DPA Assignment 4

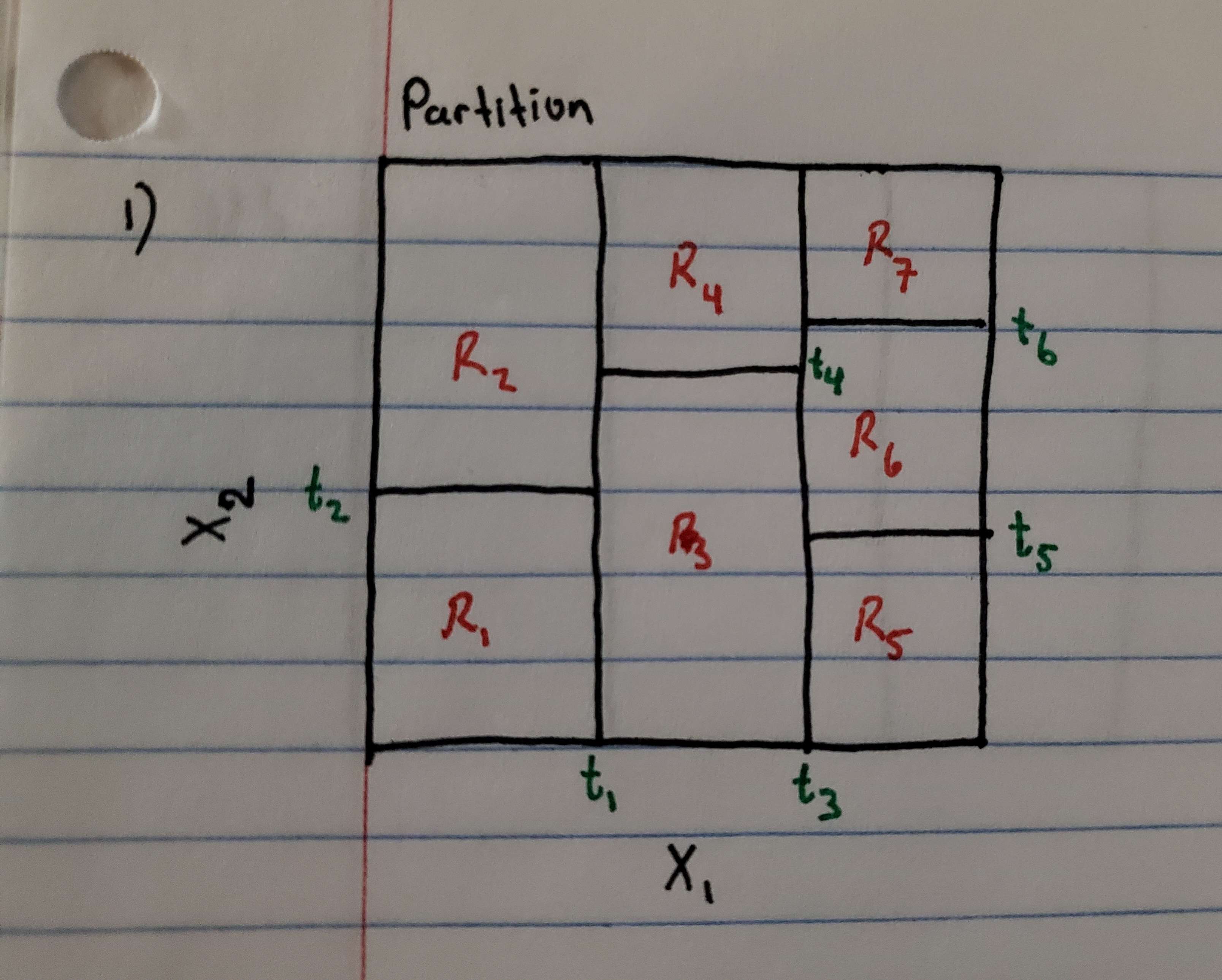
Girish Rajani

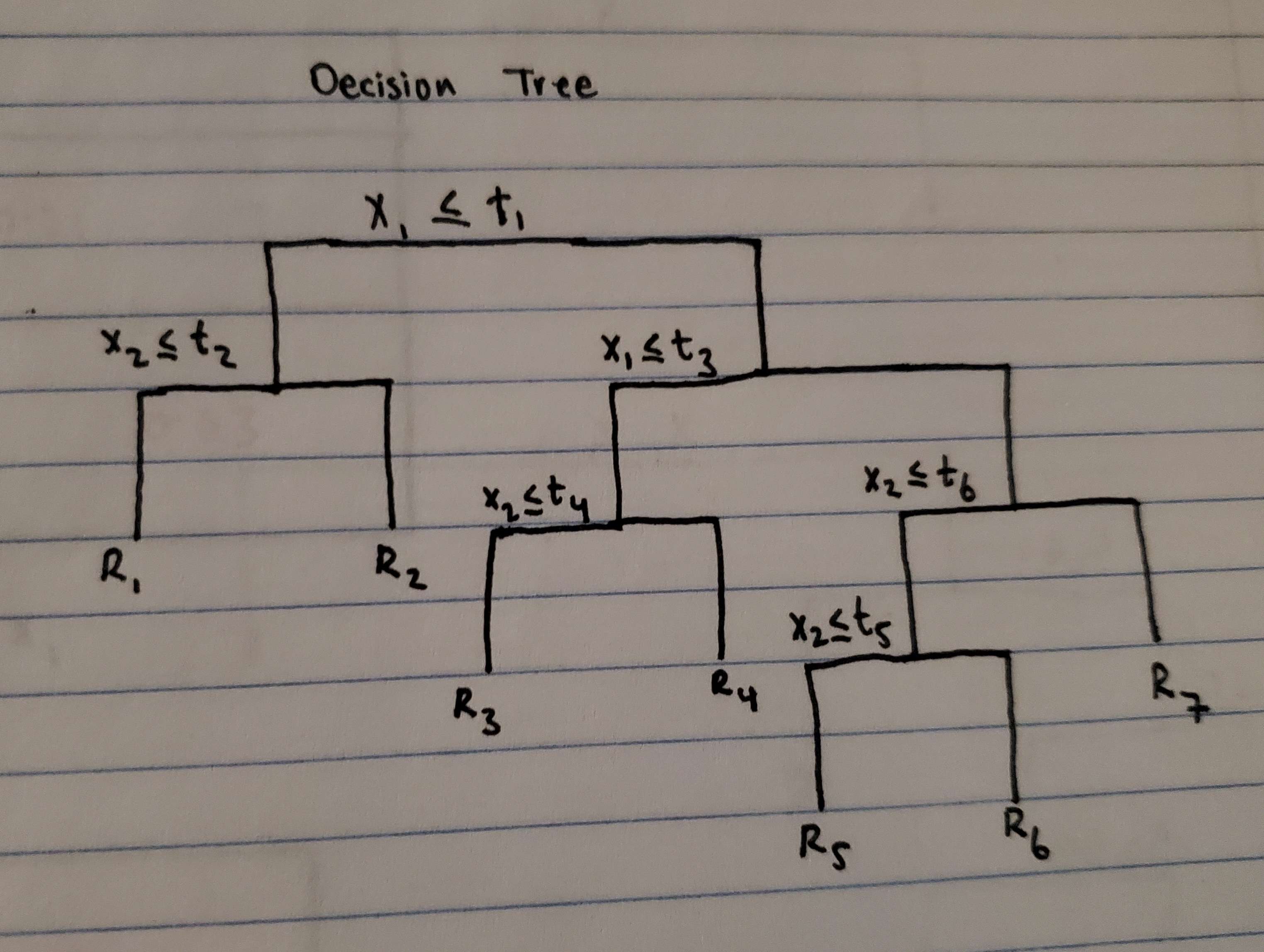
