hw1

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.

I think a situation for which a classification model would be appropriate is smart watches especially for people who would like to attain a monthly/yearly/quarterly fitness goal (ie: losing a certain amount of weight).
The classes here would be: people who lost weight and people who did not.

Predictors:

- Age
- Occupation
- Average daily steps count
- Gender
- Average daily heart rate

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.)

```
#installing the kernlab and kknn package
install.packages("kernlab")
install.packages("kknn")
library(kernlab)
library(kknn)
#read file into a  data frame in table format and stores it in my_data
my_data <- read.table("credit_card_data.txt")
#I initally used the code provided with the homework but it gave me an error
"“unused arguments (length = 4, lambda = 0.5)” error in kernlab
#After research on the internet and reading the piazza forum, I found that co
nverting the x and y inputs to matrix and factor respectively, helped with th
is error
#I chose a random number :5 and tried the formula with different powers of 5
```

```
to see what fraction of the model matches closely the actual classification
cc_model1 <- ksvm(as.matrix(my_data[,1:10]),as.factor(my_data[,11]), C=0.05,
scaled = T, kernel="vanilladot", type = "C-svc")

##  Setting default kernel parameters

a_1<- colSums(cc_model1@xmatrix[[1]] * cc_model1@coef[[1]])
a_1

##            V1            V2            V3            V4            V5
## -0.0011704691 -0.0005878213 -0.0012076263  0.0028419410  1.0007612933
##            V6            V7            V8            V9           V10
## -0.0030549796  0.0012087228 -0.0009896919 -0.0014851677  0.1060907827

a0_1 <- -cc_model1@b
a0_1

## [1] 0.0836002

pred1<- predict(cc_model1, my_data[,1:10])
sum(pred1 == my_data[,11]) / nrow(my_data)

## [1] 0.8639144

cc_model2 <- ksvm(as.matrix(my_data[,1:10]),as.factor(my_data[,11]), C=0.5, s
caled = T, kernel="vanilladot", type = "C-svc")

##  Setting default kernel parameters

a_2<- colSums(cc_model2@xmatrix[[1]] * cc_model2@coef[[1]])
a_2

##            V1            V2            V3            V4            V5
## -0.0011076132 -0.0008917150 -0.0015779256  0.0028841598  1.0048076534
##            V6            V7            V8            V9           V10
## -0.0028963067 -0.0002095800 -0.0005780909 -0.0011782094  0.1064133151

a0_2 <- -cc_model1@b
a0_2

## [1] 0.0836002

pred2 <- predict(cc_model2, my_data[,1:10])
sum(pred2 == my_data[,11]) / nrow(my_data)

## [1] 0.8639144

cc_model3 <- ksvm(as.matrix(my_data[,1:10]),as.factor(my_data[,11]), C=5, sca
led = T, kernel="vanilladot", type = "C-svc")

##  Setting default kernel parameters

a_3<- colSums(cc_model3@xmatrix[[1]] * cc_model3@coef[[1]])
a_3
```

```
##            V1            V2            V3            V4            V5
## -0.0009059129 -0.0009822588 -0.0016646387  0.0025578654  1.0052684085
##            V6            V7            V8            V9           V10
## -0.0025973024 -0.0002203001 -0.0003290890 -0.0012589283  0.1064307652
```

```
a0_3<- -cc_model3@b
a0_3
```

```
## [1] 0.08162012
```

```
pred3 <- predict(cc_model3, my_data[,1:10])
sum(pred3 == my_data[,11]) / nrow(my_data)
```

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

```
cc_model4 <- ksvm(as.matrix(my_data[,1:10]),as.factor(my_data[,11]), C=50, sc
aled = T, kernel="vanilladot", type = "C-svc")
```

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

```
a_4<- colSums(cc_model4@xmatrix[[1]] * cc_model4@coef[[1]])
a_4
```

```
##            V1            V2            V3            V4            V5
## -0.0010523630 -0.0012025131 -0.0015382662  0.0028761998  1.0052764944
##            V6            V7            V8            V9           V10
## -0.0024958086  0.0001810245 -0.0006514829 -0.0013757143  0.1064002847
```

```
a0_4 <- -cc_model4@b
a0_4
```

```
## [1] 0.08147145
```

```
pred4 <- predict(cc_model4, my_data[,1:10])
sum(pred4 == my_data[,11]) / nrow(my_data)
```

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

```
cc_model5 <- ksvm(as.matrix(my_data[,1:10]),as.factor(my_data[,11]), C=500, s
caled = T, kernel="vanilladot", type = "C-svc")
```

