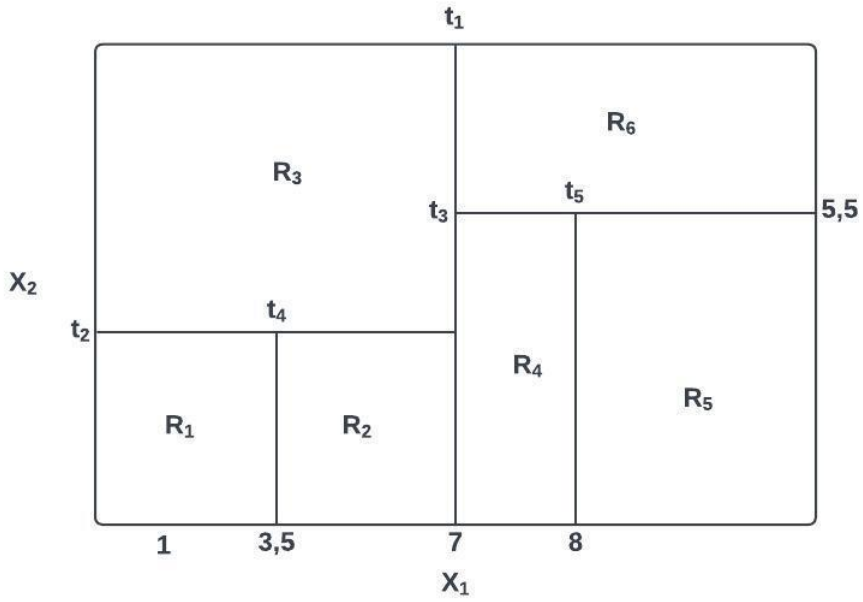
Assignment 4

Abhiram Ravipati

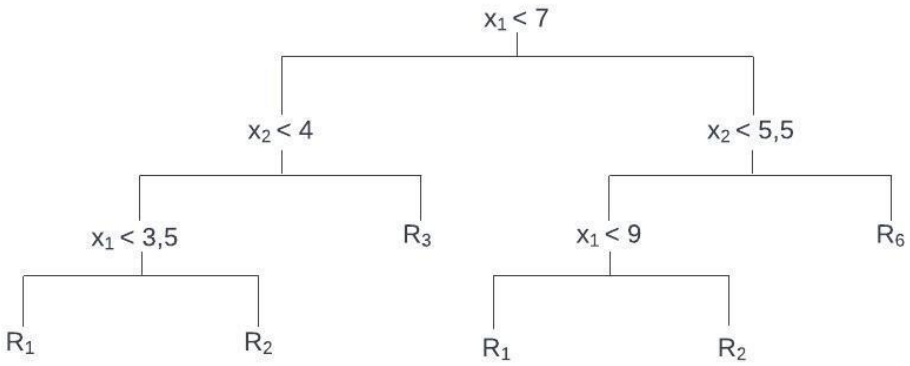
2024-04-09

Chapter 8

1)

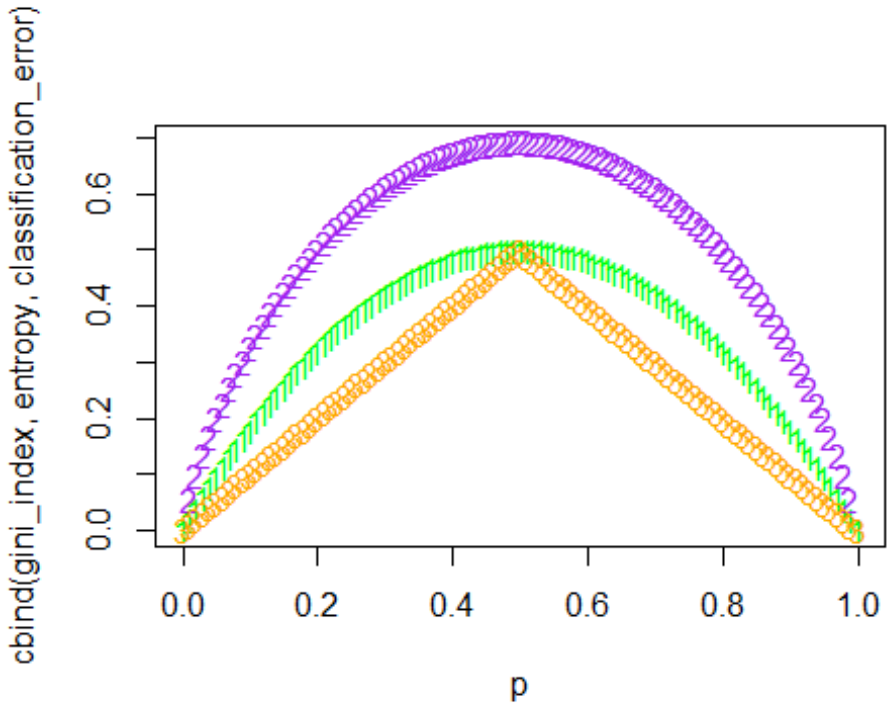


Decision Tree:

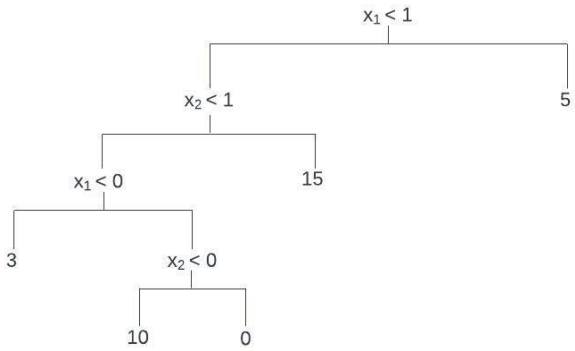


3)

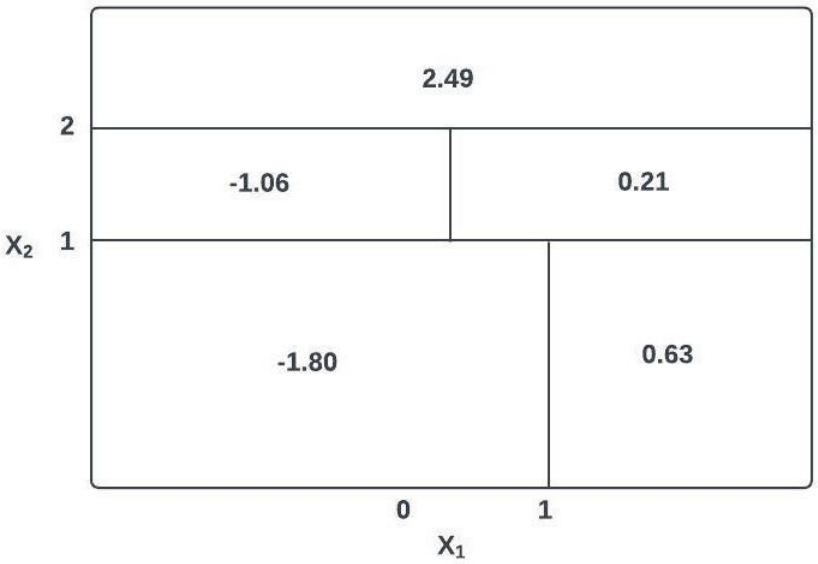
```r
#Consider the value of p as follows
p = seq(0, 1, 0.01)

#Let us calculate the values gini index, entropy, classification error
gini_index = p * (1 - p) + (1 - p) * p
entropy = -(p * log(p) + (1 - p) * log(1 - p))
classification_error = 1 - pmax(p, 1 - p)
matplot(p, cbind(gini_index, entropy, classification_error), col = c("green",
"purple", "orange"))
```



