Practicum1

Sagar

February 3, 2018

Q1

Loading the dataset and renaming columns

```
glass <- read.csv("C:/Users/Sagar Ghiya/Desktop/glass.txt", header = F)
colnames(glass) <- c("ID","RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","GlassType")
```

Q2 Exploring the data set

```
str(glass)
```

```
## 'data.frame':    214 obs. of  11 variables:
##  $ ID       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ RI       : num  1.52 1.52 1.52 1.52 1.52 ...
##  $ Na       : num  13.6 13.9 13.5 13.2 13.3 ...
##  $ Mg       : num  4.49 3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 ...
##  $ Al       : num  1.1 1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 ...
##  $ Si       : num  71.8 72.7 73 72.6 73.1 ...
##  $ K        : num  0.06 0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 ...
##  $ Ca       : num  8.75 7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 ...
##  $ Ba       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Fe       : num  0 0 0 0 0 0.26 0 0 0 0.11 ...
##  $ GlassType: int  1 1 1 1 1 1 1 1 1 1 ...
```

```
summary(glass)
```

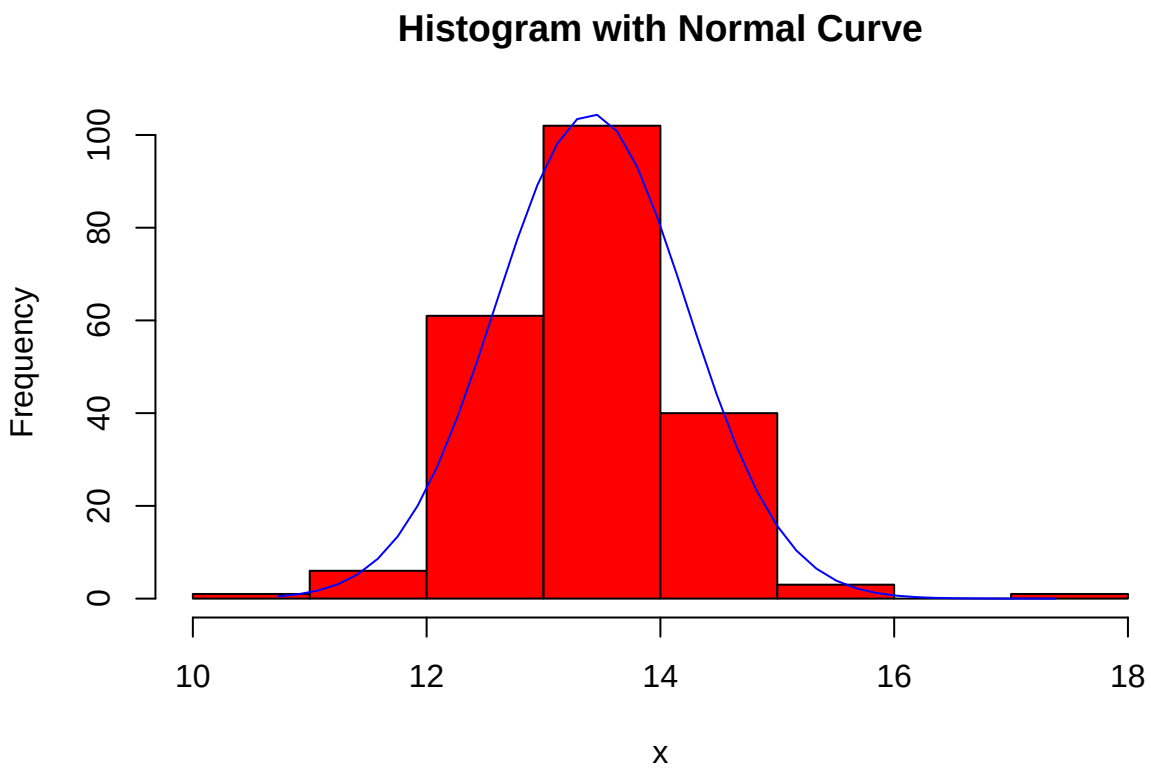
```
##        ID               RI              Na              Mg        
##  Min.   :  1.00   Min.   :1.511   Min.   :10.73   Min.   :0.000  
##  1st Qu.: 54.25   1st Qu.:1.517   1st Qu.:12.91   1st Qu.:2.115  
##  Median :107.50   Median :1.518   Median :13.30   Median :3.480  
##  Mean   :107.50   Mean   :1.518   Mean   :13.41   Mean   :2.685  
##  3rd Qu.:160.75   3rd Qu.:1.519   3rd Qu.:13.82   3rd Qu.:3.600  
##  Max.   :214.00   Max.   :1.534   Max.   :17.38   Max.   :4.490  
##        Al              Si              K               Ca        
##  Min.   :0.290   Min.   :69.81   Min.   :0.0000   Min.   : 5.430  
##  1st Qu.:1.190   1st Qu.:72.28   1st Qu.:0.1225   1st Qu.: 8.240  
##  Median :1.360   Median :72.79   Median :0.5550   Median : 8.600  
##  Mean   :1.445   Mean   :72.65   Mean   :0.4971   Mean   : 8.957  
##  3rd Qu.:1.630   3rd Qu.:73.09   3rd Qu.:0.6100   3rd Qu.: 9.172  
##  Max.   :3.500   Max.   :75.41   Max.   :6.2100   Max.   :16.190  
##        Ba              Fe            GlassType     
##  Min.   :0.000   Min.   :0.00000   Min.   :1.00  
##  1st Qu.:0.000   1st Qu.:0.00000   1st Qu.:1.00  
##  Median :0.000   Median :0.00000   Median :2.00  
##  Mean   :0.175   Mean   :0.05701   Mean   :2.78  
##  3rd Qu.:0.000   3rd Qu.:0.10000   3rd Qu.:3.00  
##  Max.   :3.150   Max.   :0.51000   Max.   :7.00
```

```
sum(is.na(glass))
```

```
## [1] 0
```

