Homework Week 1-Neil Lagundino

Question 2.1 Describe a situation or problem from your job, everyday life, current events, etc., for which a classification model would be appropriate. List some (up to 5) predictors that you might use.

Answer 2.1: Classification model can be applied in our daily lives and major life events.Getting approved for a mortgage is a classic example that I am going through right now. The probability of getting approved lies on many different predictors such as: 1) income, 2) credit score 3) down payment 4) Gross Debt Service (GDS) Ratio and 5) Total Debt Service (TDS) Ratio. The lender uses classification model to determine if a borrower like me gets approved or not.

Question 2.2.1 1. Using the support vector machine function ksvm contained in the R package kernlab, find a good classifier for this data. Show the equation of your classifier, 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.)

Answer 2.2.1

```r
#Clear all objects from the current workspace
rm(list = ls())

#import the package kernlab
library(kernlab)

#Import Dataset
filename= "~/Desktop/MicroMaster GTX/week_1_data-summer (1)/data 2.2/credit_card_data.txt"
credit_card_data <- read.table(filename,stringsAsFactors = FALSE, header=FALSE)

#Execute head and tail function to ensure data is read accurately
head(credit_card_data)
```

```
##   V1    V2    V3   V4 V5 V6 V7 V8  V9 V10 V11
## 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
```

```r
tail(credit_card_data)
```

```
##     V1    V2     V3   V4 V5 V6 V7 V8  V9 V10 V11
## 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
```

```r
# Set random number generator seed so that results are reproducible
set.seed(1)

#Using scaled=True
```

```r
#Call ksvm. Vanilladot is a simple linear kernel. Also use C-clasification method
model1<- ksvm(as.matrix(credit_card_data[,1:10]),as.factor(credit_card_data[,11]),
              type ="C-svc",       #c
              kernel="vanilladot", #simple linear kernel
              C=100,scaled=TRUE)   #scaled =True
```

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

```r
#display model1
model1
```

```
## 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
```

```r
#calculate coefficients
a <- colSums(model1@xmatrix[[1]] * model1@coef[[1]] )

#display a
a
```

```
##            V1            V2            V3            V4            V5
## -0.0010065348 -0.0011729048 -0.0016261967  0.0030064203  1.0049405641
##            V6            V7            V8            V9           V10
## -0.0028259432  0.0002600295 -0.0005349551 -0.0012283758  0.1063633995
```

```r
#calculate a0
a0 <- -model1@b

#display a0
a0
```

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

Therefore, classifier equation is: -0.0010065348v1 - 0.0011729048v2 - 0.0016261967v3 + 0.0030064203v4 + 1.0049405641v5 - 0.0028259432v6 + 0.0002600295v7 - 0.0005349551v8 - 0.0012283758v9 + 0.1063633995v10 + 0.08158492v0 = 0

```r
#model prediction
pred <- predict(model1,credit_card_data[,1:10])

#display prediction
pred
```

```
##   [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
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 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 1 1 1 1 1 1 1 1
## [106] 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
## [141] 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 1 1 1 1 1 1 1 1 1
## [176] 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
## [211] 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 1 1 1 1
## [246] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
## [281] 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0
## [316] 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 0 0 0 0 0 0 0 0
## [351] 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
## [386] 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
## [421] 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
## [456] 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 1 1 1 1 1 1 1 1
## [491] 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
## [526] 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 1 1 1 1 1
## [561] 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 0 0 0
## [596] 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 0
## [631] 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
```

```r
# see what fraction of the model's predictions match the actual classification
sum(pred == credit_card_data[,11]) / nrow(credit_card_data)
```

```
## [1] 0.8639144
```

```r
#model's aacuracy is 86.39%
```

Question 2.2.2 You are welcome, but not required, to try other (nonlinear) kernels as well; we're not covering them in this course, but they can sometimes be useful and might provide better predictions than vanilladot.

