HW1Q2.1A.R

2020-08-26

```
#start with a clean environment
rm(list=ls())

#Homework Assignment - 1 By Haritha Pulletikurti

##############################################################################
###########
# QUestion 2.2.
#
#Question 1.    Using the support vector machine function ksvm contained in t
he R package#
#kernlab,find a good classifier for this data. Show the equation of your clas
sifier,  #
#and how well it classifies the data points in the full data set.
#
#(Don't worry about test/validation data yet; we'll cover that topic soon.)
#
##############################################################################
###########





#loading libaries
library(kernlab)
library(kknn)
library(e1071)
library(data.table)
#set the working directory
setwd("C:\\R Projects")

#make sure to reproduce the test results
set.seed(50)

#import the  credit card dataset with headers
credit_card_dataset <- read.delim("credit_card_data-headers.txt")


#Take a sample set of the first and last few rows of credit_card_dataset
head(credit_card_dataset)
```

```
##   A1    A2    A3   A8 A9 A10 A11 A12 A14 A15 R1
## 1  1 30.83 0.000 1.25  1   0   1   1 202   0  1
## 2  0 58.67 4.460 3.04  1   0   6   1  43 560  1
## 3  0 24.50 0.500 1.50  1   1   0   1 280 824  1
## 4  1 27.83 1.540 3.75  1   0   5   0 100   3  1
## 5  1 20.17 5.625 1.71  1   1   0   1 120   0  1
## 6  1 32.08 4.000 2.50  1   1   0   0 360   0  1
```

```
tail(credit_card_dataset)
```

```
##     A1    A2     A3   A8 A9 A10 A11 A12 A14 A15 R1
## 649  1 40.58  3.290 3.50  0   1   0   0 400   0  0
## 650  1 21.08 10.085 1.25  0   1   0   1 260   0  0
## 651  0 22.67  0.750 2.00  0   0   2   0 200 394  0
## 652  0 25.25 13.500 2.00  0   0   1   0 200   1  0
## 653  1 17.92  0.205 0.04  0   1   0   1 280 750  0
## 654  1 35.00  3.375 8.29  0   1   0   0   0   0  0
```

```
# Find the Optimum C Value and generate the optimum classifier for the the da
ta.

# Ranges of C
C_Ranges = c(0.1,1,10,100,1000,10000)
AccuracyResults_Based_On_Full_C_Range=list()
SVMmodel=list()
svmmodelprediction=list()
# Get the Model Predictions for all C's in the list ans store them for analys
is
for(i in 1:length(C_Ranges))
{
  # call ksvm.  Vanilladot is a simple linear kernel.
  SVMmodel[[i]] <- ksvm(as.matrix(credit_card_dataset[,1:10]),as.factor(credi
t_card_dataset[,11]),type="C-svc",kernel="vanilladot",C=C_Ranges[[i]],scaled=
TRUE)
  # see what the model predicts
  svmmodelprediction[[i]] <- predict(SVMmodel[[i]],credit_card_dataset[,1:10]
)

  # see what percentage of the model’s predictions match the actual classific
ation
  AccuracyResults_Based_On_Full_C_Range[[i]]= sum(svmmodelprediction[[i]] ==
credit_card_dataset[,11]) / nrow(credit_card_dataset)*100
}
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

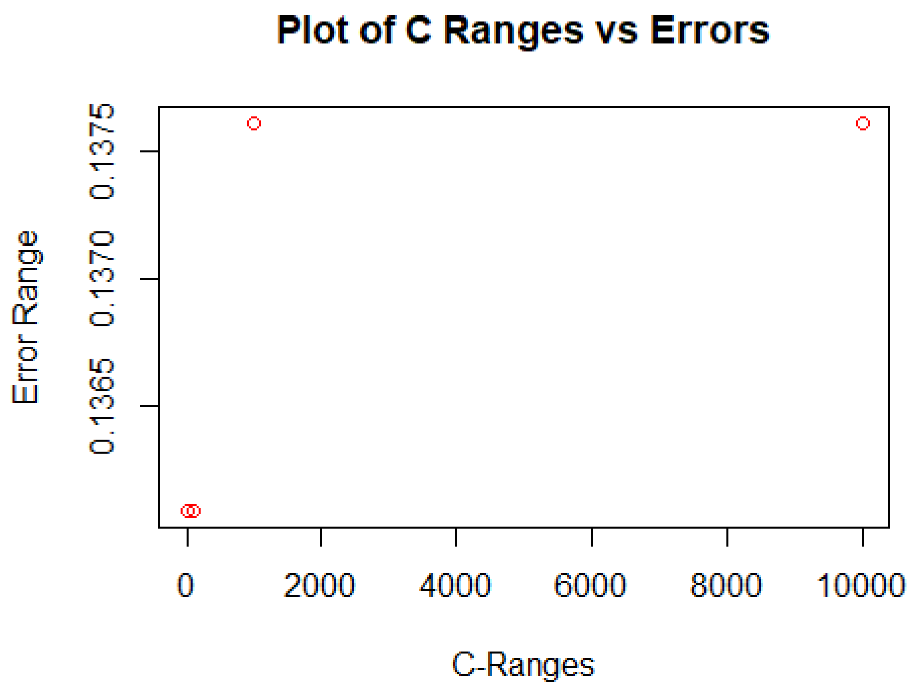
```
#Order of Accuracy from least to highest
Max_AccuracyIndices = order(unlist(AccuracyResults_Based_On_Full_C_Range))