2023-04-09

#Recitation Problems

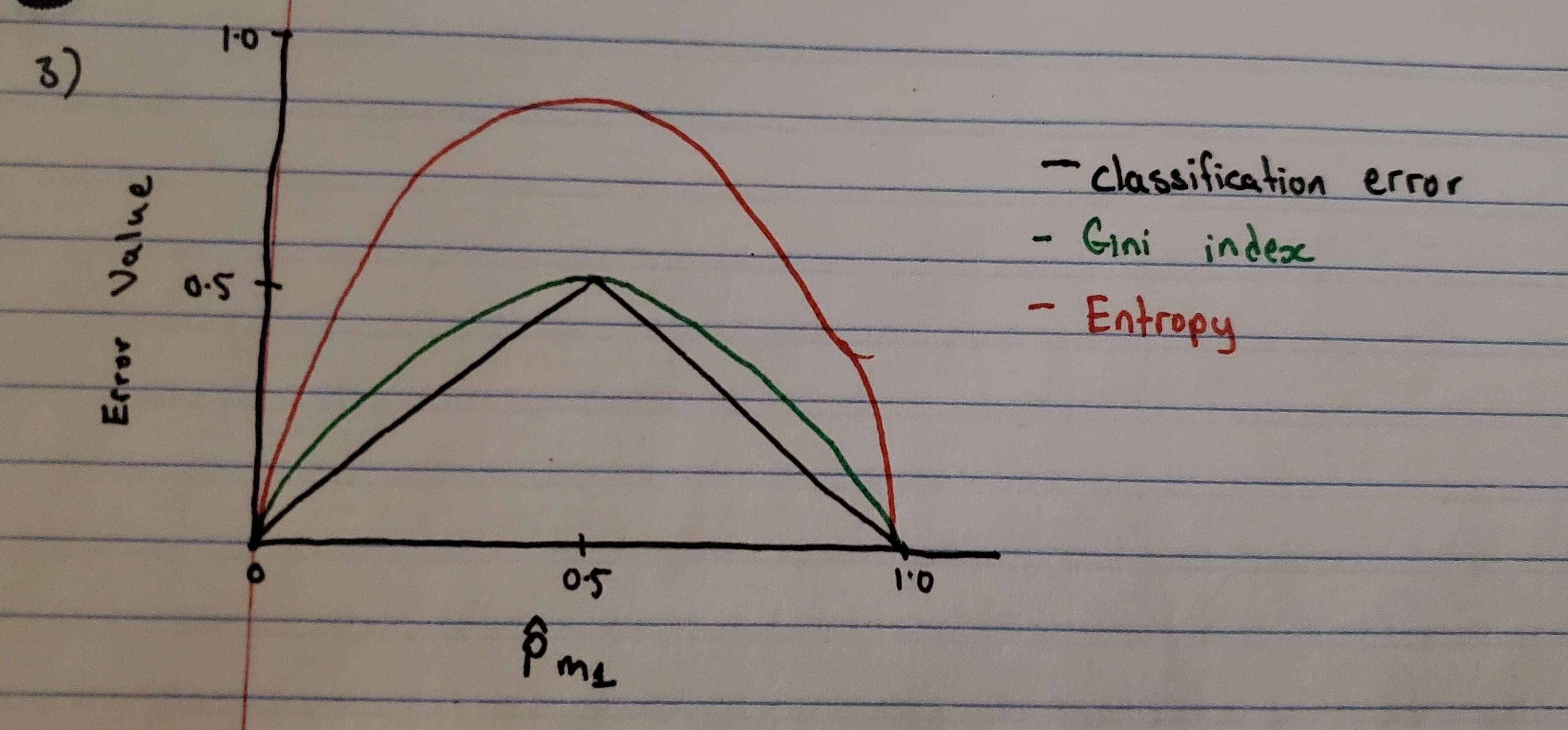
#Chapter 8

1)

