CSP 571 Assignment 4

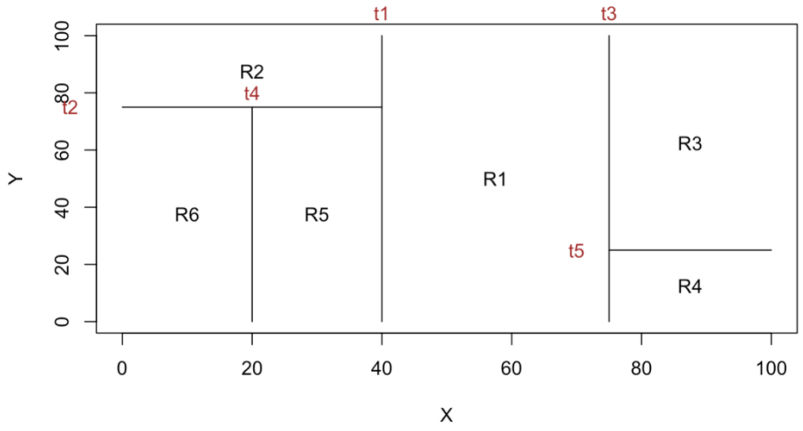
Name: Prabhu Avula
A#: A20522815

1.1 Textbook Exercises:

Chapter 8

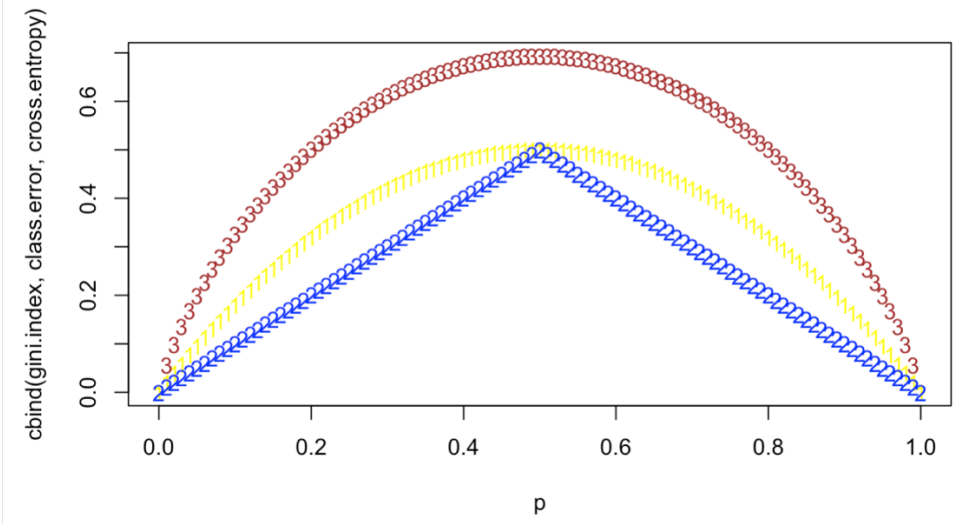
1.

```
#Create the plot
par(xpd = NA)
plot(NA, NA, type = "n", xlim = c(0,100), ylim = c(0,100), xlab = "X", ylab =
"Y")

#Add the lines
lines(x = c(40,40), y = c(0,100))
lines(x = c(0,40), y = c(75,75))
lines(x = c(75,75), y = c(0,100))
lines(x = c(20,20), y = c(0,75))
lines(x = c(75,100), y = c(25,25))

#Add the labels and colors
text(x = 40, y = 108, labels = c("t1"), col = "brown")
text(x = -8, y = 75, labels = c("t2"), col = "brown")
text(x = 75, y = 108, labels = c("t3"), col = "brown")
text(x = 20, y = 80, labels = c("t4"), col = "brown")
text(x = 70, y = 25, labels = c("t5"), col = "brown")

#Add the labels and names
text(x = (40+75)/2, y = 50, labels = c("R1"))
text(x = 20, y = (100+75)/2, labels = c("R2"))
text(x = (75+100)/2, y = (100+25)/2, labels = c("R3"))
text(x = (75+100)/2, y = 25/2, labels = c("R4"))
text(x = 30, y = 75/2, labels = c("R5"))
text(x = 10, y = 75/2, labels = c("R6"))
```

