week_1_homework

Chen Yi-Ju

2020 5 19

Question2.1

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.

As a Taiwanese, politics are quite a big part of my life. A situation that is appropriate for using classification is that of the presidential elections. There are (ususally) only two candidates running for the two major parties. As for parameters: age,area of living, origin native place , education and income would be influential parameters.

Question 2.2

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.

First download the data and the neccessary packages:

```
setwd("D:/ernie/self-study/GTxMicroMasters/Introduction to Analytics Modeling/week 1")
credit <- read.table("credit_card_data-headers.txt" , header = T)
head(credit)
```

```
##   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
```

```
library(kernlab)
library(kknn)
library(ggplot2)
library(dplyr)
library(caTools)
```

This is the first model set, using “vanilladot”(linear) and a C-value of 100

```
set.seed(101)
model.1 <- ksvm(x = as.matrix(credit[,1:10]),
                y = as.factor(credit[,11]),
                type = "C-svc" ,
                scaled = TRUE ,
                kernel = "vanilladot" ,
                C = 100)
```

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

```
model.1
```

```
## 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
```

```
a <- colSums(model.1@xmatrix[[1]]*model.1@coef[[1]])
a0 <- model.1@b
pred1 <- predict(model.1,credit[,1:10])
res1 <- sum(pred1 == credit [,11]) / nrow(credit)
```

the summary for the model is as follows:

```
##   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
```

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

```
## 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
```

```
##            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
```

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

The result is as follows:

```
##   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
```

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

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

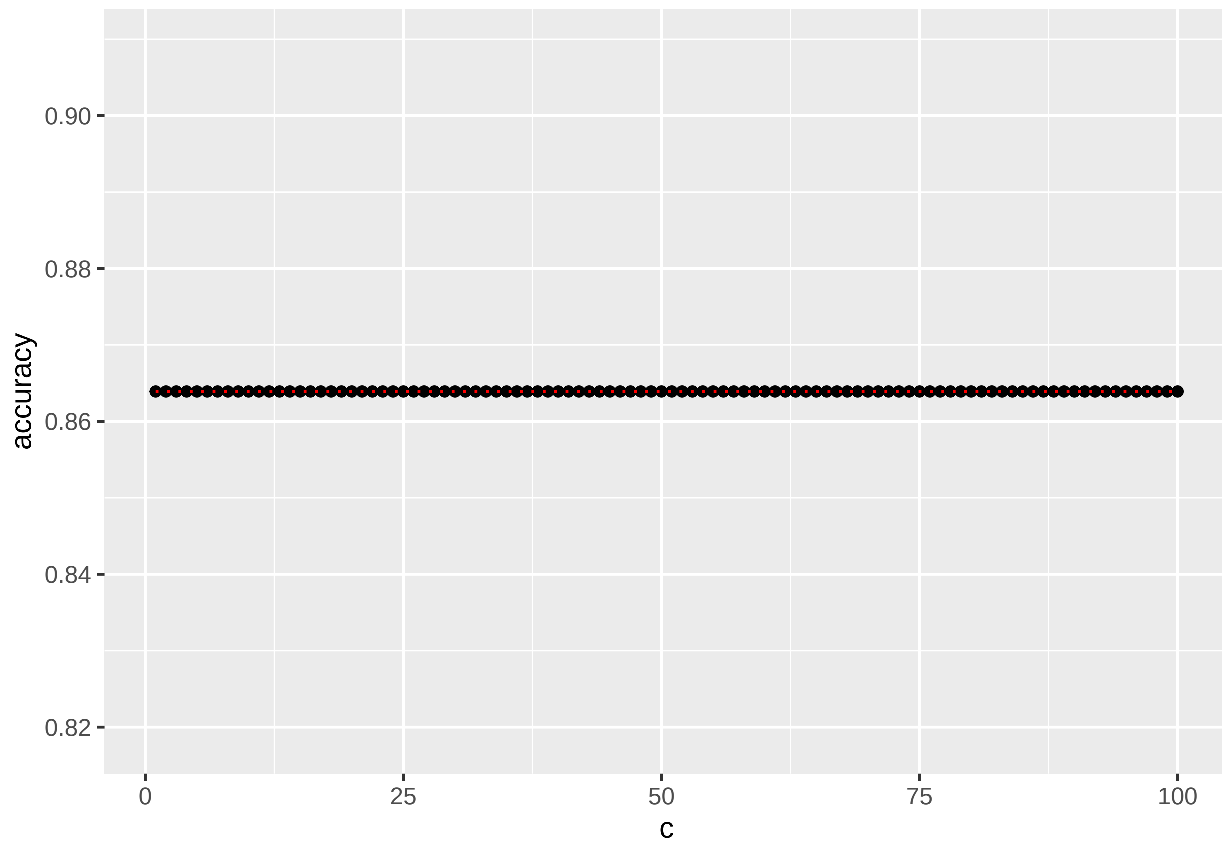
We can see that the model has an accuracy of 86.3% approximately. Next, I try to choose an appropriate C-value using a for-loop.

```
#choosing best model : c =1 ~100
set.seed(101)
test.c <- list(1:100)
acc <- data.frame(matrix(ncol = 2, nrow = 100))
names(acc) <- c("c","accuracy")
for (i in test.c){
  model <- ksvm(x = as.matrix(credit[,1:10]),
                y = as.factor(credit[,11]),
                type = "C-svc" ,
                scaled = TRUE ,
                kernel = "vanilladot" ,
                C = i)
  pred <- predict(model,credit[,1:10])
  res.0 <- sum(pred1 == credit [,11]) / nrow(credit)
  acc[i,1] <- i
  acc[i,2] <- res.0
}
```