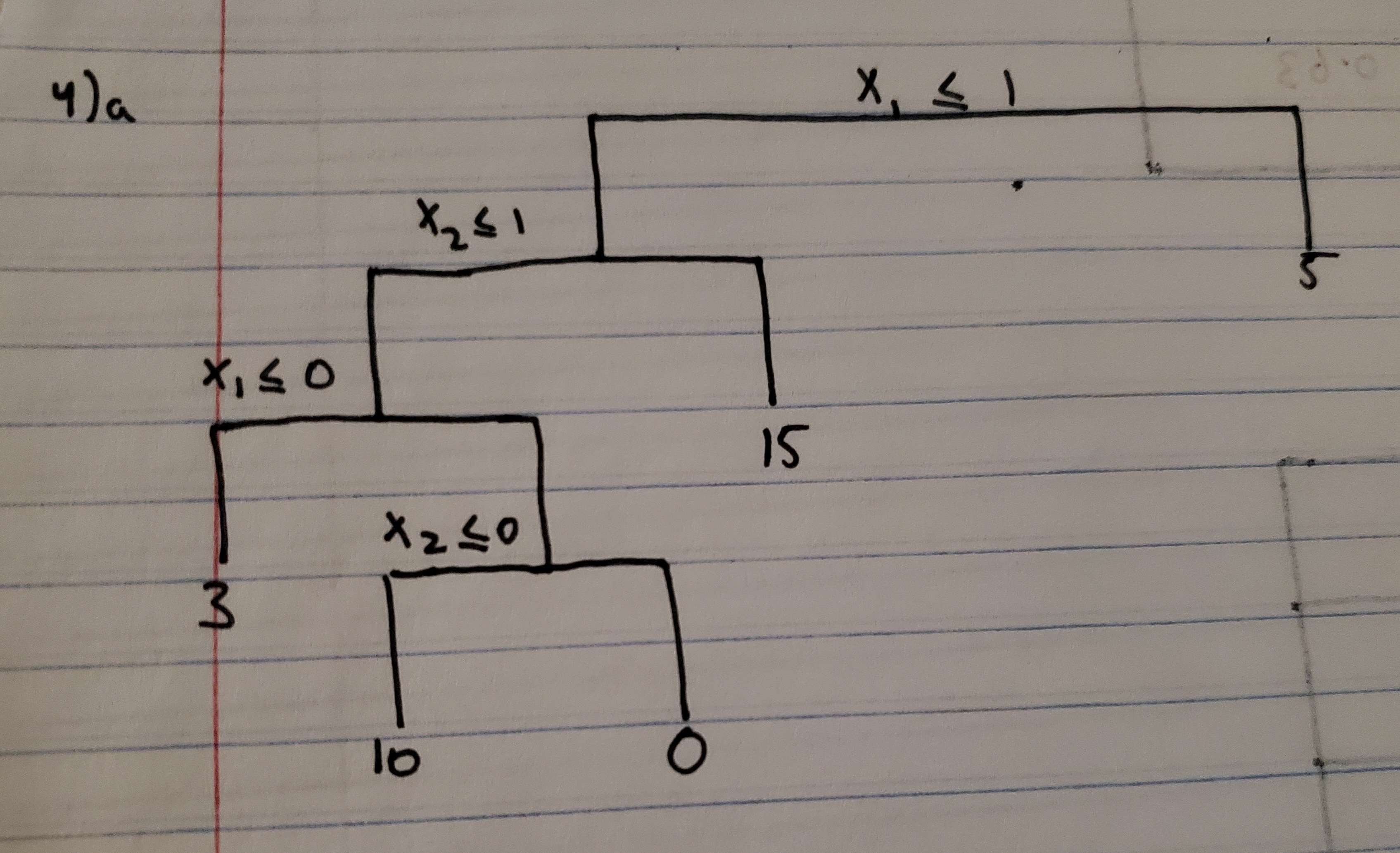


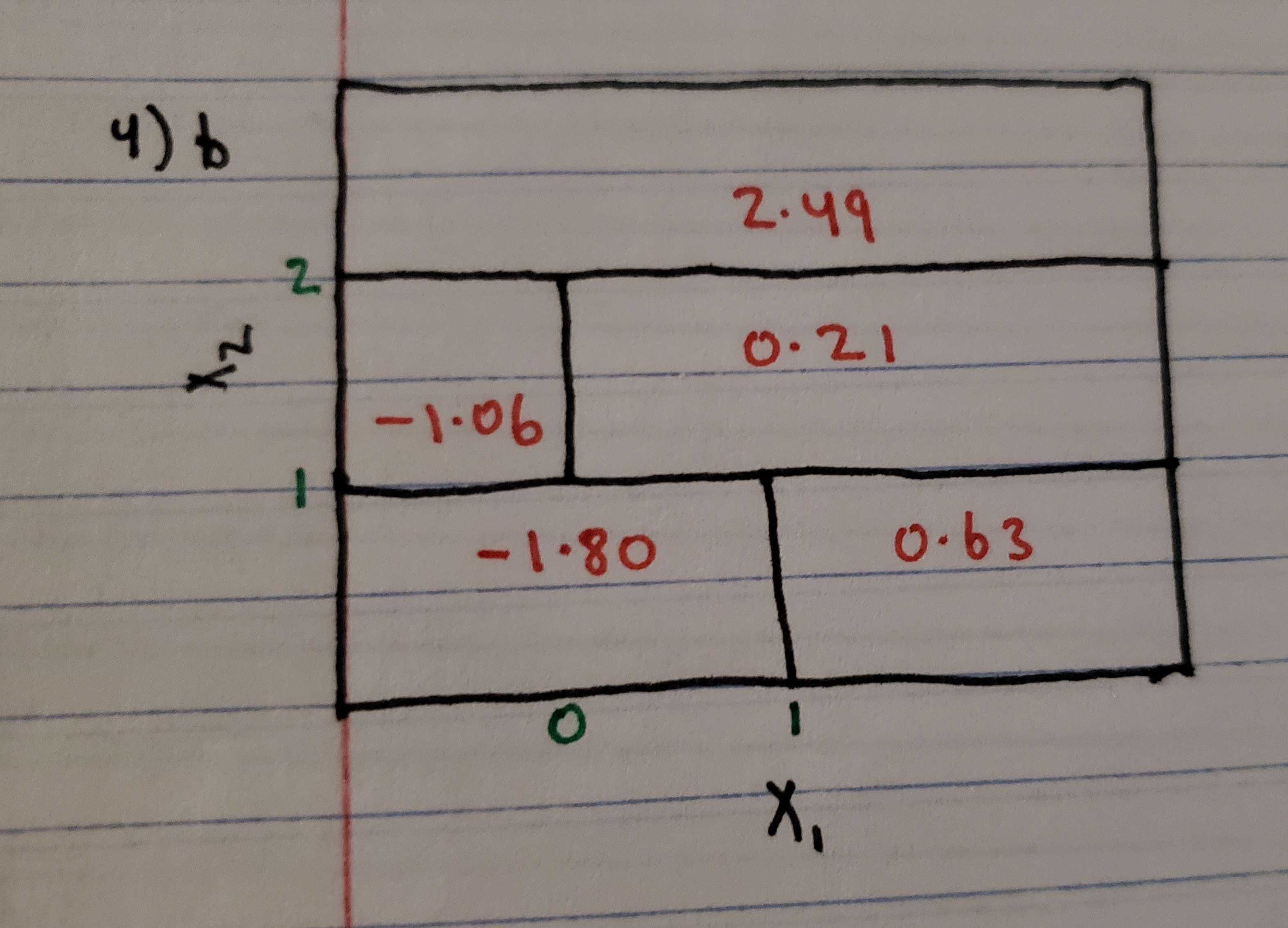


3)



4)





5)

Majority Vote Approach: Number of estimates for P(Class is Red | X) < 0.5 is 4 (4 estimates belong to Green class) Number of estimates for P(Class is Red | X) >= 0.5 is 6 (6 estimates belong to Red class)

Using majority vote, the specific value X would be classified as Red since majority of the estimates belong to Red Class

Average Probability Approach: Here, we calculate the average of the P(Class is Red | X) from the 10 estimates and if the average value is less than 0.5 we classify the specific value X as Green, else classify it as Red.

Avg Probability = (0.1 + 0.15 + 0.2 + 0.2 + 0.55 + 0.6 + 0.6 + 0.65 + 0.7 + 0.75) / 10 = 4.5/10 = 0.45

Since the average probability is less than 0.5, the specific value X would be classified as Green

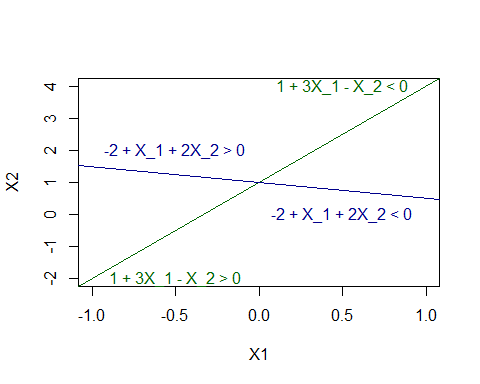
#Chapter 9

#1

1a. To sketch the hyperplane we can rewrite it in the form of which is

1b. To sketch the hyperplane we can rewrite it in the form of which is

#Create an empty plot  
plot(0,0,xlab = "X1", ylab = "X2", xlim = c(-1, 1), ylim = c(-2, 4),type='l')  
  
#Use abline to create the hyperplane using the intercept (1) and slope (3) in X2 = 3X1 + 1  
abline(a = 1, b = 3, col="darkgreen")  
  
#Indicating the set of points / region for which 1 + 3X\_1 - X\_2 < 0 and 1 + 3X\_1 - X\_2 > 0  
text(c(0.5), c(4), "1 + 3X\_1 - X\_2 < 0", col = "darkgreen")   
text(c(-0.5), c(-2), "1 + 3X\_1 - X\_2 > 0", col = "darkgreen")  
  
#Use abline to create the hyperplane using the intercept (1) and slope (-1/2) in X\_2 = - X\_1 / 2 + 1  
abline(a = 1, b = -1/2, col="darkblue")  
  