TopFiveCRanges=c(C_Ranges[Max_AccuracyIndices[length(Max_AccuracyIndices)]],C
_Ranges[Max_AccuracyIndices[length(Max_AccuracyIndices)-1]],
                 C_Ranges[Max_AccuracyIndices[length(Max_AccuracyIndices)-2]]
,C_Ranges[Max_AccuracyIndices[length(Max_AccuracyIndices)-3]],
                 C_Ranges[Max_AccuracyIndices[length(Max_AccuracyIndices)-4]]
)
#Top Five C Ranges based on highest Accuracy Values
TopFiveCRanges

## [1] 1e+02 1e+01 1e+00 1e-01 1e+04

#Analysis of Top Five C Ranges
CPlotValues = c()
nSVPlotValues = c()
errorPlotValues = c()
accuracyplotvalues = c()
j=1
for(i in length(Max_AccuracyIndices):1)
{

  CPlotValues[j]=C_Ranges[Max_AccuracyIndices[i]]
  CPlotValues[j]
  SVMmodel[Max_AccuracyIndices[i]]
  svmmodelprediction[Max_AccuracyIndices[i]]
  accuracyplotvalues[j]= AccuracyResults_Based_On_Full_C_Range[Max_AccuracyIn
dices[i]]
  accuracyplotvalues[j]
  nSVPlotValues[j]=SVMmodel[Max_AccuracyIndices[i]][[1]]@nSV
  nSVPlotValues[j]
  errorPlotValues[j]=SVMmodel[Max_AccuracyIndices[i]][[1]]@error
  errorPlotValues[j]
  j=j+1
}
#Analysis and determining the Best C Values:
# C Ranges : Best to Least based on Accuracy of Prediction
CRange = sprintf(CPlotValues,fmt='%#.2f')
CRange

## [1] "100.00"   "10.00"    "1.00"     "0.10"     "10000.00" "1000.00"

#KSVM Model Accuracy Prediction Values from Best to Least
unlist(accuracyplotvalues)