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

However, the results show that most C do not change the accuracy that much.

```
svm.plt <-ggplot(acc, aes(x = c , y = accuracy)) + geom_point() + geom_line(lty = "dotted" , color = "re
svm.plt
```



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.

On top of the linear kernels, I also attempted 3 other kernals: "Radial Basis" ,"Polynomial" and "Hyperbolic tangent" To keep thing relaitively simple, I keep the C-value at 100.

```
#2.2.2
#Using other non-linear models

#Radial Basis kernel "Gaussian"
set.seed(101)
model.2 <- ksvm(x = as.matrix(credit[,1:10]),
                y = as.factor(credit[,11]),
                type = "C-svc" ,
                scaled = TRUE ,
                kernel = "rbfdot" ,
                C = 100)
b <- colSums(model.2@xmatrix[[1]]*model.2@coef[[1]])
b0 <- model.1@b

pred2 <- predict(model.2,credit[,1:10])
res2 <- sum(pred2 == credit [,11]) / nrow(credit)

#Polynomial kernel
set.seed(101)
```

```
model.3 <- ksvm(x = as.matrix(credit[,1:10]),
                y = as.factor(credit[,11]),
                type = "C-svc" ,
                scaled = TRUE ,
                kernel = "polydot" ,
                C = 100)
```

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

```
c <- colSums(model.3@xmatrix[[1]]*model.3@coef[[1]])
c0 <- model.3@b

pred3 <- predict(model.3,credit[,1:10])
res3 <- sum(pred3 == credit [,11]) / nrow(credit)

# Hyperbolic tangent kernel
set.seed(101)
model.4 <- ksvm(x = as.matrix(credit[,1:10]),
                y = as.factor(credit[,11]),
                type = "C-svc" ,
                scaled = TRUE ,
                kernel = "tanhdot" ,
                C = 100)
```

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

```
d <- colSums(model.4@xmatrix[[1]]*model.4@coef[[1]])
d0 <- model.4@b

pred4 <- predict(model.4,credit[,1:10])
res4 <- sum(pred4 == credit [,11]) / nrow(credit)

#summaring up results
pred.list <- c(res1,res2,res3,res4)
kernel.list <- c("Linear","Radial Basis" ,"Polynomial" ,"Hyperbolic tangent")
result.df <- data.frame(kernel.list ,pred.list)
```