Q3

```
x <- glass$Na
h<-hist(x,col = "red",main="Histogram with Normal Curve")
xfit<-seq(min(x),max(x),length=40)
yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))
yfit <- yfit*diff(h$mids[1:2])*length(x)
lines(xfit,yfit,col='blue')
```



The histogram very closely resembles to the blue lined normal distribution plot. Hence the data is normally distributed. KNN is a non-parametric method as it does not try to estimate parameters and then predict. It just takes the new sample and assigns it to a class based on votes from nearest neighbours. Thus the distribution of data does not matter in case of KNN.

Q4

Min-max normalization

```
glass1 <- glass[-1]
```

```
glass1[,1] <- (glass1[,1]-min(glass1[,1]))/(max(glass1[,1])-min(glass1[,1]))
glass1[,2] <- (glass1[,2]-min(glass1[,2]))/(max(glass1[,2])-min(glass1[,2]))
```

Q5 z score standardization

```
zStandardization <- function(x) {
  return (x-mean(x))/sd(x)
}
```

```
glass1[3:9] <- lapply(glass1[3:9],zStandardization)
```

Q6 Stratified sample. Here each glass type is distributed as 50-50% to train and test data.

```
library(splitstackshape)
```

```
## Warning: package 'splitstackshape' was built under R version 3.3.3
```

```
## Loading required package: data.table
```

```
## Warning: package 'data.table' was built under R version 3.3.3
```

```
set.seed(70)
sample <- stratified(glass1, group = "GlassType", size =0.5, bothSets = T)
test <- data.frame(sample[[1]])
train <- data.frame(sample[[2]])
```

Q7 Normalizing the new cases. Doing it the same way as previous. Binding the new cases to the original non-normalized data frame and then norrmalizing each and every record again. The stratified sample of train and test data are kept as it is and will be used only in future questions.

```
u1 <- c(1.51621,12.53,3.48,1.39,73.39,0.60,8.55,0.00,0.05)
u2 <- c(1.5098,12.77,1.85,1.81,72.69,0.59,10.01,0.00,0.01)
u <- data.frame(t(data.frame(u1,u2)))
colnames(u) <- c("RI","Na","Mg","Al","Si","K","Ca","Ba","Fe")

glass_new <- data.frame(rbind(glass[,2:10],u))

glass_new[,1] <- (glass_new[,1]-min(glass_new[,1]))/(max(glass_new[,1])-min(glass_new[,1]))
glass_new[,2] <- (glass_new[,2]-min(glass_new[,2]))/(max(glass_new[,2])-min(glass_new[,2]))
```

```
glass_new[3:9] <- lapply(glass_new[3:9],zStandardization)
```

