Assignment-04

Sumanth Donthula

2023-04-09

Recitation Exercises

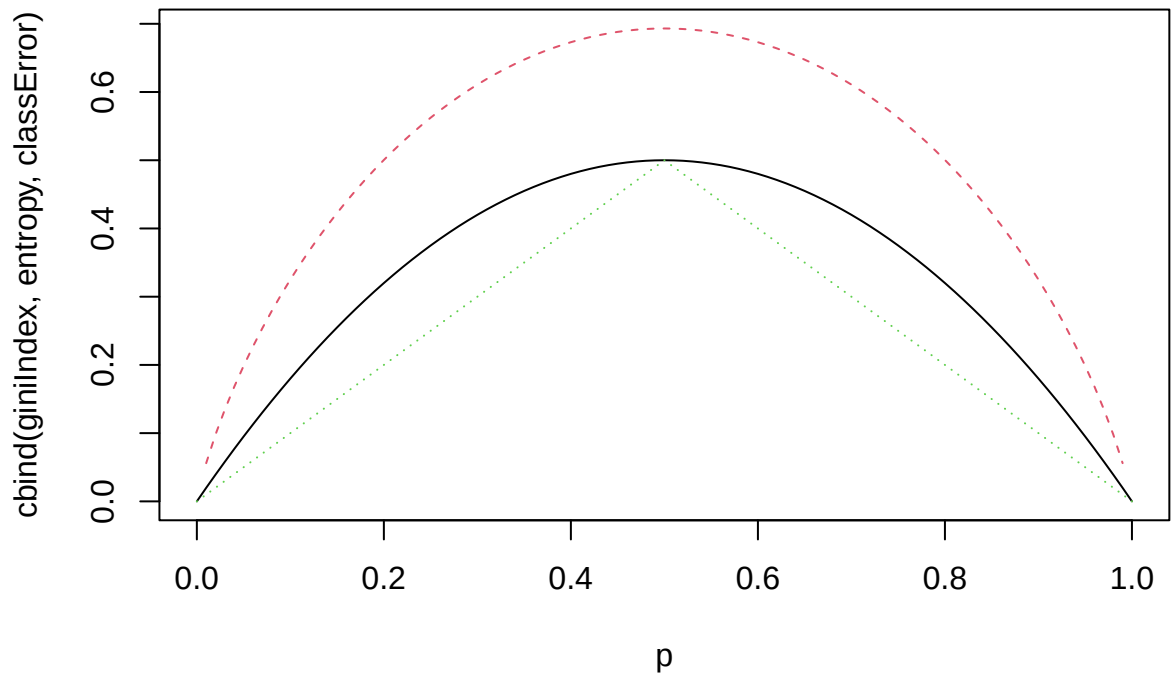
Chapter-8

Excercise-1:

Handwritten and attached seperately

Exercise-3:

```
p = seq(0, 1, 0.01)

giniIndex = p * (1 - p) +  (1-p)*p
entropy = -(p * log(p) + (1 - p) * log(1 - p))
classError = 1 - pmax(p, 1 - p)
matplot(p, cbind(giniIndex, entropy, classError), type = "l")
```



Excercise-4:

Handwritten and attached seperately

Exerciise-5:

Majority vote method: Notice that in the above estimates, we have 6 values more than and 4 values less than 0.5, which is utilized as a class boundary.

With this method, we assign X to the class RED.

The average approach: With this method, we take the average of the above estimations. The average is 0.45, which is less than 0.5. As a result, given a certain value of X, the above-mentioned estimations were obtained.

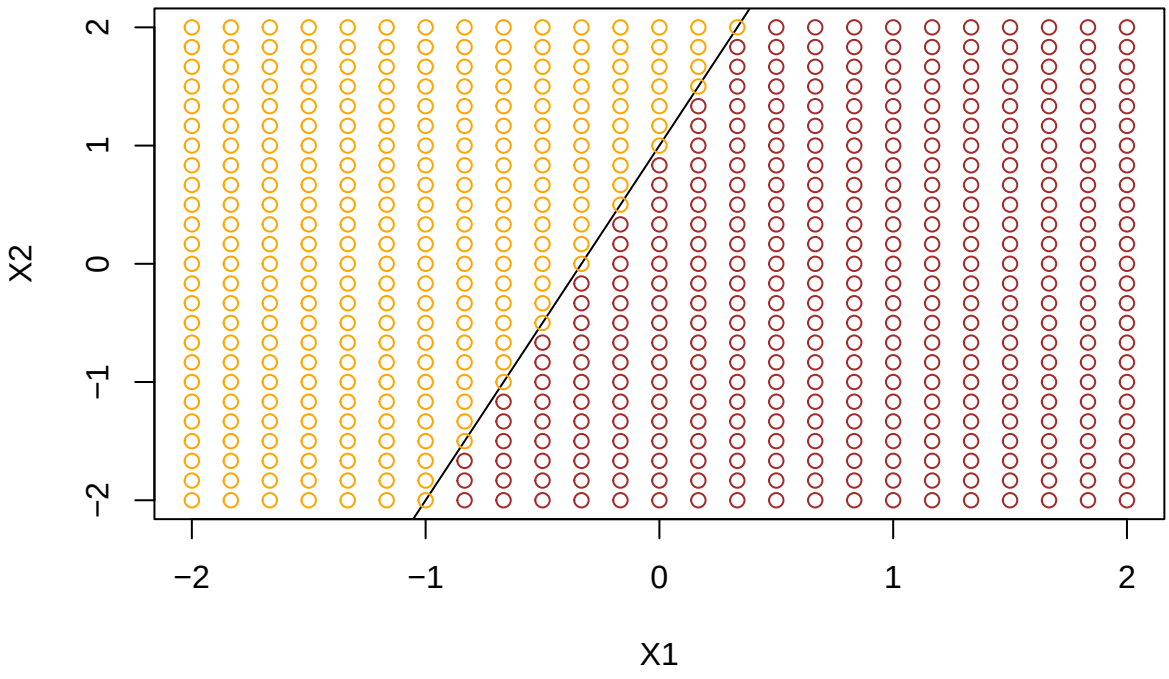
Using the AVERAGE method, we would categorize it as GREEN.

