (2, 1.5) (2, 1.5) and (4, 3.5) (4, 3.5) and hence, the equation is x1-x2-0.5=0.

(c) Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if β0 + β1X1 + β2X2 > 0 and classify to Blue otherwise." Provide the values for β0, β1, and β2

The classification rule is to "Classify to Red if x1-x2-0.5 < 0 and Classify to Blue otherwise". The values of β0, β1, and β2 are (0.5, -1,1).

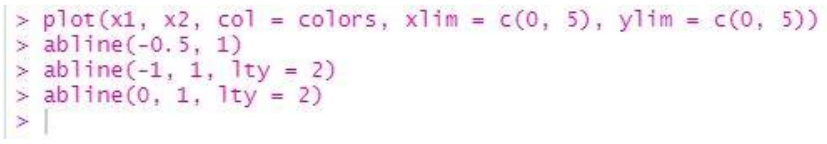
(d) On your sketch, indicate the margin for the maximal margin hyperplane.

plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))

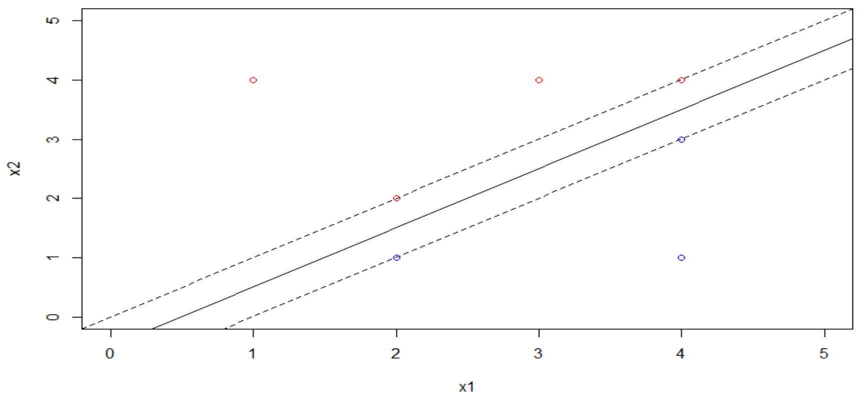
abline(-0.5, 1)

abline(-1, 1, lty = 2)

abline(0, 1, lty = 2)



```
> plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
> abline(-0.5, 1)
> abline(-1, 1, lty = 2)
> abline(0, 1, lty = 2)
> 
```



The margin here is 1/4.

(e) Indicate the support vectors for the maximal margin classifier.

The support vectors are points (2,1), (2,2), (4,3), and (4,4).

(f) Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

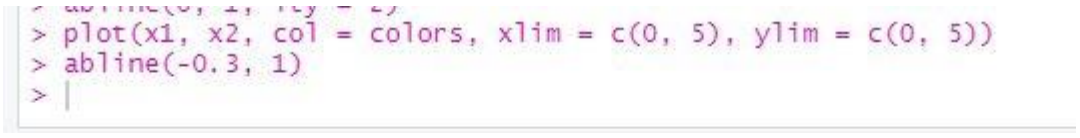
By seeing the plot, we come to know observation 7 is not a support vector. Any small movement of it in any direction is not going to change the thing that x2-x1+0.5 =0 gives us a good separation of observations with a margin of M=√2/4. In this case, observation 7 will move inside the margin to start influencing the position of the maximal margin hyperplane.

(g) Sketch a hyperplane that is not the optimal separating hyperplane and provide the equation for this hyperplane.

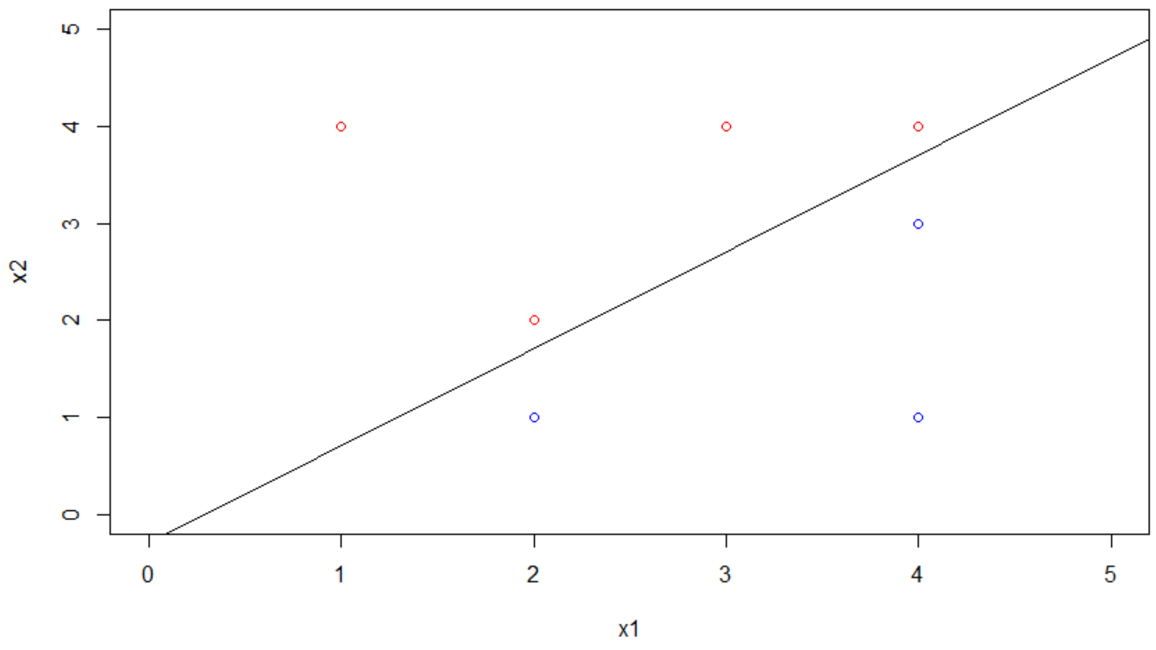
The hyperplane related to equation x1- x2- 0.3 =0 is not the optimal separating hyperplane.

plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))

abline(-0.3, 1)



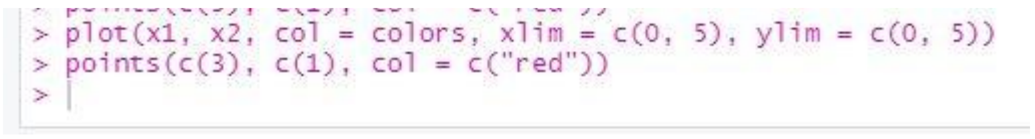
```
> plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
> abline(-0.3, 1)
> |
```



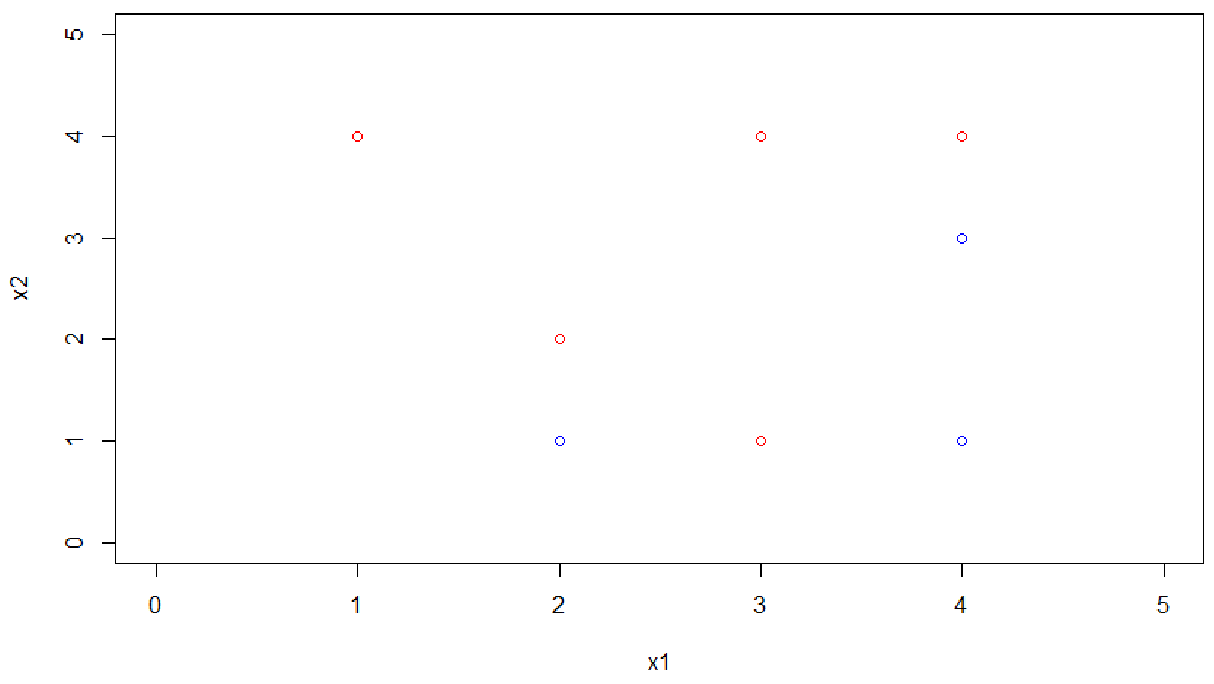
(h) Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))

points(c(3), c(1), col = c("red"))



```
> plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
> points(c(3), c(1), col = c("red"))
> |
```



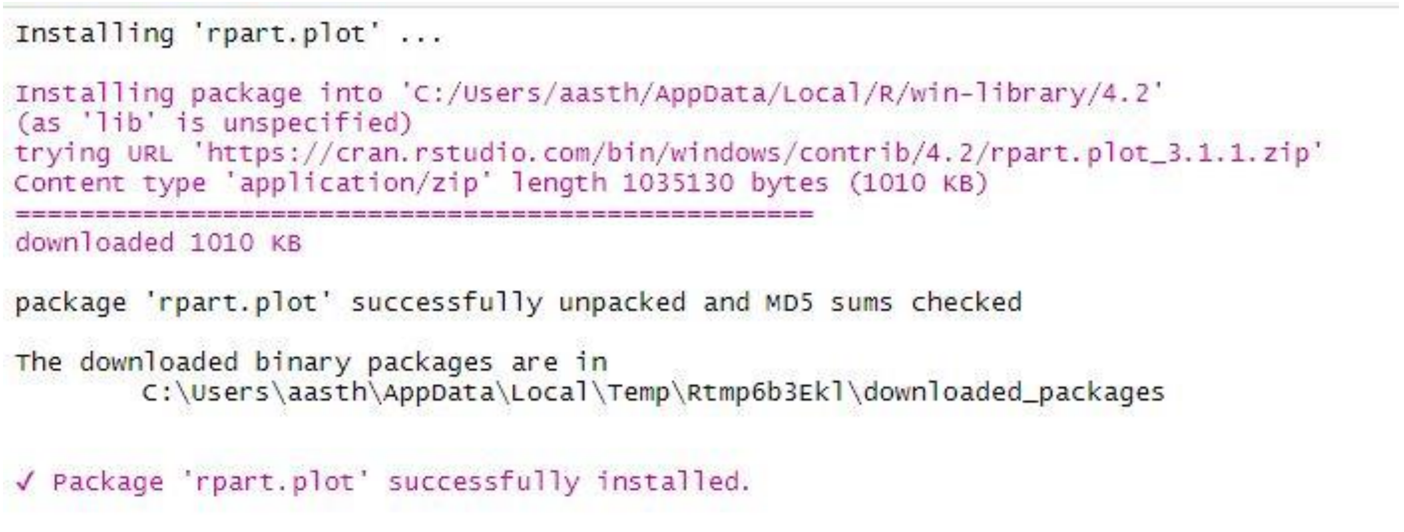
When point (3,1) (Red dot) is added to the plot, the two classes are not separable by hyperplane anymore.

2. Practicum Problems

Problem #1

library(rpart)

library(rpart.plot)



```
Installing 'rpart.plot' ...

Installing package into 'C:/Users/aasth/AppData/Local/R/win-library/4.2'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.2/rpart.plot_3.1.1.zip'
Content type 'application/zip' length 1035130 bytes (1010 KB)
==================================================
downloaded 1010 KB

package 'rpart.plot' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\aasth\AppData\Local\Temp\Rtmp6b3Ek1\downloaded_packages


✓ Package 'rpart.plot' successfully installed.
```

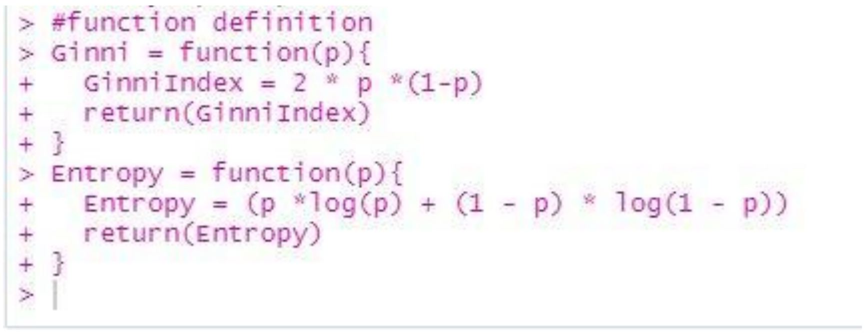
```
> library(rpart)
> library(rpart.plot)
> |
```

#function definition

Ginni = function(p){

 GinniIndex = 2 * p *(1-p)

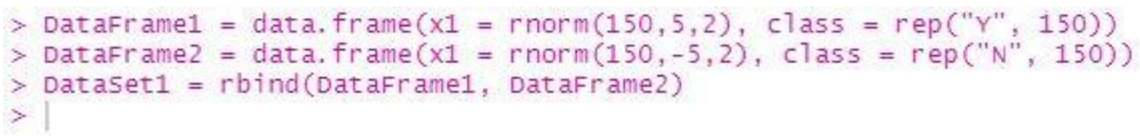
```
 return(GinniIndex)

}

Entropy = function(p){

 Entropy = (p *log(p) + (1 - p) * log(1 - p))

 return(Entropy)

}
```



```
> #function definition
> Ginni = function(p){
+   GinniIndex = 2 * p *(1-p)
+   return(GinniIndex)
+ }
> Entropy = function(p){
+   Entropy = (p *log(p) + (1 - p) * log(1 - p))
+   return(Entropy)
+ }
> |
```

The normal distribution parameters (using the function rnorm) should be (5,2) and (-5,2) for the pair of samples - you can determine an appropriate number of samples.