Preparing unknown data and seperating the training data from test. Previously it was binded for normalization.

```
glass_testdata <- glass_new[(nrow(glass_new)-1):nrow(glass_new),]
unknown1 <- as.numeric(glass_new[(nrow(glass_new)-1),])
unknown2 <- as.numeric(glass_new[nrow(glass_new),])
glass_traindata <- glass_new[1:(nrow(glass_new)-2),]
glass_traindata <- cbind.data.frame(glass_traindata,glass[,11])
colnames(glass_traindata)[10] <- "GlassType"
```

Implementing KNN Algorithm: Function to calculate distance between two points.

```
dist <- function(p,q) {
  d <- 0
  for( i in 1:length(p)) {
    d <- d + (p[i]-q[i])^2
  }
  dist <- sqrt(d)
}
```

Neighbours function:

```
neighbors <- function(train_data,s) {

  m <- nrow(train_data)
  ds <- numeric(m)
```

```
  q <- as.numeric(s[c(1,2,3,4,5,6,7,8,9)])

  for( i in 1:m) {
    p <- train_data[i,c(1,2,3,4,5,6,7,8,9)]

    ds[i] <- dist(p,q)
  }

  neighbors <- ds
    }
```

Function to figure out k closest neighbours

```
k.closest <- function(neighbors,k) {
  ordered.neighbors <- order(neighbors)
  k.closest <- ordered.neighbors[1:k]
}
```

Mode function:

```
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x,ux)))]
}
```

Combining all functions into one i.e you only need to call this function and pass test data.

```
knn1 <- function(train_data,s,k) {
  nb <- neighbors(train_data,s)
  f <- k.closest(nb,k)
  knn1 <- Mode(train_data$GlassType[f])

}
```

Prediction for case 1:

```
nn1 <- knn1(glass_traindata,unknown1,10)
nn1
```

```
## [1] 1
```

Prediction for case 2:

```
nn2 <- knn1(glass_traindata,unknown2,10)
nn2
```

```
## [1] 2
```

Q8 Applying knn from class with k=14

```
library(class)
```

```
## Warning: package 'class' was built under R version 3.3.3
```

```
##
## Attaching package: 'class'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     knn1
```

```
test_prediction <- knn(train = glass_traindata[,1:9], test = glass_testdata, cl = glass_traindata[,10],
```

```
test_prediction
```

```
## [1] 1 6
## Levels: 1 2 3 5 6 7
```