We can see from the results that a Radial Basis Kernal as the best performance of the 4, with a 95.7%approx. accuracy

```
print(result.df)
```

```
##          kernel.list pred.list
## 1             Linear 0.8639144
## 2       Radial Basis 0.9571865
## 3         Polynomial 0.8639144
## 4 Hyperbolic tangent 0.7217125
```

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.

```
R1 <-credit[,11]
pred5<- rep(0,(nrow(credit)))
set.seed(101)
for (i in 1:nrow(credit)){
  #making sure that i won't use it self
```

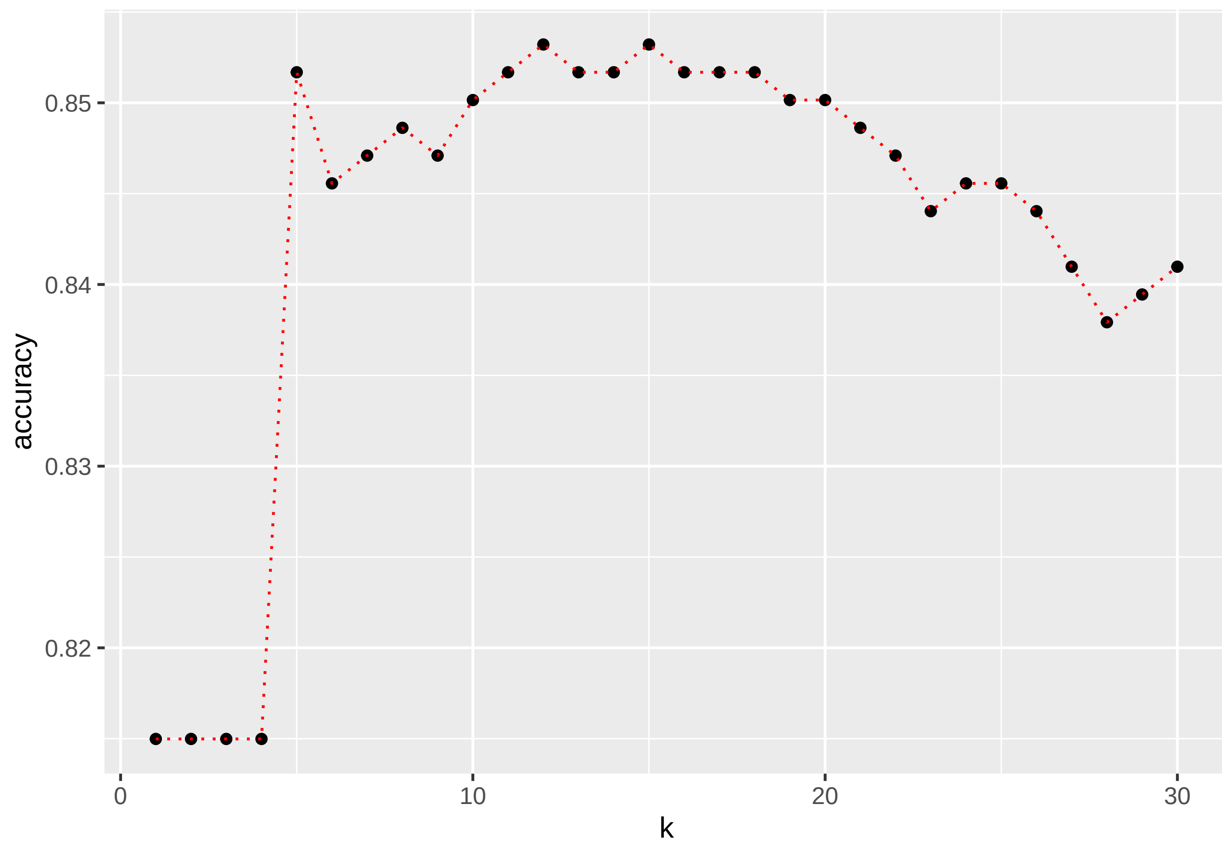
```
  knn.model=kknn(R1~., credit[-i,],credit[i,],k=1, scale = T)
  pred5[i]<- as.integer(fitted(knn.model)+0.5)
}

res5 = sum(pred5 == R1) / nrow(credit)
res5
```

```
## [1] 0.8149847
```