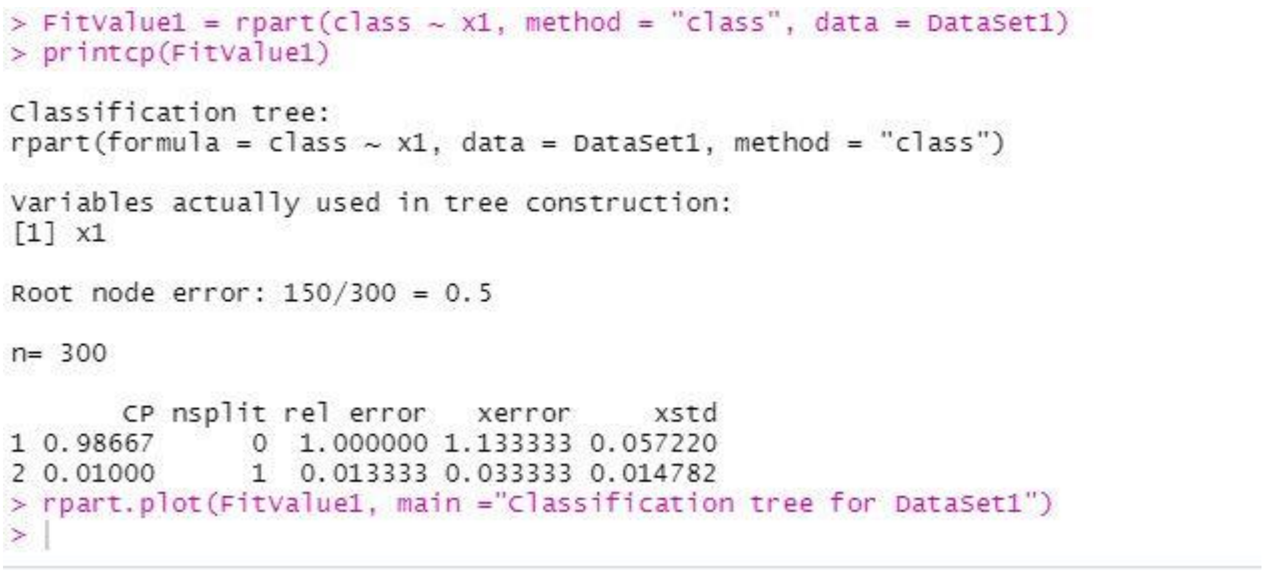
```
DataFrame1 = data.frame(x1 = rnorm(150,5,2), class = rep("Y", 150))

DataFrame2 = data.frame(x1 = rnorm(150,-5,2), class = rep("N", 150))

DataSet1 = rbind(DataFrame1, DataFrame2)
```



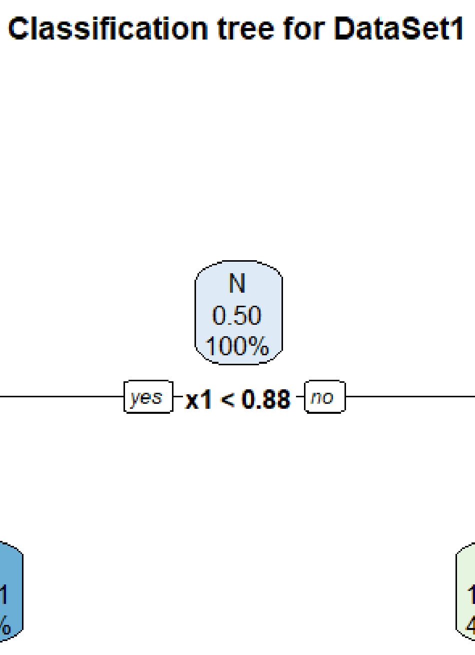
```
> DataFrame1 = data.frame(x1 = rnorm(150,5,2), class = rep("Y", 150))
> DataFrame2 = data.frame(x1 = rnorm(150,-5,2), class = rep("N", 150))
> DataSet1 = rbind(DataFrame1, DataFrame2)
> |
```

Induce a binary decision tree (using rpart), and obtain the threshold value for the feature in the first split.

```
FitValue1 = rpart(class ~ x1, method = "class", data = DataSet1)

printcp(FitValue1)

rpart.plot(FitValue1, main ="Classification tree for DataSet1")
```



```
> FitValue1 = rpart(class ~ x1, method = "class", data = DataSet1)
> printcp(FitValue1)

Classification tree:
rpart(formula = class ~ x1, data = DataSet1, method = "class")

Variables actually used in tree construction:
[1] x1

Root node error: 150/300 = 0.5

n= 300

        CP nsplit rel error   xerror     xstd
1 0.98667      0  1.000000 1.133333 0.057220
2 0.01000      1  0.013333 0.033333 0.014782
> rpart.plot(FitValue1, main ="Classification tree for DataSet1")
> |
```



#How does this value compare to the empirical distribution of the feature?

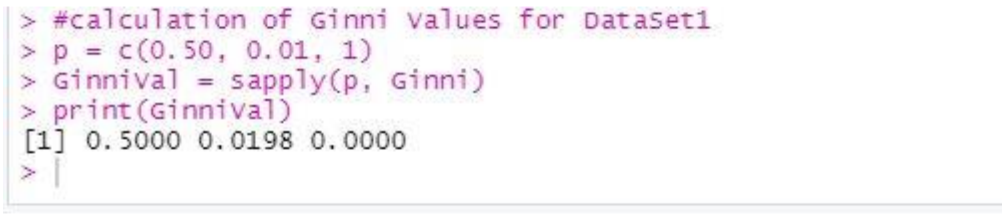
In this case, the threshold value for 1st split is less than 0.88 and the tree has 3 nodes i.e. 1 Root and 2 Leaf. The dataset is created using normal distribution in which 150 samples are used with mean = 5 and standard derivation=2 along with 150 samples with mean of -5 and the standard deviation is 2. We get at least 50-50% class distribution.

#calculation of Ginni Values for DataSet1

p = c(0.50, 0.01, 1)

GinniVal = sapply(p, Ginni)

print(GinniVal)

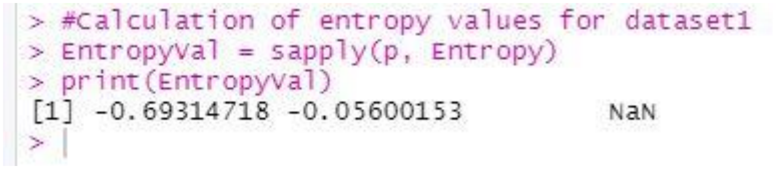


```
> #calculation of Ginni values for DataSet1
> p = c(0.50, 0.01, 1)
> GinniVal = sapply(p, Ginni)
> print(GinniVal)
[1] 0.5000 0.0198 0.0000
> |
```

#Calculation of entropy values for dataset1

EntropyVal = sapply(p, Entropy)

print(EntropyVal)

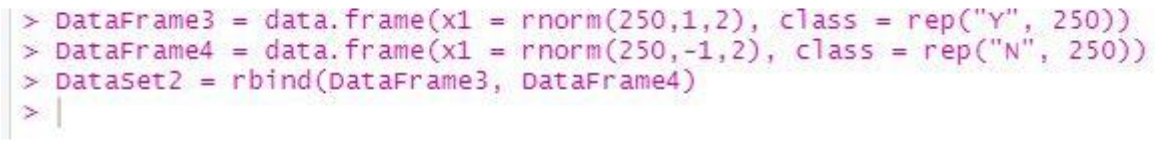


```
> #Calculation of entropy values for dataset1
> EntropyVal = sapply(p, Entropy)
> print(EntropyVal)
[1] -0.69314718 -0.05600153         NaN
> |
```

DataFrame3 = data.frame(x1 = rnorm(250,1,2), class = rep("Y", 250))

DataFrame4 = data.frame(x1 = rnorm(250,-1,2), class = rep("N", 250))

DataSet2 = rbind(DataFrame3, DataFrame4)

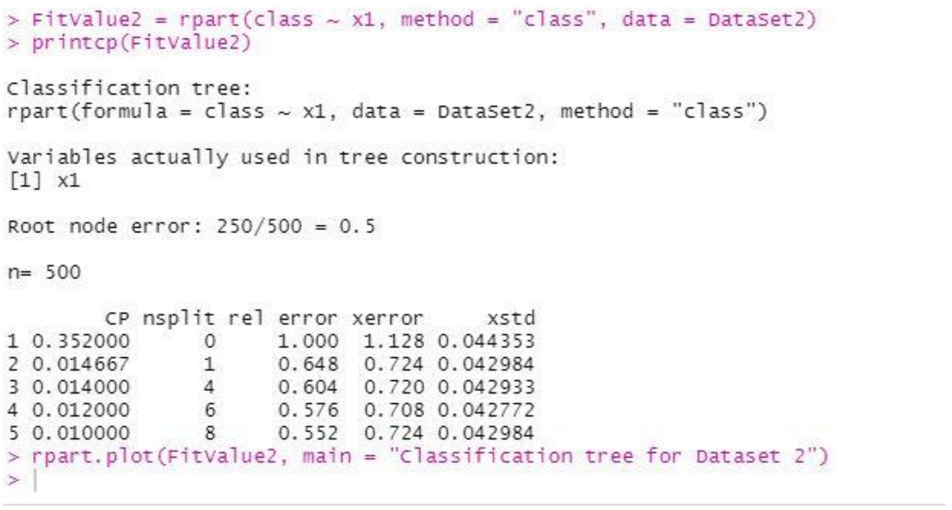


```
> DataFrame3 = data.frame(x1 = rnorm(250,1,2), class = rep("Y", 250))
> DataFrame4 = data.frame(x1 = rnorm(250,-1,2), class = rep("N", 250))
> DataSet2 = rbind(DataFrame3, DataFrame4)
> 
```

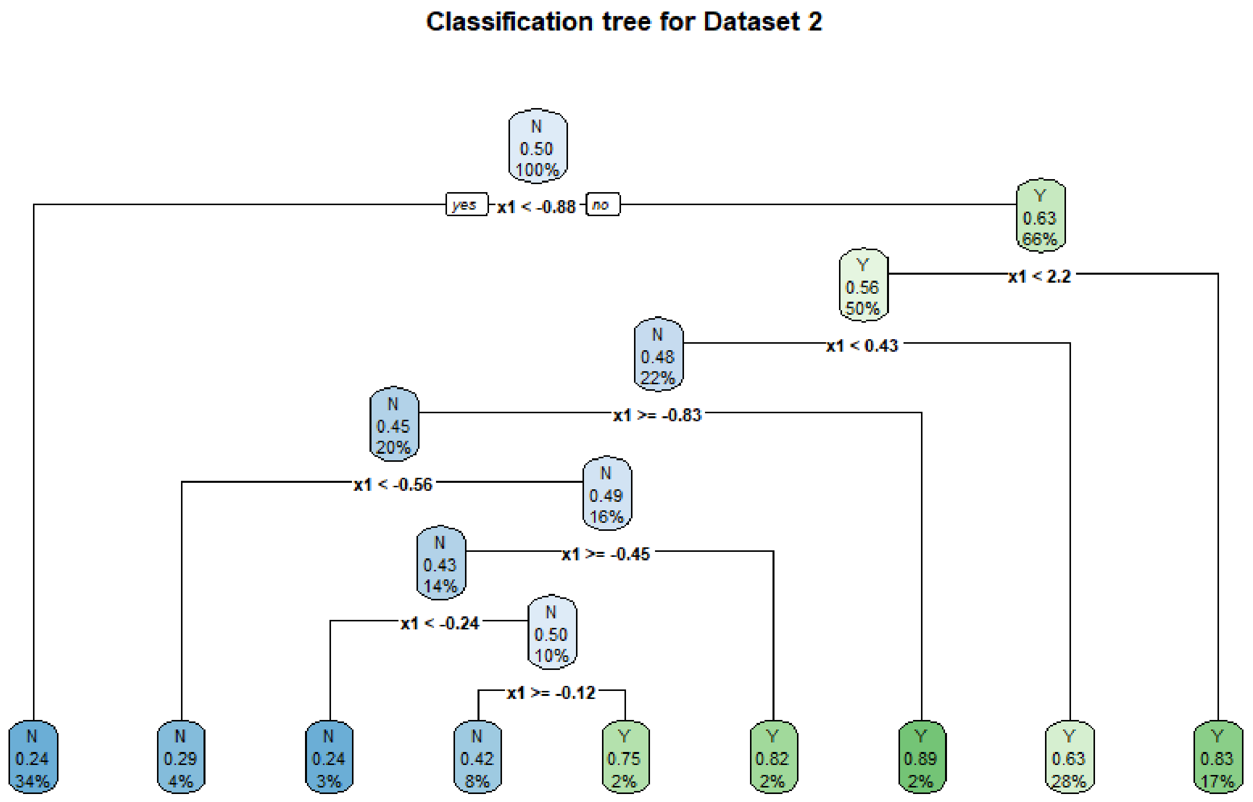
FitValue2 = rpart(class ~ x1, method = "class", data = DataSet2)

printcp(FitValue2)

rpart.plot(FitValue2, main = "Classification tree for Dataset 2")



```
> FitValue2 = rpart(class ~ x1, method = "class", data = DataSet2)
> printcp(FitValue2)

Classification tree:
rpart(formula = class ~ x1, data = DataSet2, method = "class")

Variables actually used in tree construction:
[1] x1

Root node error: 250/500 = 0.5

n= 500

        CP nsplit rel error xerror     xstd
1 0.352000      0     1.000  1.128 0.044353
2 0.014667      1     0.648  0.724 0.042984
3 0.014000      4     0.604  0.720 0.042933
4 0.012000      6     0.576  0.708 0.042772
5 0.010000      8     0.552  0.724 0.042984
> rpart.plot(FitValue2, main = "Classification tree for Dataset 2")
> 
```



Answer- We can see that the threshold value for the 1st split is less than -0.88. The tree has a total of 17 nodes of which one node is the root node and our dataset is created with rnorm values(1,2) and (-1,2)against(5,2)and(-5,2).In this part, more nodes are created as more values in the shorter range result in overlapping, and hence to come out of this situation we use the pruning technique.

calculation of Ginni Values for DataSet2

p = c(.50, 0.38, 0.49,0.53,0.51,0.54,0.51,0.55,0.51,0.56,0.22,0.34,0.27,0.27,

0.38,0.44,0.68,0.80,0.68,0.71,0.81)

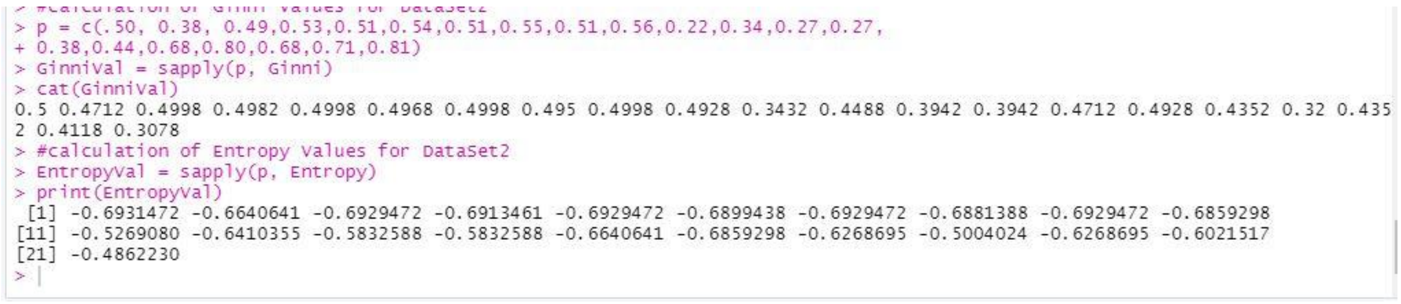
GinniVal = sapply(p, Ginni)

cat(GinniVal)

calculation of Entropy Values for DataSet2

EntropyVal = sapply(p, Entropy)

print(EntropyVal)