Chapter-9

Exercise-1:

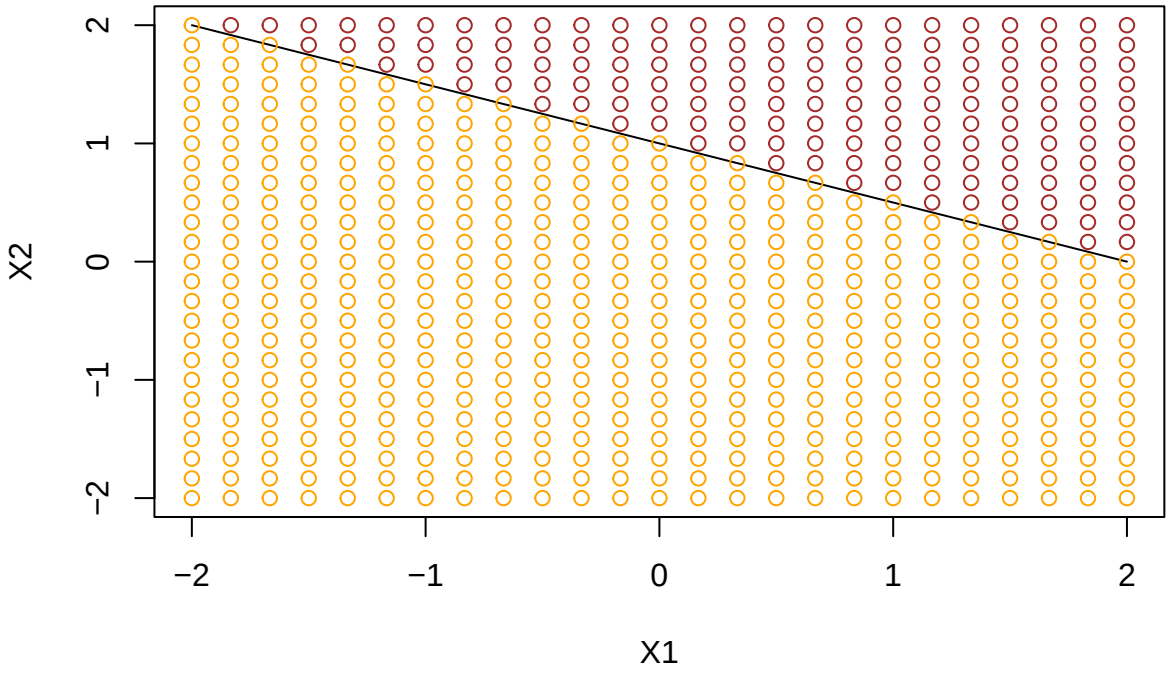
a) Sketching the hyperplane 1 + 3X1 - X2 = 0. All the brown points fall on 1 + 3X1 - X2>0 region and orange points fall on 1 + 3X1 - X2<0 region

```
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)){
  pts=data.frame(rep(i,25),seq(-2,2,length.out = 25))
  points(pts,col=ifelse(1+3*pts[,1]-pts[,2]>0,'brown','orange'))
}
```



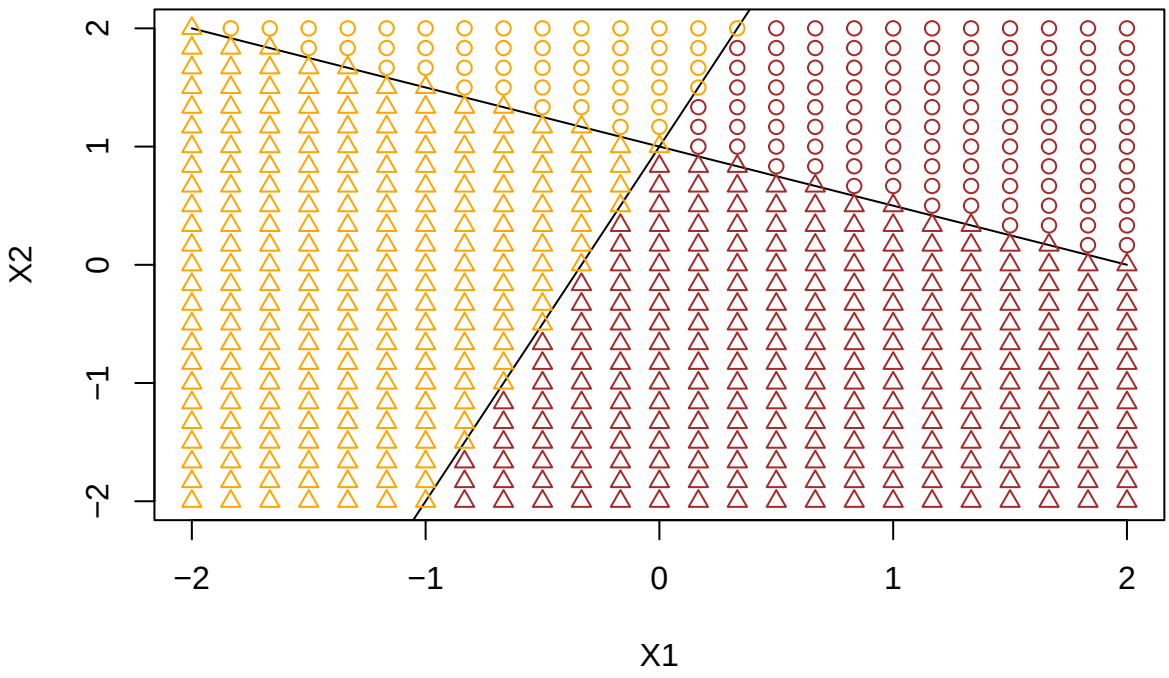
b) Plotting -2 + X1 + 2X2 = 0 seperately. All the brown points fall on -2 + X1 + 2X2>0 region and orange points fall on -2 + X1 + 2X2<0 region

```
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)){
  pts=data.frame(rep(i,25),seq(-2,2,length.out = 25))
  points(pts,col=ifelse(-2+pts[,1]+pts[,2]*2>0,'brown','orange'))

}
```