Next, I try different ks(1~30)

```
knn.df <- data.frame(matrix(nrow = 30, ncol = 2))
colnames(knn.df) <- c("k" , "accuracy")
set.seed(101)
for(n in 1:30){
  for (i in 1:nrow(credit)){
    knn_model=kknn(R1~., credit[-i,],credit[i,],k=n, scale = T)
    pred5[i] <- as.integer(fitted(knn_model)+0.5)
    res.00 <- sum(pred5 == R1) / nrow(credit)
    knn.df[n,1]<- n
    knn.df[n,2]<- res.00
  }
}
```

```
knn.plt <-ggplot(knn.df, aes(x = k , y = accuracy)) + geom_point() + geom_line(lty = "dotted" , color =
knn.plt
```



We can see that k = 12 and 15 has the biggest accuracy of 85.3%

```
##     k accuracy
## 12 12 0.853211
## 15 15 0.853211
## 5   5 0.851682
## 11 11 0.851682
## 13 13 0.851682
## 14 14 0.851682
```

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);

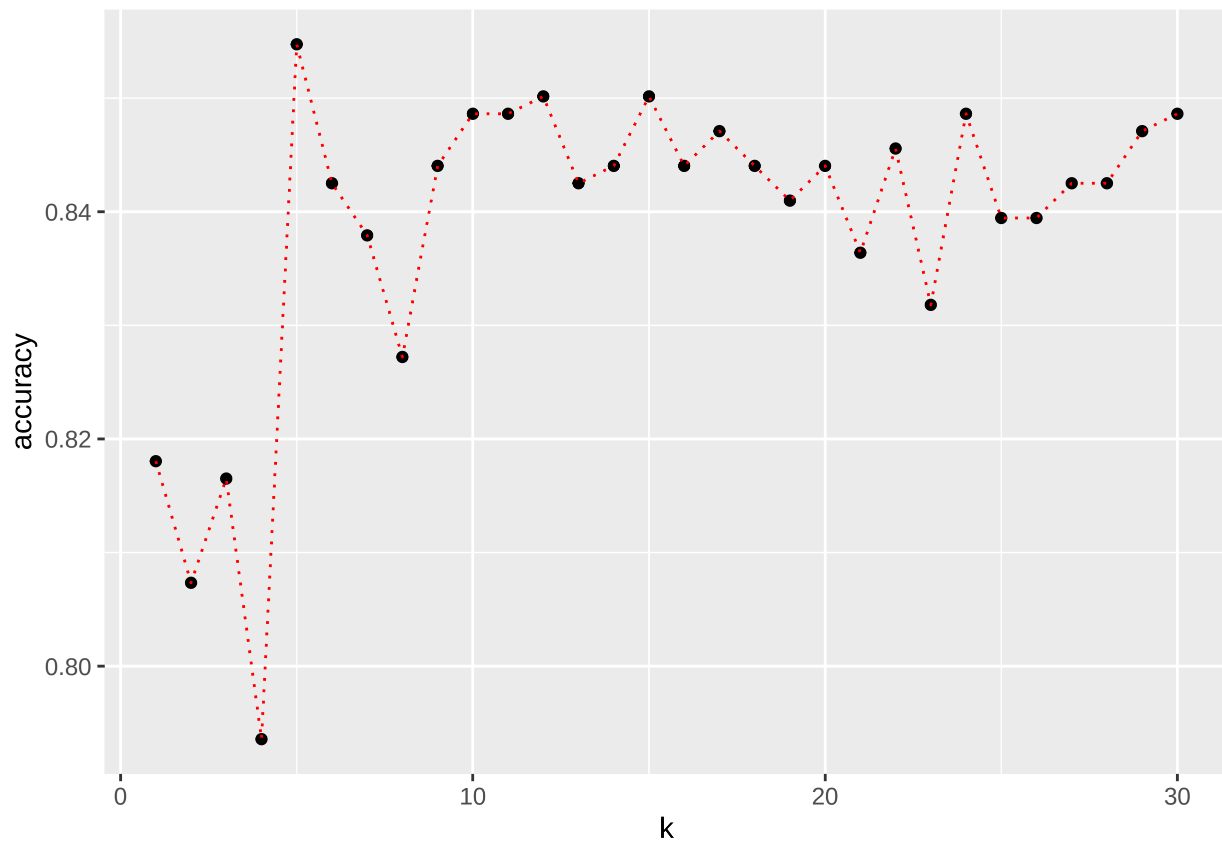
I choose to use the cv.kknn function to run a k = 10 k-fold Cross Validation

```r
#3.1
#a
#k-fold Cross validation
acc2 <- data.frame(matrix(nrow = 30 ,ncol = 2))
names(acc2) <- c("k" , "accuracy")
set.seed(101)
for (i in 1:30){
  knn_model2 <- cv.kknn(R1~ ., credit , kcv = 10 , k = i, scale = T)
  pred6 <- round(knn_model2[[1]][,2])
  res6 <- sum(pred6 == credit [,11]) / nrow(credit)
  acc2[i,1] <- i
  acc2[i,2] <- res6
}
head(acc2[order(-acc2["accuracy"]),])
```

```
##     k  accuracy
## 5   5 0.8547401
## 12 12 0.8501529
## 15 15 0.8501529
## 10 10 0.8486239
## 11 11 0.8486239
## 24 24 0.8486239
```