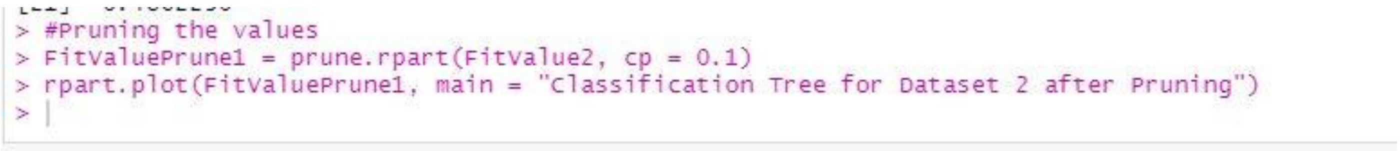
```
> p = c(.50, 0.38, 0.49,0.53,0.51,0.54,0.51,0.55,0.51,0.56,0.22,0.34,0.27,0.27,
+ 0.38,0.44,0.68,0.80,0.68,0.71,0.81)
> GinniVal = sapply(p, Ginni)
> cat(GinniVal)
0.5 0.4712 0.4998 0.4982 0.4998 0.4968 0.4998 0.495 0.4998 0.4928 0.3432 0.4488 0.3942 0.3942 0.4712 0.4928 0.4352 0.32 0.435
2 0.4118 0.3078
> #calculation of Entropy Values for DataSet2
> EntropyVal = sapply(p, Entropy)
> print(EntropyVal)
 [1] -0.6931472 -0.6640641 -0.6929472 -0.6913461 -0.6929472 -0.6899438 -0.6929472 -0.6881388 -0.6929472 -0.6859298
[11] -0.5269080 -0.6410355 -0.5832588 -0.5832588 -0.6640641 -0.6859298 -0.6268695 -0.5004024 -0.6268695 -0.6021517
[21] -0.4862230
> |
```

Pruning the values

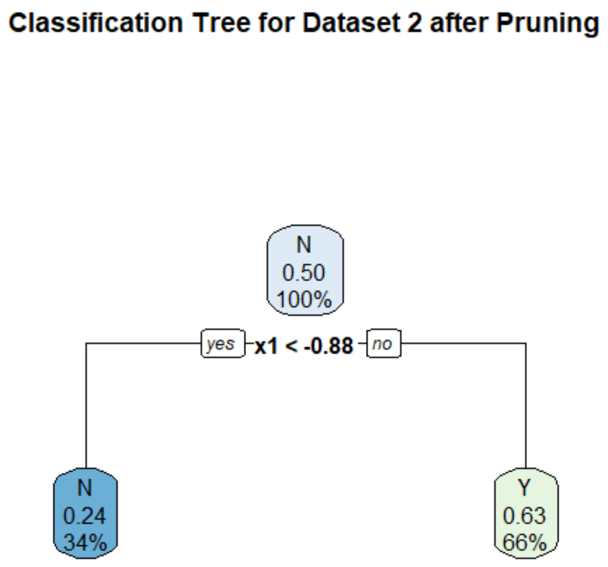
FitValuePrune1 = prune.rpart(FitValue2, cp = 0.1)

rpart.plot(FitValuePrune1, main = "Classification Tree for Dataset 2 after Pruning")

After Pruning the tree number of nodes reduce from 11 to 3 with threshold value less than -0.88.



```
> #Pruning the values
> FitValuePrune1 = prune.rpart(FitValue2, cp = 0.1)
> rpart.plot(FitValuePrune1, main = "Classification Tree for Dataset 2 after Pruning")
> |
```



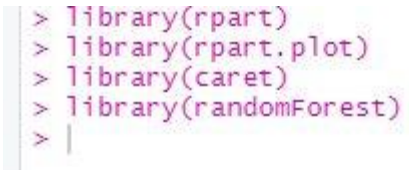
Problem#2

library(rpart)

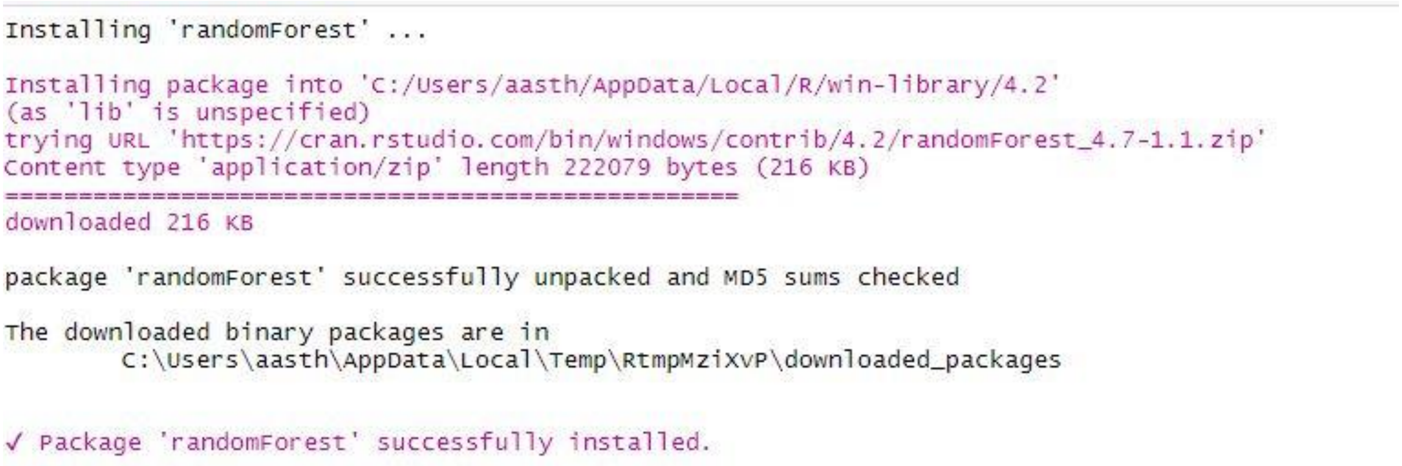
library(rpart.plot)

library(caret)

library(randomForest)



```
> library(rpart)
> library(rpart.plot)
> library(caret)
> library(randomForest)
> |
```



```
Installing 'randomForest' ...

Installing package into 'C:/Users/aasth/AppData/Local/R/win-library/4.2'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.2/randomForest_4.7-1.1.zip'
Content type 'application/zip' length 222079 bytes (216 KB)
==================================================
downloaded 216 KB

package 'randomForest' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\aasth\AppData\Local\Temp\RtmpMziXvP\downloaded_packages


✓ Package 'randomForest' successfully installed.
```

RedData = read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv",

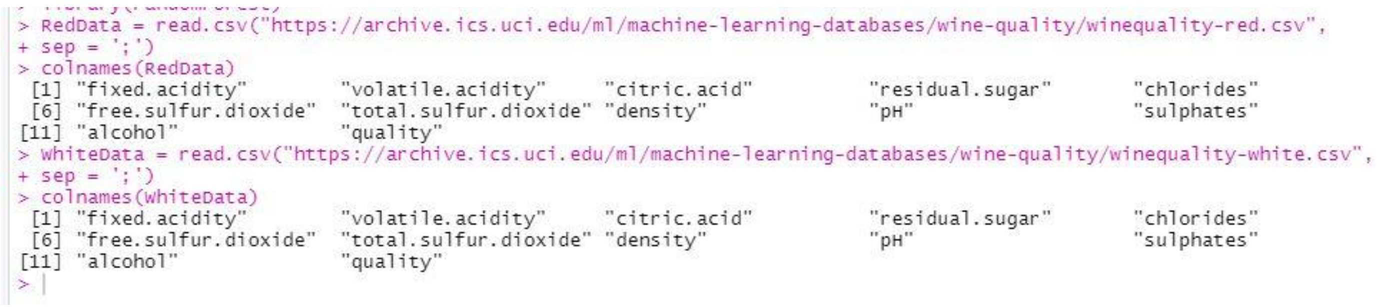
sep = ';')

WhiteData = read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv",

sep = ';')

colnames(RedData)

colnames(WhiteData)



```
> RedData = read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv",
+ sep = ';')
> colnames(RedData)
 [1] "fixed.acidity"        "volatile.acidity"     "citric.acid"          "residual.sugar"       "chlorides"
 [6] "free.sulfur.dioxide"  "total.sulfur.dioxide" "density"              "pH"                   "sulphates"
[11] "alcohol"              "quality"
> WhiteData = read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv",
+ sep = ';')
> colnames(WhiteData)
 [1] "fixed.acidity"        "volatile.acidity"     "citric.acid"          "residual.sugar"       "chlorides"
 [6] "free.sulfur.dioxide"  "total.sulfur.dioxide" "density"              "pH"                   "sulphates"
[11] "alcohol"              "quality"
> |
```

80-20 Test-train split

Create an 80/20 test-train split of each wine data frame, and use the rpart package to induce a decision tree of both the red and white wines, targeting the quality variable.

80-20 test-Train split of Red Wine

RedWineDataIndex = createDataPartition(y = RedData$quality, p = 0.8, list = FALSE)

TrainingDataRedWine = RedData[RedWineDataIndex,]

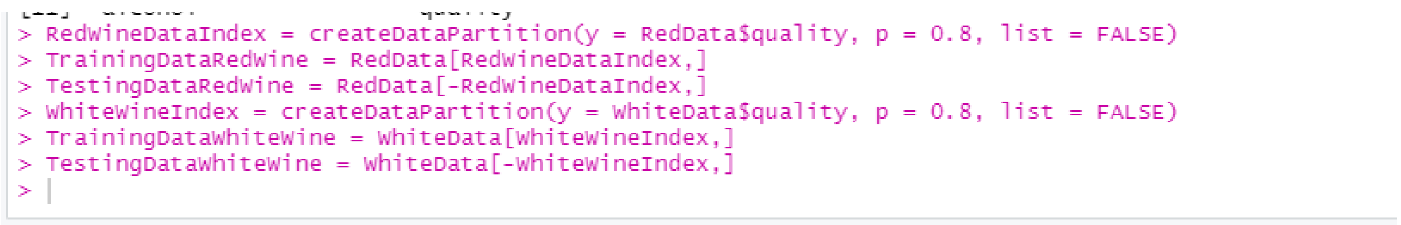
TestingDataRedWine = RedData[-RedWineDataIndex,]

80-20 test-train split of white wine

WhiteWineIndex = createDataPartition(y = WhiteData$quality, p = 0.8, list = FALSE)

TrainingDataWhiteWine = WhiteData[WhiteWineIndex,]

TestingDataWhiteWine = WhiteData[-WhiteWineIndex,]



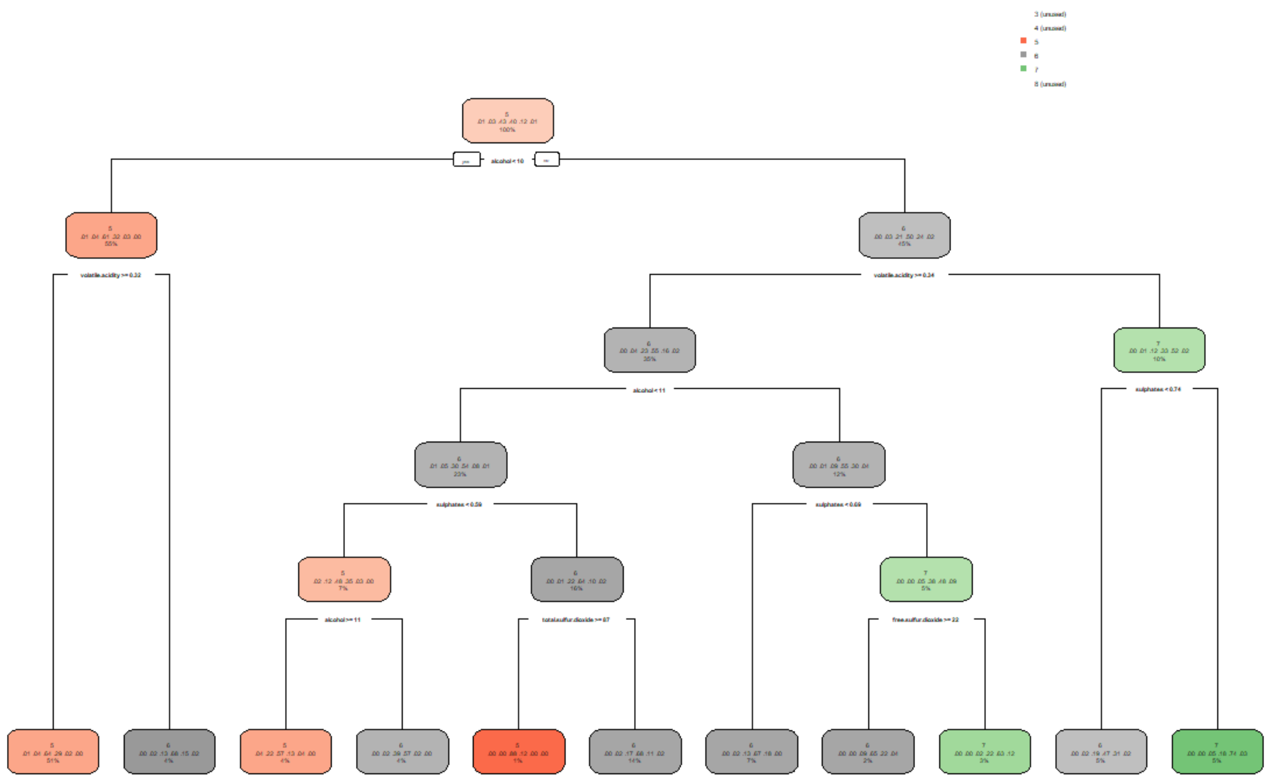
```
> RedWineDataIndex = createDataPartition(y = RedData$quality, p = 0.8, list = FALSE)
> TrainingDataRedWine = RedData[RedWineDataIndex,]
> TestingDataRedWine = RedData[-RedWineDataIndex,]
> WhiteWineIndex = createDataPartition(y = WhiteData$quality, p = 0.8, list = FALSE)
> TrainingDataWhiteWine = WhiteData[WhiteWineIndex,]
> TestingDataWhiteWine = WhiteData[-WhiteWineIndex,]
> 
```

Visualize the tree using the rpart.plot library, and use the caret package confusionMatrix method to determine the decision tree accuracy on the test set. Compare the decision trees for red and white wine what differences in terms of tree structure and variables of interest can be noted?

#Red Wine decision tree

RedWineDecisionTree = rpart(quality ~. , data = TrainingDataRedWine, method = "class")

rpart.plot(RedWineDecisionTree)



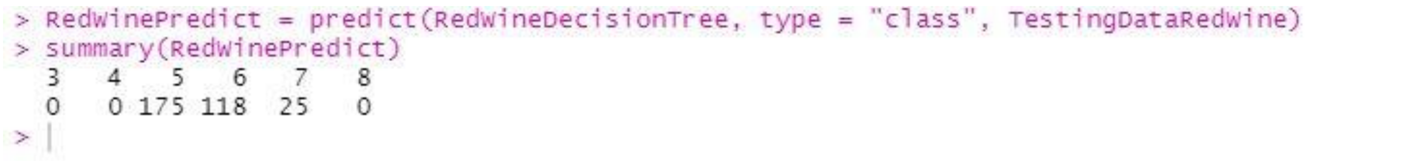
Answer-The above decision tree for red wine data set has 21 nodes in total. The problem can be viewed as a classification problem and the response i.e. the quality variable has classes between 0-10. Out of these 10 classes, most of the data samples come under 3 classes which are 5, 6 and 7. The decision tree has overlapping labels.