Q9

Applying knn from class against stratified test data with k =14.

```
test_prediction_Q9 <- knn(train = train[,1:9], test = test[,1:9], cl = train[,10], k=14)
```

```
library(gmodels)
```

```
## Warning: package 'gmodels' was built under R version 3.3.3
```

```
CrossTable(x = test[,10], y = test_prediction_Q9, chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  105
##
##
##              | test_prediction_Q9
##   test[, 10] |         1 |         2 |         3 |         5 |         7 | Row Total |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            1 |        29 |         5 |         1 |         0 |         0 |        35 |
##              |     9.823 |     4.360 |     0.167 |     1.000 |     4.667 |           |
##              |     0.829 |     0.143 |     0.029 |     0.000 |     0.000 |     0.333 |
##              |     0.592 |     0.135 |     0.500 |     0.000 |     0.000 |           |
##              |     0.276 |     0.048 |     0.010 |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            2 |        11 |        24 |         1 |         1 |         1 |        38 |
##              |     2.557 |     8.406 |     0.105 |     0.007 |     3.264 |           |
##              |     0.289 |     0.632 |     0.026 |     0.026 |     0.026 |     0.362 |
##              |     0.224 |     0.649 |     0.500 |     0.333 |     0.071 |           |
##              |     0.105 |     0.229 |     0.010 |     0.010 |     0.010 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            3 |         6 |         2 |         0 |         0 |         0 |         8 |
##              |     1.376 |     0.238 |     0.152 |     0.229 |     1.067 |           |
##              |     0.750 |     0.250 |     0.000 |     0.000 |     0.000 |     0.076 |
##              |     0.122 |     0.054 |     0.000 |     0.000 |     0.000 |           |
##              |     0.057 |     0.019 |     0.000 |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            5 |         0 |         3 |         0 |         2 |         1 |         6 |
##              |     2.800 |     0.371 |     0.114 |    19.505 |     0.050 |           |
```

```
##              |     0.000 |     0.500 |     0.000 |     0.333 |     0.167 |     0.057 |
##              |     0.000 |     0.081 |     0.000 |     0.667 |     0.071 |           |
##              |     0.000 |     0.029 |     0.000 |     0.019 |     0.010 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            6 |         2 |         1 |         0 |         0 |         1 |         4 |
##              |     0.010 |     0.119 |     0.076 |     0.114 |     0.408 |           |
##              |     0.500 |     0.250 |     0.000 |     0.000 |     0.250 |     0.038 |
##              |     0.041 |     0.027 |     0.000 |     0.000 |     0.071 |           |
##              |     0.019 |     0.010 |     0.000 |     0.000 |     0.010 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            7 |         1 |         2 |         0 |         0 |        11 |        14 |
##              |     4.686 |     1.744 |     0.267 |     0.400 |    44.688 |           |
##              |     0.071 |     0.143 |     0.000 |     0.000 |     0.786 |     0.133 |
##              |     0.020 |     0.054 |     0.000 |     0.000 |     0.786 |           |
##              |     0.010 |     0.019 |     0.000 |     0.000 |     0.105 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
## Column Total |        49 |        37 |         2 |         3 |        14 |       105 |
##              |     0.467 |     0.352 |     0.019 |     0.029 |     0.133 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##
##
```

Accuracy = (66/105) *100 = 62.86%

Q10

Calculating accuracy for k=5:14. looping knn function for each row of test data.

```
nn <- vector()
k <- c(5,6,7,8,9,10,11,12,13,14)
accuracy <- numeric(length(k))
inc <- 1
for(j in 5:14) {
for ( i in 1:nrow(test)) {
  nn[i] <- knn1(train,test[i,1:9],j)
}
a <- table(nn,test[,10])
accuracy[inc] <- (sum(diag(a))/sum(a)) *100
inc <- inc + 1
}
```

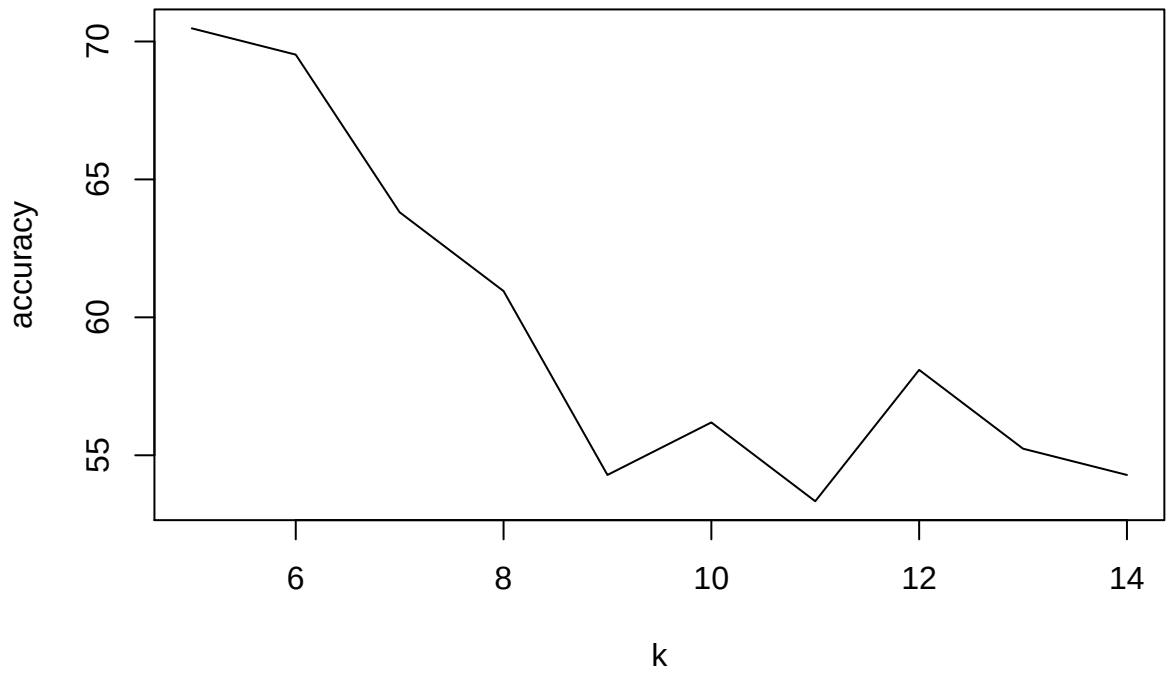
```
which.max(accuracy)
```

```
## [1] 1
```