Plotting both on a 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)){
  pts=data.frame(rep(i,25),seq(-2,2,length.out = 25))
  points(pts,col=ifelse(1+3*pts[,1]-pts[,2]>0,'brown','orange'),
  pch=ifelse(-2+pts[,1]+pts[,2]*2>0,1,2))
}
```



Exercise-2:

(1 + X1)2 + (2 - X2)2 = 4 is an equation of circle. Plotting a,b,c on a same plot.

a)&b)&c)

The brown points indicate (1 + X1)^2 + (2 - X2)^2 < 4 region

The orange points indicate (1 + X1)^2 + (2 - X2)^2 > 4 region

Note that the point (0, 0) , (2, 2), (3, 8) are outside the circle and are classified as brown The point (-1, 1) lies inside the circle and is classified as orange.

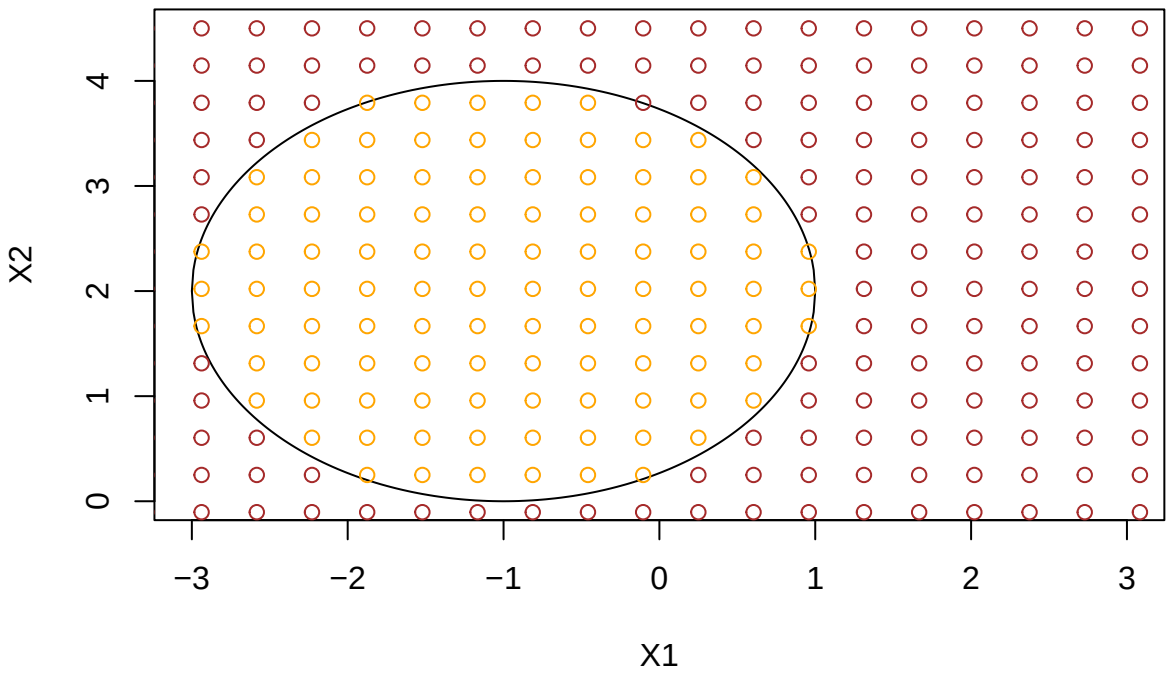
```
X1=seq(-3,10,0.01)
X2=2 - sqrt ( 4-(1+X1)^2)
```

```
## 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)){
  pts=data.frame(rep(i,25),seq(-4,4.5,length.out = 25))
  points(pts,col=ifelse((1+pts[,1])^2+(2-pts[,2])^2>4,'brown','orange'))
}
```



d)

The decision boundary is a circle in our case

(1 + X1)^2 + (2 - X2)^2 -4 = 0 which can be simplified as : 1+ X1^2 + 2X1 + 4 + X2^2 - 4X2 - 4 = 0 1+ 2X1 - 4X2 + X1^2 + X2^2 = 0

This is of the form a+bf(x1)+cf(x2)+df(x3)+ef(x4) = 0, where

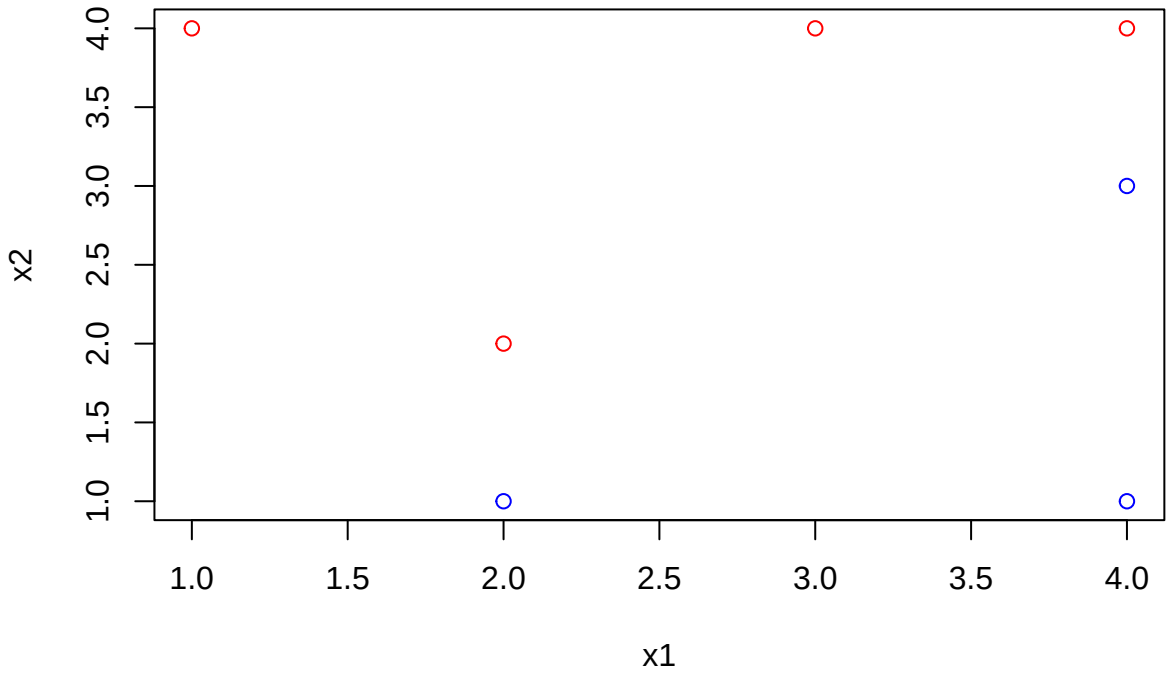
f(x1)=X1 f(x2)=X2 f(x3)=X1^2 f(x4)=X2^2

From the above, we can argue that while the decision boundary in (c) is not linear in terms of X1 and X2, it is linear in terms of X1,X1^{2,X2,X2}2.

Exercise-3:

a) Plotting Observations