#Indicating the set of points / region for which -2 + X\_1 + 2X\_2 < 0 and -2 + X\_1 + 2X\_2 > 0  
text(c(0.5), c(0), "-2 + X\_1 + 2X\_2 < 0", col = "darkblue")  
text(c(-0.5), c(2), "-2 + X\_1 + 2X\_2 > 0", col = "darkblue")

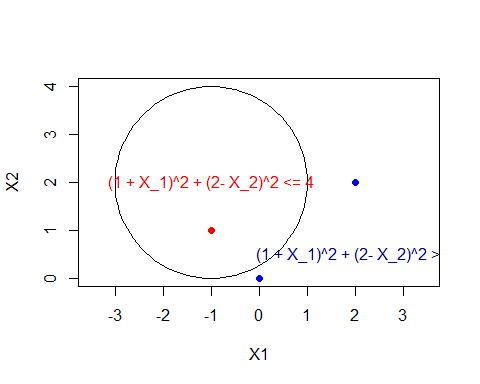


#2a and b

The curve will be a circle and it can be rewritten in the form: where h and k are the coordinates of the center of the circle and r is the radius of the circle (these 3 parameters will be required to draw the circle)

Rewritten form: where the coordinates of the center is (-1,2) and radius is 2

#install.packages('plotrix')  
library(plotrix)  
plot(x=seq(-2,2), y=seq(0,4),type="n", xlab='X1', ylab='X2', asp =1)  
  
#draw circle using coordinates of the center as (-1,2) and radius as 2  
draw.circle(-1,2,2)  
  
#Indicating the set of points / region for which (1 + X\_1)^2 + (2- X\_2)^2 > 4 and (1 + X\_1)^2 + (2- X\_2)^2 <= 4  
text(c(2), c(0.5), "(1 + X\_1)^2 + (2- X\_2)^2 > 4", col = "darkblue")  
text(c(-1), c(2), "(1 + X\_1)^2 + (2- X\_2)^2 <= 4", col = "red")  
points(c(0,2,3),c(0,2,8), col="blue",pch=19)  
points(-1,1, col="red",pch=19)



#2c

Blue class if and red class otherwise

(0,0) –> which is > 4 so it belongs to class blue

(-1,1) –> which is < 4 so it belongs to class red

(2,2) –> which is > 4 so it belongs to class blue

(3,8) –> which is > 4 so it belongs to class blue

#2d

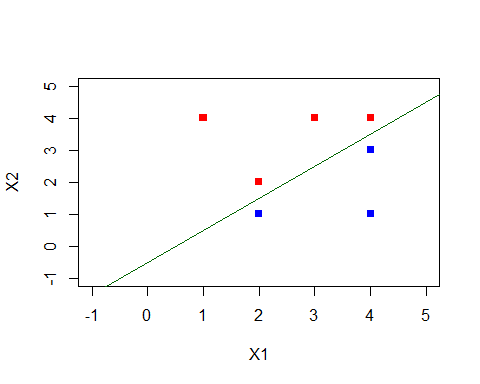
We can expand on equation to include and :

Combine to get:

As we can see from above, when we include and , the decision boundary in (c) results in a linear equation

#3a and b

plot(-1:5,-1:5,type="n",xlab='X1', ylab='X2')  
points(c(3,2,4,1),c(4,2,4,4), col="red", pch=15)  
points(c(2,4,4),c(1,3,1), col="blue", pch=15)  
  
#3b plotting the optimal separating hyperplane  
abline(a = -0.5, b = 1, col="darkgreen")



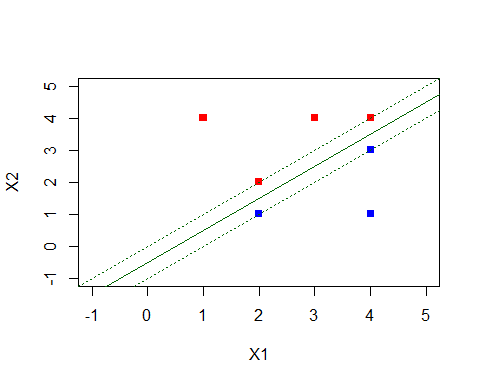
#3b continued Using the format , the equation for the hyperplane above would be

In the form of 9.1 with and , we get or

#3c Classify to red if else blue otherwise where , , and

#3d

plot(-1:5,-1:5,type="n",xlab='X1', ylab='X2')  
points(c(3,2,4,1),c(4,2,4,4), col="red", pch=15)  
points(c(2,4,4),c(1,3,1), col="blue", pch=15)  
  
#3b plotting the optimal separating hyperplane  
abline(a = -0.5, b = 1, col="darkgreen")  
  
#3d margin for maximal margin hyperplane  
abline(-1, 1, col='darkgreen',lty='dotted')  
abline(0, 1, col='darkgreen',lty='dotted')

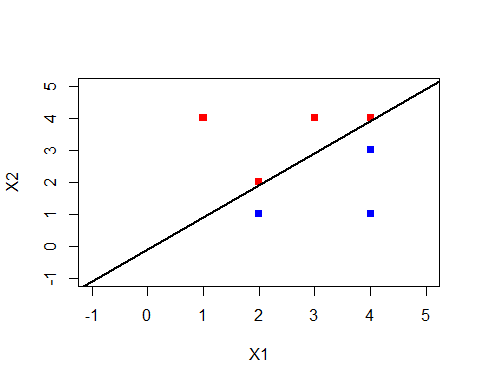


#3e the support vectors for the maximal margin classifier:(2,1), (2,2), (4,3) and (4,4)

#3f A slight movement of the seventh observation (4,1) would not affect the maximal margin hyperplane because it is not a support vector (only those mentioned in 3e are). Since only movements in support vectors will affect the maximal margin hyperplane, a slight movement in the seventh observation will not affect the maximal margin hyperplane.

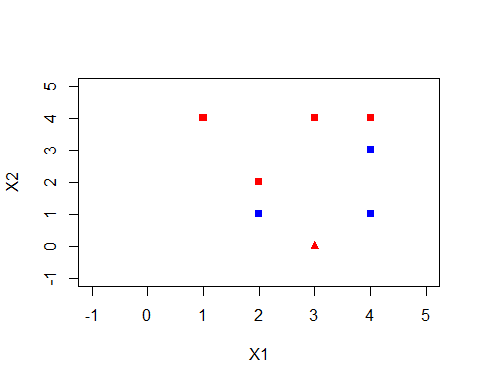
#3g

plot(-1:5,-1:5,type="n",xlab='X1', ylab='X2')  
points(c(3,2,4,1),c(4,2,4,4), col="red", pch=15)  
points(c(2,4,4),c(1,3,1), col="blue", pch=15)  
  
#3g plotting non optimal separating hyperparameter where intercept a is -0.1 and slope b is 1 which results in $-0.1 + X\_1 - X\_2 = 0$ as the equation for the hyperplane  
abline(-0.1,1, col="black", lwd=2)



#3h

plot(-1:5,-1:5,type="n",xlab='X1', ylab='X2')  
points(c(3,2,4,1),c(4,2,4,4), col="red", pch=15)  
points(c(2,4,4),c(1,3,1), col="blue", pch=15)  
  
#3h (additional observation represented by red triangle)  
points(3,0, col="red", pch=17)