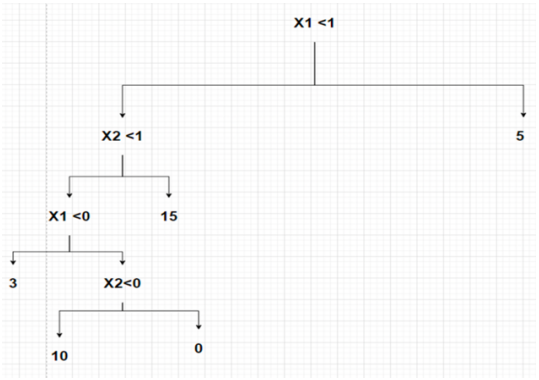


3.

```r
#Chapter 8, Question 3
p <- seq(0, 1, 0.01)
gini.index <- 2 * p * (1 - p)
class.error <- 1 - pmax(p, 1 - p)
cross.entropy <- - (p * log(p) + (1 - p) * log(1 - p))
matplot(p, cbind(gini.index, class.error, cross.entropy), col = c("yellow", "blue", "brown"))
```

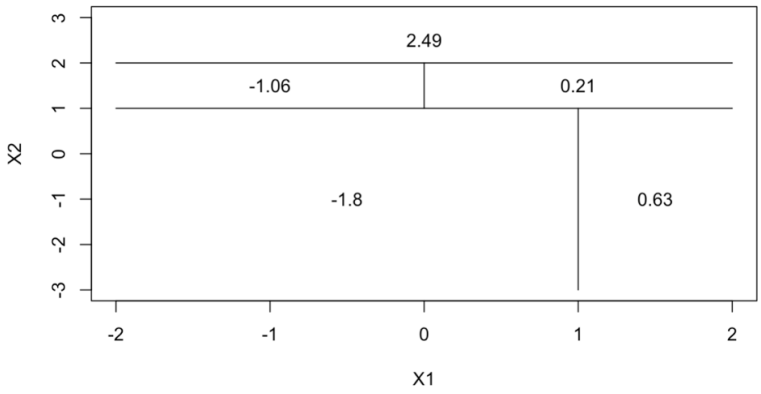


4. A.



B.

```
#C8Q4b:
par(xpd = NA)
plot(NA, NA, type = "n", xlim = c(-2, 2), ylim = c(-3, 3), xlab = "X1", ylab = "X2")

# X2 < 1
lines(x = c(-2, 2), y = c(1, 1))

# X1 < 1 with X2 < 1
lines(x = c(1, 1), y = c(-3, 1))
text(x = (-2 + 1)/2, y = -1, labels = c(-1.8))
text(x = 1.5, y = -1, labels = c(0.63))

# X2 < 2 with X2 >= 1
lines(x = c(-2, 2), y = c(2, 2))
text(x = 0, y = 2.5, labels = c(2.49))

# X1 < 0 with X2<2 and X2>=1
lines(x = c(0, 0), y = c(1, 2))
text(x = -1, y = 1.5, labels = c(-1.06))
text(x = 1, y = 1.5, labels = c(0.21))
```



5.

The probabilities for P (Class is Red|X) are given as follows:
0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, and 0.75. To consolidate these probabilities into a single class prediction, two main methods are employed:

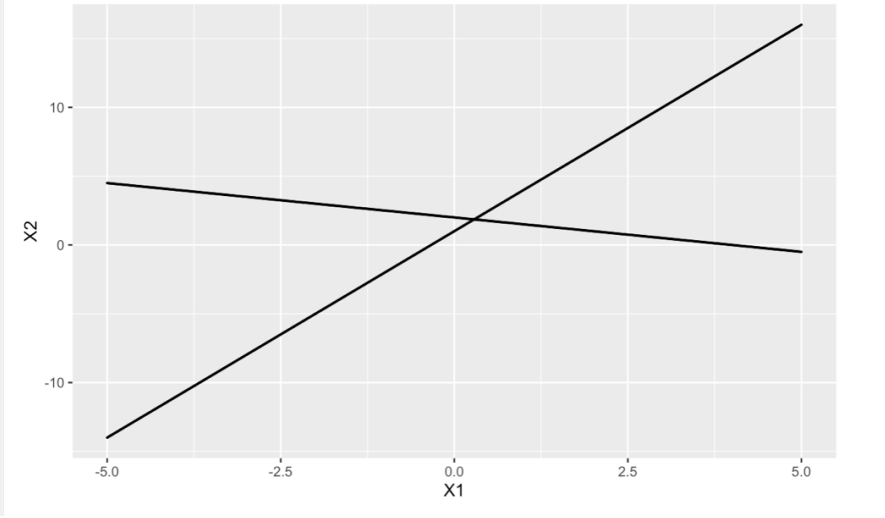
1. **Majority Vote Approach:** Among the 1q0 estimates, 6 have probabilities exceeding 0.5, while 4 fall below 0.5. With the majority favoring the red class (6 votes), we can conclude that X is assigned to the red class.