## [1] 86.39144 86.39144 86.39144 86.39144 86.23853 86.23853

#KSVM Model Error Prediction Values from Best to Least
errorPlotValues
```

```
## [1] 0.1360856 0.1360856 0.1360856 0.1360856 0.1376147 0.1376147
```

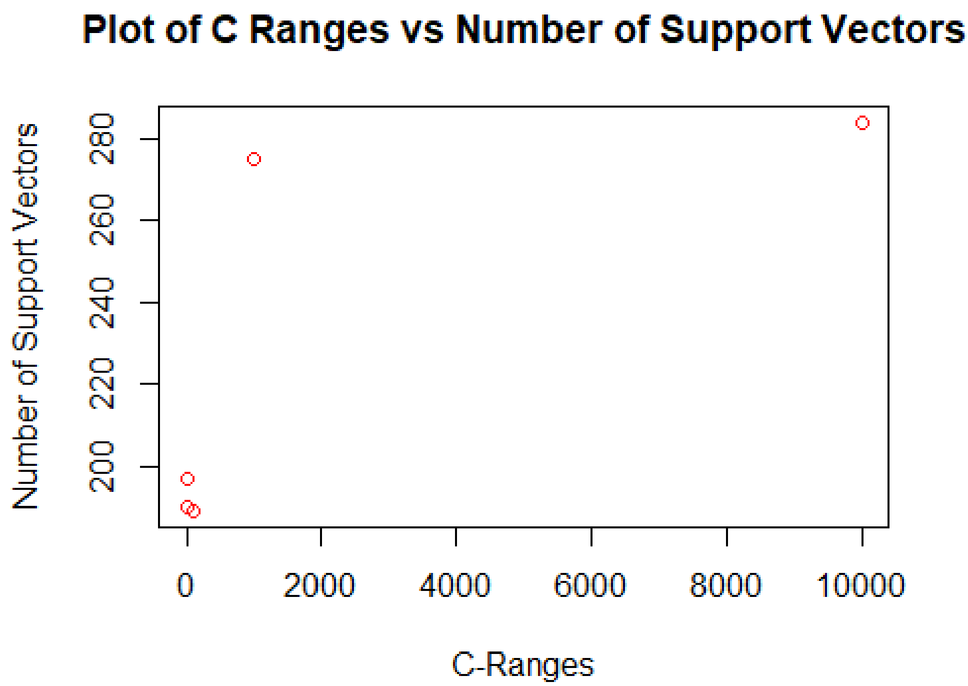
```r
#KSVM Model "Number of Support Vectors: Prediction Values from Best to Least
nSVPlotValues
```

```
## [1] 189 190 190 197 284 275
```

```r
plot(CRange, errorPlotValues, xlab="C-Ranges", ylab="Error Range" , main="Plo
t of C Ranges vs Errors" , col="red" , varwidth=T,horizontal=T)
```




```r
plot(CRange, nSVPlotValues, xlab="C-Ranges", ylab="Number of Support Vectors"
, main="Plot of C Ranges vs Number of Support Vectors" , col="red" , varwidth
=T,horizontal=T)
```




```
# Based on the above analysis,
# Top C Values based on Accuracy of Prodiction are : C=100,  C=10 and C=1 whe
re accuracies are all 80.39144%
# Top Error Allowed of the Top C Values are : for all C= 100, 10, 1 is 0.1360
856
# Top Number of Support Vectors(NSV) : C= 100 and NSV = 189 , C=10 and NSV =
190 , C=1 and NSV = 190.
# For More larger C Values like C=1000 and 10000 the NSV > 275 implying the m
argin is too large. So not optimal
# For C values 0.1 and the margin is very small. The new data points can be m
isclassified to a great extent.
# Hence the optimum value of C=100 and the second best choice would be C=10 b
ased on the above Research.

BestSVMmodel <- ksvm(as.matrix(credit_card_dataset[,1:10]),as.factor(credit_c
ard_dataset[,11]),type="C-svc",kernel="vanilladot",C=100,scaled=TRUE)

##  Setting default kernel parameters

BestSVMmodel

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
## Linear (vanilla) kernel function.
```

```
##
## Number of Support Vectors : 189
##
## Objective Function Value : -17887.92
## Training error : 0.136086
```

```
# calculate the cofficients a1…am of the  model predicted support vectors
a1ToamCoeff<- colSums(BestSVMmodel@xmatrix[[1]] * BestSVMmodel@coef[[1]])
a1ToamCoeff
```

```
##            A1            A2            A3            A8            A9
## -0.0010065348 -0.0011729048 -0.0016261967  0.0030064203  1.0049405641
##           A10           A11           A12           A14           A15
## -0.0028259432  0.0002600295 -0.0005349551 -0.0012283758  0.1063633995
```

```
# calculate a0
a0Intercept <- BestSVMmodel@b
a0Intercept
```

```
## [1] -0.08158492
```

```
Bestsvmmodelprediction <- predict(BestSVMmodel,credit_card_dataset[,1:10])
Bestsvmmodelprediction
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1
## [260] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0
## [297] 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
1 1 1
## [519] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1
## [556] 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0
## [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
```

```
# see what percentage of the model's predictions match the actual classificat
ion
AccuracyResults= sum(Bestsvmmodelprediction == credit_card_dataset[,11]) / nr
ow(credit_card_dataset)*100

AccuracyResults
```

```
## [1] 86.39144
```

```
#############################################################################
#############################
#Question 2.    You are welcome, but not required, to try other (nonlinear) k
ernels as well; we're not covering them #
#in this course, but they can sometimes be useful and might provide better pr
edictions than vanilladot.###
#############################################################################
##############################




