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.

1. Classifying candidates, who apply for a job, after interview as “No hire”, “Leaning No hire”, “Leaning Hire”, “Hire”, “Strong Hire”
   a. Predictors: no. of questions, no. of correct, no. of incorrect, experience of the candidate
2. After grocery shopping classify items as perishable vs non-perishable and store them either in the refrigerator or outside on the shelf
   a. Predictors: type of item, vegetables, packed goods, frozen or not
3. While Playing lego blocks with my daughter, helping her classify the blocks by color, shapes and sizes
   a. Predictors: colors, shape, size
4. Classifying good vs bad apples when buying them
   a. Predictors: color, shape, firmness
5. Classifying weekend hikes by intensity and time before deciding on the hike to take
   a. Predictors: distance, location from home, elevation

Question 2.2

Import Libraries

```
library(kernlab)
library(rmarkdown)
require(data.table)
```

Loading required package: data.table

Read data

```
data<-read.table(/Assignments/ISYE-6501/week1/data\ 2.2/credit_card_data-headers.txt", header=TRUE)
datamatrix <- as.matrix(data)
```

Train Model

trainLoop function to loop through different values of C for a given svm kernal. Return the best performing model

```
trainLoop <- function(k) {
  results=list()
```

```
 model <- NULL
 cval <- c(25,80,95,100,500,1000)
 count= 0

 for ( val in cval) {
   count= count + 1
   model <- ksvm(x=datamatrix[,1:10],y=datamatrix[,11], type="C-svc",  kernel=k, C=val, scaled=TRUE)
   a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
   a0 <- -model@b
   pred <- predict(model,datamatrix[, 1:10])

   results[[count]] = data.table(kernal_=k, a0_=a0, a_=a,C_=val,accuracy_=sum(pred == data[,11]) /
nrow(data))
 }

 return(rbindlist(results)[order(-accuracy_)][1:10])
}
```

Call the train function to train models

Define the list of kernals to try and then call the trainLoop function.

```
kernals <- c("vanilladot", "rbfdot", "polydot")


 results_merged=list()
 kcount<-0
 #loop through the kernals
 for (k in kernals) {
   kcount=kcount+1
   results_merged[[kcount]] = trainLoop(k)

 }
```

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

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

Top Results for each Kernal

```
## [[1]]
##        kernal_        a0_            a_ C_ accuracy_
##  1: vanilladot 0.08159128 -0.0009884196 25 0.8639144
##  2: vanilladot 0.08159128 -0.0008760683 25 0.8639144
##  3: vanilladot 0.08159128 -0.0016321668 25 0.8639144
##  4: vanilladot 0.08159128  0.0026042255 25 0.8639144
##  5: vanilladot 0.08159128  1.0052357680 25 0.8639144
##  6: vanilladot 0.08159128 -0.0025888253 25 0.8639144
##  7: vanilladot 0.08159128 -0.0001812252 25 0.8639144
##  8: vanilladot 0.08159128 -0.0003848578 25 0.8639144
##  9: vanilladot 0.08159128 -0.0012054701 25 0.8639144
## 10: vanilladot 0.08159128  0.1064407684 25 0.8639144
##
## [[2]]
##     kernal_       a0_        a_   C_ accuracy_
##  1:  rbfdot 0.7059756 -58.31781 1000 0.9801223
##  2:  rbfdot 0.7059756 -13.54450 1000 0.9801223
##  3:  rbfdot 0.7059756 -41.33221 1000 0.9801223
##  4:  rbfdot 0.7059756 135.03942 1000 0.9801223
##  5:  rbfdot 0.7059756  83.16611 1000 0.9801223
##  6:  rbfdot 0.7059756 -95.43087 1000 0.9801223
##  7:  rbfdot 0.7059756 110.28435 1000 0.9801223
##  8:  rbfdot 0.7059756 -70.23276 1000 0.9801223
##  9:  rbfdot 0.7059756 -81.11027 1000 0.9801223
## 10:  rbfdot 0.7059756 100.73060 1000 0.9801223
##
## [[3]]
##     kernal_        a0_            a_ C_ accuracy_
##  1: polydot 0.08155816 -0.0010504747 25 0.8639144
##  2: polydot 0.08155816 -0.0012724073 25 0.8639144
##  3: polydot 0.08155816 -0.0015648400 25 0.8639144
##  4: polydot 0.08155816  0.0024923798 25 0.8639144
##  5: polydot 0.08155816  1.0053108361 25 0.8639144
##  6: polydot 0.08155816 -0.0027017651 25 0.8639144
##  7: polydot 0.08155816 -0.0002493278 25 0.8639144
##  8: polydot 0.08155816 -0.0003655242 25 0.8639144
##  9: polydot 0.08155816 -0.0013374678 25 0.8639144
## 10: polydot 0.08155816  0.1063986013 25 0.8639144
```

For the given kernals, rbfdot seems to perform the best.