2. **Average Probability Approach:** This approach determines the assigned class for X based on whether the average probability surpasses 0.5. Computing the average of all 10 estimates results in
P = (0.1+0.15+0.2+0.2+0.55+0.6+0.6+0.65+0.7+0.75)/10, equaling P = 0.45. Since this average probability is below 0.5, X will be allocated to the green class.

Chapter 9

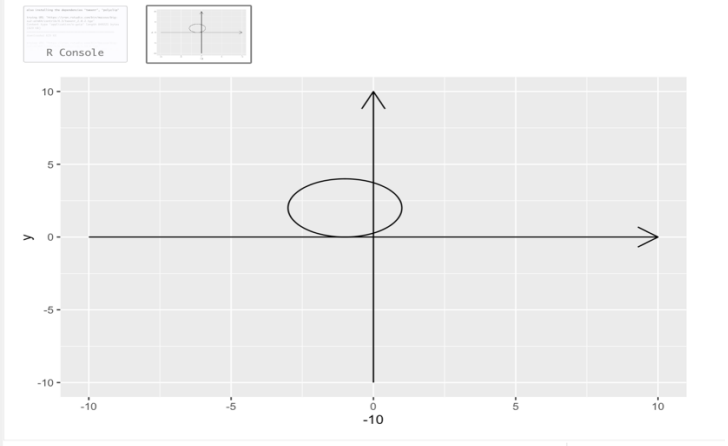
1.

```
#C9Q1a
library(tibble)
library(ggplot2)
tibble(
  X1 = seq(-5, 5, .01),
  X2 = 3 * X1 + 1,
  X2A = 2 - X1/2
) %>%
ggplot() +
geom_point(aes(X1, X2), size = .1) +
geom_point(aes(X1, X2A), size = .1)
```



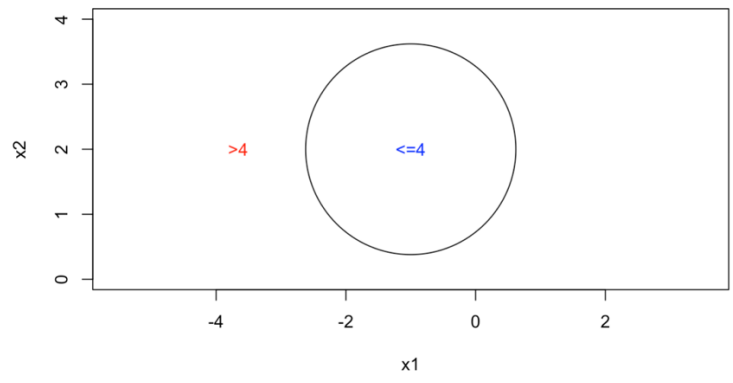
2. A.

```
install.packages("ggforce")
library(ggforce)
ggplot() +
geom_segment(aes(x = -10, y = 0, xend = 10, yend = 0), arrow = arrow()) +
geom_segment(aes(x = 0, y = -10, xend = 0, yend = 10), arrow = arrow()) +
geom_circle(aes(x0 = -1, y0 = 2, r = 2))
```



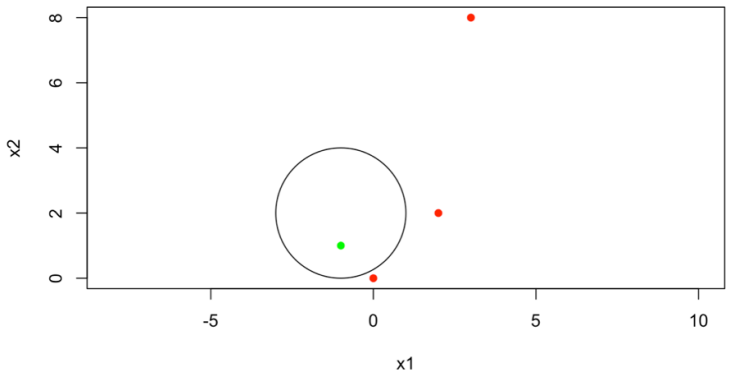
b.

```r
plot (NA, NA, type = "n", xlim = c(-4,2) ,ylim = c(0,4) ,asp =1 ,xlab = "x1", ylab = "x2")
symbols (c(-1),c(2), circles = c(2) ,add = TRUE, inches=TRUE)
text(c(-1),c(2),"<=4", col = "blue")
text(c(-3.66),c(2),">4",col="red")
```