4) a)



b)



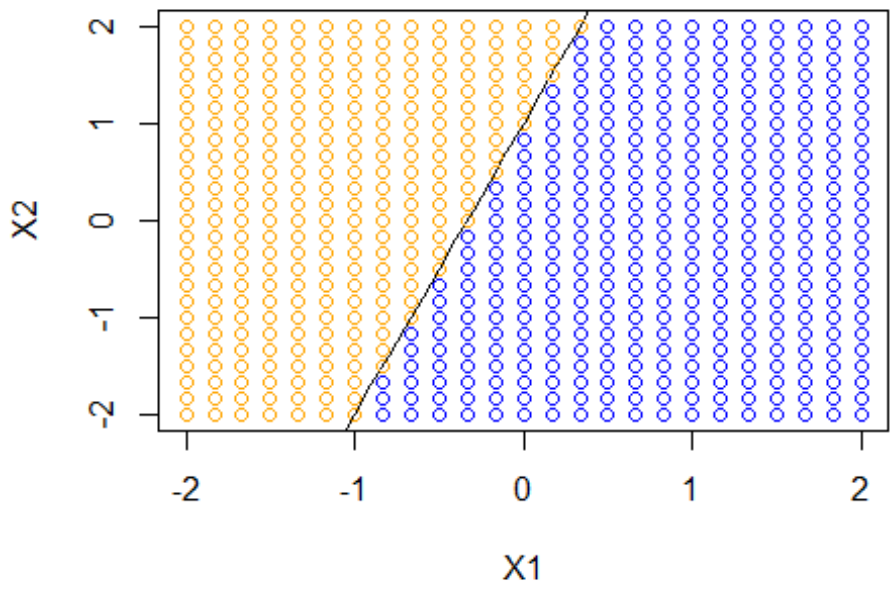
5) Major Vote Approach: When making predictions based on the estimates provided we observe that there are 6 values greater than 0.5 and 4 values less than 0.5 which serves as the boundary for class assignment. Therefore, we classify the specific value of X into the class labeled as RED.

Average Approach: Using this procedure, we will take the average of the supplied values and get an average value of 0.45, which is less than 0.5. As a result using the AVERAGE method to identify a particular value of X that is related to the previously described estimates would result in its classification as GREEN.

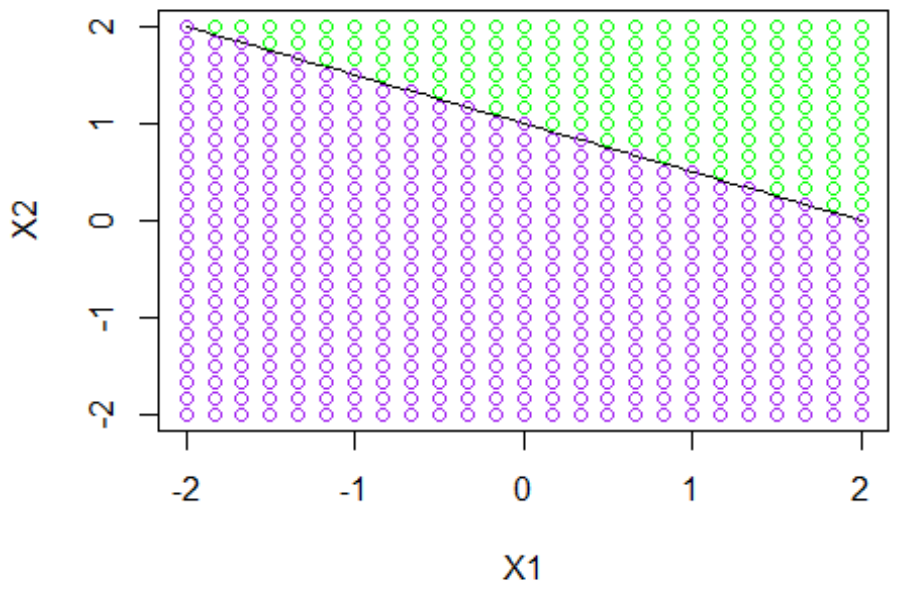
Chapter 9:

1) a, b)

```
X1 = seq(-2, 2, 0.1)
X2 = 1 + 3 * X1
plot(X1, X2, xlab = "X1", ylab = "X2", type = "l", xlim = c(-2, 2), ylim =
c(-2, 2))

for(i in seq(-2, 2, length.out = 25)){
  point = data.frame(rep(i, 25), seq(-2, 2, length.out = 25))
  points(point, col = ifelse(1 + 3 * point[,1]-point[,2] > 0,
"blue","orange"))
}
```

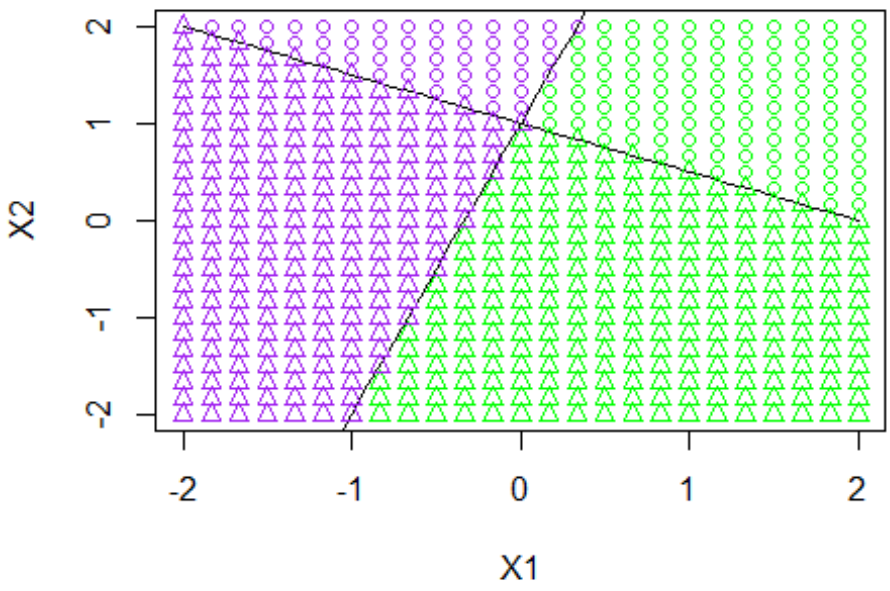



```r
X1 = seq(-2, 2, 0.1)
X2 = 1 - X1 / 2
plot(X1, X2, xlab = "X1", ylab = "X2", type = "l", xlim = c(-2, 2), ylim =
c(-2, 2))

for(i in seq(-2, 2, length.out = 25)){
  point = data.frame(rep(i, 25), seq(-2, 2, length.out = 25))
  points(point, col = ifelse(-2 + point[,1]+point[,2]*2 > 0,
"green","purple"))
}
```