Equations

Kernel=rbfdot

```
a0
```

[1] **0.7059756**

(-58.31781* A1 + -13.54450* A2 + -41.33221*A3 + 135.03942*A8 + 83.16611*A9 + -95.43087*A10 + 110.28435*A11 + -70.23276*A12 + -81.11027*A14 + 100.73060*A15 + **0.7059756)* R1 >=1**

Kernel=vanilladot

```
a0
```

[1] **0.08159128**

(-0.0009884196 * A1 + -0.0008760683* A2 + -0.0016321668*A3 + 0.0026042255*A8 + 1.0052357680*A9 + -0.0025888253*A10 + -0.0001812252*A11 + -0.0003848578*A12 + -0.0012054701*A14 + 0.1064407684*A15 + **0.08159128)* R1 >=1**

KKNN

```
library(kknn)
library(data.table)
```

Load data

```
data<-read.table("/Assignments/ISYE-6501/week1/data\ 2.2/credit_card_data-headers.txt",
header=TRUE)
```

Looping Method

Train function to loop for every value of K for a given kernel, for each i

```
trainkknnloop <- function(kknn_kernel){

  predicted <- rep(0,(nrow(data)))
  kvalues = seq(2,3)
  results = list()
  count=0;
  m = dim(data)[1]
  for (kval in kvalues){


  for(i in 1:m){
    kknnmodel_2 <-NULL
    kknnmodel_2 <- kknn(R1~.,data[-i,],data[i,], k=kval, kernel=kknn_kernel, scale=TRUE)
    fit <- fitted(kknnmodel_2)
    predicted[i] = fit

  }
  count = count +1
  results[[count]] = data.table(accuracy=sum(predicted == data[,11])/m, kval=kval, kernel=kknn_kernel)
  }
  return(rbindlist(results)[order(-accuracy)][1])
}
```

Call Training function for each kernel and try different K values from 2 to 10

```
kkcount <-0

accurracy_list= list()
kknn_kernels = c("rectangular", "triangular","gaussian")
for (kknn_kernel in kknn_kernels) {

    kkcount = kkcount+1
    accurracy_list[[kkcount]] = trainkknnloop(kknn_kernel)
}
```

Top K value for each kernel

```
## [[1]]
##    accuracy kval      kernel
## 1: 0.6865443    2 rectangular
##
## [[2]]
##    accuracy kval     kernel
## 1: 0.6865443    2 triangular
##
```

```
## [[3]]
##    accuracy kval   kernel
## 1: 0.6865443    2 gaussian
```

Method 2 using train.knn function

Sample data and break them into train and validation sets

```
m <- dim(data)[1]
 test_sample <- sample(1:m, size = round(m/3), replace = FALSE, prob = rep(1/m, m))

 testing <- data[test_sample, ]
 learning <- data[-test_sample, ]

 dim(learning)
```

```
## [1] 436  11
```

```
dim(testing)
```

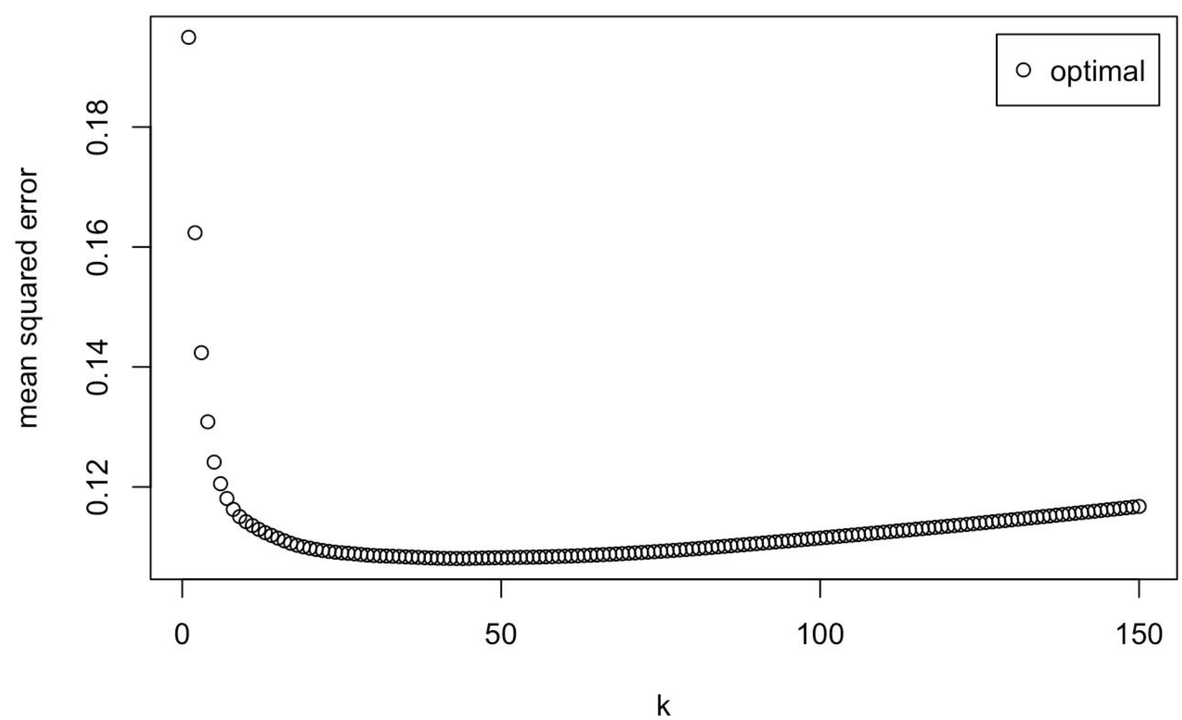
```
## [1] 218  11
```

Train the model on the training set

```
kknnmodel <- train.kknn(R1 ~ ., data = learning, kmax = 150, scale=TRUE)
```

Optimal K value

```
## [1] 43
```



Accuracy

```
prediction <- predict(kknnmodel, testing[, -11])
 pred<-round(prediction)
 pred_accuracy<-table(pred,testing$R1)
 pred_accuracy
```

```
##
## pred   0   1
##    0 103  15
##    1  17  83
```

```
sum(pred==testing$R1)/length(testing$R1)
```

```
## [1] 0.853211
```