```
x1 = c(3, 2, 4, 1, 2, 4, 4)
x2 = c(4, 2, 4, 4, 1, 3, 1)

plot(x1, x2, col = c("red", "red", "red", "red", "blue", "blue", "blue"))
```

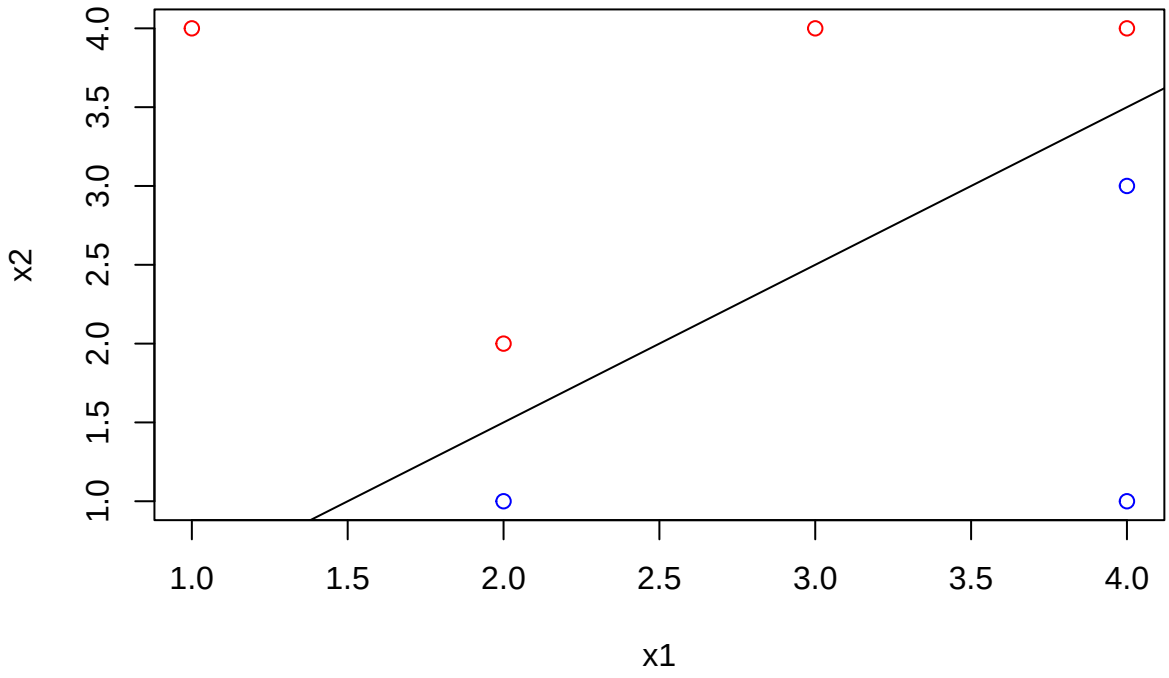


b)Finding Optimal hyper plane:

From the above plot we can see that the optimal hyperplane should pass through the midpoint of (2,1)and (2,2) and midpoint of (4,3) and (4,4), so the plain passes thro(2,1.5) and (4,3.5)

then the line will be in the form of, X2 - X1 + 0.5 = 0

```
plot(x1, x2, col = c("red", "red", "red", "red", "blue", "blue", "blue"))
abline(-0.5, 1)
```



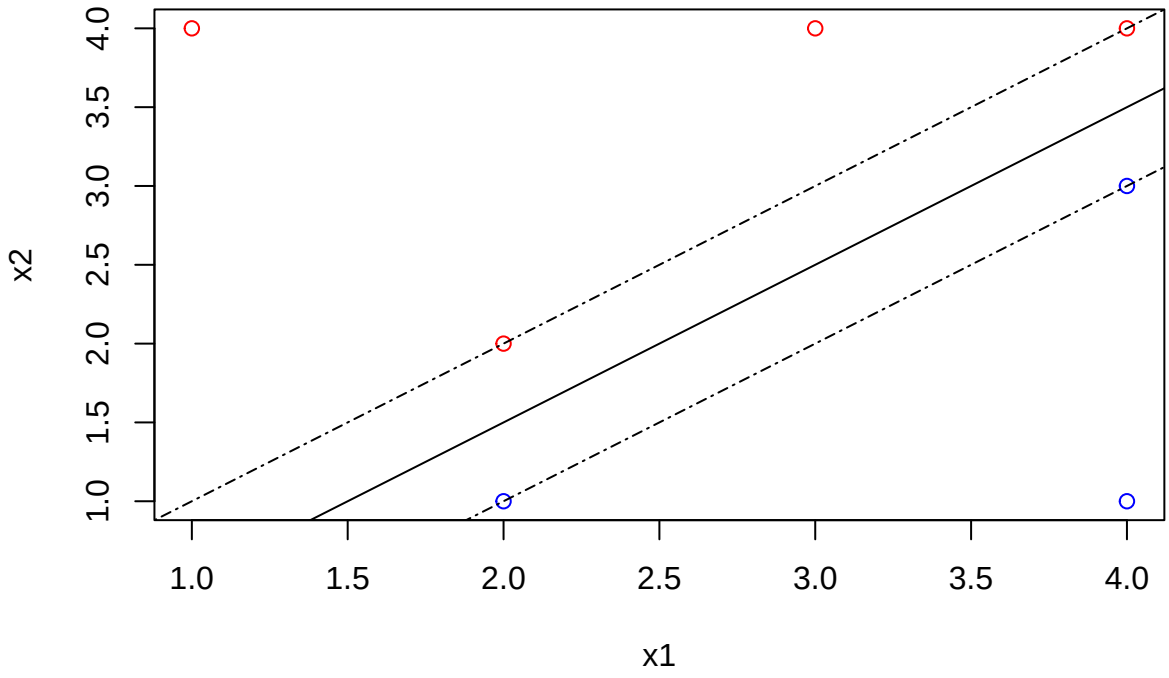
c)Maximum margin Classifier

we know that line equation is X2-X1+0.5=0, it will classify the data point as red if X2-X1+0.5>0 and blue otherwise.

(Beta0,Beta1,Beta2)=(1,-1,0.5)

d) Margin for Maximum Margin Hyperplane:

```
plot(x1, x2, col = c("red", "red", "red", "red", "blue", "blue", "blue"))
abline(-0.5, 1)
abline(-1, 1, lty = 6)
abline(0, 1, lty = 6)
```

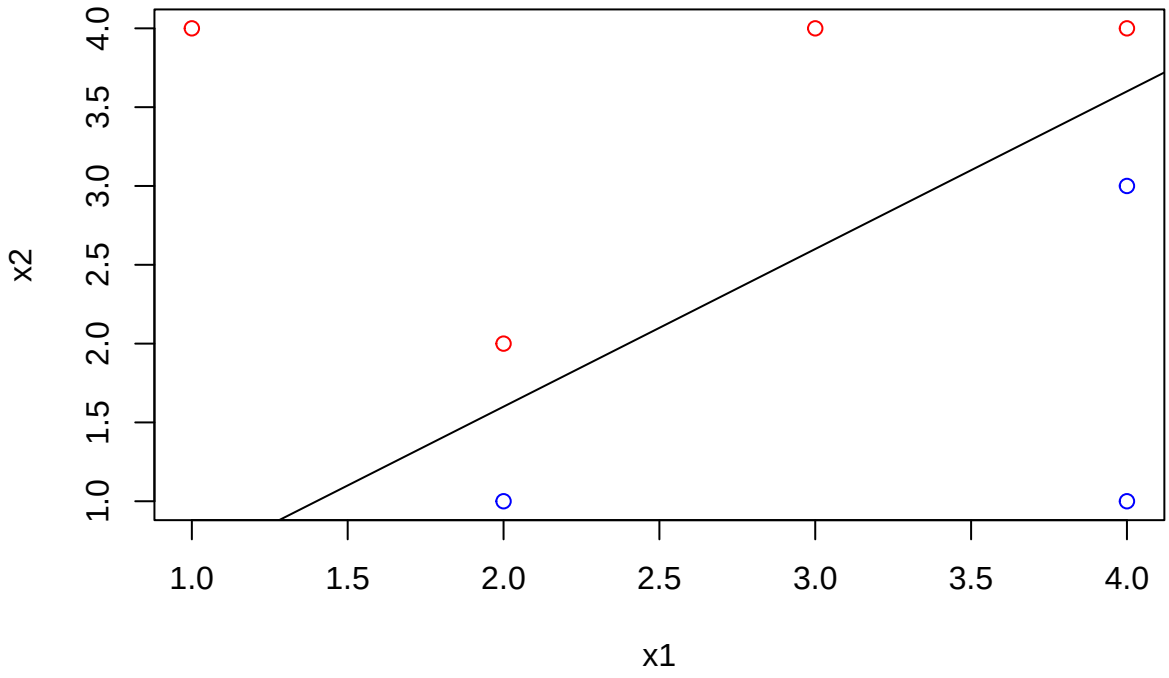


e) Support vectors from the above plot are points (2,1),(2,2),(4,3) and (4,4).

f) 7 point (4,1) is not a support vector and it did not present on the margin, even if we change it it won't effect the hyperplane.

g) Other hyper plane The equation x2-x1+0.4=0 is also a hyper plane which cuts resd and blue region but it's not the optimal one.

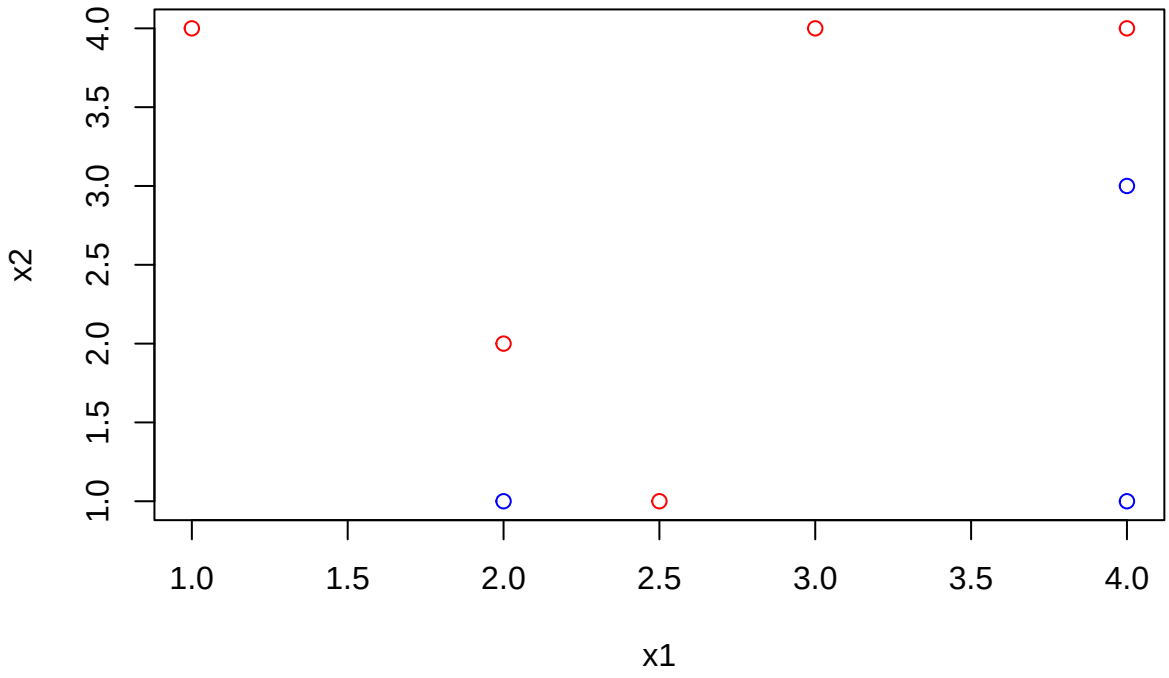
```
plot(x1, x2, col = c("red", "red", "red", "red", "blue", "blue", "blue"))
abline(-0.4, 1)
```



h) adding an additional observation

The additional poin made hyperplane,no loner seperable

```
plot(x1, x2, col = c("red", "red", "red", "red", "blue", "blue", "blue"))
points(c(2.5), c(1), col = c("red"))
```



Practicum Problems:

Problem-1:

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.2.3
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.2.3
```

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.2.3
```

```
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 4.2.3
```

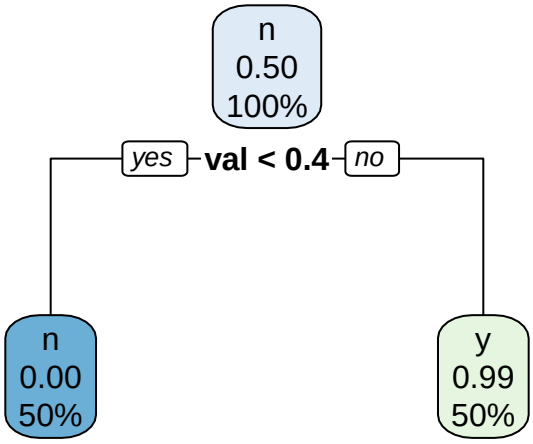
```
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.2.3
```

```
giniScore = function(p)
{
giniValue = 2 * p * (1 - p)
return (giniValue)
}

entropyValue = function(p)
{
entropyVal = (p * log(p) + (1 - p) * log(1 - p))
return (entropyVal)
}
```

Given normal distribution parameters of (5,2) and (-5,2) for pair of samples considered

```
set.seed(200)

f1=rnorm(n=200,mean=5,sd=2)
f2=rnorm(n=200,mean=-5,sd=2)

dataFrame1 = data.frame(val = f1,label=rep("y",200))
dataFrame2 = data.frame(val = f2,label=rep("n",200))
dataFrame = rbind(dataFrame1,dataFrame2)

dataFrame$label = as.factor(dataFrame$label)
decisionTree1 = rpart(label~val,dataFrame,method="class")
rpart.plot(decisionTree1)
```



From above we can see that the threshold value for first split will be 0.4. It has one root and two leaf nodes. The tree is able to classify both classes and shows empirical distribution.

Calculating Gini Score and Entropy for every Node: p=probability of correct class at each node

The gini values for nodes of tree are 0.49, 0.0, 0.0197. The entropy values for above tree will be -0.683, NaN, -0.056