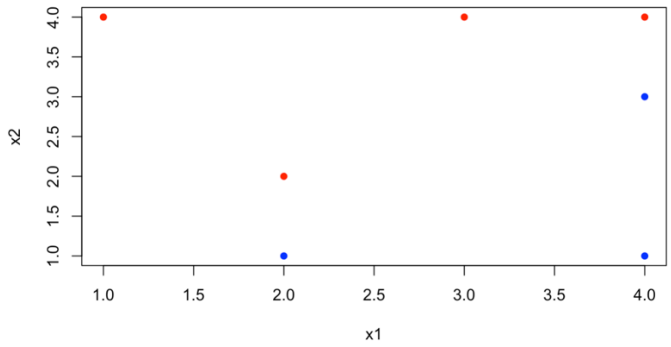
c.

```r
plot(c(0,-1,2,3),c(0,1,2,8), col=c("red", "green", "red","red"),pch=16,asp=1,xlab="x1", ylab="x2")
symbols(c(-1),c(2),circles= c(2),add= TRUE, inches=FALSE)
```



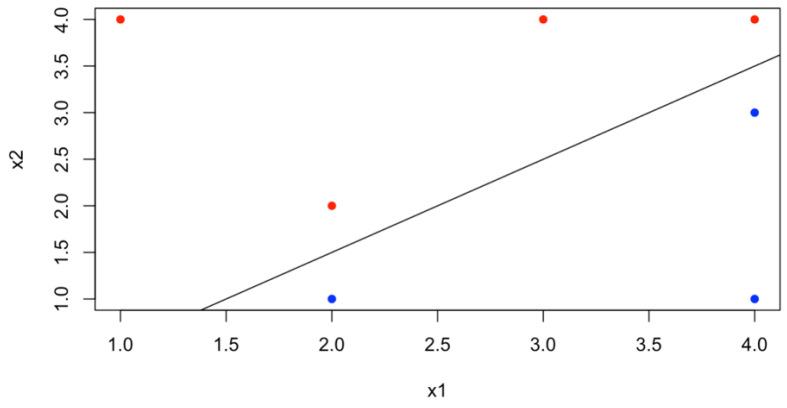
3. a.

```r
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, pch=16)
```



b.

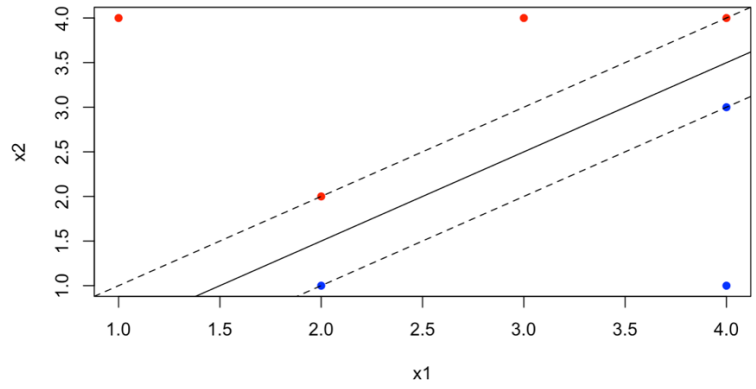
```r
x1=c(3,2,4,1,2,4,4)
x2=c(4,2,4,4,1,3,1)
colors = c("red", "red","red", "red", "blue", "blue","blue")
plot(x1 , x2 , col= colors, pch=16)
abline(-0.5, 1)
```



c. The maximal margin classifier is represented by the hyperplane that maximizes the distance from the observations. In our formulation, the maximal margin classifier is expressed as $\beta 0+\beta 1X1+\beta 2X2$, with specific coefficients given by $(\beta_0, \beta_1, \beta_2) = (-12, 78, -1)$. Now, classification is based on the rule that when the formula is less than 0, we assign the observation to the red class, else we assign it to the blue class.

D.

```r
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, pch=16)
abline(-0.5, 1)
abline(-1,1,lty=2)
abline(0,1,lty=2)
```

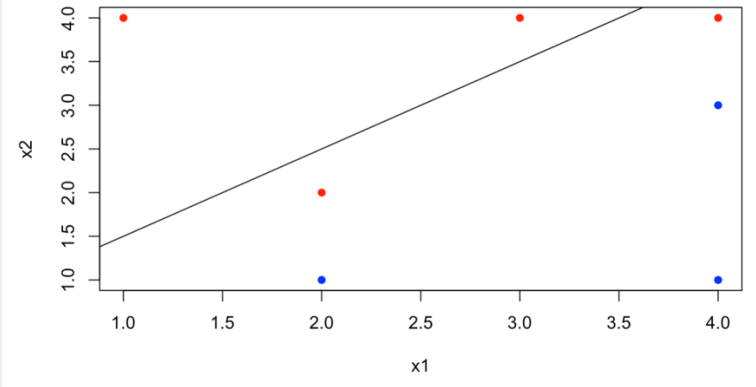


e. The coordinates (2,1), (2,2), (4,3), and (4,4) function as the support vectors for the maximal margin classifier.

f. It would have no effect.