Representing both in the same Plot:

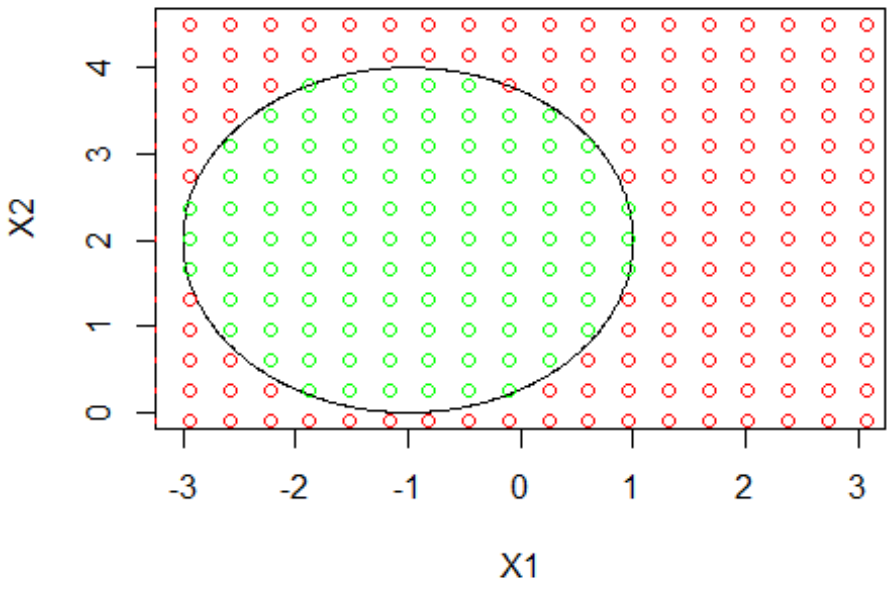
```
X1 = seq(-2, 2, 0.1)
X2 = 1 + 3 * X1
plot(X1, X2, xlab = "X1", ylab = "X2", type = "l", xlim = c(-2, 2), ylim =
c(-2, 2))
lines(X1,1 - 1/2*X1)

for(i in seq(-2, 2, length.out = 25)){
  point = data.frame(rep(i, 25), seq(-2, 2, length.out = 25))
  points(point, col = ifelse(1 + 3 * point[,1]-point[,2] > 0,
"green","purple"),
  pch = ifelse(-2 + point[,1]+point[,2]*2>0,1,2))
}
```



2) a, b, c)

Consider that the equation $(1 + X_1)^2 + (2 - X_2)^2 = 4$ represents a circle with a radius of 2. Upon further solving, we obtain $X_2 = 2 - \sqrt{4 - (1 + X_1)^2}$. This expression is utilized in our program.

```
X1 = seq(-3, 10, 0.01)
X2 <- 2 - sqrt (4-(1 + X1)^2)

## Warning in sqrt(4 - (1 + X1)^2): NaNs produced

X3 <- 2 + sqrt (4-(1 + X1)^2)

## Warning in sqrt(4 - (1 + X1)^2): NaNs produced

plot(X1, X2, xlab = "X1", ylab = "X2", type = "l", xlim = c(-3, 3), ylim =
c(0, 4.5))
lines(X1, X3)

for (i in seq(-4, 4.5, length.out = 25)){
  point = data.frame(rep(i,25), seq(-4, 4.5, length.out = 25))
  points(point, col = ifelse((1+point[,1])^2+(2-point[,2])^2>4, "red",
"green"))
}
```



The green points depicted in the image above indicate the points that meet the condition

$$(1 + X_1)^2 + (2 - X_2)^2 < 4$$

The red points shown in the image above denote the points that fulfill the condition

$$(1 + X_1)^2 + (2 - X_2)^2 > 4$$

The points (0, 0), (2, 2), and (3, 8) are located outside the circle and are classified as red. Conversely, the point (-1, 1) falls inside the circle and is classified as green.

d) The decision boundary for the above mentioned scenario corresponds to the equation of a circle, namely $(1 + X_1)^2 + (2 - X_2)^2 - 4 = 0$, which can be simplified as follows:

$$1 + X_1^2 + 2X_1 + 4 + X_2^2 - 4X_2 - 4 = 0$$

This further simplifies to:

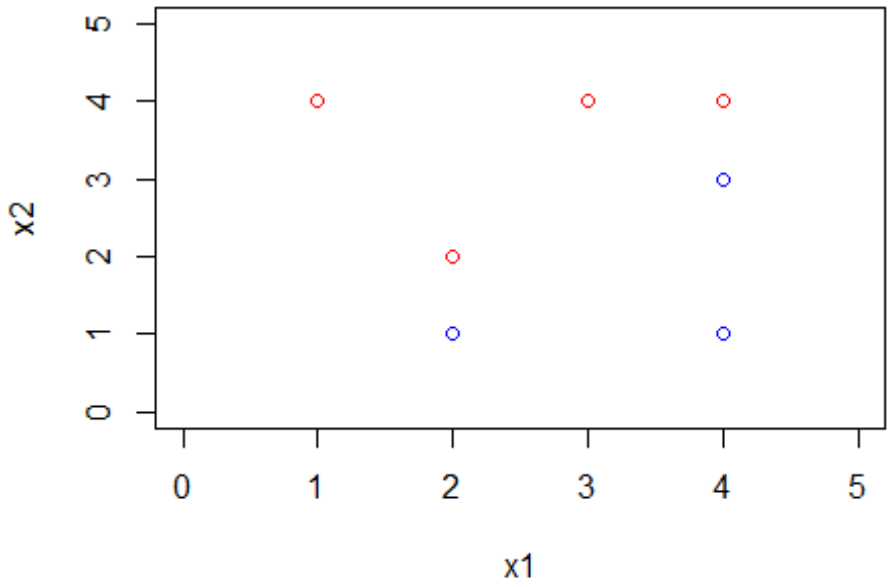
$$1 + 2X_1 - 4X_2 + X_1^2 + X_2^2 = 0$$

which is of the form $a + bT1 + cT2 + dT3 + eT4 = 0$, where $T1 = X_1$, $T2 = X_2$, $T3 = X_1^2$, and $T4 = X_2^2$.

From the above, we can say that while the decision boundary in (c) is not linear in terms of X_1 and X_2, it is linear in terms of X_1, X_1^2, X_2, and X_2^2.