#Model Evaluation in case of red wine

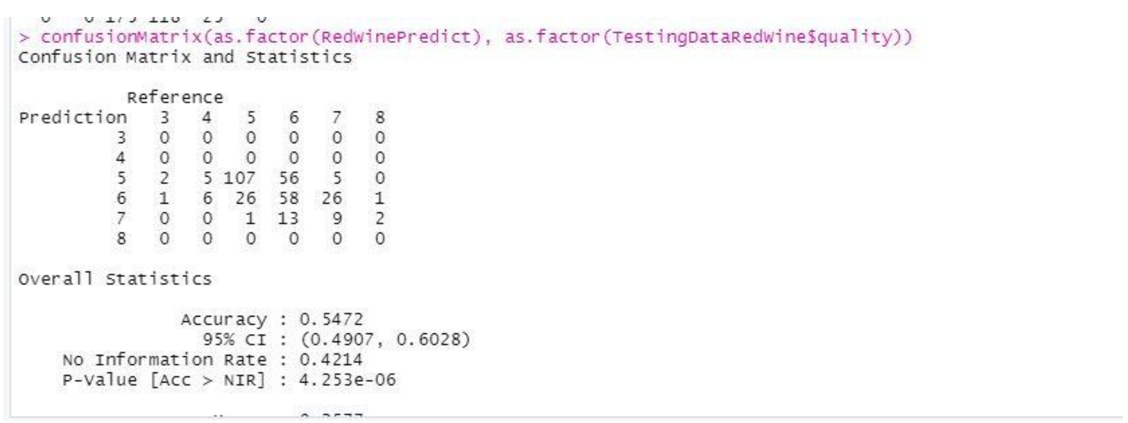
RedWinePredict = predict(RedWineDecisionTree, type = "class", TestingDataRedWine)

summary(RedWinePredict)

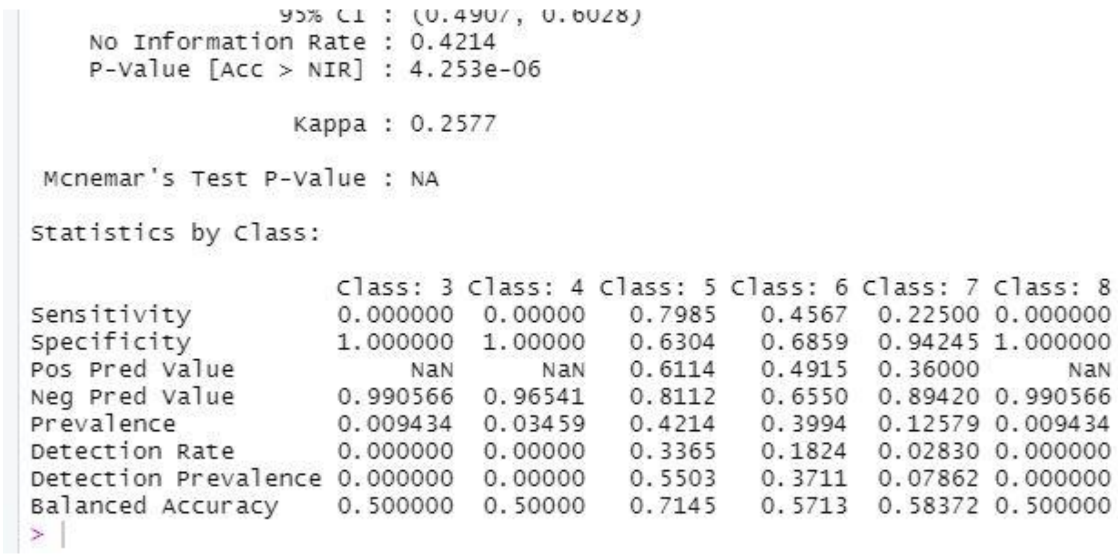
confusionMatrix(as.factor(RedWinePredict), as.factor(TestingDataRedWine$quality))



```
> RedWinePredict = predict(RedWineDecisionTree, type = "class", TestingDataRedWine)
> summary(RedWinePredict)
  3   4   5   6   7   8 
  0   0 175 118  25   0 
> |
```



```
> confusionMatrix(as.factor(RedWinePredict), as.factor(TestingDataRedWine$quality))
Confusion Matrix and Statistics

          Reference
Prediction   3   4   5   6   7   8
         3   0   0   0   0   0   0
         4   0   0   0   0   0   0
         5   2   5 107  56   5   0
         6   1   6  26  58  26   1
         7   0   0   1  13   9   2
         8   0   0   0   0   0   0

Overall Statistics
                                          
               Accuracy : 0.5472          
                 95% CI : (0.4907, 0.6028)
    No Information Rate : 0.4214          
    P-Value [Acc > NIR] : 4.253e-06       
```



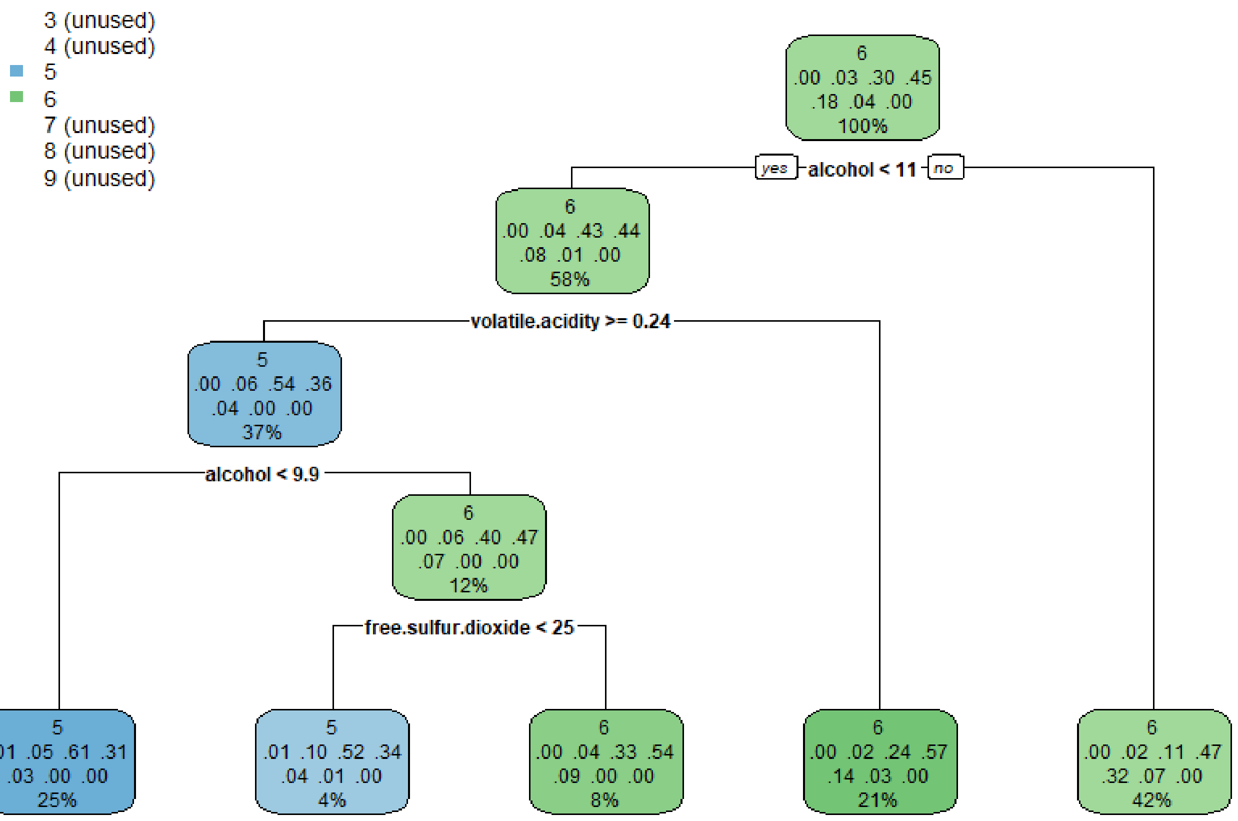
```
                 95% CI : (0.4907, 0.6028)
    No Information Rate : 0.4214          
    P-Value [Acc > NIR] : 4.253e-06       
                                          
                  Kappa : 0.2577          
                                          
 Mcnemar's Test P-Value : NA              

Statistics by Class:

                     Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
Sensitivity          0.000000  0.00000   0.7985   0.4567  0.22500 0.000000
Specificity          1.000000  1.00000   0.6304   0.6859  0.94245 1.000000
Pos Pred Value            NaN      NaN   0.6114   0.4915  0.36000      NaN
Neg Pred Value       0.990566  0.96541   0.8112   0.6550  0.89420 0.990566
Prevalence           0.009434  0.03459   0.4214   0.3994  0.12579 0.009434
Detection Rate       0.000000  0.00000   0.3365   0.1824  0.02830 0.000000
Detection Prevalence 0.000000  0.00000   0.5503   0.3711  0.07862 0.000000
Balanced Accuracy    0.500000  0.50000   0.7145   0.5713  0.58372 0.500000
> |
```

Answer-Accuracy is 54.72% and this is not a good one.

#White wine decision tree

WhiteWineDecisionTree = rpart(quality ~. , data = TrainingDataWhiteWine, method = "class")

rpart.plot(WhiteWineDecisionTree)



The above decision tree for the white wine data set has 9 nodes in total. This problem can be viewed as a classification problem and the response i.e., the quality variable has classes between 0-10. Out of these 10 classes, most of the data samples come under 2 classes which are 5, and 6. The decision tree has overlapping labels. There are differences between tree structures of red wine and white wine data sets and variables of interest. We can see that the decision tree of red wine has more nodes i.e., 21 nodes as compared to white wine which has only 9 nodes. In both cases, the most commonly occurring classes are 5 and 6. In the red wine decision tree, there are more labels overlapping than the white wine decision tree. In white wine, the features based on which the splits are taking place are alcohol and volatile acidity whereas in the case of the red wine decision tree, there are more than 2 features based on which splits are taking place which are alcohol, volatile acidity, sulfates, residual sugar, fixed.acidity. So, these are the features that play a major role in deciding the outcome. A greater number of split nodes are created in the red wine tree as more values are present in the smaller r range, which leads to overlapping labels in nodes.

#Model Evaluation in the case of white wine

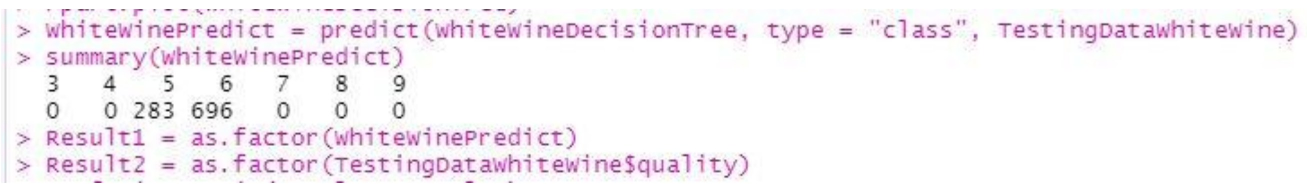
WhiteWinePredict = predict(WhiteWineDecisionTree, type = "class", TestingDataWhiteWine)

summary(WhiteWinePredict)

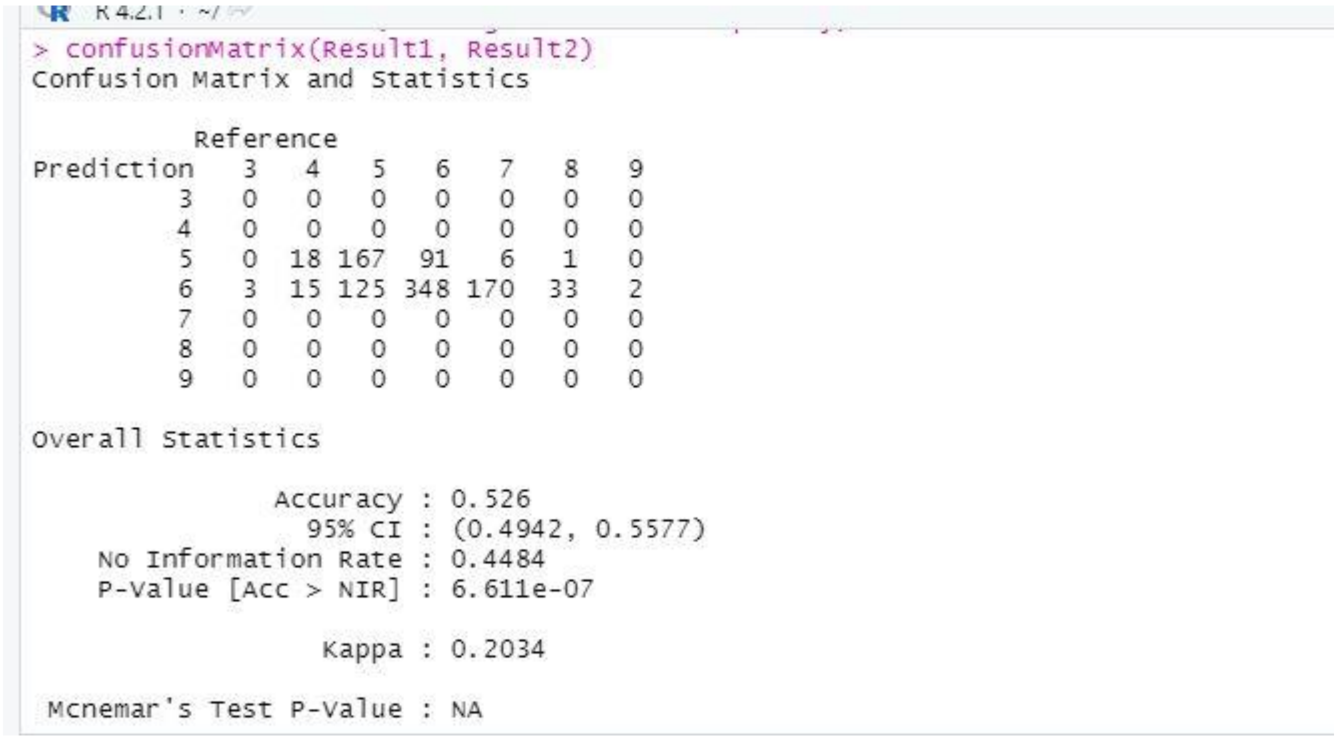
Result1 = as.factor(WhiteWinePredict)

Result2 = as.factor(TestingDataWhiteWine$quality)