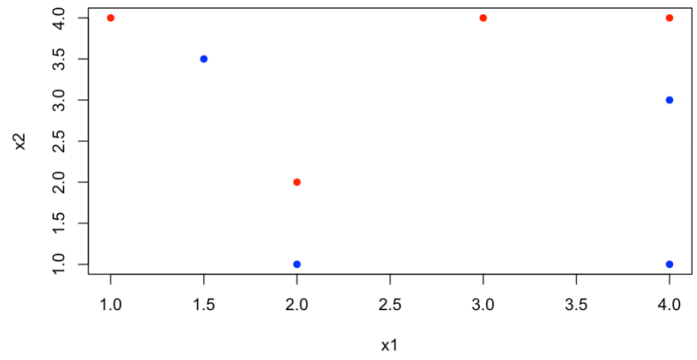
g.

```r
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, pch=16)
abline(0.5, 1)
```



h.

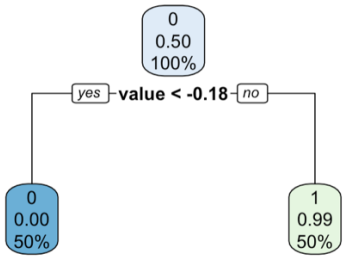
```r
x1= c(3,2,4,1,2,4,4)
x2= c(4,2,4,4,1,3,1)
colors = c("red", "red", "red", "red", "blue", "blue", "blue")
plot(x1, x2, col = colors, pch=16)
points (c(1.5), c(3.5), col="blue", pch=16)
```



Practicum Problems

2.1

```r
#Loading the required libraries
library(rpart)
library(rpart.plot)

#Setting the random seed for reproducibility
set.seed(123)

#Generating the data with normal distributions
class_1 <- data.frame(value = rnorm(100, mean = 5, sd = 2), class = 1)
class_2 <- data.frame(value = rnorm(100, mean = -5, sd = 2), class = 0)

#Combining the data frames
data <- rbind(class_1, class_2)

#Inducing a decision tree
tree <- rpart(class ~ value, data = data, method = "class")
rpart.plot(tree)
```



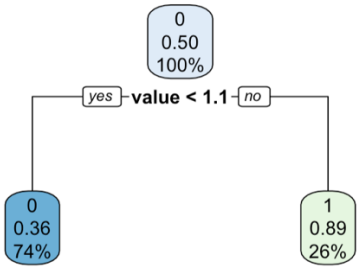
```{r}
#Getting the threshold value for the first split
threshold <- tree$frame$yval[1]

#Getting the number of nodes in the tree
num_nodes <- nrow(tree$frame)

#Calculating the entropy and Gini for each node
entropy <- tree$frame$yval
gini <- 1 - (tree$frame$yval^2 + (1 - tree$frame$yval)^2)

cat("Threshold for the first split:", threshold, "\n")
cat("Number of nodes in the tree:", num_nodes, "\n")
cat("Entropy at each node:", entropy, "\n")
cat("Gini at each node:", gini, "\n")

#Repeating the same with the given normal distributions
class_1 <- data.frame(value = rnorm(100, mean = 1, sd = 2), class = 1)
class_2 <- data.frame(value = rnorm(100, mean = -1, sd = 2), class = 0)
data <- rbind(class_1, class_2)
tree <- rpart(class ~ value, data = data, method = "class")

#Getting the number of nodes in the new tree
num_nodes_new <- nrow(tree$frame)
cat("Number of nodes in the new tree:", num_nodes_new, "\n")

#Pruning the tree
pruned_tree <- prune(tree, cp = 0.1)
```

```
Threshold for the first split: 1
Number of nodes in the tree: 9
Entropy at each node: 1 1 1 2 1 1 2 2 2
Gini at each node: 0 0 0 -4 0 0 -4 -4 -4
Number of nodes in the new tree: 11
```

```{r}
#Getting the pruned tree's summary and its plot
summary(pruned_tree)
rpart.plot(pruned_tree)

```

```
1 0.41      0      1.00   1.12 0.07019972
2 0.10      1      0.59   0.68 0.06699254

Variable importance
value
  100

Node number 1: 200 observations,    complexity param=0.41
  predicted class=0  expected loss=0.5  P(node) =1
    class counts:   100   100
   probabilities: 0.500 0.500
  left son=2 (147 obs) right son=3 (53 obs)
  Primary splits:
      value < 1.133877 to the left,  improve=21.57618, (0 missing)

Node number 2: 147 observations
  predicted class=0  expected loss=0.3605442  P(node) =0.735
    class counts:    94    53
   probabilities: 0.639 0.361

Node number 3: 53 observations
  predicted class=1  expected loss=0.1132075  P(node) =0.265
    class counts:     6    47
   probabilities: 0.113 0.887
```