3)a)Plotting the Observations given

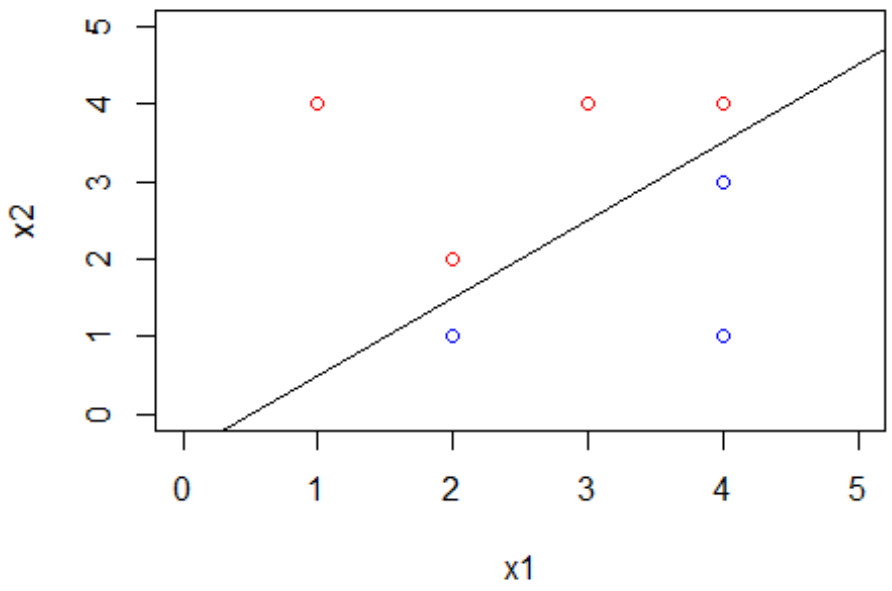
```
x1 = c(3, 2, 4, 1, 2, 4, 4)
x2 = c(4, 2, 4, 4, 1, 3, 1)
colors = c("red", "red", "red", "red", "blue", "blue", "blue", "blue")
plot(x1, x2, col = colors, xlim = c(0,5), ylim = c(0,5))
```



b) Optimal Hyper Plane: From the plot above, we deduce that the optimal hyperplane, derived from the provided observations, should intersect the midpoints of (2,1) and (2,2) as well as the midpoints of (4,3) and (4,4). This suggests a plane passing through the points (2,1.5) and (4,3.5), which can be expressed by the equation $X_2 - X_1 + 0.5 = 0$.

Plotting the above in graph

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

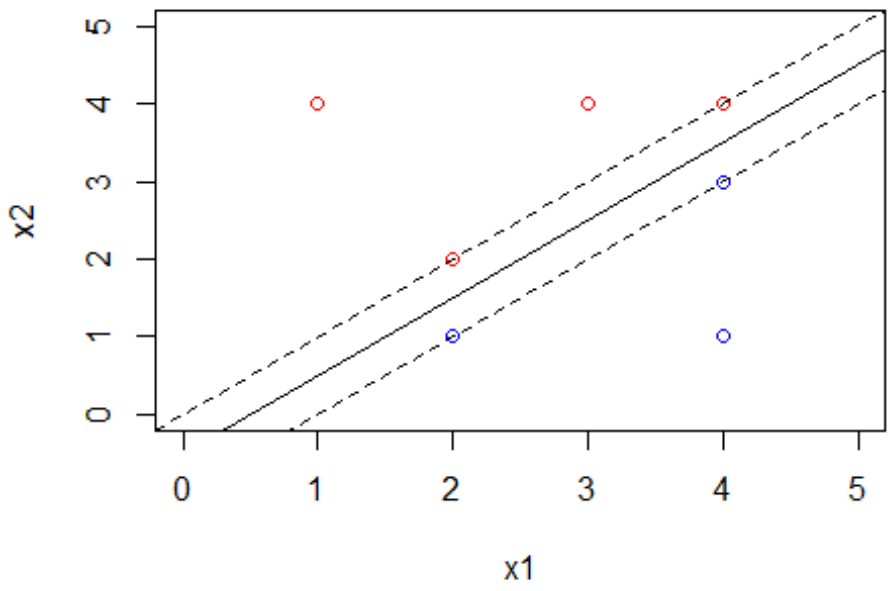


c) Maximum Margin Classifier: The optimal separating hyperplane has the equation $X_2 = X_1 - 0.5$. So the classification will be Red when $X_2 > X_1 - 0.5 \Leftrightarrow X_2 - X_1 + 0.5 > 0$, and Blue otherwise.

We have $(\beta_0, \beta_1, \beta_2) = (0.5, -1, 1)$.

d)Margin for Maximum 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)
```



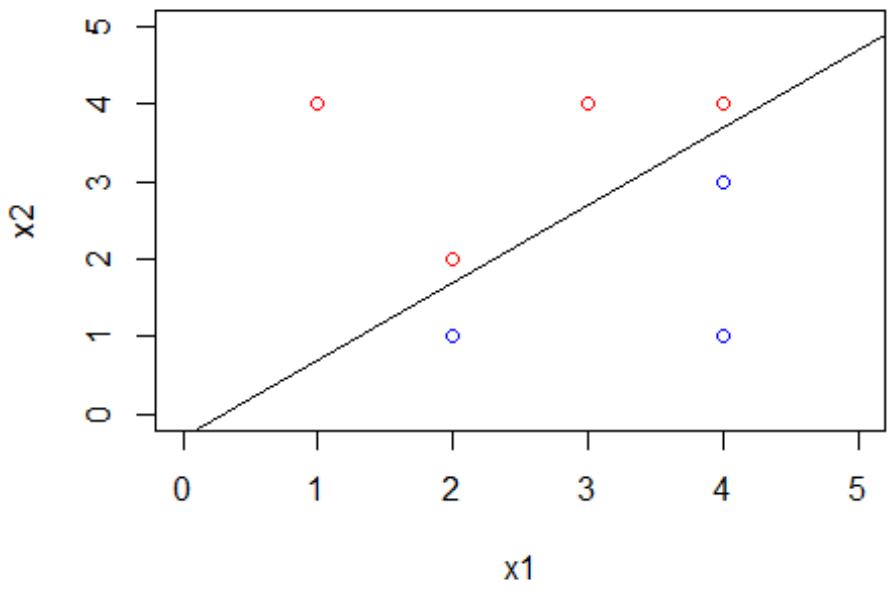
The margin refers to the distance from the solid line to either of the dashed lines.

e) Support Vectors: The graph clearly indicates that the support vectors correspond to the points (2,1), (2,2), (4,3), and (4,4).

f) Upon analyzing the plot above it's clear that even if we relocate the 7th observation (4,1) it wouldn't affect the maximal margin hyperplane since it isn't a support vector.

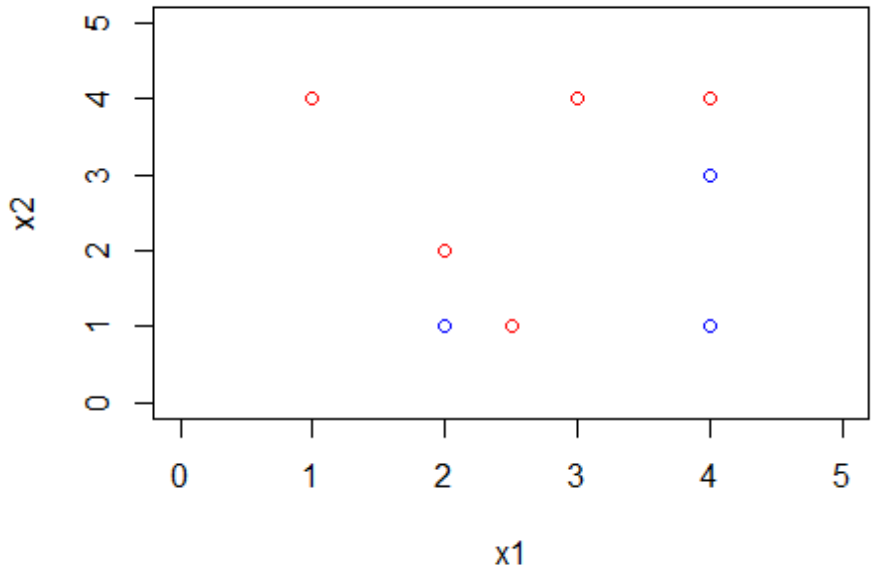
g) We have the maximal margin hyperplane as $X_1 - X_2 - 0.5 = 0$. For example, the hyperplane with the equation $X_1 - X_2 - 0.1 = 0$ is not the optimal separating hyperplane.

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



h)Drawing Additional Observations:

```
plot(x1, x2, col = colors, xlim = c(0,5), ylim = c(0,5))
points(c(2.5), c(1), col = c("red"))
```



It should be noted that the present maximal hyperplane is no longer an accurate classifier when it contains only one observation (2.5,1) and classifies it as red.

Practicum Problems:

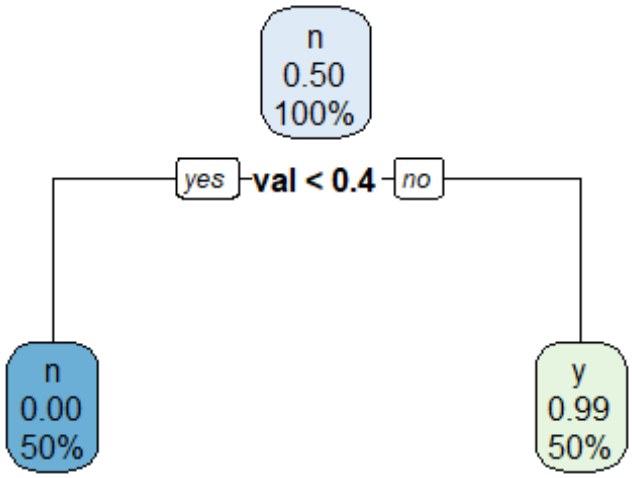
Problem 1:

```
library(caret)

## Warning: package 'caret' was built under R version 4.3.3

## Loading required package: ggplot2

## Loading required package: lattice

library(tree)

## Warning: package 'tree' was built under R version 4.3.3

library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 4.3.3

## Loading required package: rpart

gini <- function(p)
{
  gini_index = 2 * p * (1 - p)
  return (gini_index)
}
entropy <- function(p)
{
  value_entropy = (p * log(p) + (1 - p) * log(1 - p))
  return (value_entropy)
}
```

The parameters for the normal distribution should be (5,2) and (-5,2) for the given pair of samples.

```
set.seed(200)

a <- rnorm(n=200, mean = 5, sd = 2)
b <- rnorm(n=200, mean = -5, sd = 2)

dataframe1 <- data.frame(val = a, label = rep("y", 200))

dataframe2 <- data.frame(val = b, label = rep("n", 200))

dataframe <- rbind(dataframe1, dataframe2)

dataframe$label <- as.factor(dataframe$label)
```

```
decisiontree <- rpart(label~val, dataframe, method = "class")

rpart.plot(decisiontree)
```



The analysis indicates that the threshold value for the initial split will be 0.4. The tree comprises one root node and two leaf nodes. Furthermore, it effectively classifies both classes individually, demonstrating a clear empirical distribution.

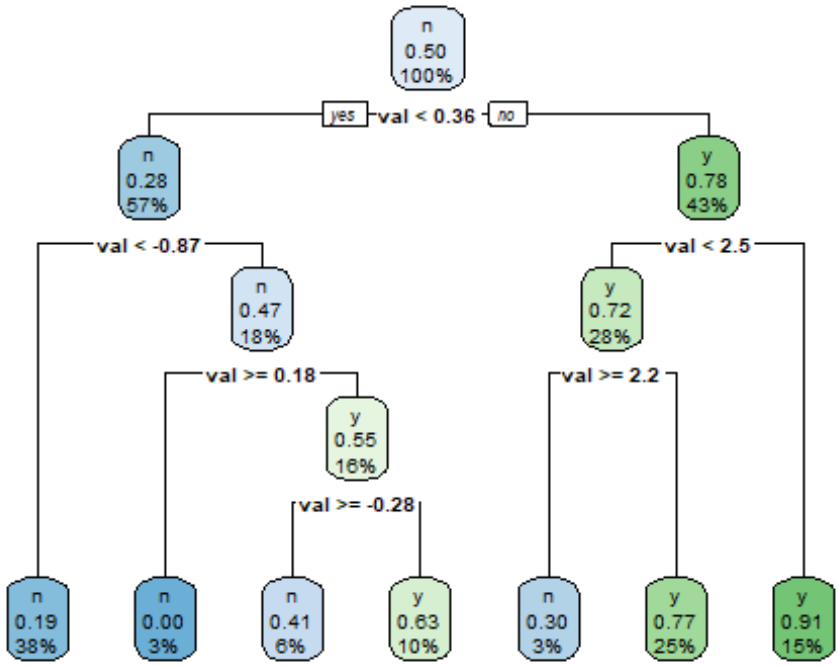
When computing the Gini impurity and entropy for each node, we utilize the probabilities associated with the node's classes. Let's denote prob as the probability of each class within a node.

```
prob = c(0.4, 0, 0.99)
ginival = sapply(prob, gini)
ginival

## [1] 0.4800 0.0000 0.0198

entropyval = sapply(prob, entropy)
entropyval

## [1] -0.67301167         NaN -0.05600153
```

The Gini values for the given above tree are 0.48, 0.0, and 0.0198, respectively. For entropy, the corresponding values are -0.673, NaN, and -0.056.

```
set.seed(150)
a1 <- rnorm(n = 150, mean = 1, sd = 2)
```

```
b1 <- rnorm(n = 150, mean = -1, sd = 2)
dataframe_three <- data.frame(val = a1, label = rep("y", 150))
dataframe_four <- data.frame(val = b1, label = rep("n",150))
dataframebind <- rbind(dataframe_three, dataframe_four)

dataframebind$label <- as.factor(dataframebind$label)
decisiontree1 <- rpart(label~val, dataframebind, method = "class")
rpart.plot(decisiontree1)
```



The given tree indicates a threshold value of 0.36 for the initial split. It comprises a total of 13 nodes, with one serving as the root node and seven as leaf nodes. The sizeable tree suggests the presence of numerous distinct labels within its nodes, leading to its extensive structure. Consequently, this tree exhibits a higher degree of label overlap across its nodes.

When computing the Gini and entropy for each node, we consider the probabilities associated with each node's classes. Let's denote prob1 as the probability of each class within a node. Then, we can use these probabilities to calculate the Gini impurity and entropy for each node.