Answer 2.2.2

```r
#Using scaled=True
#Call ksvm. rbfdot is a nonlinear kernel. Also use C-clasification method

model1<- ksvm(as.matrix(credit_card_data[,1:10]),as.factor(credit_card_data[,11]),
              type ="C-svc",    #c classification
              kernel="rbfdot",   #nonlinear model
              C=100,scaled=TRUE) #scaled=True

#display model1
model1
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0890718384846732
##
## Number of Support Vectors : 244
```

```
##
## Objective Function Value : -9313.911
## Training error : 0.04893
```

```r
#calculate coefficients
a <- colSums(model1@xmatrix[[1]] * model1@coef[[1]] )

#display a
a
```

```
##         V1         V2         V3         V4         V5         V6         V7
## -19.30440 -38.81550  -8.82235  56.79429  49.16483 -22.83664  11.46010
##         V8         V9        V10
## -23.06623 -59.18260  49.67840
```

```r
#calculate a0
a0 <- -model1@b

#display a0
a0
```

```
## [1] 0.8469984
```

```r
#model prediction
pred <- predict(model1,credit_card_data[,1:10])

#display prediction
pred
```

```
##   [1] 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0
## [106] 1 0 0 0 1 0 0 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
## [141] 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 1 1 1 1 1 1 1 1 1
## [176] 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
## [211] 1 1 1 1 1 1 1 1 0 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 1
## [246] 0 0 0 0 0 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
## [281] 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 1 0 0 1 0 1 0 0 0 0
## [316] 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
## [351] 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
## [386] 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
## [421] 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
## [456] 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 1 1 1 1 1 1 1 1
## [491] 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
## [526] 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
## [561] 1 1 1 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0
## [596] 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 0
## [631] 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
```

```r
# see what fraction of the model's predictions match the actual classification
sum(pred == credit_card_data[,11]) / nrow(credit_card_data)
```

```
## [1] 0.9510703
```

```r
#model's aacuracy is 95.11%
```

Conclusion: By applying the Gaussian RBF kernel, the accuracy rate is much higher than the vanilla dot.Thus, a better prediction model

Question 2.2.3 Using the k-nearest-neighbors classification function kknn contained 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)

Answer 2.2.3

```r
#Clear all objects from the current workspace
rm(list = ls())

#import the package kknn
library("kknn")

#Import Dataset
filename= "~/Desktop/MicroMaster GTX/week_1_data-summer (1)/data 2.2/credit_card_data.txt"
credit_card_data <- read.table(filename,stringsAsFactors = FALSE, header=FALSE)

#Execute head and tail function to ensure data is read accurately
head(credit_card_data)
```

```
##   V1    V2    V3   V4 V5 V6 V7 V8  V9 V10 V11
## 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
```

```r
tail(credit_card_data)
```

```
##     V1    V2     V3   V4 V5 V6 V7 V8  V9 V10 V11
## 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
```