R Console


0
0.50
100%
yes
value < 1.1
no
0
0.36
74%
1
0.89
26%

2.2

```r
#Practicum Problem 2
library(caret)
library(randomForest)
library(rpart)
library(rpart.plot)

redwine = read.csv("/Users/prabhuavula7/Desktop/wine+quality/winequality-red.csv", header = T, sep = ';')
whitewine = read.csv("/Users/prabhuavula7/Desktop/wine+quality/winequality-white.csv", header = T, sep = ';')
str(whitewine)
str(redwine)
```

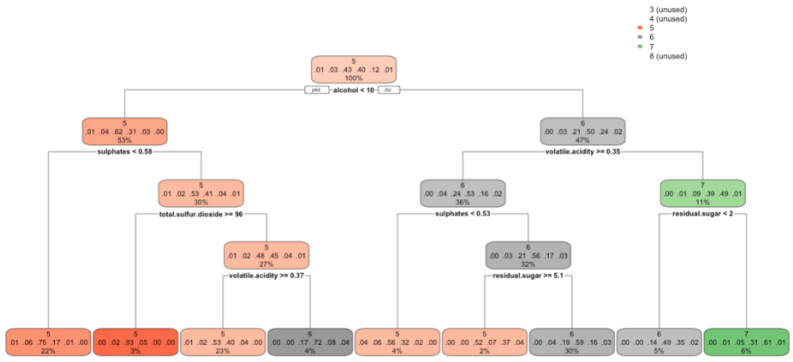
```
'data.frame':   4898 obs. of  12 variables:
 $ fixed.acidity       : num  7 6.3 8.1 7.2 7.2 8.1 6.2 7 6.3 8.1 ...
 $ volatile.acidity    : num  0.27 0.3 0.28 0.23 0.23 0.28 0.32 0.27 0.3 0.22 ...
 $ citric.acid         : num  0.36 0.34 0.4 0.32 0.32 0.4 0.16 0.36 0.34 0.43 ...
 $ residual.sugar      : 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 ...
'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 ...
```

```r
#Handling the 'quality' output variable
whitewine$quality = as.factor(whitewine$quality)
redwine$quality = as.factor(redwine$quality)
str(whitewine)
str(redwine)
```