```
prob1 <- c(0.5, 0.22, 0.72, 0.28, 0.53, 0.45, 0.09, 0.23, 0.70, 0.37, 0.59,
1.0, 0.81)
ginival1 <- sapply(prob1, gini)
ginival1

##  [1] 0.5000 0.3432 0.4032 0.4032 0.4982 0.4950 0.1638 0.3542 0.4200 0.4662
## [11] 0.4838 0.0000 0.3078
```

```
entropyval1 <- sapply(prob1, entropy)
entropyval1
```

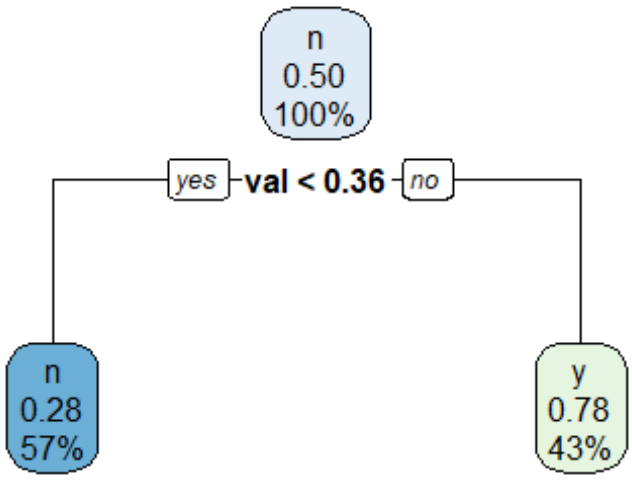
```
##  [1] -0.6931472 -0.5269080 -0.5929533 -0.5929533 -0.6913461 -0.6881388
##  [7] -0.3025378 -0.5392763 -0.6108643 -0.6589557 -0.6768585        NaN
## [13] -0.4862230
```

The Gini values for the given tree nodes are as follows: 0.5000, 0.3432, 0.4032, 0.4032, 0.4982, 0.4950, 0.1638, 0.3542, 0.4200, 0.4662, 0.4838, 0.0000, 0.3078

The entropy values for the provided tree nodes are as follows: -0.6931472, -0.5269080, -0.5929533, -0.5929533, -0.6913461, -0.6881388, -0.3025378, -0.5392763, -0.6108643, -0.6589557, -0.6768585, NaN, -0.4862230

Pruning:

```
tree_new <- prune.rpart(decisiontree1, cp = 0.1)
rpart.plot(tree_new)
```



The pruned tree consists of one root node and two leaf nodes. Comparatively, this pruned version exhibits significant improvement over the previous tree as it features only two leaf nodes resulting in reduced overlap among labels.

When computing the Gini and entropy for each node, we utilize the probabilities associated with each node's classes. Let's denote prob2 as the probability of each class within a node. Then, we can apply these probabilities to calculate the Gini and entropy for each node.

```
prob2 <- c(0.5, 0.22, 0.72)
ginival2 <- sapply(prob2, gini)
ginival2

## [1] 0.5000 0.3432 0.4032

entropyval2 = sapply(prob2, entropy)
entropyval2

## [1] -0.6931472 -0.5269080 -0.5929533
```

The Gini values for the given tree nodes are: 0.5000, 0.3432, 0.4032. The entropy values for the provided tree nodes are: -0.6931472, -0.5269080, -0.5929533.

Problem 2:

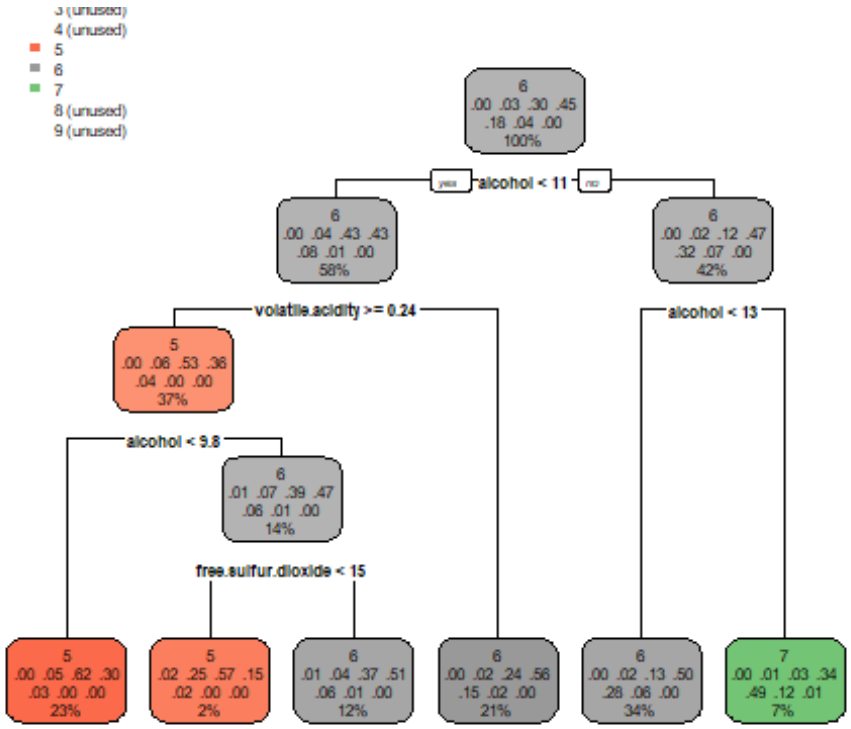
```
library(rpart.plot)
library(randomForest)

## Warning: package 'randomForest' was built under R version 4.3.3

## randomForest 4.7-1.1

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

library(caret)
library(rpart)

redwine_dataset <-
read.csv("C:/Users/Abhiram/Downloads/wine+quality/winequality-red.csv", sep =
";")

whitewine_dataset <-
  read.csv("C:/Users/Abhiram/Downloads/wine+quality/winequality-white.csv",
sep = ";")

str(redwine_dataset)

## 'data.frame':    1599 obs. of  12 variables:
##  $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58
0.5 ...
##  $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
##  $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069
0.065 0.073 0.071 ...
```

```
##  $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
##  $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
##  $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
##  $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36
3.35 ...
##  $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57
0.8 ...
##  $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
##  $ quality             : int  5 5 5 6 5 5 5 7 7 5 ...

str(whitewine_dataset)