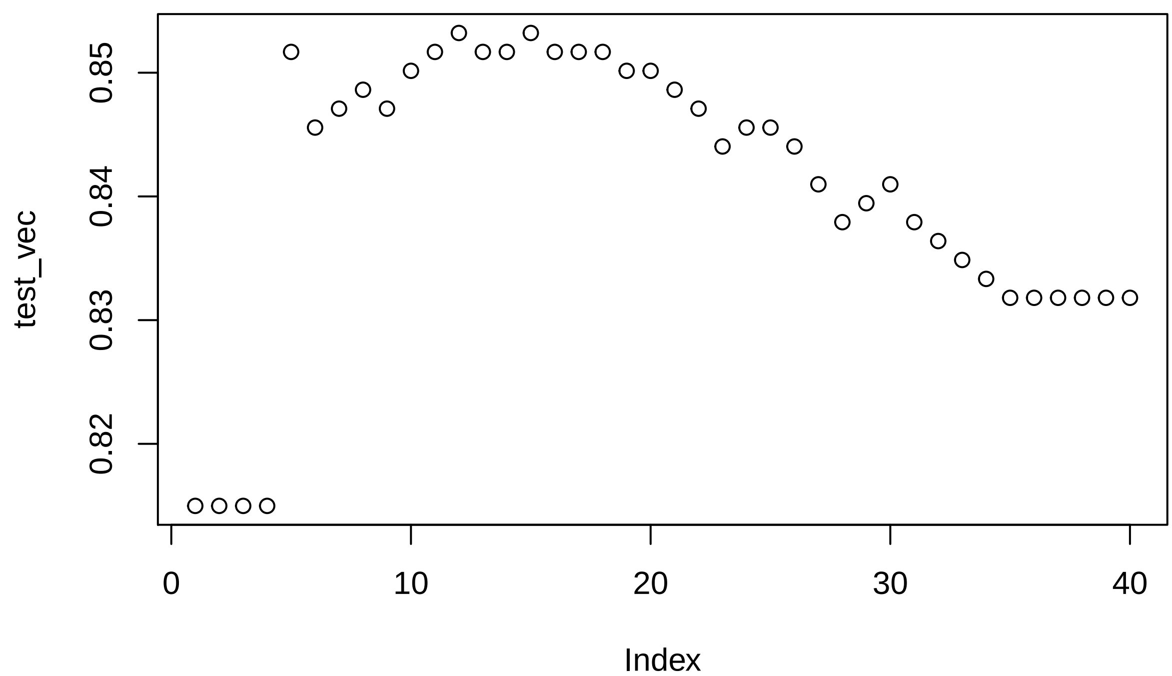
```r
# Set random number generator seed so that results are reproducible
set.seed(589)

#Create a function to calculate the accuracy of the model with k=Z
chk_accuracy = function(Z){
```

```r
  predicted <- rep(0,(nrow(credit_card_data)))
  for (i in 1:nrow(credit_card_data)){
  model1=kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10,credit_card_data[-i,],credit_card_data[i,],
              k=Z, scale = TRUE)
  predicted[i] <- as.integer(fitted(model1)+0.5)} # for rounding

#calculate correct predictions
  acc = sum(predicted == credit_card_data[,11]) / nrow(credit_card_data)
  return(acc)
}

test_vec <- rep(0,40) #accuracy test (knn values 1 to 40)
for (Z in 1:40){
   test_vec[Z] = chk_accuracy(Z)
}

knn_acc <- as.matrix(test_vec * 100) #accuracy in percentage
knn_acc
```

```
##           [,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
```

```r
knn_val <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,
             23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40)

plot(test_vec)#observe accuracies per knn value
```




```r
max(knn_acc)
```

```
## [1] 85.3211
```

Conclusion: The knn value that best classifies the data points is 12, having an accuracy rate of 85.3211%

Question 3.1 Using the same data set (credit_card_data.txt or credit_card_data-headers.txt) as in Question 2.2, use the ksvm or kknn function to find a good classifier: A) using cross-validation(do this for the k-nearest-neighbors model;SVM is optional);and B) splitting the data into training,validation,and test datasets(pick either KNN or SVM;the other is optional).

Question 3.1.A and B

```r
#Clear all objects from the current workspace
rm(list = ls())

#import the package and kknn
library("kknn")

#Import Dataset
filename= "~/Desktop/MicroMaster GTX/week_1_data-summer (1)/data 2.2/credit_card_data.txt"
credit_card_data <- read.table(filename,stringsAsFactors = FALSE, header=FALSE)

#Execute head and tail function to ensure data is read accurately
head(credit_card_data)
```

```
##   V1    V2    V3   V4 V5 V6 V7 V8  V9 V10 V11
## 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
```

```r
tail(credit_card_data)
```