Accuracy is maximum for k = 5. '1' output refers to first value of accuracy which is for k = 5. Optimal k is 5.

Plotting k vs accuracy

```
plot(k,accuracy,type='l')
```



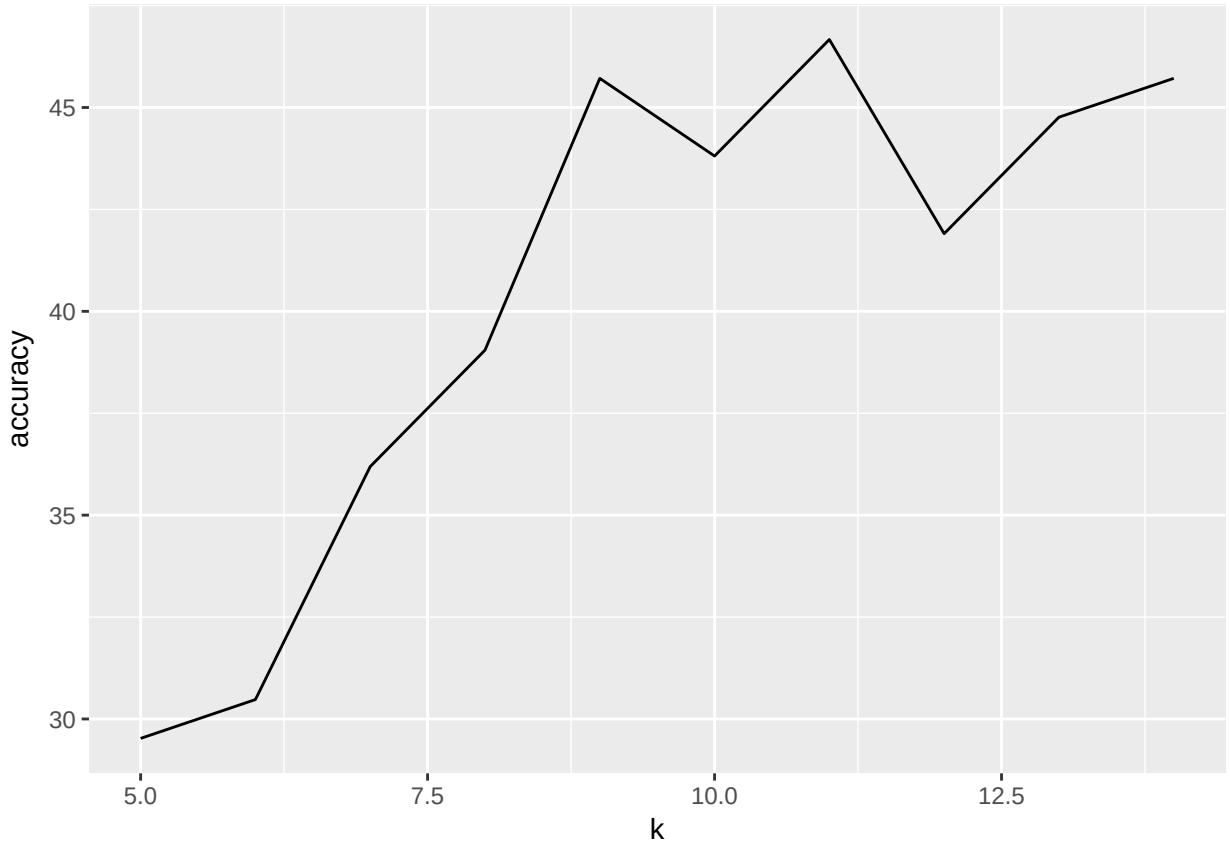
Q11

Plotting k vs incorrect classifications

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.3.3
```

```
df<- data.frame(k,(100-accuracy))
colnames(df)[2] <- 'accuracy'
ggplot(df,aes(x=k,y=accuracy)) + geom_line()
```



Q12 Packge of my choice = class k of my choice = 13

```
library(class)
test_prediction_Q12 <- knn(train = train[,1:9], test = test[,1:9], cl = train[,10], k=13)
CrossTable(x = test[,10], y = test_prediction_Q12, chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  105
##
##
##              | test_prediction_Q12
##   test[, 10] |         1 |         2 |         3 |         5 |         7 | Row Total |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            1 |        30 |         5 |         0 |         0 |         0 |        35 |
##              |    10.667 |     4.360 |     0.333 |     1.667 |     4.000 |           |
##              |     0.857 |     0.143 |     0.000 |     0.000 |     0.000 |     0.333 |
```

```
##              |     0.600 |     0.135 |     0.000 |     0.000 |     0.000 |           |
##              |     0.286 |     0.048 |     0.000 |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            2 |        12 |        24 |         0 |         2 |         0 |        38 |
##              |     2.053 |     8.406 |     0.362 |     0.020 |     4.343 |           |
##              |     0.316 |     0.632 |     0.000 |     0.053 |     0.000 |     0.362 |
##              |     0.240 |     0.649 |     0.000 |     0.400 |     0.000 |           |
##              |     0.114 |     0.229 |     0.000 |     0.019 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            3 |         6 |         1 |         1 |         0 |         0 |         8 |
##              |     1.260 |     1.174 |    11.201 |     0.381 |     0.914 |           |
##              |     0.750 |     0.125 |     0.125 |     0.000 |     0.000 |     0.076 |
##              |     0.120 |     0.027 |     1.000 |     0.000 |     