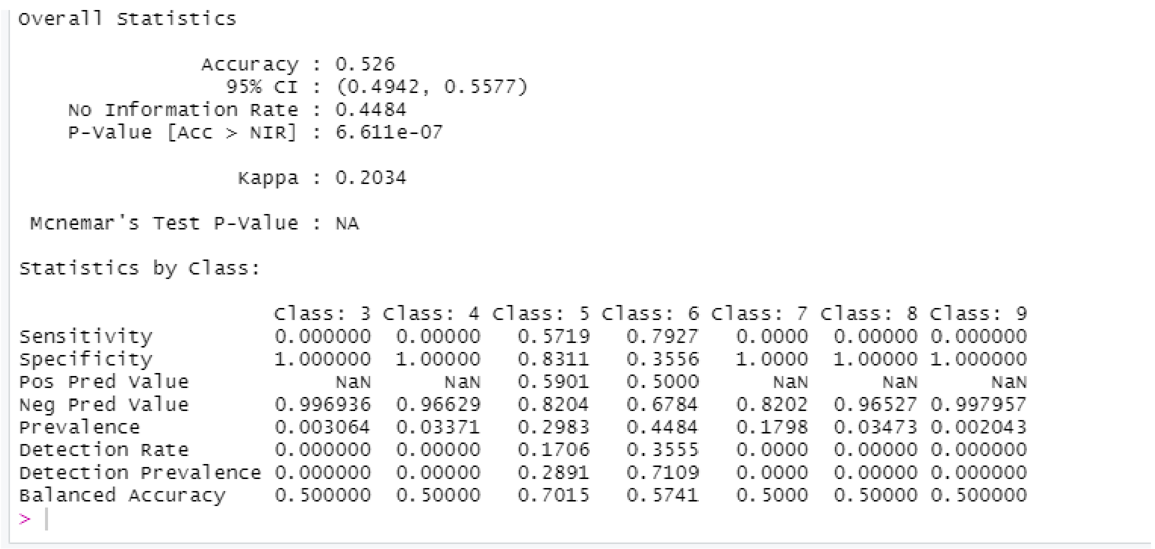
confusionMatrix(Result1, Result2)



```
> whiteWinePredict = predict(whiteWineDecisionTree, type = "class", TestingDataWhiteWine)
> summary(whiteWinePredict)
  3   4   5   6   7   8   9 
  0   0 283 696   0   0   0 
> Result1 = as.factor(whiteWinePredict)
> Result2 = as.factor(TestingDataWhiteWine$quality)
```



```
R 4.2.1 · ~/
> confusionMatrix(Result1, Result2)
Confusion Matrix and Statistics

          Reference
Prediction   3   4   5   6   7   8   9
         3   0   0   0   0   0   0   0
         4   0   0   0   0   0   0   0
         5   0  18 167  91   6   1   0
         6   3  15 125 348 170  33   2
         7   0   0   0   0   0   0   0
         8   0   0   0   0   0   0   0
         9   0   0   0   0   0   0   0

Overall Statistics

               Accuracy : 0.526
                 95% CI : (0.4942, 0.5577)
    No Information Rate : 0.4484
    P-Value [Acc > NIR] : 6.611e-07

                  Kappa : 0.2034

 Mcnemar's Test P-Value : NA
```



```
Overall Statistics

               Accuracy : 0.526
                 95% CI : (0.4942, 0.5577)
    No Information Rate : 0.4484
    P-Value [Acc > NIR] : 6.611e-07

                  Kappa : 0.2034

 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity          0.000000  0.00000   0.5719   0.7927   0.0000  0.00000 0.000000
Specificity          1.000000  1.00000   0.8311   0.3556   1.0000  1.00000 1.000000
Pos Pred Value            NaN      NaN   0.5901   0.5000      NaN      NaN      NaN
Neg Pred Value       0.996936  0.96629   0.8204   0.6784   0.8202  0.96527 0.997957
Prevalence           0.003064  0.03371   0.2983   0.4484   0.1798  0.03473 0.002043
Detection Rate       0.000000  0.00000   0.1706   0.3555   0.0000  0.00000 0.000000
Detection Prevalence 0.000000  0.00000   0.2891   0.7109   0.0000  0.00000 0.000000
Balanced Accuracy    0.500000  0.50000   0.7015   0.5741   0.5000  0.50000 0.500000
> |
```

#In this the accuracy is 52.6% and is also not a good one.

#Building Random Forest Model

Use the randomForest package to repeat the fit with a random forest tree model, and compare the resulting test accuracy against the original single tree model.

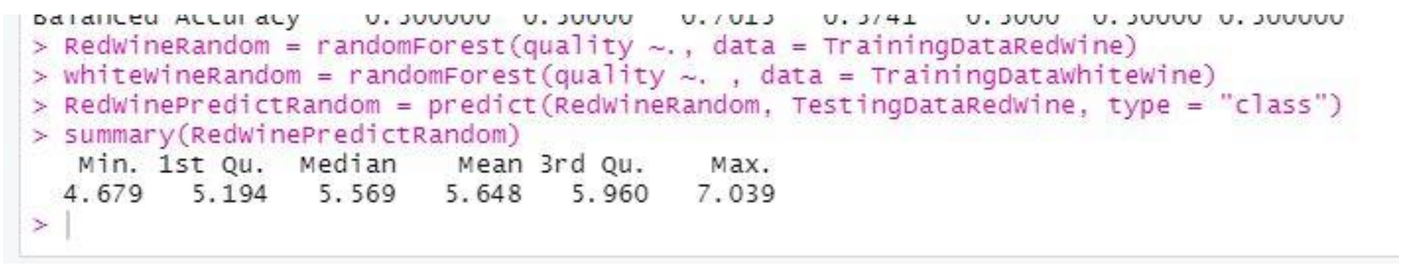
RedWineRandom = randomForest(quality ~., data = TrainingDataRedWine)

whiteWineRandom = randomForest(quality ~. , data = TrainingDataWhiteWine)

#Model Evaluation of Red Wine

RedWinePredictRandom = predict(RedWineRandom, TestingDataRedWine, type = "class")

summary(RedWinePredictRandom)

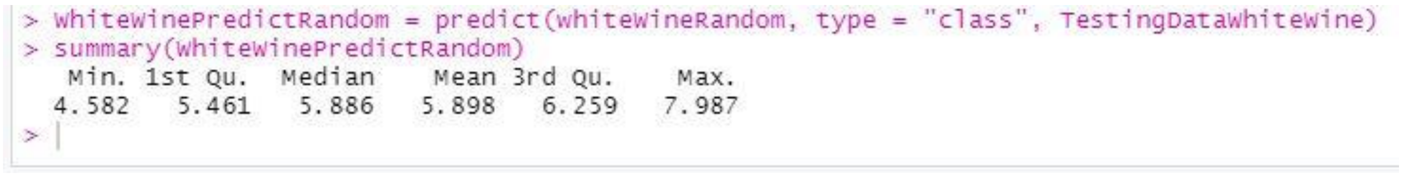


```
Balanced Accuracy     0.500000   0.50000    0.7013    0.5741    0.5000   0.50000 0.500000
> RedWineRandom = randomForest(quality ~., data = TrainingDataRedWine)
> whiteWineRandom = randomForest(quality ~.  , data = TrainingDataWhiteWine)
> RedWinePredictRandom = predict(RedWineRandom, TestingDataRedWine, type = "class")
> summary(RedWinePredictRandom)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  4.679   5.194   5.569   5.648   5.960   7.039 
> 
```

#Model evaluation on white wine test data set

WhiteWinePredictRandom = predict(whiteWineRandom, type = "class", TestingDataWhiteWine)

summary(WhiteWinePredictRandom)



```
> WhiteWinePredictRandom = predict(whiteWineRandom, type = "class", TestingDataWhiteWine)
> summary(WhiteWinePredictRandom)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  4.582   5.461   5.886   5.898   6.259   7.987 
> 
```

#Confusion Matrix for white wine

Result3 <- as.factor(c(WhiteWinePredictRandom))

levels(Result3)

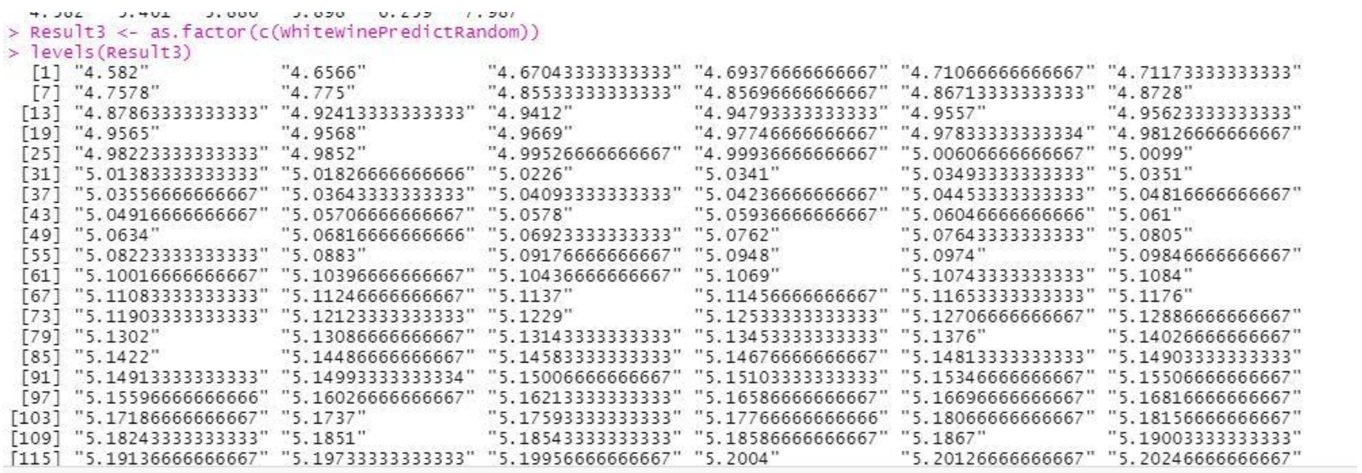
Result4 <- as.factor(c(TestingDataWhiteWine$quality))

levels(Result4)

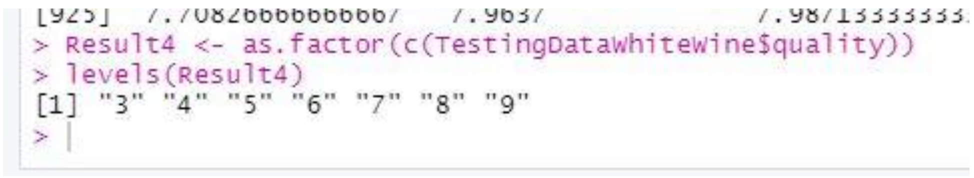
levels(Result3) <-c(Result4)

levels(Result3)

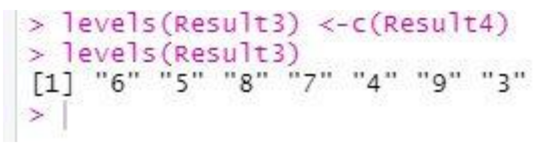
confusionMatrix(Result3, Result4)



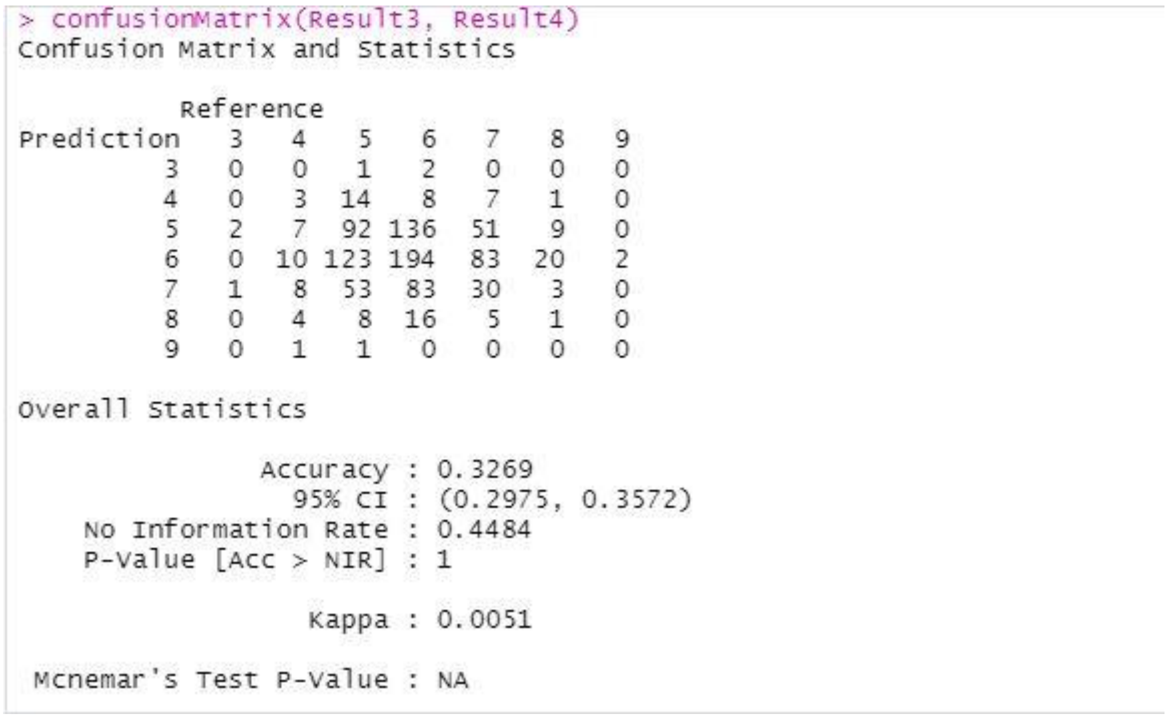
```
  4.582   5.461   5.886   5.898   6.259   7.987 
> Result3 <- as.factor(c(WhiteWinePredictRandom))
> levels(Result3)
  [1] "4.582"            "4.6566"           "4.67043333333333" "4.69376666666667" "4.71066666666667" "4.71173333333333"
  [7] "4.7578"           "4.775"            "4.85533333333333" "4.85696666666667" "4.86713333333333" "4.8728"          
 [13] "4.87863333333333" "4.92413333333333" "4.9412"           "4.94793333333333" "4.9557"           "4.95623333333333"
 [19] "4.9565"           "4.9568"           "4.9669"           "4.97746666666667" "4.97833333333334" "4.98126666666667"
 [25] "4.98223333333333" "4.9852"           "4.99526666666667" "4.99936666666667" "5.00606666666667" "5.0099"          
 [31] "5.01383333333333" "5.01826666666666" "5.0226"           "5.0341"           "5.03493333333333" "5.0351"          
 [37] "5.03556666666667" "5.03643333333333" "5.04093333333333" "5.04236666666667" "5.04453333333333" "5.04816666666667"
 [43] "5.04916666666667" "5.05706666666667" "5.0578"           "5.05936666666667" "5.06046666666666" "5.061"           
 [49] "5.0634"           "5.06816666666666" "5.06923333333333" "5.0762"           "5.07643333333333" "5.0805"          
 [55] "5.08223333333333" "5.0883"           "5.09176666666667" "5.0948"           "5.0974"           "5.09846666666667"
 [61] "5.10016666666667" "5.10396666666667" "5.10436666666667" "5.1069"           "5.10743333333333" "5.1084"          
 [67] "5.11083333333333" "5.11246666666667" "5.1137"           "5.11456666666667" "5.11653333333333" "5.1176"          
 [73] "5.11903333333333" "5.12123333333333" "5.1229"           "5.12533333333333" "5.12706666666667" "5.12886666666667"
 [79] "5.1302"           "5.13086666666667" "5.13143333333333" "5.13453333333333" "5.1376"           "5.14026666666667"
 [85] "5.1422"           "5.14486666666667" "5.14583333333333" "5.14676666666667" "5.14813333333333" "5.14903333333333"
 [91] "5.14913333333333" "5.14993333333334" "5.15006666666667" "5.15103333333333" "5.15346666666667" "5.15506666666667"
 [97] "5.15596666666666" "5.16026666666667" "5.16213333333333" "5.16586666666667" "5.16696666666667" "5.16816666666667"
[103] "5.17186666666667" "5.1737"           "5.17593333333333" "5.17766666666666" "5.18066666666667" "5.18156666666667"
[109] "5.18243333333333" "5.1851"           "5.18543333333333" "5.18586666666667" "5.1867"           "5.19003333333333"
[115] "5.19136666666667" "5.19733333333333" "5.19956666666667" "5.2004"           "5.20126666666667" "5.20246666666667"
```