## 'data.frame':    4898 obs. of  12 variables:
##  $ fixed.acidity       : num  7 6.3 8.1 7.2 7.2 8.1 6.2 7 6.3 8.1 ...
##  $ volatile.acidity    : num  0.27 0.3 0.28 0.23 0.23 0.28 0.32 0.27 0.3
0.22 ...
##  $ citric.acid         : num  0.36 0.34 0.4 0.32 0.32 0.4 0.16 0.36 0.34
0.43 ...
##  $ residual.sugar      : num  20.7 1.6 6.9 8.5 8.5 6.9 7 20.7 1.6 1.5 ...
##  $ chlorides           : num  0.045 0.049 0.05 0.058 0.058 0.05 0.045
0.045 0.049 0.044 ...
##  $ free.sulfur.dioxide : num  45 14 30 47 47 30 30 45 14 28 ...
##  $ total.sulfur.dioxide: num  170 132 97 186 186 97 136 170 132 129 ...
##  $ density             : num  1.001 0.994 0.995 0.996 0.996 ...
##  $ pH                  : num  3 3.3 3.26 3.19 3.19 3.26 3.18 3 3.3 3.22
...
##  $ sulphates           : num  0.45 0.49 0.44 0.4 0.4 0.44 0.47 0.45 0.49
0.45 ...
##  $ alcohol             : num  8.8 9.5 10.1 9.9 9.9 10.1 9.6 8.8 9.5 11 ...
##  $ quality             : int  6 6 6 6 6 6 6 6 6 6 ...
```

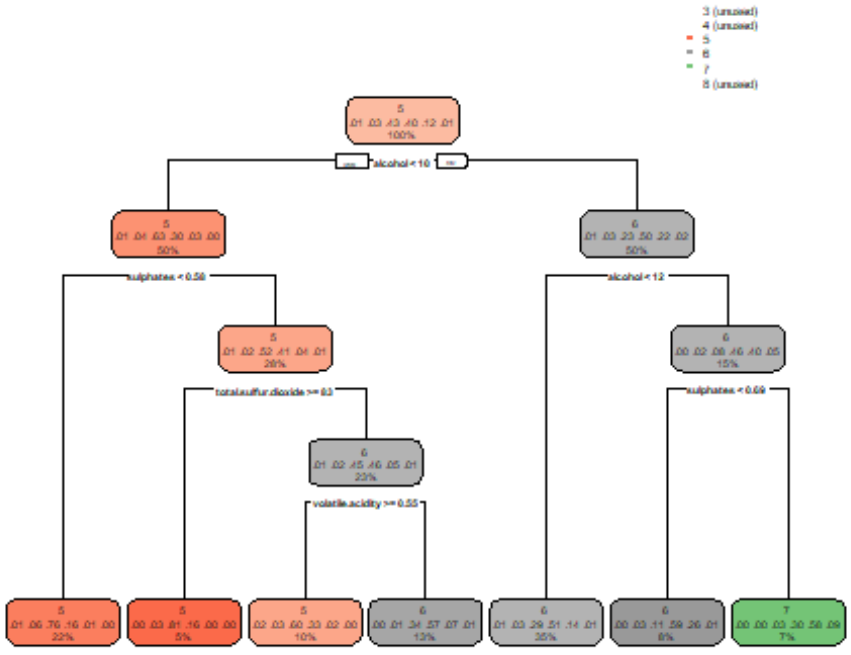
The white wine dataset comprises 4898 samples and 12 variables, while the red wine dataset consists of 1599 samples and 12 variables.

Splitting Train and Test Sets:

```
whitewine_dataset$quality <- as.factor(whitewine_dataset$quality)
whitewine_Train <- createDataPartition(whitewine_dataset$quality, p = 0.8,
list = F)
whitewine_TrainSet <- whitewine_dataset[whitewine_Train,]
whitewine_TestSet <- whitewine_dataset[-whitewine_Train,]

redwine_dataset$quality <- as.factor(redwine_dataset$quality)
redwine_Train <- createDataPartition(redwine_dataset$quality, p = 0.8, list =
F)
redwine_TrainSet <- redwine_dataset[redwine_Train,]
redwine_TestSet <- redwine_dataset[-redwine_Train,]

whitewine_decisiontree <- rpart(quality~., data = whitewine_TrainSet)
rpart.plot(whitewine_decisiontree)
```




```
redwine_decisiontree <- rpart(quality~., data = redwine_TrainSet)
rpart.plot(redwine_decisiontree)
```



Confusion Matrix for Red and White wine:

```
redwine_decisiontree_predict <- predict(redwine_decisiontree,
redwine_TestSet, type = "class")
confusionMatrix(redwine_decisiontree_predict, redwine_TestSet$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  1  4 84 32  0  0
##          6  1  6 52 85 26  0
##          7  0  0  0 10 13  3
##          8  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.5741
##                  95% CI : (0.5176, 0.6292)
##     No Information Rate : 0.429
##     P-Value [Acc > NIR] : 1.451e-07
##
##                   Kappa : 0.3033
##
##  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   0.6176   0.6693  0.33333 0.000000
## Specificity          1.000000  1.00000   0.7956   0.5526  0.95324 1.000000
## Pos Pred Value            NaN      NaN   0.6942   0.5000  0.50000      NaN
## Neg Pred Value       0.993691  0.96845   0.7347   0.7143  0.91065 0.990536
## Prevalence           0.006309  0.03155   0.4290   0.4006  0.12303 0.009464
## Detection Rate       0.000000  0.00000   0.2650   0.2681  0.04101 0.000000
## Detection Prevalence 0.000000  0.00000   0.3817   0.5363  0.08202 0.000000
## Balanced Accuracy    0.500000  0.50000   0.7066   0.6110  0.64329 0.500000
```

```
whitewine_decisiontree_predict <- predict(whitewine_decisiontree,
whitewine_TestSet, type = "class")
confusionMatrix(whitewine_decisiontree_predict, whitewine_TestSet$quality)
```

```
## 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   1  16 148  86   3   1   0
```

```
##          6   3  16 142 331 139  23   1
##          7   0   0   1  22  34  11   0
##          8   0   0   0   0   0   0   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5245
##                  95% CI : (0.4927, 0.5562)
##     No Information Rate : 0.4489
##     P-Value [Acc > NIR] : 1.235e-06
##
##                   Kappa : 0.2196
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000  0.00000   0.5086   0.7540  0.19318  0.00000
## Specificity           1.00000  1.00000   0.8443   0.3989  0.95761  1.00000
## Pos Pred Value            NaN      NaN   0.5804   0.5053  0.50000      NaN
## Neg Pred Value        0.99591  0.96728   0.8022   0.6656  0.84396  0.96421
## Prevalence            0.00409  0.03272   0.2975   0.4489  0.17996  0.03579
## Detection Rate        0.00000  0.00000   0.1513   0.3384  0.03476  0.00000
## Detection Prevalence  0.00000  0.00000   0.2607   0.6697  0.06953  0.00000
## Balanced Accuracy     0.50000  0.50000   0.6764   0.5764  0.57539  0.50000
##                      Class: 9
## Sensitivity          0.000000
## Specificity          1.000000
## Pos Pred Value            NaN
## Neg Pred Value       0.998978
## Prevalence           0.001022
## Detection Rate       0.000000
## Detection Prevalence 0.000000
## Balanced Accuracy    0.500000
```

For the White Wine Dataset, the Decision Tree achieved an accuracy of 52.45%, while for the Red Wine Dataset, it attained an accuracy of 57.41%.

In the White Wine Dataset, the initial split occurred at “alcohol < 11”, while in the Red Wine Dataset, it took place at “alcohol < 10”. Additionally, Sulphates and Total Sulfur Dioxide were considered in the Red Wine Dataset but not in the White Wine Dataset, whereas Free Sulfur Dioxide was considered in the White Wine Dataset but not in the Red Wine Dataset.