PolyDotKernelSVMmodel <- ksvm(as.matrix(credit_card_dataset[,1:10]),as.factor
(credit_card_dataset[,11]),type="C-svc",kernel="polydot",C=100,scaled=TRUE)
```

```
##  Setting default kernel parameters
```

```
PolyDotKernelSVMmodel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
## Polynomial kernel function.
##  Hyperparameters : degree =  1  scale =  1  offset =  1
##
## Number of Support Vectors : 190
##
## Objective Function Value : -17887.98
## Training error : 0.136086
```

```
# calculate the cofficients a1…am of the  model predicted support vectors
a1ToamCoeffofPolydotKernel<- colSums(PolyDotKernelSVMmodel@xmatrix[[1]] * PolyDotKernelSVMmodel@coef[[1]])
a1ToamCoeffofPolydotKernel

##            A1            A2            A3            A8            A9
## -0.0010929705 -0.0012425741 -0.0015628157  0.0027739329  1.0051781402
##           A10           A11           A12           A14           A15
## -0.0026901076 -0.0001935512 -0.0005270357 -0.0014583698  0.1063997443

# calculate a0
a0InterceptofPolydotKernel <- PolyDotKernelSVMmodel@b
a0InterceptofPolydotKernel

## [1] -0.08157716

PolyDotKernelsvmmodelprediction <- predict(PolyDotKernelSVMmodel,credit_card_dataset[,1:10])
PolyDotKernelsvmmodelprediction

##   [1] 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1
## [260] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0
## [297] 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [519] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1
## [556] 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
```

```
0 0 0
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 
0 0 0
## [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
```

```
# see what percentage of the model's predictions match the actual classificat
ion
AccuracyResultsforPolyDotKernel= sum(PolyDotKernelsvmmodelprediction == credi
t_card_dataset[,11]) / nrow(credit_card_dataset)*100

AccuracyResultsforPolyDotKernel
```

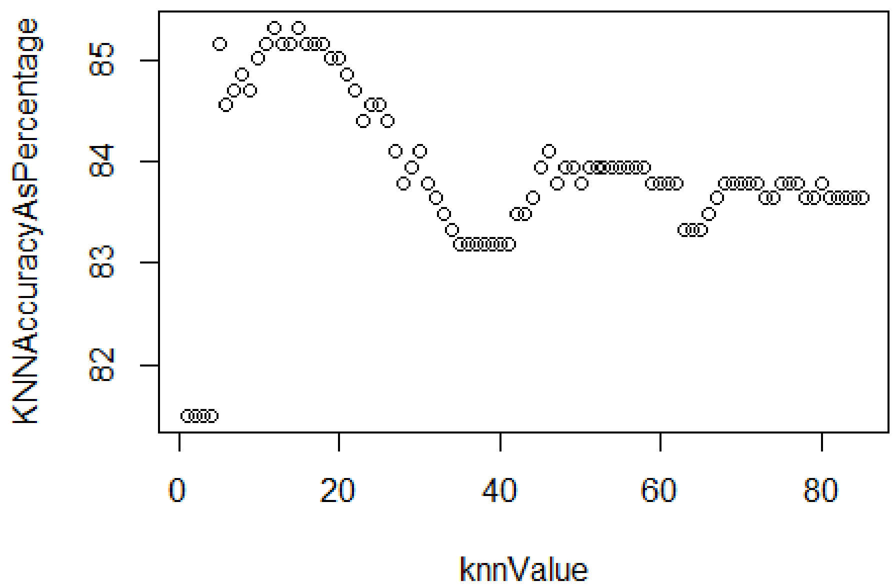
```
## [1] 86.39144
```

```
#Inference: The Polydot Kernel gives 86.39144 % of accuracy rate with 0.13608
6 Training error, Number of Support Vectors = 190 for C=100.