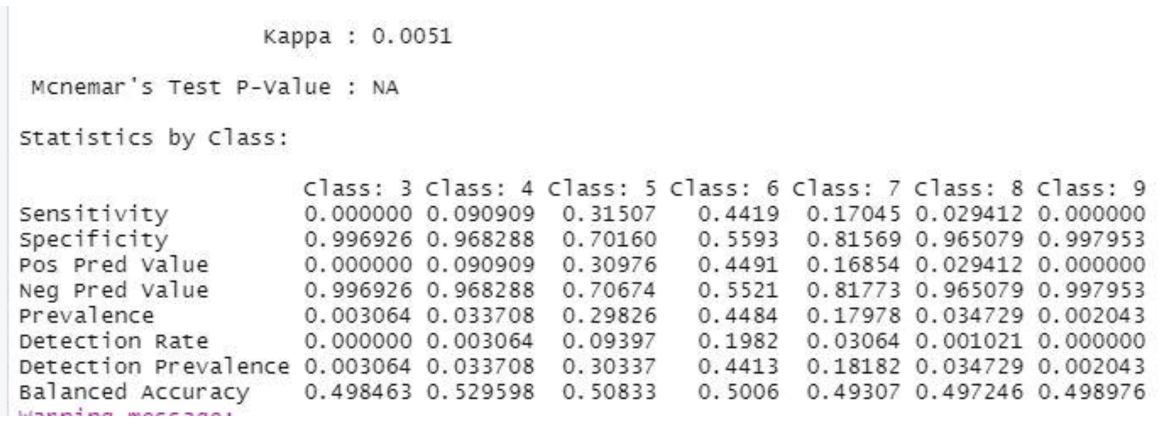
```
[925] "7.70826666666667"   "7.9637"            "7.98713333333
> Result4 <- as.factor(c(TestingDataWhiteWine$quality))
> levels(Result4)
[1] "3" "4" "5" "6" "7" "8" "9"
> 
```



```
> levels(Result3) <-c(Result4)
> levels(Result3)
[1] "6" "5" "8" "7" "4" "9" "3"
> 
```



```
> confusionMatrix(Result3, Result4)
Confusion Matrix and Statistics

          Reference
Prediction   3   4   5   6   7   8   9
         3   0   0   1   2   0   0   0
         4   0   3  14   8   7   1   0
         5   2   7  92 136  51   9   0
         6   0  10 123 194  83  20   2
         7   1   8  53  83  30   3   0
         8   0   4   8  16   5   1   0
         9   0   1   1   0   0   0   0

Overall Statistics

               Accuracy : 0.3269
                 95% CI : (0.2975, 0.3572)
    No Information Rate : 0.4484
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0051

 Mcnemar's Test P-Value : NA
```



```
                  Kappa : 0.0051

 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity          0.000000 0.090909  0.31507   0.4419  0.17045 0.029412 0.000000
Specificity          0.996926 0.968288  0.70160   0.5593  0.81569 0.965079 0.997953
Pos Pred Value       0.000000 0.090909  0.30976   0.4491  0.16854 0.029412 0.000000
Neg Pred Value       0.996926 0.968288  0.70674   0.5521  0.81773 0.965079 0.997953
Prevalence           0.003064 0.033708  0.29826   0.4484  0.17978 0.034729 0.002043
Detection Rate       0.000000 0.003064  0.09397   0.1982  0.03064 0.001021 0.000000
Detection Prevalence 0.003064 0.033708  0.30337   0.4413  0.18182 0.034729 0.002043
Balanced Accuracy    0.498463 0.529598  0.50833   0.5006  0.49307 0.497246 0.498976
```

Answer- The accuracy here is 32.69% which is less than the previous one which was 52.6%. This accuracy is also not good.

#confusion Matrix for red wine

Result5 <- as.factor(c(RedWinePredictRandom))

levels(Result5)

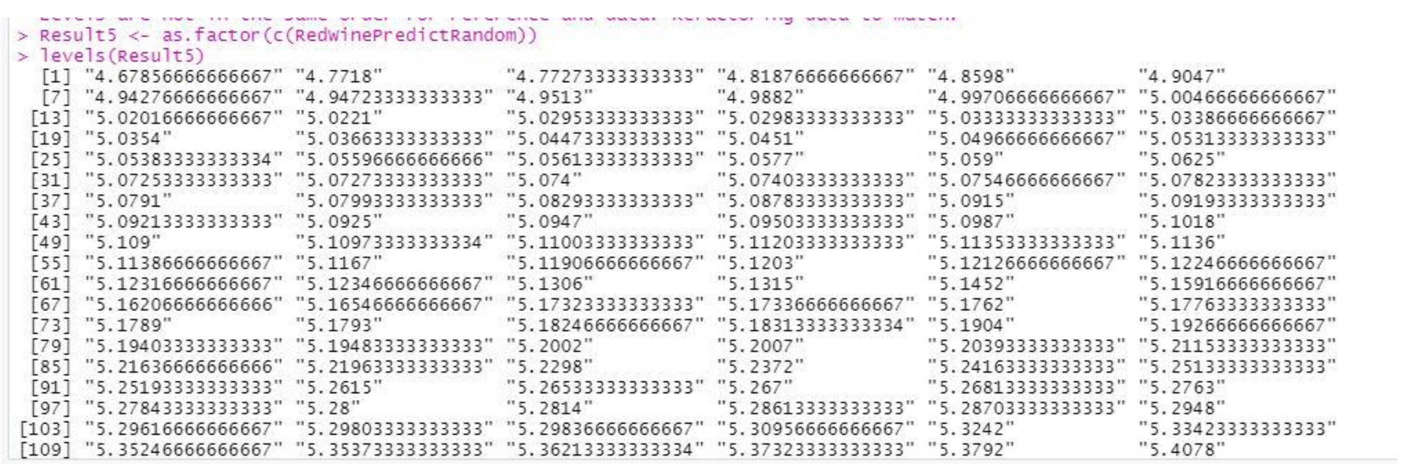
Result6 <- as.factor(c(TestingDataRedWine$quality))

levels(Result6)

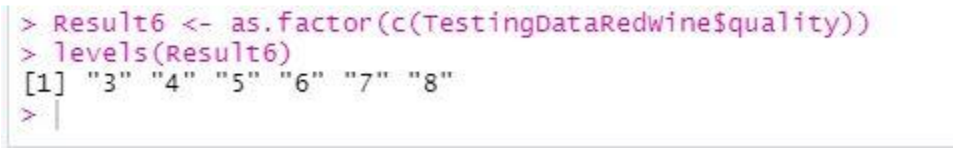
levels(Result5) <-c(Result6)

levels(Result5)

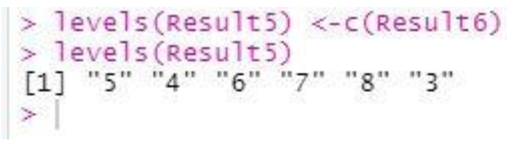
confusionMatrix(Result5, Result6)



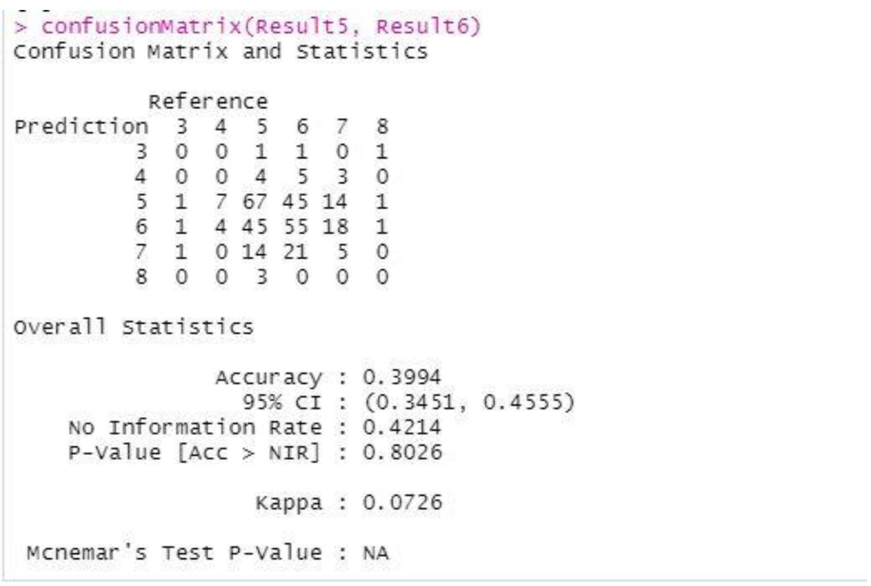
```
> Result5 <- as.factor(c(RedwinePredictRandom))
> levels(Result5)
  [1] "4.67856666666667" "4.7718"           "4.77273333333333" "4.81876666666667" "4.8598"           "4.9047"
  [7] "4.94276666666667" "4.94723333333333" "4.9513"           "4.9882"           "4.99706666666667" "5.00466666666667"
 [13] "5.02016666666667" "5.0221"           "5.02953333333333" "5.02983333333333" "5.03333333333333" "5.03386666666667"
 [19] "5.0354"           "5.03663333333333" "5.04473333333333" "5.0451"           "5.04966666666667" "5.05313333333333"
 [25] "5.05383333333334" "5.05596666666666" "5.05613333333333" "5.0577"           "5.059"            "5.0625"
 [31] "5.07253333333333" "5.07273333333333" "5.074"            "5.07403333333333" "5.07546666666667" "5.07823333333333"
 [37] "5.0791"           "5.07993333333333" "5.08293333333333" "5.08783333333333" "5.0915"           "5.09193333333333"
 [43] "5.09213333333333" "5.0925"           "5.0947"           "5.09503333333333" "5.0987"           "5.1018"
 [49] "5.109"            "5.10973333333334" "5.11003333333333" "5.11203333333333" "5.11353333333333" "5.1136"
 [55] "5.11386666666667" "5.1167"           "5.11906666666667" "5.1203"           "5.12126666666667" "5.12246666666667"
 [61] "5.12316666666667" "5.12346666666667" "5.1306"           "5.1315"           "5.1452"           "5.15916666666667"
 [67] "5.16206666666666" "5.16546666666667" "5.17323333333333" "5.17336666666667" "5.1762"           "5.17763333333333"
 [73] "5.1789"           "5.1793"           "5.18246666666667" "5.18313333333334" "5.1904"           "5.19266666666667"
 [79] "5.19403333333333" "5.19483333333333" "5.2002"           "5.2007"           "5.20393333333333" "5.21153333333333"
 [85] "5.21636666666666" "5.21963333333333" "5.2298"           "5.2372"           "5.24163333333333" "5.25133333333333"
 [91] "5.25193333333333" "5.2615"           "5.26533333333333" "5.267"            "5.26813333333333" "5.2763"
 [97] "5.27843333333333" "5.28"             "5.2814"           "5.28613333333333" "5.28703333333333" "5.2948"
[103] "5.29616666666667" "5.29803333333333" "5.29836666666667" "5.30956666666667" "5.3242"           "5.33423333333333"
[109] "5.35246666666667" "5.35373333333333" "5.36213333333334" "5.37323333333333" "5.3792"           "5.4078"
```



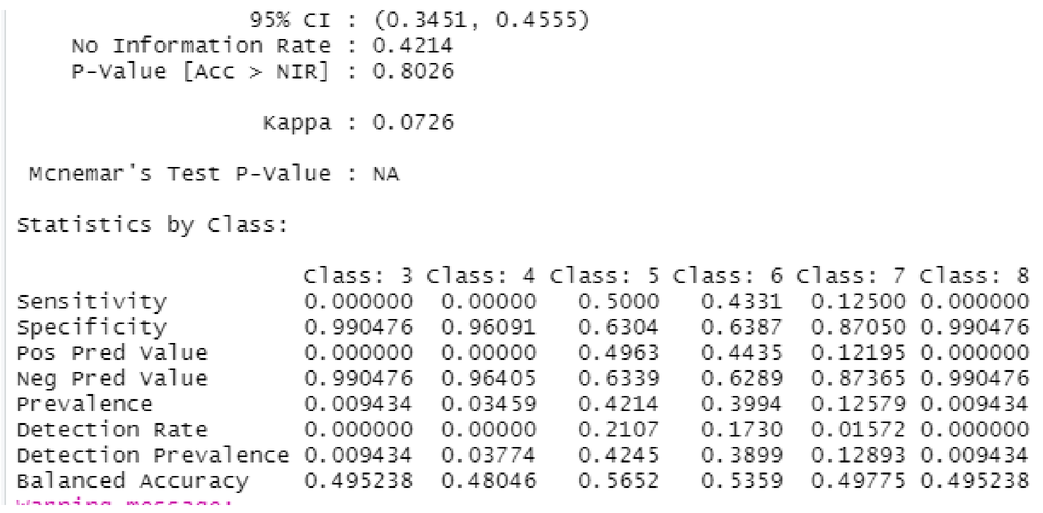
```
> Result6 <- as.factor(c(TestingDataRedwine$quality))
> levels(Result6)
[1] "3" "4" "5" "6" "7" "8"
> 
```



```
> levels(Result5) <-c(Result6)
> levels(Result5)
[1] "5" "4" "6" "7" "8" "3"
> 
```



```
> confusionMatrix(Result5, Result6)
Confusion Matrix and Statistics

          Reference
Prediction  3  4  5  6  7  8
         3  0  0  1  1  0  1
         4  0  0  4  5  3  0
         5  1  7 67 45 14  1
         6  1  4 45 55 18  1
         7  1  0 14 21  5  0
         8  0  0  3  0  0  0

Overall Statistics
                                          
               Accuracy : 0.3994          
                 95% CI : (0.3451, 0.4555)
    No Information Rate : 0.4214          
    P-Value [Acc > NIR] : 0.8026          
                                          
                  Kappa : 0.0726          
                                          
 Mcnemar's Test P-Value : NA              
```



```
                 95% CI : (0.3451, 0.4555)
    No Information Rate : 0.4214          
    P-Value [Acc > NIR] : 0.8026          
                                          
                  Kappa : 0.0726          
                                          
 Mcnemar's Test P-Value : NA              
```

```
Statistics by Class:

                     Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
Sensitivity          0.000000  0.00000   0.5000   0.4331  0.12500 0.000000
Specificity          0.990476  0.96091   0.6304   0.6387  0.87050 0.990476
Pos Pred Value       0.000000  0.00000   0.4963   0.4435  0.12195 0.000000
Neg Pred Value       0.990476  0.96405   0.6339   0.6289  0.87365 0.990476
Prevalence           0.009434  0.03459   0.4214   0.3994  0.12579 0.009434
Detection Rate       0.000000  0.00000   0.2107   0.1730  0.01572 0.000000
Detection Prevalence 0.009434  0.03774   0.4245   0.3899  0.12893 0.009434
Balanced Accuracy    0.495238  0.48046   0.5652   0.5359  0.49775 0.495238
```

The accuracy here is 39.94%, which is less than the previously calculated accuracy of 54.72%. This is also not a good percentage of accuracy as well.