```
p=c(0.4, 0, 0.99)
giniValues=sapply(p, giniScore)
giniValues
```

```
## [1] 0.4800 0.0000 0.0198
```

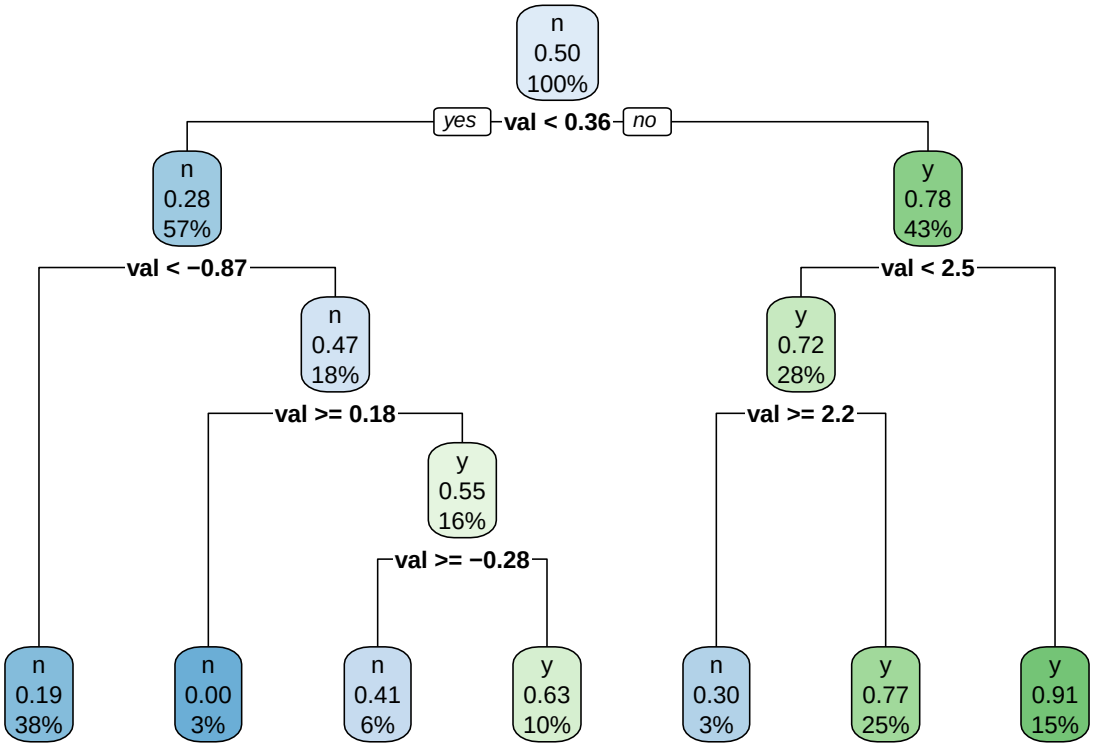
```
entropyValues=sapply(p, entropyValue)
entropyValues
```

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

The tree shows that the threshold value for the first split is 0.36. The tree comprises a total of 13 nodes, one of which is the root node, and 7 leaf nodes. A big tree size indicates the existence of more diverse labels in nodes, resulting in a huge tree. As a result, this tree has greater label overlap in nodes.

```
set.seed(150)
```

```
f1=rnorm(n=150,mean=1,sd=2)
f2=rnorm(n=150,mean=-1,sd=2)

dataFrame1 = data.frame(val = f1,label=rep("y",150))
dataFrame2 = data.frame(val = f2,label=rep("n",150))
dataFrame = rbind(dataFrame1,dataFrame2)

dataFrame$label = as.factor(dataFrame$label)
decisionTree2 = rpart(label~val,dataFrame,method="class")
rpart.plot(decisionTree2)
```



The gini values are 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 are -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

```
p=c(.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)
giniValues=sapply(p, giniScore)
giniValues
```

```
##  [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
```

```
entropyValues=sapply(p, entropyValue)
entropyValues
```

```
##  [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
```

```
p=c(.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)
giniValues=sapply(p, giniScore)
giniValues
```

```
##  [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
```

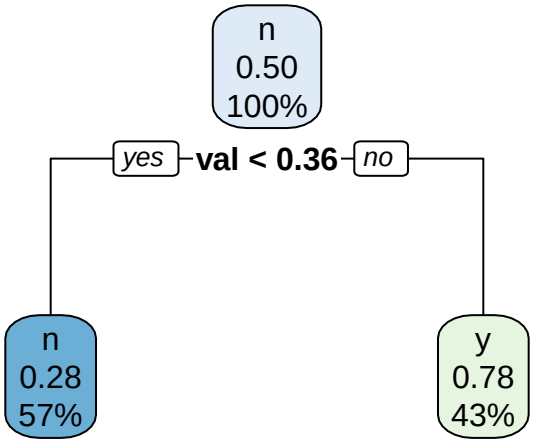
```
entropyValues=sapply(p, entropyValue)
entropyValues
```

```
##  [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 pruned tree has one root node and 2 leaf nodes. Also, the pruned tree is better than previous on with with two leafs and less overlapping labels.

Pruning:

```
prunedTree=prune.rpart(decisionTree2,cp=0.1)

rpart.plot(prunedTree)
```



Calculating Gini and Entropy for all Nodes: p=probability of each node

The gini values are 0.5000, 0.3432, 0.4032.The entropy values are -0.6931472,-0.5269080, -0.5929533.

```
p=c(.5,0.28,0.78)
giniValues=sapply(p, giniScore)
giniValues
```

```
## [1] 0.5000 0.4032 0.3432
```

```
entropyValues=sapply(p, entropyValue)
entropyValues
```

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

Problem-2:

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

```
## Warning: package 'randomForest' was built under R version 4.2.3
```

```
## 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
```

Loading Dataset

```
winequalityWhite = read.csv("winequality-white.csv", sep=";")

winequalityRed = read.csv("winequality-red.csv", sep=";")
```

Creating test train split

```
winequalityWhite$quality = as.factor(winequalityWhite$quality)
trainInd = createDataPartition(winequalityWhite$quality, p = 0.8, list = F)
train.white = winequalityWhite[trainInd,]
test.white = winequalityWhite[-trainInd,]

winequalityRed$quality = as.factor(winequalityRed$quality)
trainIndRed = createDataPartition(winequalityRed$quality, p = 0.8, list = F)
train.red = winequalityRed[trainIndRed,]
test.red = winequalityRed[-trainIndRed,]
```

Building Decision Trees for Red and white wine qualities.

Comparison Red and White Decision Trees:

White Wine Dataset received a decision tree accuracy of 52.45%(+-2). With the Red Wine Dataset, Decision Tree has an accuracy of 57.4%(+-2).