##############################################################################
##############################
#################Question : 2.2.3 ############################################
##############################
#Question 3.    Using the k-nearest-neighbors classification function kknn co
ntained in the R kknn package,##
#suggest a good value of k, and show how well it classifies that data points 
in the full data set.#########
#Don't forget to scale the data (scale=TRUE in kknn).########################
#############################
##############################################################################
#############################

#Function(Knearest) takes a value for k and predicts which class the new data 
point belongs to.

classify_Based_on_kknn = function(Knearest){
  #Create n zero predictions
  predictions<- rep(0,(nrow(credit_card_dataset)))

  for (i in 1:nrow(credit_card_dataset)){
    #Run the kknn function and ensure it doesn't use i itself
    KNNModel=kknn(R1~A1+A2+A3+A8+A9+A10+A11+A12+A14+A15,credit_card_dataset[-
i,],
                  credit_card_dataset[i,],k=Knearest, distance = 2,kernel = "
optimal",scale = TRUE)
    predictions[i] <- as.integer(fitted(KNNModel)+0.5)
  }
```

```
  #Check the accuracy of the prediction.
  account = sum(predictions == credit_card_dataset[,11]) / nrow(credit_card_d
ataset)
  return(account)
}

# Create 85 zeroed vector for accuracy test.
RunTestVectors <- rep(0,85)
#Run the K value (Knearest_Value) from 1 to 85 .
for (Knearest_Value in 1:85){
  RunTestVectors[Knearest_Value] = classify_Based_on_kknn(Knearest_Value)
}

#see accuracy as percentage
KNNAccuracyAsPercentage <- as.matrix(RunTestVectors * 100)

#print out knn values and percentage of accuracy
KNNAccuracyAsPercentage

##           [,1]
##  [1,] 81.49847
##  [2,] 81.49847
##  [3,] 81.49847
##  [4,] 81.49847
##  [5,] 85.16820
##  [6,] 84.55657
##  [7,] 84.70948
##  [8,] 84.86239
##  [9,] 84.70948
## [10,] 85.01529
## [11,] 85.16820
## [12,] 85.32110
## [13,] 85.16820
## [14,] 85.16820
## [15,] 85.32110
## [16,] 85.16820
## [17,] 85.16820
## [18,] 85.16820
## [19,] 85.01529
## [20,] 85.01529
## [21,] 84.86239
## [22,] 84.70948
## [23,] 84.40367
## [24,] 84.55657
## [25,] 84.55657
## [26,] 84.40367
## [27,] 84.09786
## [28,] 83.79205
## [29,] 83.94495
## [30,] 84.09786
```

```
## [31,] 83.79205
## [32,] 83.63914
## [33,] 83.48624
## [34,] 83.33333
## [35,] 83.18043
## [36,] 83.18043
## [37,] 83.18043
## [38,] 83.18043
## [39,] 83.18043
## [40,] 83.18043
## [41,] 83.18043
## [42,] 83.48624
## [43,] 83.48624
## [44,] 83.63914
## [45,] 83.94495
## [46,] 84.09786
## [47,] 83.79205
## [48,] 83.94495
## [49,] 83.94495
## [50,] 83.79205
## [51,] 83.94495
## [52,] 83.94495
## [53,] 83.94495
## [54,] 83.94495
## [55,] 83.94495
## [56,] 83.94495
## [57,] 83.94495
## [58,] 83.94495
## [59,] 83.79205
## [60,] 83.79205
## [61,] 83.79205
## [62,] 83.79205
## [63,] 83.33333
## [64,] 83.33333
## [65,] 83.33333
## [66,] 83.48624
## [67,] 83.63914
## [68,] 83.79205
## [69,] 83.79205
## [70,] 83.79205
## [71,] 83.79205
## [72,] 83.79205
## [73,] 83.63914
## [74,] 83.63914
## [75,] 83.79205
## [76,] 83.79205
## [77,] 83.79205
## [78,] 83.63914
## [79,] 83.63914
## [80,] 83.79205
```

```
## [81,] 83.63914
## [82,] 83.63914
## [83,] 83.63914
## [84,] 83.63914
## [85,] 83.63914

knnValue <- c(1:85)

#Plot the  KKN accuracies as Percentage per Kth Nearest Neighbour value
plot(knnValue,KNNAccuracyAsPercentage)
```




```
#Maximum Percentage
max(KNNAccuracyAsPercentage)

## [1] 85.3211

#Inference

#The  highest accuracy  of 85.321110 is at k = 15. So, the classifier is opti
mal at k = 15.
#The  highest accuracy  of 85.321110 is at k = 12. So, the classifier is also
optimal at k = 12.
# The accuracy value seems to decrease as K value increases more than 15.
# Hence the optimal value of k = 12 for this data set.
```