0.000 |           |
##              |     0.057 |     0.010 |     0.010 |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            5 |         0 |         3 |         0 |         2 |         1 |         6 |
##              |     2.857 |     0.371 |     0.057 |    10.286 |     0.144 |           |
##              |     0.000 |     0.500 |     0.000 |     0.333 |     0.167 |     0.057 |
##              |     0.000 |     0.081 |     0.000 |     0.400 |     0.083 |           |
##              |     0.000 |     0.029 |     0.000 |     0.019 |     0.010 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            6 |         1 |         2 |         0 |         1 |         0 |         4 |
##              |     0.430 |     0.247 |     0.038 |     3.440 |     0.457 |           |
##              |     0.250 |     0.500 |     0.000 |     0.250 |     0.000 |     0.038 |
##              |     0.020 |     0.054 |     0.000 |     0.200 |     0.000 |           |
##              |     0.010 |     0.019 |     0.000 |     0.010 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            7 |         1 |         2 |         0 |         0 |        11 |        14 |
##              |     4.817 |     1.744 |     0.133 |     0.667 |    55.225 |           |
##              |     0.071 |     0.143 |     0.000 |     0.000 |     0.786 |     0.133 |
##              |     0.020 |     0.054 |     0.000 |     0.000 |     0.917 |           |
##              |     0.010 |     0.019 |     0.000 |     0.000 |     0.105 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
## Column Total |        50 |        37 |         1 |         5 |        12 |       105 |
##              |     0.476 |     0.352 |     0.010 |     0.048 |     0.114 |           |
## -------------|-----------|-----------|-----------|-----------|-----------|-----------|
##
##
```

Accuracy = (68/105) *100 = 64.76%

Q13

For each new case, the algorithm needs to calculate distance from all points in the training data set for all features. So the run-time complexity should be O(wnm).

The algorithm takes more time as w,m and n increase. The algorithm would get slower as m and n increase as the computation increases.