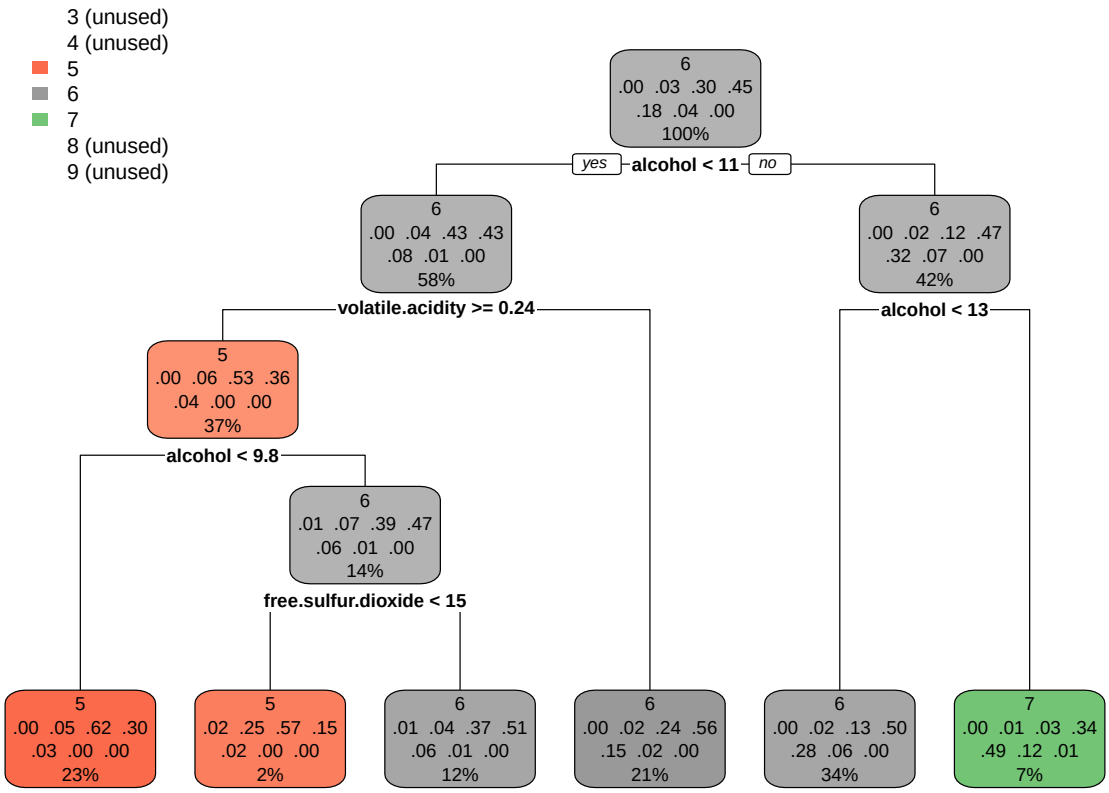
The first split in the White Wine Dataset was at alcohol 11,while the first break in the Red Wine Dataset was at alcohol 10.

Sulphates were considered in the Red Wine Dataset but not in the White Wine Dataset.

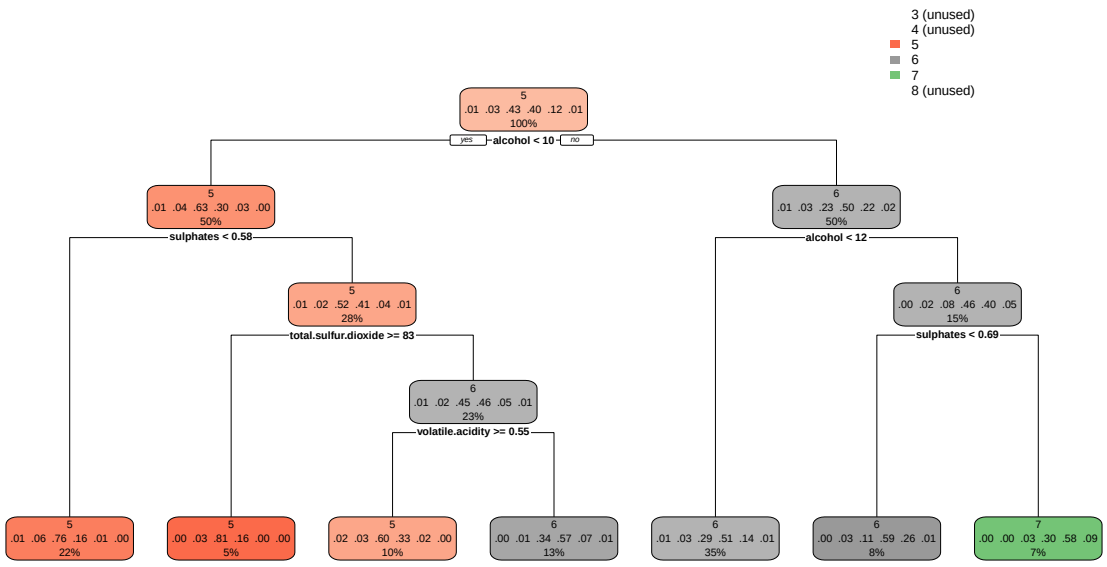
Total Sulfur Dioxide was considered in the Red Wine Dataset but not in the White Wine Dataset.

Free Sulfur Dioxide was considered in the White Wine Dataset but not in the Red Wine Dataset.

```
decisionTreeWhite = rpart(quality~., data=train.white )
rpart.plot(decisionTreeWhite)
```




```
decisionTreeRed = rpart(quality~., data=train.red )
rpart.plot(decisionTreeRed)
```



Confusion Matrix for white and red wine:

```
predWhite = predict(decisionTreeWhite, test.white, type = 'class')
confusionMatrix(predWhite, test.white$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
```

```
predRed = predict(decisionTreeRed, test.red, type = 'class')
confusionMatrix(predRed, test.red$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
```

Random forest Model and its confusion matrix:

Note that accuracy increased to 69.63%(+-2) and 74.45%(+-2) for white and red wine after using RF classifier.

```
RFWhiteModel = train(quality ~ ., data = train.white, method = "rf",preProcess = c("center","scale"))
```

```
## Warning: model fit failed for Resample22: mtry= 2 Error in randomForest.default(x, y, mtry = param$m
##   Can't have empty classes in y.

## Warning: model fit failed for Resample22: mtry= 6 Error in randomForest.default(x, y, mtry = param$m
##   Can't have empty classes in y.

## Warning: model fit failed for Resample22: mtry=11 Error in randomForest.default(x, y, mtry = param$m
##   Can't have empty classes in y.

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
RFpredWhite = predict(RFWhiteModel, test.white)
confusionMatrix(RFpredWhite, test.white$quality)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   0   0   0   0   0
##          4   0   9   2   1   0   0   0
##          5   1  14 206  64   1   1   0
##          6   3   9  82 353  71   8   0
##          7   0   0   1  21 102   9   1
##          8   0   0   0   0   2  17   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.7025
##                  95% CI : (0.6727, 0.731)
##     No Information Rate : 0.4489
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5391
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