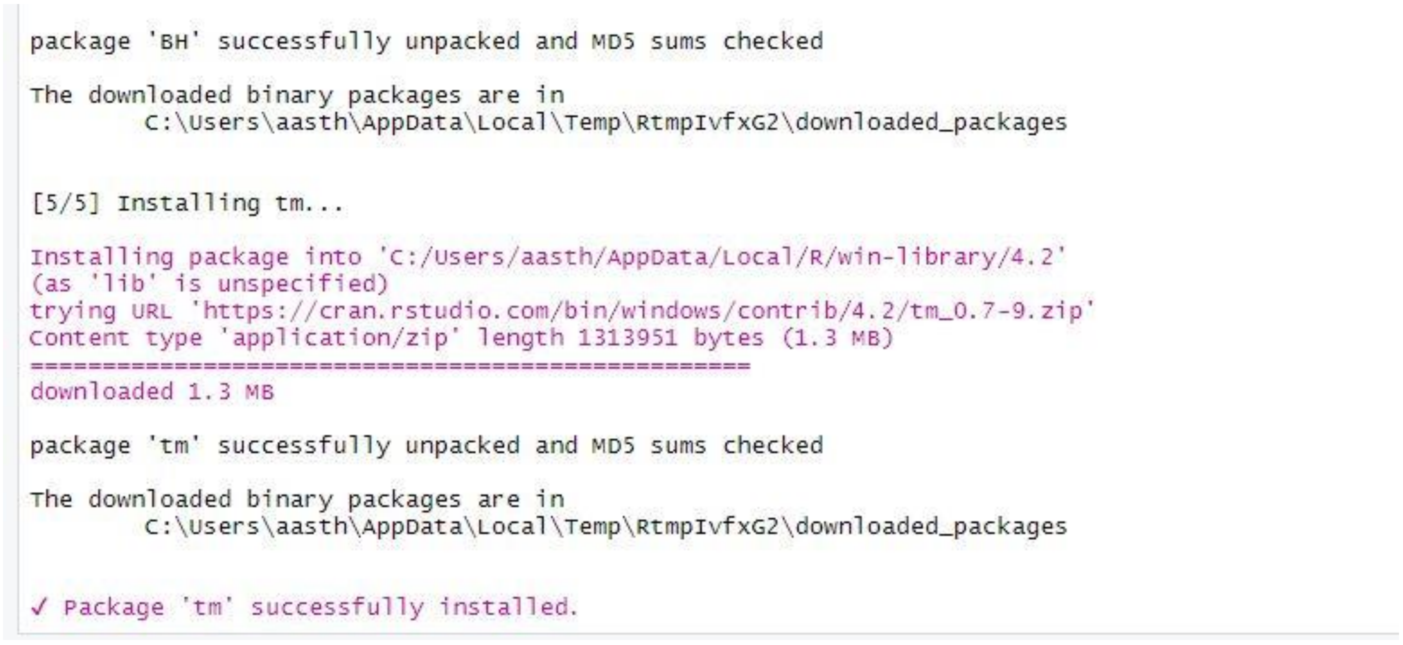
Problem#3

library(readxl)

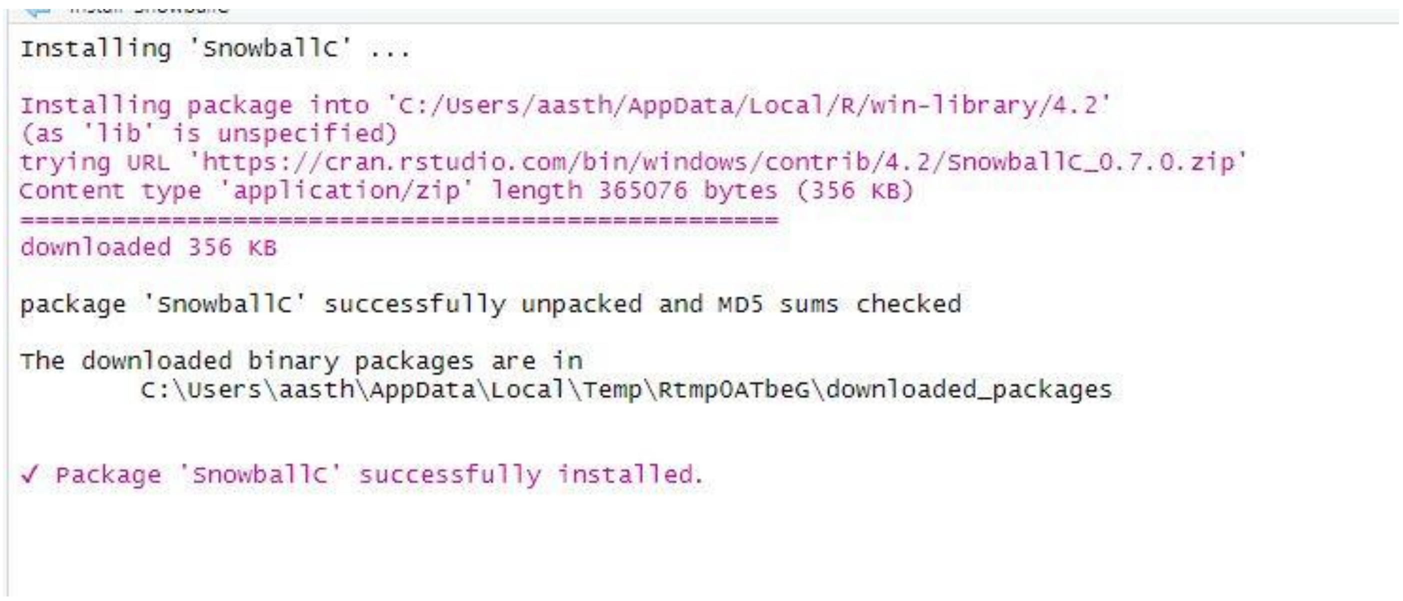
library(tm)

library(e1071)

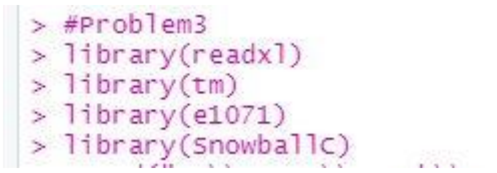
library(SnowballC)



```
package 'BH' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\aasth\AppData\Local\Temp\RtmpIvfxG2\downloaded_packages


[5/5] Installing tm...

Installing package into 'C:/Users/aasth/AppData/Local/R/win-library/4.2'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.2/tm_0.7-9.zip'
Content type 'application/zip' length 1313951 bytes (1.3 MB)
==================================================
downloaded 1.3 MB

package 'tm' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\aasth\AppData\Local\Temp\RtmpIvfxG2\downloaded_packages


✓ Package 'tm' successfully installed.
```



```
Installing 'SnowballC' ...

Installing package into 'C:/Users/aasth/AppData/Local/R/win-library/4.2'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.2/SnowballC_0.7.0.zip'
Content type 'application/zip' length 365076 bytes (356 KB)
==================================================
downloaded 356 KB

package 'SnowballC' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\aasth\AppData\Local\Temp\Rtmp0ATbeG\downloaded_packages


✓ Package 'SnowballC' successfully installed.
```



```
> #Problem3
> library(readxl)
> library(tm)
> library(e1071)
> library(SnowballC)
```

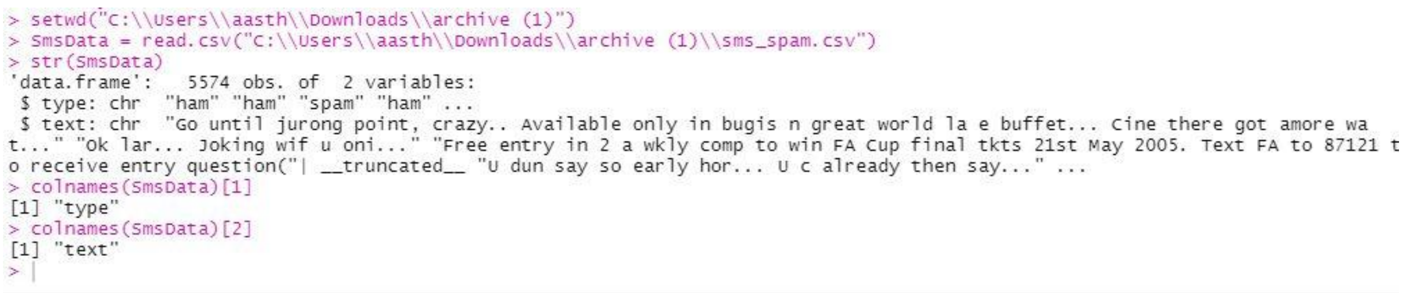
setwd("C:\\Users\\aasth\\Downloads\\archive (1)")

SmsData = read.csv("C:\\Users\\aasth\\Downloads\\archive (1)\\sms_spam.csv")

str(SmsData)

colnames(SmsData)[1]

colnames(SmsData)[2]

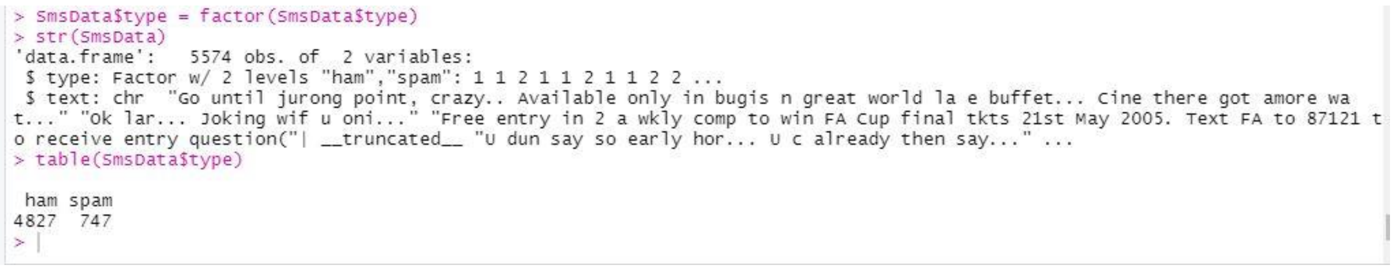


```
> setwd("C:\\Users\\aasth\\Downloads\\archive (1)")
> SmsData = read.csv("C:\\Users\\aasth\\Downloads\\archive (1)\\sms_spam.csv")
> str(SmsData)
'data.frame':	5574 obs. of  2 variables:
 $ type: chr  "ham" "ham" "spam" "ham" ...
 $ text: chr  "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wa
t..." "Ok lar... Joking wif u oni..." "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 t
o receive entry question("| __truncated__ "U dun say so early hor... U c already then say..." ...
> colnames(SmsData)[1]
[1] "type"
> colnames(SmsData)[2]
[1] "text"
> |
```

SmsData$type = factor(SmsData$type)

str(SmsData)

table(SmsData$type)

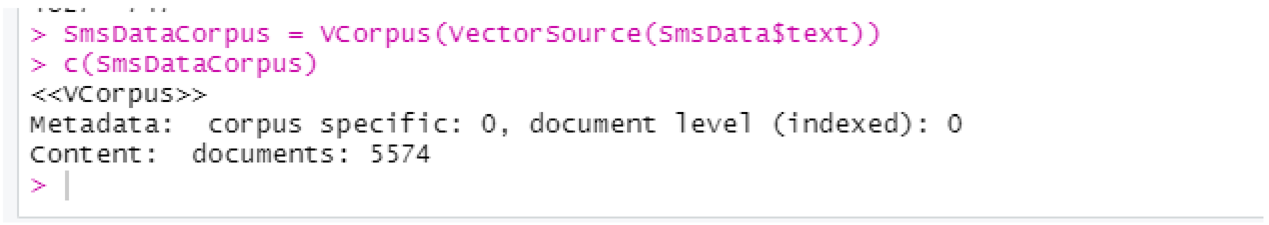


```
> SmsData$type = factor(SmsData$type)
> str(SmsData)
'data.frame':	5574 obs. of  2 variables:
 $ type: Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
 $ text: chr  "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wa
t..." "Ok lar... Joking wif u oni..." "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 t
o receive entry question("| __truncated__ "U dun say so early hor... U c already then say..." ...
> table(SmsData$type)

 ham spam 
4827  747 
> |
```

#Use the tm package to create a Corpus of documents (Hint: Construct the corpus using a VectorSource of the text column).

SmsDataCorpus = VCorpus(VectorSource(SmsData$text))

c(SmsDataCorpus)



```
> SmsDataCorpus = VCorpus(VectorSource(SmsData$text))
> c(SmsDataCorpus)
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 5574
> |
```

#Apply the following transformations from the tm package to the corpus in order prepare the data: a) Convert lowercase, b) Remove stopwords, c) Strip whitespace, and d) Remove punctuation.

#Stemming

SmsDataCorpusClean = tm_map(SmsDataCorpus, stemDocument)

#Convert to lower case

SmsDataCorpusClean = tm_map(SmsDataCorpusClean, content_transformer(tolower))

as.character(SmsDataCorpusClean[[1]])

#Remove stop words

SmsDataCorpusClean = tm_map(SmsDataCorpusClean, removeWords, stopwords())

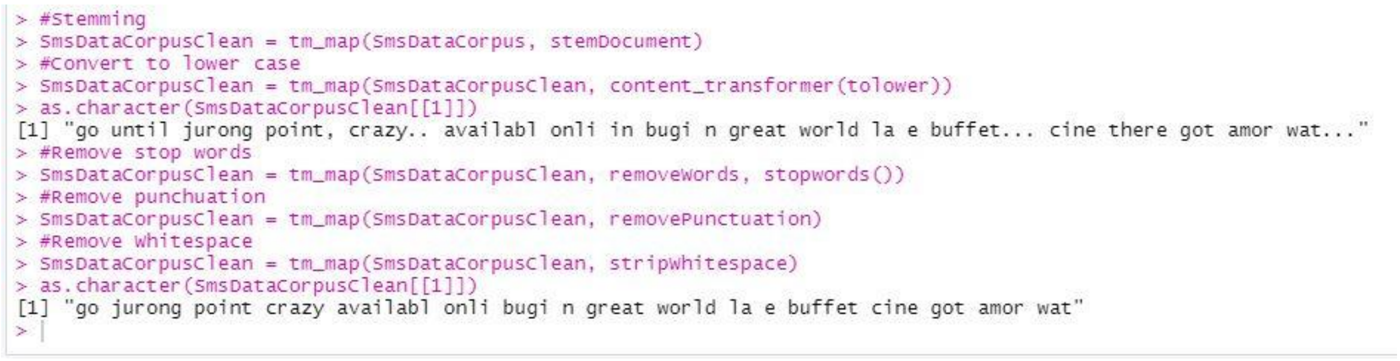
#Remove punchuation

SmsDataCorpusClean = tm_map(SmsDataCorpusClean, removePunctuation)

#Remove Whitespace

SmsDataCorpusClean = tm_map(SmsDataCorpusClean, stripWhitespace)

as.character(SmsDataCorpusClean[[1]])

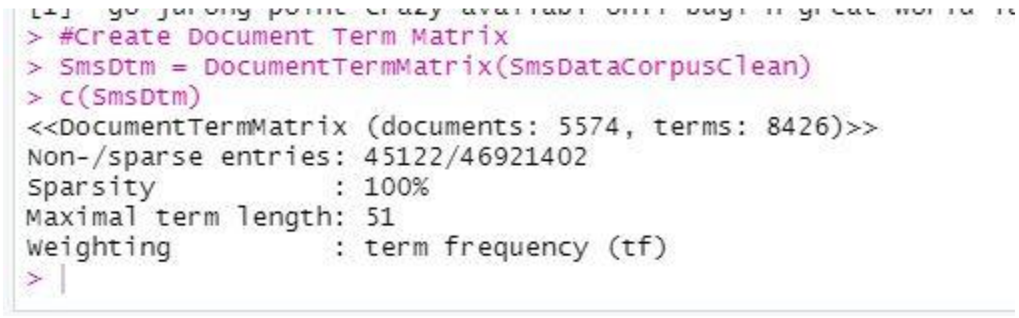


```
> #Stemming
> SmsDataCorpusClean = tm_map(SmsDataCorpus, stemDocument)
> #Convert to lower case
> SmsDataCorpusClean = tm_map(SmsDataCorpusClean, content_transformer(tolower))
> as.character(SmsDataCorpusClean[[1]])
[1] "go until jurong point, crazy.. availabl onli in bugi n great world la e buffet... cine there got amor wat..."
> #Remove stop words
> SmsDataCorpusClean = tm_map(SmsDataCorpusClean, removeWords, stopwords())
> #Remove punchuation
> SmsDataCorpusClean = tm_map(SmsDataCorpusClean, removePunctuation)
> #Remove whitespace
> SmsDataCorpusClean = tm_map(SmsDataCorpusClean, stripWhitespace)
> as.character(SmsDataCorpusClean[[1]])
[1] "go jurong point crazy availabl onli bugi n great world la e buffet cine got amor wat"
> 
```