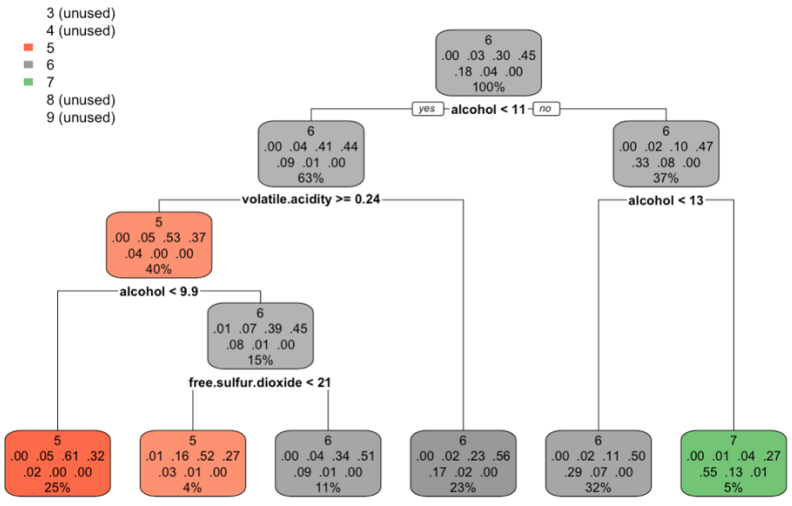
```
'data.frame':   4898 obs. of  12 variables:
 $ fixed.acidity       : num  7 6.3 8.1 7.2 7.2 8.1 6.2 7 6.3 8.1 ...
 $ volatile.acidity    : num  0.27 0.3 0.28 0.23 0.23 0.28 0.32 0.27 0.3 0.22 ...
 $ citric.acid         : num  0.36 0.34 0.4 0.32 0.32 0.4 0.16 0.36 0.34 0.43 ...
 $ residual.sugar      : 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             : Factor w/ 7 levels "3","4","5","6",..: 4 4 4 4 4 4 4 4 4 4 ...
'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             : Factor w/ 6 levels "3","4","5","6",..: 3 3 3 4 3 3 3 5 5 3 ...
```

```{r}
#Using the caret package to perform a 80/20 split
datapartred = createDataPartition(y=redwine$quality,p=0.8,list=FALSE)
traindatared = redwine[datapartred,]
testdatared = redwine[-datapartred,]
```

```{r}
#Using the caret package to perform a 80/20 split
datapartwhite = createDataPartition(y=whitewine$quality,p=0.8,list=FALSE)
traindatawhite = whitewine[datapartwhite,]
testdatawhite = whitewine[-datapartwhite,]
#inducing a decision tree for red wine targeting the quality output variable
dtred = rpart(quality~., data=traindatared)
#plotting
rpart.plot(dtred)
```




```{r}
#inducing a decision tree for white wine
dtwhite = rpart(quality~., data=traindatawhite)
#plotting
rpart.plot(dtwhite)
```



```r
#Confusion Matrix for red wine
redpred = predict(dtred, testdatared, type = 'class')
confusionMatrix(redpred, testdatared$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   7 100  53   6   0
         6   1   3  30  67  27   2
         7   0   0   6   7   6   1
         8   0   0   0   0   0   0

Overall Statistics

               Accuracy : 0.5457
                 95% CI : (0.4891, 0.6015)
    No Information Rate : 0.429
    P-Value [Acc > NIR] : 1.923e-05

                  Kappa : 0.2453

 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   0.7353   0.5276  0.15385 0.000000
Specificity          1.000000  1.00000   0.6298   0.6684  0.94964 1.000000
Pos Pred Value            NaN      NaN   0.5988   0.5154  0.30000      NaN
Neg Pred Value       0.993691  0.96845   0.7600   0.6791  0.88889 0.990536
Prevalence           0.006309  0.03155   0.4290   0.4006  0.12303 0.009464
Detection Rate       0.000000  0.00000   0.3155   0.2114  0.01893 0.000000
Detection Prevalence 0.000000  0.00000   0.5268   0.4101  0.06309 0.000000
Balanced Accuracy    0.500000  0.50000   0.6826   0.5980  0.55174 0.500000
```

```r
#Confusion Matrix for white wine
whitepred = predict(dtwhite, testdatawhite, type = 'class')
confusionMatrix(whitepred, testdatawhite$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  21 167  91   9   0   0
         6   3  10 124 332 151  29   1
         7   0   1   0  16  16   6   0
         8   0   0   0   0   0   0   0
         9   0   0   0   0   0   0   0

Overall Statistics

               Accuracy : 0.5266
                 95% CI : (0.4947, 0.5583)
    No Information Rate : 0.4489
    P-Value [Acc > NIR] : 6.538e-07

                  Kappa : 0.2195

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity          0.00000  0.00000   0.5739   0.7563  0.09091  0.00000 0.000000
Specificity          1.00000  1.00000   0.8224   0.4100  0.97132  1.00000 1.000000
Pos Pred Value           NaN      NaN   0.5779   0.5108  0.41026      NaN      NaN
Neg Pred Value       0.99591  0.96728   0.8200   0.6738  0.82961  0.96421 0.998978
Prevalence           0.00409  0.03272   0.2975   0.4489  0.17996  0.03579 0.001022
Detection Rate       0.00000  0.00000   0.1708   0.3395  0.01636  0.00000 0.000000
Detection Prevalence 0.00000  0.00000   0.2955   0.6646  0.03988  0.00000 0.000000
Balanced Accuracy    0.50000  0.50000   0.6981   0.5831  0.53112  0.50000 0.500000
```