#Question 1

#install.packages("rpart.plot")  
#install.packages("randomForest")  
library(rpart)  
library(rpart.plot)

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

library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

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

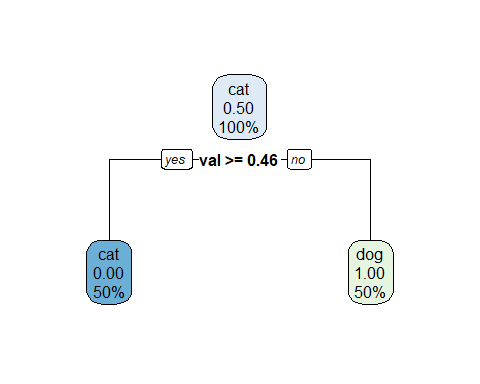
set.seed(100)  
  
#Use rnorm to simulate the normal distributions for the 2 datasets   
normal\_dist\_1 <- rnorm(n = 200, mean = 5, sd = 2)  
normal\_dist\_2 <- rnorm(n = 200, mean = -5, sd = 2)  
  
#Use the normal distribution to simulate dataframes with a class label then combine them together  
data\_frame1 <- data.frame(val = normal\_dist\_1, label = rep("cat",200))  
data\_frame2 <- data.frame(val = normal\_dist\_2, label = rep("dog",200))  
dataset1 <- rbind(data\_frame1, data\_frame2)  
  
#Convert the label feature to a categorical variable  
dataset1$label <- as.factor(dataset1$label)  
  
head(dataset1)

## val label  
## 1 3.995615 cat  
## 2 5.263062 cat  
## 3 4.842166 cat  
## 4 6.773570 cat  
## 5 5.233943 cat  
## 6 5.637260 cat

summary(dataset1)

## val label   
## Min. :-11.0416 cat:200   
## 1st Qu.: -5.0323 dog:200   
## Median : 0.5916   
## Mean : -0.0282   
## 3rd Qu.: 4.8694   
## Max. : 10.1639

#Inducing a binary decision tree using rpart then plotting it using rpart.plot  
binary\_decision\_tree <- rpart(label~val, dataset1, method = "class")  
rpart.plot(binary\_decision\_tree)



The threshold value for the feature in the first split is 0.46 as shown above. Since the tree is able to classify both classes, it follows an empirical distribution. The tree above has three nodes (1 root node and 2 leaf nodes).

#Write our functions to calculate gini and entropy  
entropy\_func <- function(p){  
entropy = (p\*log(p) + (1-p) \* log(1-p))  
return (entropy)}  
  
gini\_func <- function(p){  
gini = 2 \* p \* (1-p)  
return (gini)}

#Use the above entropy and gini functions to compute the entropy and gini at each node  
p\_tree1 <- c(0.5, 0, 1)  
  
entropy\_tree1 <- sapply(p\_tree1, entropy\_func)  
cat("Entropy at each node of tree 1: ",entropy\_tree1)

## Entropy at each node of tree 1: -0.6931472 NaN NaN

gini\_tree1 <- sapply(p\_tree1, gini\_func)  
cat("\nGini at each node of tree 1: ",gini\_tree1)

##   
## Gini at each node of tree 1: 0.5 0 0

#Repeat the same process using normal distributions of (1,2) and (-1,2)  
set.seed(200)  
#Use rnorm to simulate the normal distributions for the 2 datasets   
normal\_dist\_1 <- rnorm(n = 100, mean = 1, sd = 2)  
normal\_dist\_2 <- rnorm(n = 100, mean = -1, sd = 2)  
  
#Use the normal distribution to simulate dataframes with a class label then combine them together  
data\_frame1 <- data.frame(val = normal\_dist\_1, label = rep("cat",100))  
data\_frame2 <- data.frame(val = normal\_dist\_2, label = rep("dog",100))  
dataset2 <- rbind(data\_frame1, data\_frame2)  
  
#Convert the label feature to a categorical variable  
dataset2$label <- as.factor(dataset2$label)  
  
head(dataset2)

## val label  
## 1 1.1695127 cat  
## 2 1.4529207 cat  
## 3 1.8651130 cat  
## 4 2.1161305 cat  
## 5 1.1195105 cat  
## 6 0.7707183 cat

summary(dataset2)

## val label   
## Min. :-5.461128 cat:100   
## 1st Qu.:-1.421417 dog:100   
## Median : 0.009538   
## Mean :-0.012207   
## 3rd Qu.: 1.313753   
## Max. : 7.175955

#Inducing a binary decision tree using rpart then plotting it using rpart.plot  
binary\_decision\_tree2 <- rpart(label~val, dataset2, method = "class")  
rpart.plot(binary\_decision\_tree2)

Timeline

Description automatically generated

The threshold value for the feature in the first split is 0.68 as shown above. The tree above has 13 nodes. Due to this large size of decision tree, there are overlapping of labels. This means that the decision tree is more likely to overfit and be less interpretable.

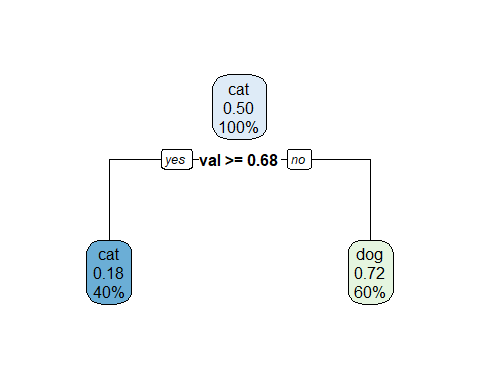
#Use the above entropy and gini functions to compute the entropy and gini at each node  
p\_tree2 <- c(0.5,0.18,0.72,0.63,0.54,0.22,0.64,0.71,0.57,0.36,0.80,0.85,0.88)  
  
entropy\_tree2 <- sapply(p\_tree2, entropy\_func)  
cat("Entropy at each node of tree 2: ",entropy\_tree2)

## Entropy at each node of tree 2: -0.6931472 -0.4713935 -0.5929533 -0.6589557 -0.6899438 -0.526908 -0.6534182 -0.6021517 -0.6833149 -0.6534182 -0.5004024 -0.4227091 -0.366925

gini\_tree2 <- sapply(p\_tree2, gini\_func)  
cat("\nGini at each node of tree 2: ",gini\_tree2)

##   
## Gini at each node of tree 2: 0.5 0.2952 0.4032 0.4662 0.4968 0.3432 0.4608 0.4118 0.4902 0.4608 0.32 0.255 0.2112

#Pruning the tree  
  
binary\_decision\_tree2\_prune <- prune.rpart(binary\_decision\_tree2, cp = 0.1)  
rpart.plot(binary\_decision\_tree2\_prune)



From above, we can see that the pruned tree has a lot less nodes, in this case, 3 nodes (1 root and 2 leaf nodes). We can observe that the results of the pruned tree is better than the initial tree due to less nodes and less overlapping labels. This means that parts of the tree that do not contribute to the classifier have been removed. This decreases the risk of overfitting.