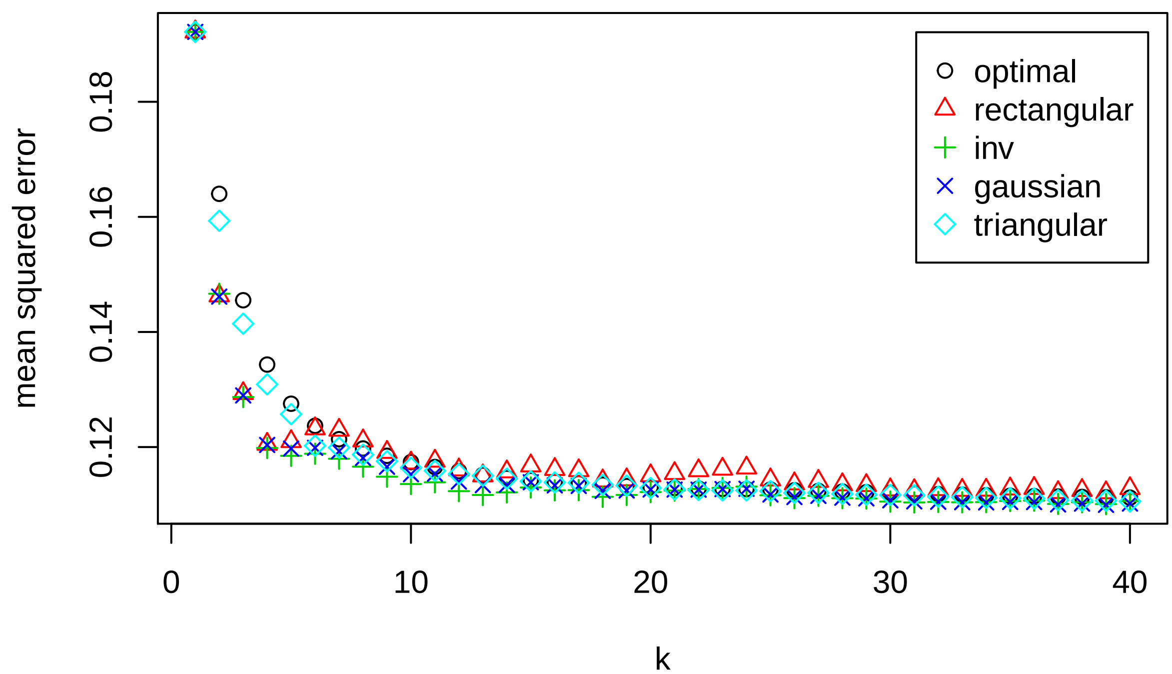
```
##     V1    V2     V3   V4 V5 V6 V7 V8  V9 V10 V11
## 649  1 40.58  3.290 3.50  0  1  0  0 400   0   0
## 650  1 21.08 10.085 1.25  0  1  0  1 260   0   0
## 651  0 22.67  0.750 2.00  0  0  2  0 200 394   0
## 652  0 25.25 13.500 2.00  0  0  1  0 200   1   0
## 653  1 17.92  0.205 0.04  0  1  0  1 280 750   0
## 654  1 35.00  3.375 8.29  0  1  0  0   0   0   0
```

```r
#Set random number generator seed so that results are reproducible
set.seed(10)

#Take 70% data for training and the rest for testing

#Number of rows in cc data
numrows = nrow(credit_card_data)

#Select 30% of indexes among 654 indexes
numsample = sample(1:numrows, size = round(numrows*.3), replace = FALSE)

#Training data selected by excluding the 30% sample
traindata = credit_card_data[-numsample,]
#Test data seleted by including the 70% sample
traintest = credit_card_data[numsample,]

#Training of kknn method via (train.kknn) cross validation, we want to find the optimal value of 'k'
crossval=train.kknn(V11~ .,data = traindata, kmax = 40,
          kernel = c("optimal","rectangular", "inv", "gaussian", "triangular"), scale = TRUE)
crossval
```

```
##
```

```
## Call:
## train.kknn(formula = V11 ~ ., data = traindata, kmax = 40, kernel = c("optimal",     "rectangular",
##
## Type of response variable: continuous
## minimal mean absolute error: 0.1921397
## Minimal mean squared error: 0.1099627
## Best kernel: gaussian
## Best k: 39
```

```
#Plot classification error rate for k values
plot(crossval)
```




```
#Test model on test data
pred<-predict(crossval, traintest)
pred_bin<-round(pred)
pred_acc<-table(pred_bin,traintest$V11)
pred_acc
```

```
##
## pred_bin  0  1
##        0 94 19
##        1  9 74
```

```r
#Calculate prediction accuracy
sum(pred_bin==traintest$V11)/length(traintest$V11)
```

```
## [1] 0.8571429
```

Conclusion: KNN(leave-one-out method) prediction acuracy is 85.71% Best K: 39

Overall conclusion: The best prediction accuracy for this data is SVM (nonlinear and simple linear) with 95.11% and 86.39% accuracy peercentage. KKNN and TRAIN.KKNN both have 85.32% and 85.71%