Represented Graphically

```r
knn.plt2 <-ggplot(acc2, aes(x = k , y = accuracy)) + geom_point() + geom_line(lty = "dotted" , color =
knn.plt2
```




```
##     k  accuracy
## 5   5 0.8547401
## 12 12 0.8501529
## 15 15 0.8501529
## 10 10 0.8486239
## 11 11 0.8486239
## 24 24 0.8486239
```

(b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).

I choose to split my three data sets to the porpotion of 70:15:15(training:validation:test), the data are choosen randomly using the sample()function

```
#splitting data
train.index <- sample(nrow(credit),nrow(credit) * 0.7)
train.data <-  credit[train.index,]
remaining_data <- credit[-train.index,]
vad.index <- sample(nrow(remaining_data),nrow(remaining_data) * 0.5)
vad.data <- remaining_data[vad.index,]
test.data <- remaining_data[-vad.index,]

#testing whether total rows are correct
nrow(test.data) + nrow(vad.data) + nrow(train.data) == nrow(credit)
```

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

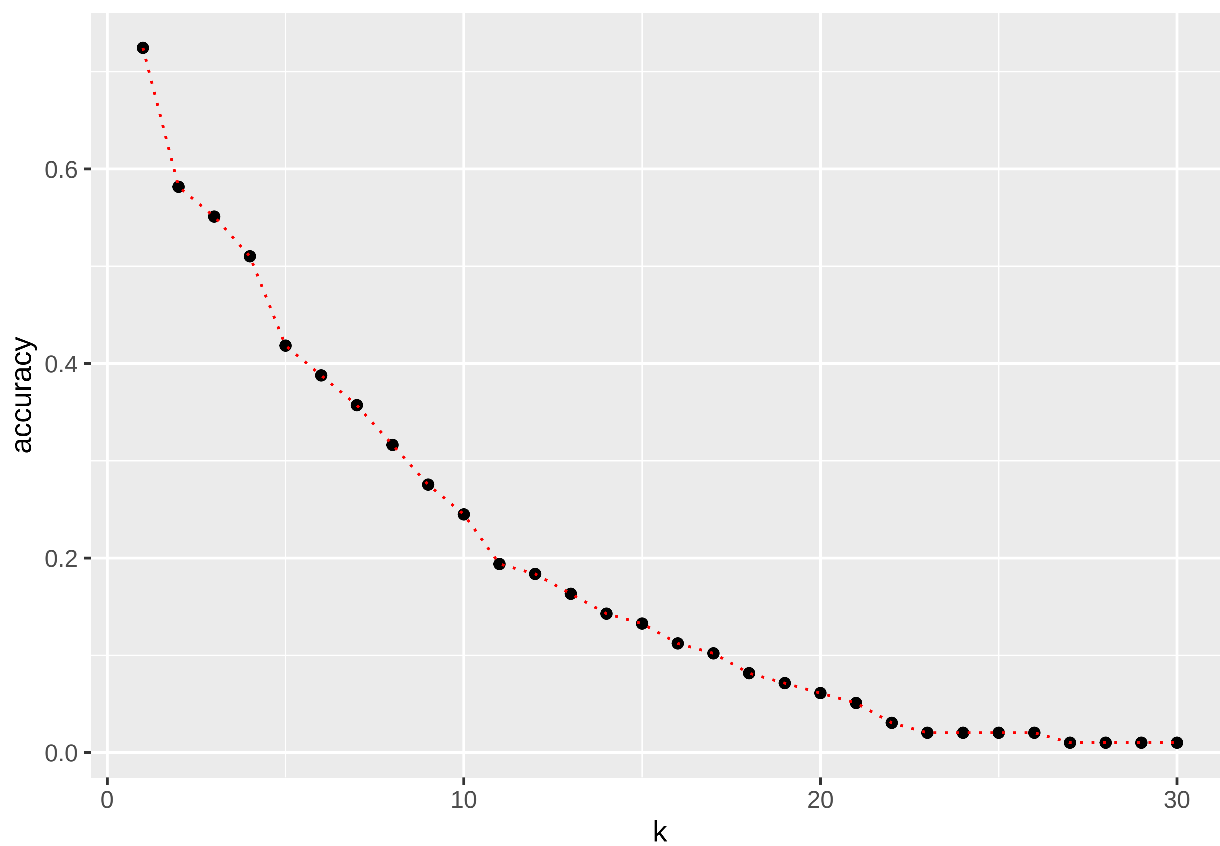
I then run a for loop over for the KNN model with k from 1 to 30 using the training data set to train and validation data set to test.