#Use the entropy and gini functions to compute the entropy and gini at each node  
p\_tree3 <- c(0.5,0.18,0.72)  
  
entropy\_tree3 <- sapply(p\_tree3, entropy\_func)  
cat("Entropy at each node of tree 3: ",entropy\_tree3)

## Entropy at each node of tree 3: -0.6931472 -0.4713935 -0.5929533

gini\_tree3 <- sapply(p\_tree3, gini\_func)  
cat("\nGini at each node of tree 3: ",gini\_tree3)

##   
## Gini at each node of tree 3: 0.5 0.2952 0.4032

#Question 2

set.seed(2)  
white\_wine\_data <- read.csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv', header = TRUE, sep = ";")  
summary(white\_wine\_data)

## fixed.acidity volatile.acidity citric.acid residual.sugar   
## Min. : 3.800 Min. :0.0800 Min. :0.0000 Min. : 0.600   
## 1st Qu.: 6.300 1st Qu.:0.2100 1st Qu.:0.2700 1st Qu.: 1.700   
## Median : 6.800 Median :0.2600 Median :0.3200 Median : 5.200   
## Mean : 6.855 Mean :0.2782 Mean :0.3342 Mean : 6.391   
## 3rd Qu.: 7.300 3rd Qu.:0.3200 3rd Qu.:0.3900 3rd Qu.: 9.900   
## Max. :14.200 Max. :1.1000 Max. :1.6600 Max. :65.800   
## chlorides free.sulfur.dioxide total.sulfur.dioxide density   
## Min. :0.00900 Min. : 2.00 Min. : 9.0 Min. :0.9871   
## 1st Qu.:0.03600 1st Qu.: 23.00 1st Qu.:108.0 1st Qu.:0.9917   
## Median :0.04300 Median : 34.00 Median :134.0 Median :0.9937   
## Mean :0.04577 Mean : 35.31 Mean :138.4 Mean :0.9940   
## 3rd Qu.:0.05000 3rd Qu.: 46.00 3rd Qu.:167.0 3rd Qu.:0.9961   
## Max. :0.34600 Max. :289.00 Max. :440.0 Max. :1.0390   
## pH sulphates alcohol quality   
## Min. :2.720 Min. :0.2200 Min. : 8.00 Min. :3.000   
## 1st Qu.:3.090 1st Qu.:0.4100 1st Qu.: 9.50 1st Qu.:5.000   
## Median :3.180 Median :0.4700 Median :10.40 Median :6.000   
## Mean :3.188 Mean :0.4898 Mean :10.51 Mean :5.878   
## 3rd Qu.:3.280 3rd Qu.:0.5500 3rd Qu.:11.40 3rd Qu.:6.000   
## Max. :3.820 Max. :1.0800 Max. :14.20 Max. :9.000

head(white\_wine\_data)

## fixed.acidity volatile.acidity citric.acid residual.sugar chlorides  
## 1 7.0 0.27 0.36 20.7 0.045  
## 2 6.3 0.30 0.34 1.6 0.049  
## 3 8.1 0.28 0.40 6.9 0.050  
## 4 7.2 0.23 0.32 8.5 0.058  
## 5 7.2 0.23 0.32 8.5 0.058  
## 6 8.1 0.28 0.40 6.9 0.050  
## free.sulfur.dioxide total.sulfur.dioxide density pH sulphates alcohol  
## 1 45 170 1.0010 3.00 0.45 8.8  
## 2 14 132 0.9940 3.30 0.49 9.5  
## 3 30 97 0.9951 3.26 0.44 10.1  
## 4 47 186 0.9956 3.19 0.40 9.9  
## 5 47 186 0.9956 3.19 0.40 9.9  
## 6 30 97 0.9951 3.26 0.44 10.1  
## quality  
## 1 6  
## 2 6  
## 3 6  
## 4 6  
## 5 6  
## 6 6

red\_wine\_data <- read.csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv', header = TRUE, sep = ";")  
#str(red\_wine\_data)  
summary(red\_wine\_data)

## fixed.acidity volatile.acidity citric.acid residual.sugar   
## Min. : 4.60 Min. :0.1200 Min. :0.000 Min. : 0.900   
## 1st Qu.: 7.10 1st Qu.:0.3900 1st Qu.:0.090 1st Qu.: 1.900   
## Median : 7.90 Median :0.5200 Median :0.260 Median : 2.200   
## Mean : 8.32 Mean :0.5278 Mean :0.271 Mean : 2.539   
## 3rd Qu.: 9.20 3rd Qu.:0.6400 3rd Qu.:0.420 3rd Qu.: 2.600   
## Max. :15.90 Max. :1.5800 Max. :1.000 Max. :15.500   
## chlorides free.sulfur.dioxide total.sulfur.dioxide density   
## Min. :0.01200 Min. : 1.00 Min. : 6.00 Min. :0.9901   
## 1st Qu.:0.07000 1st Qu.: 7.00 1st Qu.: 22.00 1st Qu.:0.9956   
## Median :0.07900 Median :14.00 Median : 38.00 Median :0.9968   
## Mean :0.08747 Mean :15.87 Mean : 46.47 Mean :0.9967   
## 3rd Qu.:0.09000 3rd Qu.:21.00 3rd Qu.: 62.00 3rd Qu.:0.9978   
## Max. :0.61100 Max. :72.00 Max. :289.00 Max. :1.0037   
## pH sulphates alcohol quality   
## Min. :2.740 Min. :0.3300 Min. : 8.40 Min. :3.000   
## 1st Qu.:3.210 1st Qu.:0.5500 1st Qu.: 9.50 1st Qu.:5.000   
## Median :3.310 Median :0.6200 Median :10.20 Median :6.000   
## Mean :3.311 Mean :0.6581 Mean :10.42 Mean :5.636   
## 3rd Qu.:3.400 3rd Qu.:0.7300 3rd Qu.:11.10 3rd Qu.:6.000   
## Max. :4.010 Max. :2.0000 Max. :14.90 Max. :8.000

head(red\_wine\_data)

## fixed.acidity volatile.acidity citric.acid residual.sugar chlorides  
## 1 7.4 0.70 0.00 1.9 0.076  
## 2 7.8 0.88 0.00 2.6 0.098  
## 3 7.8 0.76 0.04 2.3 0.092  
## 4 11.2 0.28 0.56 1.9 0.075  
## 5 7.4 0.70 0.00 1.9 0.076  
## 6 7.4 0.66 0.00 1.8 0.075  
## free.sulfur.dioxide total.sulfur.dioxide density pH sulphates alcohol  
## 1 11 34 0.9978 3.51 0.56 9.4  
## 2 25 67 0.9968 3.20 0.68 9.8  
## 3 15 54 0.9970 3.26 0.65 9.8  
## 4 17 60 0.9980 3.16 0.58 9.8  
## 5 11 34 0.9978 3.51 0.56 9.4  
## 6 13 40 0.9978 3.51 0.56 9.4  
## quality  
## 1 5  
## 2 5  
## 3 5  
## 4 6  
## 5 5  
## 6 5

#Using createDataPartition to perform 80/20 train-test split on white wine data  
  
#Changing our response variable quality to a categorical value  
white\_wine\_data$quality <- as.factor(white\_wine\_data$quality)  
  
datasetPartition <- createDataPartition(white\_wine\_data$quality, p = 0.8, list = FALSE)  
train\_white <- white\_wine\_data[datasetPartition,]  
test\_white <- white\_wine\_data[-datasetPartition,]  
  
cat("Dimensions of training data: ", dim(train\_white))

## Dimensions of training data: 3920 12

cat("\nDimensions of testing data: ", dim(test\_white))