```
whitewine_randomforest_train <- randomForest(quality ~ ., data =
whitewine_TrainSet)
```

```
whitewine_randomforest_predict <- predict(whitewine_randomforest_train,
whitewine_TestSet)
confusionMatrix(whitewine_randomforest_predict, whitewine_TestSet$quality)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   0   0   0   0   0
##          4   0  10   4   2   0   0   0
##          5   1  12 202  68   1   1   0
##          6   3  10  83 349  72   8   0
##          7   0   0   2  20 101   9   1
##          8   0   0   0   0   2  17   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6943
##                  95% CI : (0.6643, 0.723)
##     No Information Rate : 0.4489
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5272
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000  0.31250   0.6942   0.7950   0.5739  0.48571
## Specificity           1.00000  0.99366   0.8792   0.6735   0.9601  0.99788
## Pos Pred Value            NaN  0.62500   0.7088   0.6648   0.7594  0.89474
## Neg Pred Value        0.99591  0.97713   0.8716   0.8013   0.9112  0.98123
## Prevalence            0.00409  0.03272   0.2975   0.4489   0.1800  0.03579
## Detection Rate        0.00000  0.01022   0.2065   0.3569   0.1033  0.01738
## Detection Prevalence  0.00000  0.01636   0.2914   0.5368   0.1360  0.01943
## Balanced Accuracy     0.50000  0.65308   0.7867   0.7342   0.7670  0.74180
##                      Class: 9
## Sensitivity          0.000000
## Specificity          1.000000
## Pos Pred Value            NaN
## Neg Pred Value       0.998978
## Prevalence           0.001022
## Detection Rate       0.000000
## Detection Prevalence 0.000000
## Balanced Accuracy    0.500000

redwine_randomforest_train <- randomForest(quality ~ ., data =
redwine_TrainSet)
```

```
redwine_randomforest_predict <- predict(redwine_randomforest_train,
redwine_TestSet)
confusionMatrix(redwine_randomforest_predict, redwine_TestSet$quality)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   0   0   0   0
##          4   1   0   0   0   0   0
##          5   0   5 110  30   2   0
##          6   1   5  26  89  19   1
##          7   0   0   0   8  18   2
##          8   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6845
##                  95% CI : (0.6303, 0.7353)
##     No Information Rate : 0.429
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4845
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.000000   0.8088   0.7008  0.46154 0.000000
## Specificity          1.000000 0.996743   0.7956   0.7263  0.96403 1.000000
## Pos Pred Value            NaN 0.000000   0.7483   0.6312  0.64286      NaN
## Neg Pred Value       0.993691 0.968354   0.8471   0.7841  0.92734 0.990536
## Prevalence           0.006309 0.031546   0.4290   0.4006  0.12303 0.009464
## Detection Rate       0.000000 0.000000   0.3470   0.2808  0.05678 0.000000
## Detection Prevalence 0.000000 0.003155   0.4637   0.4448  0.08833 0.000000
## Balanced Accuracy    0.500000 0.498371   0.8022   0.7136  0.71278 0.500000
```

After implementing the Random Forest classifier, the accuracy improved to 69.43% for white wine and 68.45% for red wine.

Problem 3:

```
library(readxl)
library(tm)
```

```
## Warning: package 'tm' was built under R version 4.3.3
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'

## The following object is masked from 'package:ggplot2':
##
##     annotate

library(readr)
library(NLP)
library(e1071)
library(caret)

dataset_spam = read.csv("C:/Users/Abhiram/Downloads/archive (4)/spam.csv",
stringsAsFactors = FALSE,encoding = "UTF-8")
colnames(dataset_spam) <- c("type", "text")
dataset_spam$type = factor(dataset_spam$type)

corpus = VCorpus(VectorSource(dataset_spam$text))
corpus

## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 5572
```

Convert to Lower Case:

```
dataset_spam_clean <- tm_map(corpus, content_transformer(function(x) iconv(x,
"UTF-8", "UTF-8", sub = "")))

dataset_spam_clean <- tm_map(dataset_spam_clean,
content_transformer(tolower))
as.character(dataset_spam_clean[[1]])

## [1] "go until jurong point, crazy.. available only in bugis n great world
la e buffet... cine there got amore wat..."
```

Remove Stopwords:

```
dataset_spam_clean <- tm_map(dataset_spam_clean, removeWords, stopwords())
as.character(dataset_spam_clean[[1]])

## [1] "go  jurong point, crazy.. available   bugis n great world la e
buffet... cine  got amore wat..."
```

Strip Whitespace:

```
dataset_spam_clean <- tm_map(dataset_spam_clean, stripWhitespace)
as.character(dataset_spam_clean[[1]])

## [1] "go jurong point, crazy.. available bugis n great world la e buffet...
cine got amore wat..."
```

Removing Punctuation:

```
dataset_spam_clean <- tm_map(dataset_spam_clean, removePunctuation)
as.character(dataset_spam_clean[[1]])

## [1] "go jurong point crazy available bugis n great world la e buffet cine
got amore wat"
```

Document Term Matrix:

```
dataset_spam_matrix = DocumentTermMatrix(dataset_spam_clean)
dataset_spam_matrix

## <<DocumentTermMatrix (documents: 5572, terms: 8733)>>
## Non-/sparse entries: 44649/48615627
## Sparsity           : 100%
## Maximal term length: 50
## Weighting          : term frequency (tf)

dataset_spam_train_dtm <- dataset_spam_matrix[1:4179,]
dataset_spam_test_dtm <- dataset_spam_matrix[4180:5572,]

dataset_spam_train_labels <- dataset_spam[1:4179,]$type
dataset_spam_test_labels <- dataset_spam[4180:5572,]$type
```

Finding Frequent Terms:

```
dataset_spam_frequent_terms <- findFreqTerms(dataset_spam_train_dtm,10)

data_train <- dataset_spam_train_dtm[,dataset_spam_frequent_terms]
data_test <- dataset_spam_test_dtm[,dataset_spam_frequent_terms]

count = function(x){
  x = ifelse(x >0, "Yes", "No")
}
spam_train = apply(data_train, MARGIN = 2, count)
spam_test = apply(data_test, MARGIN = 2, count)

svm = naiveBayes(spam_train,dataset_spam_train_labels)

train_prediction <- predict(svm, spam_train)
test_prediction <- predict(svm, spam_test)

accuracy_training = confusionMatrix(train_prediction,
dataset_spam_train_labels)
cat("Accuracy for Trainig is:",accuracy_training$overall[1])

## Accuracy for Trainig is: 0.9834889

accuracy_testing = confusionMatrix(test_prediction, dataset_spam_test_labels)
cat("Accuracy for Testing is:",accuracy_testing$overall[1])

## Accuracy for Testing is: 0.9741565
```