```
set.seed(101)
acc3 <- data.frame(matrix(nrow = 30 ,ncol = 2))
names(acc3) <- c("k" , "accuracy")
pred7<- rep(0,(nrow(vad.data)))
for(n in 1:30){
    knn_model3=kknn(R1~., train = train.data,test = vad.data,k=n, scale = T)
    res7 <- sum(knn_model3$fitted.values == vad.data$R1) / nrow(vad.data)
    acc3[n,1]<- n
    acc3[n,2]<- res7
  }
```

```
head(acc3[order(-acc3["accuracy"]),])
```

```
##   k  accuracy
## 1 1 0.7244898
## 2 2 0.5816327
## 3 3 0.5510204
## 4 4 0.5102041
## 5 5 0.4183673
## 6 6 0.3877551
```

```
knn.plt3 <-ggplot(acc3, aes(x = k , y = accuracy)) + geom_point() + geom_line(lty = "dotted" , color =
knn.plt3
```



The results, represented in a matrix and in a graph show that the accuracy would be at its highest if k = 1, giving and accuracy of 72.4%(approx.) Therefore, using k = 1, I use the model again to test on the test data.

```r
set.seed(101)
knn_model4=kknn(R1~., train = train.data,test = test.data,k=1, scale = T)
sum(knn_model4$fitted.values == test.data$R1) / nrow(test.data)
```

```
## [1] 0.8181818
```

The result is better what was expected, giving an accuracy of 81.8%(approx.)