##   
## Dimensions of testing data: 978 12

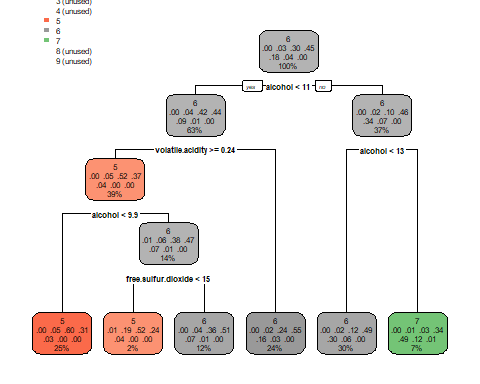
#Using createDataPartition to perform 80/20 train-test split on red wine data  
  
#Changing our response variable quality to a categorical value  
red\_wine\_data$quality <- as.factor(red\_wine\_data$quality)  
  
datasetPartition1 <- createDataPartition(red\_wine\_data$quality, p = 0.8, list = FALSE)  
train\_red <- red\_wine\_data[datasetPartition1,]  
test\_red <- red\_wine\_data[-datasetPartition1,]  
  
cat("Dimensions of training data: ", dim(train\_red))

## Dimensions of training data: 1282 12

cat("\nDimensions of testing data: ", dim(test\_red))

##   
## Dimensions of testing data: 317 12

#Decision tree for white wine  
decision\_tree\_white <- rpart(quality~., train\_white, method = "class")  
rpart.plot(decision\_tree\_white)



#Decision tree for red wine  
decision\_tree\_red <- rpart(quality~., train\_red, method = "class")  
rpart.plot(decision\_tree\_red)

Timeline

Description automatically generated

#using the caret package confusionMatrix method to determine the   
#decision tree accuracy on the white wine test set  
  
predict\_white <- predict(decision\_tree\_white, test\_white, type = 'class')  
confusionMatrix(predict\_white, 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 14 161 77 7 0 0  
## 6 2 17 129 339 132 23 1  
## 7 1 1 1 23 37 12 0  
## 8 0 0 0 0 0 0 0  
## 9 0 0 0 0 0 0 0  
##   
## Overall Statistics  
##   
## Accuracy : 0.5491   
## 95% CI : (0.5173, 0.5806)  
## No Information Rate : 0.4489   
## P-Value [Acc > NIR] : 2.093e-10   
##   
## Kappa : 0.2632   
##   
## 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 0.5533 0.7722 0.21023 0.00000  
## Specificity 1.00000 1.00000 0.8559 0.4360 0.95262 1.00000  
## Pos Pred Value NaN NaN 0.6192 0.5272 0.49333 NaN  
## Neg Pred Value 0.99591 0.96728 0.8189 0.7015 0.84607 0.96421  
## Prevalence 0.00409 0.03272 0.2975 0.4489 0.17996 0.03579  
## Detection Rate 0.00000 0.00000 0.1646 0.3466 0.03783 0.00000  
## Detection Prevalence 0.00000 0.00000 0.2658 0.6575 0.07669 0.00000  
## Balanced Accuracy 0.50000 0.50000 0.7046 0.6041 0.58142 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

#using the caret package confusionMatrix method to determine the   
#decision tree accuracy on the red wine test set  
  
predict\_red <- predict(decision\_tree\_red, test\_red, type = 'class')  
confusionMatrix(predict\_red, 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 9 108 44 3 0  
## 6 1 1 25 74 26 3  
## 7 0 0 3 9 10 0  
## 8 0 0 0 0 0 0  
##   
## Overall Statistics  
##   
## Accuracy : 0.6057   
## 95% CI : (0.5495, 0.6598)  
## No Information Rate : 0.429   
## P-Value [Acc > NIR] : 1.938e-10   
##   
## Kappa : 0.347   
##   
## 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 0.7941 0.5827 0.25641 0.000000  
## Specificity 1.000000 1.00000 0.6851 0.7053 0.95683 1.000000  
## Pos Pred Value NaN NaN 0.6545 0.5692 0.45455 NaN  
## Neg Pred Value 0.993691 0.96845 0.8158 0.7166 0.90169 0.990536  
## Prevalence 0.006309 0.03155 0.4290 0.4006 0.12303 0.009464  
## Detection Rate 0.000000 0.00000 0.3407 0.2334 0.03155 0.000000  
## Detection Prevalence 0.000000 0.00000 0.5205 0.4101 0.06940 0.000000  
## Balanced Accuracy 0.500000 0.50000 0.7396 0.6440 0.60662 0.500000

It can be seen that the decision tree for white wine had an accuracy of 54.91% while the decision tree for the red wine had an accuracy of 60.57%.

For the white wine decision tree, it was observed that the first split on alcohol < 11 while the red wine decision tree performed the first split on alcohol < 10

In terms of variables of interest:

The white wine decision tree utilized the free sulfur dioxide variable while the red wine decision tree did not

On the flip side, the red wine decision tree utilized the total sulfur dioxide, sulphates, and residual sugar variable while the white wine decision tree did not.

#Repeat with a random forest tree model for white wine data  
random\_forest\_white <- train(quality ~ ., data = train\_white, method = "rf", preProcess = c("center","scale"))

predict\_white\_random\_forest <- predict(random\_forest\_white, test\_white)  
confusionMatrix(predict\_white\_random\_forest, 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 7 1 0 0 0 0  
## 5 1 14 201 52 1 0 0  
## 6 3 11 87 358 80 7 1  
## 7 0 0 2 28 94 11 0  
## 8 0 0 0 1 1 17 0  
## 9 0 0 0 0 0 0 0  
##   
## Overall Statistics  
##   
## Accuracy : 0.6922   
## 95% CI : (0.6622, 0.7211)  
## No Information Rate : 0.4489   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.5201   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8  
## Sensitivity 0.00000 0.218750 0.6907 0.8155 0.53409 0.48571  
## Specificity 1.00000 0.998943 0.9010 0.6494 0.94888 0.99788  
## Pos Pred Value NaN 0.875000 0.7472 0.6545 0.69630 0.89474  
## Neg Pred Value 0.99591 0.974227 0.8731 0.8121 0.90273 0.98123  
## Prevalence 0.00409 0.032720 0.2975 0.4489 0.17996 0.03579  
## Detection Rate 0.00000 0.007157 0.2055 0.3661 0.09611 0.01738  
## Detection Prevalence 0.00000 0.008180 0.2751 0.5593 0.13804 0.01943  
## Balanced Accuracy 0.50000 0.608846 0.7959 0.7324 0.74148 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

From above, we can see that when we use random forest tree model on the white wine data we get an accuracy of 69.22% which when compared to the decision tree model accuracy of 54.91%, shows an improvement.