For red wine: 54.57%
For white wine: 52.66%

```{r}
#Via randomForest package to repeat the fit for red wine
rfred = train(quality~., data=traindatared, method='rf', preProcess=c('center', 'scale'))

#Via randomForest package to repeat the fit for white wine
rfwhite = train(quality~., data=traindatawhite, method='rf', preProcess=c('center', 'scale'))
```

```{r}
#Confusion Matrix of random forest model on test data of red wine
rfredpred = predict(rfred, testdatared)
confusionMatrix(rfredpred, testdatared$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   1   7 107  29   3   0
         6   0   2  28  93  14   1
         7   0   1   1   5  22   2
         8   0   0   0   0   0   0

Overall Statistics

               Accuracy : 0.7003
                 95% CI : (0.6466, 0.7502)
    No Information Rate : 0.429
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5123

 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   0.7868   0.7323  0.56410 0.000000
Specificity          1.000000 0.996743   0.7790   0.7632  0.96763 1.000000
Pos Pred Value            NaN 0.000000   0.7279   0.6739  0.70968      NaN
Neg Pred Value       0.993691 0.968354   0.8294   0.8101  0.94056 0.990536
Prevalence           0.006309 0.031546   0.4290   0.4006  0.12303 0.009464
Detection Rate       0.000000 0.000000   0.3375   0.2934  0.06940 0.000000
Detection Prevalence 0.000000 0.003155   0.4637   0.4353  0.09779 0.000000
Balanced Accuracy    0.500000 0.498371   0.7829   0.7477  0.76586 0.500000
```

```
#Confusion Matrix of random forest model on test data of white wine
rfwhitepred = predict(rfwhite, testdatawhite)
confusionMatrix(rfwhitepred, testdatawhite$quality)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   3   4   5   6   7   8   9
         3   0   0   0   0   0   0   0
         4   0   6   1   0   0   0   0
         5   2  17 207  52   5   1   0
         6   2   9  83 365  81  11   0
         7   0   0   0  21  90  10   1
         8   0   0   0   1   0  13   0
         9   0   0   0   0   0   0   0

Overall Statistics

               Accuracy : 0.6963
                 95% CI : (0.6664, 0.725)
    No Information Rate : 0.4489
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5236

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity           0.00000 0.187500   0.7113   0.8314  0.51136  0.37143 0.000000
Specificity           1.00000 0.998943   0.8879   0.6549  0.96010  0.99894 1.000000
Pos Pred Value            NaN 0.857143   0.7289   0.6624  0.73770  0.92857      NaN
Neg Pred Value        0.99591 0.973223   0.8790   0.8267  0.89953  0.97718 0.998978
Prevalence            0.00409 0.032720   0.2975   0.4489  0.17996  0.03579 0.001022
Detection Rate        0.00000 0.006135   0.2117   0.3732  0.09202  0.01329 0.000000
Detection Prevalence  0.00000 0.007157   0.2904   0.5634  0.12474  0.01431 0.000000
Balanced Accuracy     0.50000 0.593221   0.7996   0.7432  0.73573  0.68518 0.500000
```

For Red wine: 70.03%
For White wine: 69.63%

2.3

```{r}
#Loading required libraries
library(tm)
library(e1071)

#Loading the dataset
data_dir <- "/Users/prabhuavula7/Desktop/sms+spam+collection/"
sms_data <- read.table(file.path(data_dir, "SMSSpamCollection"), sep = "\t", header = FALSE)
colnames(sms_data) <- c("class", "text")

#Preprocessing the text data
corpus <- Corpus(VectorSource(sms_data$text))
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, removeWords, stopwords("en"))
corpus <- tm_map(corpus, stripWhitespace)

#Creating a Document Term Matrix
dtm <- DocumentTermMatrix(corpus)

#Finding features from words occurring more than 10 times
freq_words <- findFreqTerms(dtm, 10)

#Splitting the data
set.seed(123)
train_indices <- sample(1:nrow(sms_data), 0.7 * nrow(sms_data))
train_data <- sms_data[train_indices, ]
test_data <- sms_data[-train_indices, ]
```

```{r}
#Creating Document Term Matrices for training and test data
train_dtm <- DocumentTermMatrix(Corpus(VectorSource(train_data$text)), control = list(dictionary = freq_words))
test_dtm <- DocumentTermMatrix(Corpus(VectorSource(test_data$text)), control = list(dictionary = freq_words))

#Converting the Document Term Matrix to a Boolean representation
train_matrix <- ifelse(as.matrix(train_dtm) > 0, 1, 0)
test_matrix <- ifelse(as.matrix(test_dtm) > 0, 1, 0)

#Fitting an SVM
svm_model <- svm(as.factor(train_data$class) ~ ., data = as.data.frame(train_matrix), kernel = "linear")

#Reporting the training and test set accuracy
train_preds <- predict(svm_model, newdata = as.data.frame(train_matrix))
test_preds <- predict(svm_model, newdata = as.data.frame(test_matrix))

train_accuracy <- sum(train_preds == train_data$class) / length(train_data$class)
test_accuracy <- sum(test_preds == test_data$class) / length(test_data$class)

cat("Training Set Accuracy:", train_accuracy, "\n")
cat("Test Set Accuracy:", test_accuracy, "\n")

```

```
Warning: Variable(s) 'amp' and 'ltgt' and 'ampm' and 'ltdecimalgt' and 'mobileupd' and 'pmin' and 'pmsg' and 'voice' and 'grins' constant.
Cannot scale data.Training Set Accuracy: 0.996
Test Set Accuracy: 0.934
```

The accuracies are 99.6% and 93.4% for training and testing respectively.