#Create Document Term Matrix

SmsDtm = DocumentTermMatrix(SmsDataCorpusClean)

c(SmsDtm)



```
> #Create Document Term Matrix
> SmsDtm = DocumentTermMatrix(SmsDataCorpusClean)
> c(SmsDtm)
<<DocumentTermMatrix (documents: 5574, terms: 8426)>>
Non-/sparse entries: 45122/46921402
Sparsity           : 100%
Maximal term length: 51
Weighting          : term frequency (tf)
> 
```

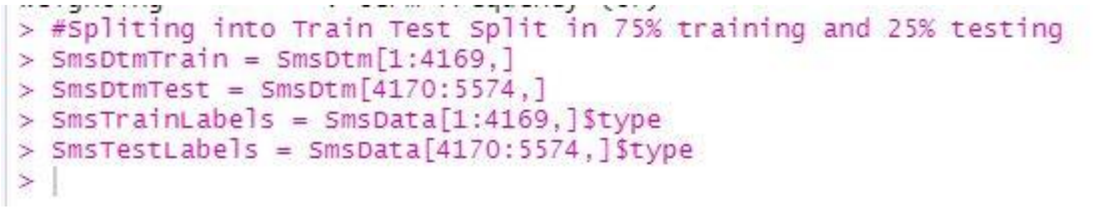
#Spliting into Train Test Split in 75% training and 25% testing

SmsDtmTrain = SmsDtm[1:4169,]

SmsDtmTest = SmsDtm[4170:5574,]

SmsTrainLabels = SmsData[1:4169,]$type

SmsTestLabels = SmsData[4170:5574,]$type



```
> #Spliting into Train Test Split in 75% training and 25% testing
> SmsDtmTrain = SmsDtm[1:4169,]
> SmsDtmTest  = SmsDtm[4170:5574,]
> SmsTrainLabels = SmsData[1:4169,]$type
> SmsTestLabels = SmsData[4170:5574,]$type
> 
```

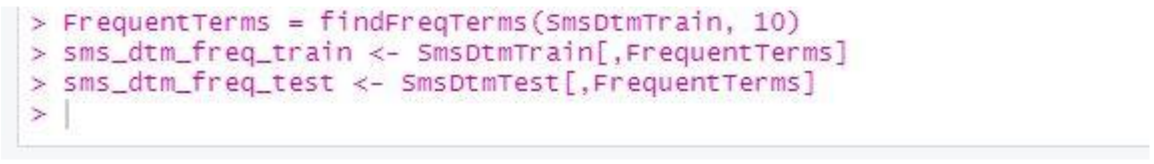
#Using FindFreqTerms

Use findFreqTerms to contruct features from words occuring more than 10 times and proceed to split the data into a training and test set - for each create a DocumentTermMatrix.

FrequentTerms = findFreqTerms(SmsDtmTrain, 10)

sms_dtm_freq_train <- SmsDtmTrain[,FrequentTerms]

sms_dtm_freq_test <- SmsDtmTest[,FrequentTerms]



```
> FrequentTerms = findFreqTerms(SmsDtmTrain, 10)
> sms_dtm_freq_train <- SmsDtmTrain[,FrequentTerms]
> sms_dtm_freq_test <- SmsDtmTest[,FrequentTerms]
> 
```

#Function to convert to boolean

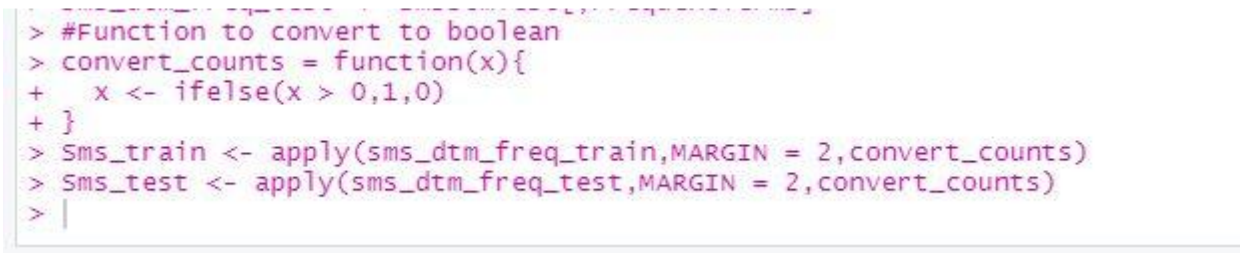
convert_counts = function(x){

  x <- ifelse(x > 0,1,0)

}

Sms_train <- apply(sms_dtm_freq_train,MARGIN = 2,convert_counts)

Sms_test <- apply(sms_dtm_freq_test,MARGIN = 2,convert_counts)



```
> #Function to convert to boolean
> convert_counts = function(x){
+   x <- ifelse(x > 0,1,0)
+ }
> Sms_train <- apply(sms_dtm_freq_train,MARGIN = 2,convert_counts)
> Sms_test <- apply(sms_dtm_freq_test,MARGIN = 2,convert_counts)
> |
```

#Building Model

SmsClassifier = naiveBayes(Sms_train, SmsTrainLabels)

sms_train_pred <- predict(SmsClassifier,Sms_train)

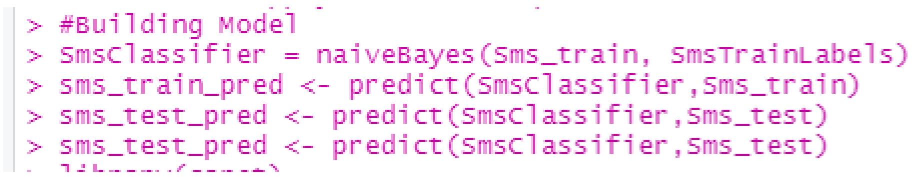
sms_test_pred <- predict(SmsClassifier,Sms_test)

#Displaying the accuracy

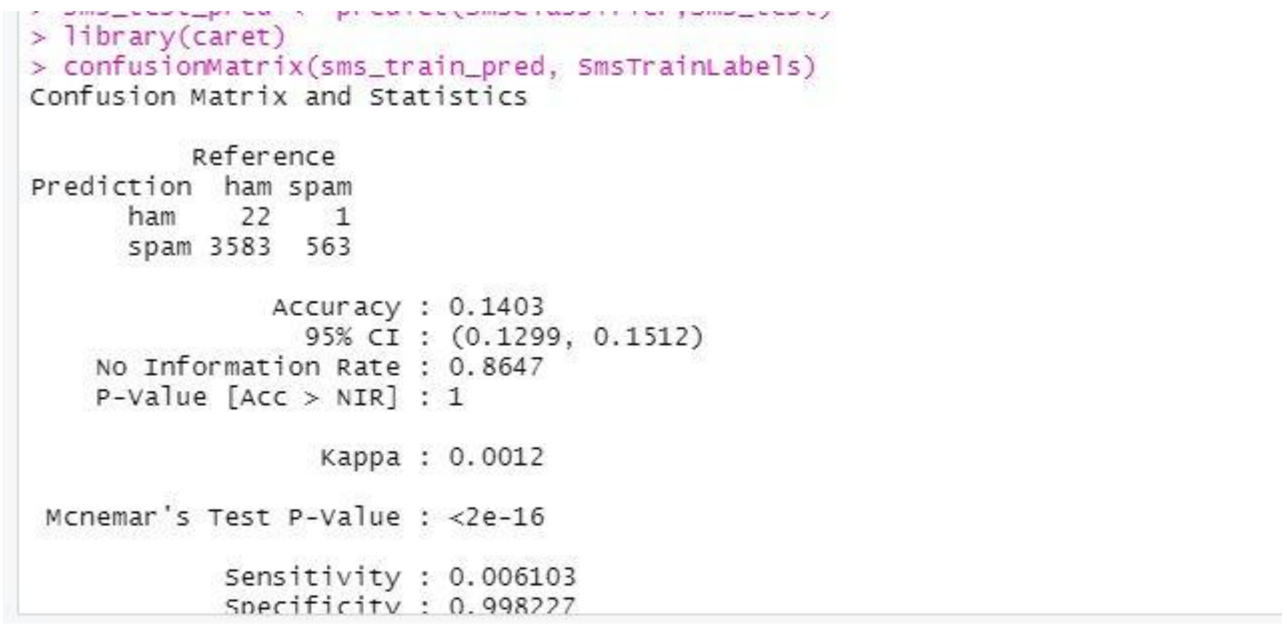
library(caret)

confusionMatrix(sms_train_pred, SmsTrainLabels)

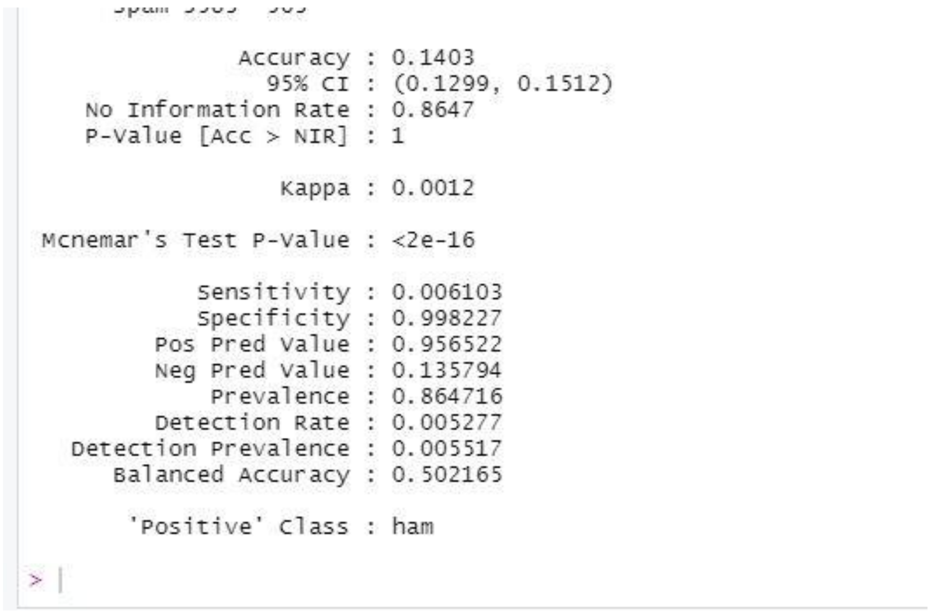
confusionMatrix(sms_test_pred, SmsTestLabels)



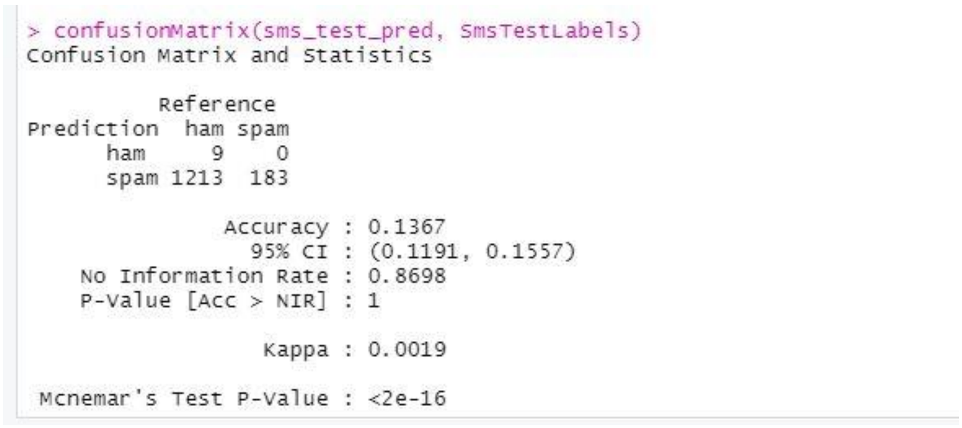
```
> #Building Model
> SmsClassifier = naiveBayes(Sms_train, SmsTrainLabels)
> sms_train_pred <- predict(SmsClassifier,Sms_train)
> sms_test_pred <- predict(SmsClassifier,Sms_test)
> sms_test_pred <- predict(SmsClassifier,Sms_test)
```



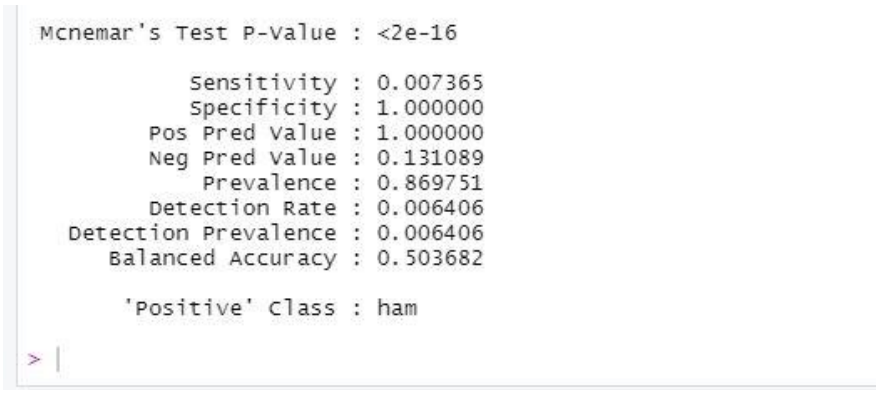
```
> library(caret)
> confusionMatrix(sms_train_pred, SmsTrainLabels)
Confusion Matrix and Statistics

          Reference
Prediction  ham spam
      ham    22    1
      spam 3583  563

               Accuracy : 0.1403
                 95% CI : (0.1299, 0.1512)
    No Information Rate : 0.8647
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0012

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.006103
            Specificity : 0.998227
```



```
               Accuracy : 0.1403
                 95% CI : (0.1299, 0.1512)
    No Information Rate : 0.8647
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0012

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.006103
            Specificity : 0.998227
         Pos Pred Value : 0.956522
         Neg Pred Value : 0.135794
             Prevalence : 0.864716
         Detection Rate : 0.005277
   Detection Prevalence : 0.005517
      Balanced Accuracy : 0.502165

       'Positive' Class : ham

> |
```



```
> confusionMatrix(sms_test_pred, SmsTestLabels)
Confusion Matrix and Statistics

          Reference
Prediction  ham spam
      ham     9    0
      spam 1213  183

               Accuracy : 0.1367
                 95% CI : (0.1191, 0.1557)
    No Information Rate : 0.8698
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0019

 Mcnemar's Test P-Value : <2e-16
```



```
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.007365
            Specificity : 1.000000
         Pos Pred Value : 1.000000
         Neg Pred Value : 0.131089
             Prevalence : 0.869751
         Detection Rate : 0.006406
   Detection Prevalence : 0.006406
      Balanced Accuracy : 0.503682

       'Positive' Class : ham

> |
```

The accuracy of the Training Set is 14.03% and the accuracy of the Testing Set is 13.67% respectively.

Submitted By: -

Aastha Dhir

CWID- A20468022

adhir2@hawk.iit.edu