#Repeat with a random forest tree model for red wine data  
random\_forest\_red <- train(quality ~ ., data = train\_red, method = "rf", preProcess = c("center","scale"))  
predict\_red\_random\_forest <- predict(random\_forest\_red, test\_red)  
confusionMatrix(predict\_red\_random\_forest, 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 7 111 21 0 0  
## 6 1 3 25 102 18 1  
## 7 0 0 0 4 21 2  
## 8 0 0 0 0 0 0  
##   
## Overall Statistics  
##   
## Accuracy : 0.7382   
## 95% CI : (0.6861, 0.7857)  
## No Information Rate : 0.429   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.5711   
##   
## 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 0.8162 0.8031 0.53846 0.000000  
## Specificity 1.000000 1.00000 0.8398 0.7474 0.97842 1.000000  
## Pos Pred Value NaN NaN 0.7929 0.6800 0.77778 NaN  
## Neg Pred Value 0.993691 0.96845 0.8588 0.8503 0.93793 0.990536  
## Prevalence 0.006309 0.03155 0.4290 0.4006 0.12303 0.009464  
## Detection Rate 0.000000 0.00000 0.3502 0.3218 0.06625 0.000000  
## Detection Prevalence 0.000000 0.00000 0.4416 0.4732 0.08517 0.000000  
## Balanced Accuracy 0.500000 0.50000 0.8280 0.7753 0.75844 0.500000

From above, we can see that when we use random forest tree model on the red wine data we get an accuracy of 73.82% which when compared to the decision tree model accuracy of 60.57%, shows an improvement.

#Question 3

#install.packages("tm")  
library(tm)

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

## Loading required package: NLP

##   
## Attaching package: 'NLP'

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

library(e1071)

sms\_data <- read.csv("D:\\Users\\giris\\Downloads\\SMSSpamCollection",header = FALSE,sep = "\t", quote = "", stringsAsFactors=FALSE)  
colnames(sms\_data) <- c("Class", "Message")  
  
#Changing our response variable quality to a categorical value  
sms\_data$Class <- as.factor(sms\_data$Class)  
  
summary(sms\_data)

## Class Message   
## ham :4827 Length:5574   
## spam: 747 Class :character   
## Mode :character

head(sms\_data)

## Class  
## 1 ham  
## 2 ham  
## 3 spam  
## 4 ham  
## 5 ham  
## 6 spam  
## Message  
## 1 Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...  
## 2 Ok lar... Joking wif u oni...  
## 3 Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's  
## 4 U dun say so early hor... U c already then say...  
## 5 Nah I don't think he goes to usf, he lives around here though  
## 6 FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv

corpusVectorSource <- VCorpus(VectorSource(sms\_data$Message))  
  
#Convert lowercase  
corpusVectorSource <- tm\_map(corpusVectorSource, content\_transformer(tolower))  
  
#Remove stopwords  
corpusVectorSource <- tm\_map(corpusVectorSource, removeWords, stopwords("en"))  
  
#Strip whitespace  
corpusVectorSource <- tm\_map(corpusVectorSource, stripWhitespace)  
  
#Remove punctuation  
corpusVectorSource <- tm\_map(corpusVectorSource, removePunctuation)

#Create DocumentTermMatrix  
doc\_term\_matrix <- DocumentTermMatrix(corpusVectorSource)  
doc\_term\_matrix

## <<DocumentTermMatrix (documents: 5574, terms: 8879)>>  
## Non-/sparse entries: 44937/49446609  
## Sparsity : 100%  
## Maximal term length: 51  
## Weighting : term frequency (tf)

#Use findFreqTerms to contruct features from words occuring more than 10 times  
const\_features <- findFreqTerms(doc\_term\_matrix, 10)

#split the data into a training and test set  
doc\_term\_matrix\_train = doc\_term\_matrix[1:3999,]  
doc\_term\_matrix\_test = doc\_term\_matrix[4000:5574,]  
  
train\_labels <- sms\_data[1:3999, ]$Class  
train\_labels <- as.factor(train\_labels)  
test\_labels <- sms\_data[4000:5574, ]$Class  
test\_labels <- as.factor(test\_labels)

#create a DocumentTermMatrix for each  
freq\_doc\_term\_matrix\_train <- doc\_term\_matrix\_train[, const\_features]  
freq\_doc\_term\_matrix\_test <- doc\_term\_matrix\_test[, const\_features]  
  
freq\_doc\_term\_matrix\_train

## <<DocumentTermMatrix (documents: 3999, terms: 855)>>  
## Non-/sparse entries: 20999/3398146  
## Sparsity : 99%  
## Maximal term length: 15  
## Weighting : term frequency (tf)

#Convert the DocumentTermMatrix train/test matrices to a Boolean representation (counts greater than zero are converted to a 1)  
count\_func <- function(x) {x <- ifelse(x > 0, "1", "0")}  
  
#Apply boolean representation function to train/test matrices  
train <- apply(freq\_doc\_term\_matrix\_train, MARGIN = 2,count\_func)  
test <- apply(freq\_doc\_term\_matrix\_test, MARGIN = 2, count\_func)  
train <- as.data.frame(train)  
test <- as.data.frame(test)

#fit a naiveBayes model for both training and testing data  
fit\_nb\_train = naiveBayes(train,train\_labels)  
fit\_nb\_test = naiveBayes(test,test\_labels)  
  
#Perform prediction using training and testing data  
nb\_train\_pred = predict(fit\_nb\_train,train)  
nb\_test\_pred = predict(fit\_nb\_test,test)

#Show confusion matrix for training data after prediction to observe accuracy  
train\_confusion\_matrix <- confusionMatrix(nb\_train\_pred,train\_labels)  
train\_confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction ham spam  
## ham 3457 54  
## spam 9 479  
##   
## Accuracy : 0.9842   
## 95% CI : (0.9799, 0.9879)  
## No Information Rate : 0.8667   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9293   
##   
## Mcnemar's Test P-Value : 2.965e-08   
##   
## Sensitivity : 0.9974   
## Specificity : 0.8987   
## Pos Pred Value : 0.9846   
## Neg Pred Value : 0.9816   
## Prevalence : 0.8667   
## Detection Rate : 0.8645   
## Detection Prevalence : 0.8780   
## Balanced Accuracy : 0.9480   
##   
## 'Positive' Class : ham   
##

#Train accuracy of naiveBayes model is 98.42%

#Show confusion matrix for testing data after prediction to observe accuracy  
test\_confusion\_matrix <- confusionMatrix(nb\_test\_pred,test\_labels)  
test\_confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction ham spam  
## ham 1357 20  
## spam 4 194  
##   
## Accuracy : 0.9848   
## 95% CI : (0.9774, 0.9902)  
## No Information Rate : 0.8641   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.933   
##   
## Mcnemar's Test P-Value : 0.0022   
##   
## Sensitivity : 0.9971   
## Specificity : 0.9065   
## Pos Pred Value : 0.9855   
## Neg Pred Value : 0.9798   
## Prevalence : 0.8641   
## Detection Rate : 0.8616   
## Detection Prevalence : 0.8743   
## Balanced Accuracy : 0.9518   
##   
## 'Positive' Class : ham   
##

#Test accuracy of naiveBayes model is 98.48%

# fit a SVM using the e1071 package  
fit\_svm\_train = svm(train\_labels~., data= train, kernel = "linear")  
  
pred\_svm\_train = predict(fit\_svm\_train, train)  
train\_acc = mean(pred\_svm\_train == train\_labels)  
cat("The train accuracy of SVM is :", train\_acc\*100, "%")

## The train accuracy of SVM is : 99.54989 %

pred\_svm\_test = predict(fit\_svm\_train, test)  
test\_acc = mean(pred\_svm\_test == test\_labels)  
cat("The test accuracy of SVM is :", test\_acc\*100, "%")

## The test accuracy of SVM is : 98.43182 %