```
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000 0.281250   0.7079   0.8041   0.5795  0.48571
## Specificity           1.00000 0.996829   0.8821   0.6790   0.9601  0.99788
## Pos Pred Value            NaN 0.750000   0.7178   0.6711   0.7612  0.89474
## Neg Pred Value        0.99591 0.976190   0.8770   0.8097   0.9123  0.98123
## Prevalence            0.00409 0.032720   0.2975   0.4489   0.1800  0.03579
## Detection Rate        0.00000 0.009202   0.2106   0.3609   0.1043  0.01738
## Detection Prevalence  0.00000 0.012270   0.2935   0.5378   0.1370  0.01943
## Balanced Accuracy     0.50000 0.639039   0.7950   0.7416   0.7698  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
```

```
RFRedModel = train(quality ~ ., data = train.red, method = "rf",preProcess = c("center", "scale"))

RFpredRed = predict(RFRedModel, test.red)
confusionMatrix(RFpredRed, test.red$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   5 112  32   1   0
##          6   1   5  24  87  22   2
##          7   0   0   0   8  16   1
##          8   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6782
##                  95% CI : (0.6237, 0.7294)
##     No Information Rate : 0.429
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4716
##
##  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   0.8235   0.6850  0.41026 0.000000
## Specificity          1.000000 0.996743   0.7901   0.7158  0.96763 1.000000
## Pos Pred Value            NaN 0.000000   0.7467   0.6170  0.64000      NaN
## Neg Pred Value       0.993691 0.968354   0.8563   0.7727  0.92123 0.990536
```

```
## Prevalence           0.006309 0.031546   0.4290   0.4006  0.12303 0.009464
## Detection Rate       0.000000 0.000000   0.3533   0.2744  0.05047 0.000000
## Detection Prevalence 0.000000 0.003155   0.4732   0.4448  0.07886 0.000000
## Balanced Accuracy    0.500000 0.498371   0.8068   0.7004  0.68894 0.500000
```

Problem-3:

Loading Spam Data and required libraries

```
library(NLP)
```

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

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

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

```
## Warning: package 'tm' was built under R version 4.2.3
```

```
library(readr)
library(SnowballC)
```

Training and Test accuracy are as follows 99.16, 97%

```
smsSpamData= read.csv("SMSSpamCollection",sep="\t",header=FALSE,quote="",stringsAsFactors=FALSE)
str(smsSpamData)
```

```
## 'data.frame':    5574 obs. of  2 variables:
##  $ V1: chr  "ham" "ham" "spam" "ham" ...
##  $ V2: chr  "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine
```

```
names(smsSpamData)[1]="type"
names(smsSpamData)[2]="text"
smsSpamData$type=factor(smsSpamData$type)
str(smsSpamData)
```

```
## '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... Ci
```

```
table(smsSpamData$type)
```

```
##
##  ham spam
## 4827  747
```

```
smsCorpus=VCorpus(VectorSource(smsSpamData$text))
print(smsCorpus)
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 5574
```

```
cleansedSMSCorpus=tm_map(smsCorpus,stemDocument)

cleansedSMSCorpus= tm_map(cleansedSMSCorpus,content_transformer(tolower))
as.character(cleansedSMSCorpus[[1]])
```

```
## [1] "go until jurong point, crazy.. availabl onli in bugi n great world la e buffet... cine there go
```

```
#Removing Stop Words:
cleansedSMSCorpus=tm_map(cleansedSMSCorpus,removeWords,stopwords())

#Stripping whitespace:
cleansedSMSCorpus=tm_map(cleansedSMSCorpus,stripWhitespace)

#Removing Punctuation:
cleansedSMSCorpus=tm_map(cleansedSMSCorpus,removePunctuation)
as.character(cleansedSMSCorpus[[1]])
```

```
## [1] "go jurong point crazy availabl onli bugi n great world la e buffet cine got amor wat"
```

```
smsDocTermMatrix=DocumentTermMatrix(cleansedSMSCorpus)
smsDocTermMatrix
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 8438)>>
## Non-/sparse entries: 45137/46988275
## Sparsity           : 100%
## Maximal term length: 51
## Weighting          : term frequency (tf)
```

```
smsDtmTrainData=smsDocTermMatrix[1:4169,]
smsDtmTestData=smsDocTermMatrix[4170:5574,]
smsTrainLabel=smsSpamData[1:4169,]$type
smsTestLabel=smsSpamData[4170:5574,]$type


frequentTerms=findFreqTerms(smsDtmTrainData,10)
smsDtmFreqTrain=smsDtmTrainData[,frequentTerms]
smsDtmFreqTest=smsDtmTestData[,frequentTerms]

convertToBoolean=function(x){x=ifelse(x>0,1,0)}
smsTrainData=apply(smsDtmFreqTrain,MARGIN = 2,convertToBoolean)
smsTestData=apply(smsDtmFreqTest,MARGIN = 2,convertToBoolean)

library(e1071)
```

```
smsClassifier=svm(smsTrainData,smsTrainLabel)

library(caret)
smsTrainPred=predict(smsClassifier,smsTrainData)
smsTestPred=predict(smsClassifier,smsTestData)
confusionMatrix(smsTrainPred,smsTrainLabel)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  ham spam
##       ham  3605   35
##       spam    0  529
##
##                Accuracy : 0.9916
##                  95% CI : (0.9883, 0.9941)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9632
##
##  Mcnemar's Test P-Value : 9.081e-09
##
##             Sensitivity : 1.0000
##             Specificity : 0.9379
##          Pos Pred Value : 0.9904
##          Neg Pred Value : 1.0000
##              Prevalence : 0.8647
##          Detection Rate : 0.8647
##    Detection Prevalence : 0.8731
##       Balanced Accuracy : 0.9690
##
##        'Positive' Class : ham
##
```

```
confusionMatrix(smsTestPred,smsTestLabel)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  ham spam
##       ham  1221   40
##       spam    1  143
##
##                Accuracy : 0.9708
##                  95% CI : (0.9606, 0.979)
##     No Information Rate : 0.8698
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8584
##
##  Mcnemar's Test P-Value : 2.946e-09
##
```

```
##             Sensitivity : 0.9992
##             Specificity : 0.7814
##          Pos Pred Value : 0.9683
##          Neg Pred Value : 0.9931
##              Prevalence : 0.8698
##          Detection Rate : 0.8690
##    Detection Prevalence : 0.8975
##       Balanced Accuracy : 0.8903
##
##        'Positive' Class : ham
##
```