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

```
a_5<- colSums(cc_model5@xmatrix[[1]] * cc_model5@coef[[1]])
a_5
```

```
##            V1            V2            V3            V4            V5
## -6.306278e-04 -1.994861e-04 -3.750699e-04  1.615496e-03  1.003398e+00
##            V6            V7            V8            V9           V10
## -3.814784e-04  6.761309e-05 -3.621798e-05 -1.272743e-04  1.057258e-01
```

```
a0_5 <- -cc_model5@b
a0_5
```

```
## [1] 0.08534036

pred5 <- predict(cc_model5, my_data[,1:10])
sum(pred5 == my_data[,11]) / nrow(my_data)

## [1] 0.8639144

cc_model6 <- ksvm(as.matrix(my_data[,1:10]),as.factor(my_data[,11]), C=5000,
scaled = T, kernel="vanilladot", type = "C-svc")

##  Setting default kernel parameters

a_6<- colSums(cc_model6@xmatrix[[1]] * cc_model6@coef[[1]])
a_6

##            V1            V2            V3            V4            V5
##  0.0006731908  0.0083010624 -0.0020956953  0.0036643813  1.0018274134
##            V6            V7            V8            V9           V10
## -0.0012200402  0.0016892488 -0.0006029131  0.0007377296  0.0044516724

a0_6 <- -cc_model6@b
a0_6

## [1] 0.07045529

pred6 <- predict(cc_model6, my_data[,1:10])
sum(pred6 == my_data[,11]) / nrow(my_data)

## [1] 0.8623853

cc_model7 <- ksvm(as.matrix(my_data[,1:10]),as.factor(my_data[,11]), C=50000,
scaled = T, kernel="vanilladot", type = "C-svc")

##  Setting default kernel parameters

a_7<- colSums(cc_model7@xmatrix[[1]] * cc_model7@coef[[1]])
a_7

##            V1            V2            V3            V4            V5
## -0.0054482536 -0.0047167277  0.0602292155 -0.0277398078  1.0200890866
##            V6            V7            V8            V9           V10
## -0.0250900645  0.0454624504 -0.0176236821  0.0070736709 -0.0007544996

a0_7 <- -cc_model7@b
a0_7

## [1] 0.07104353

pred7 <- predict(cc_model7, my_data[,1:10])
sum(pred7 == my_data[,11]) / nrow(my_data)

## [1] 0.8623853
```

```
cc_model8 <- ksvm(as.matrix(my_data[,1:10]),as.factor(my_data[,11]), C=500000
, scaled = T, kernel="vanilladot", type = "C-svc")

##  Setting default kernel parameters

a_8<- colSums(cc_model8@xmatrix[[1]] * cc_model8@coef[[1]])
a_8

##          V1          V2          V3          V4          V5          V6
##  0.65378568 -0.24204401  1.07837854 -0.02326028  0.46918159  0.04930826
##          V7          V8          V9         V10
##  0.27398589 -0.10563386 -0.43954881  0.32342971

a0_8 <- -cc_model8@b
a0_8

## [1] -0.1567435

pred8 <- predict(cc_model8, my_data[,1:10])
sum(pred8 == my_data[,11]) / nrow(my_data)

## [1] 0.7125382

cc_model9 <- ksvm(as.matrix(my_data[,1:10]),as.factor(my_data[,11]), C=500000
0, scaled = T, kernel="vanilladot", type = "C-svc")

##  Setting default kernel parameters

a_9<- colSums(cc_model9@xmatrix[[1]] * cc_model9@coef[[1]])
a_9

##         V1         V2         V3         V4         V5         V6
V7
## -0.1057036 10.8796811 -4.7489824  1.2555724  0.6754055 -1.1260273  4.22315
92
##         V8         V9        V10
##  0.8589146  4.8609166  2.2599450

a0_9 <- -cc_model9@b
a0_9

## [1] 0.8361298

pred9 <- predict(cc_model9, my_data[,1:10])
sum(pred9 == my_data[,11]) / nrow(my_data)

## [1] 0.6116208
```

Table with different values of C

Given the table below, I think the lowest powers of 5 have the best error. I will choose 500 because, it is one of the values of C with the highest accuracy, and I also wanted to minimize the number of support vectors to allow for a little less slack.

Value of C	Training error	What fraction of model matches the actual classification	Number of support vectors
0.05	0.136086	0.8639144	209
0.5	0.136086	0.8639144	191
5	0.136086	0.8639144	191
50	0.136086	0.8639144	189
500	0.136086	0.8639144	193
5000	0.137615	0.8623853	291
50000	0.137615	0.8623853	290
500000	0.287462	0.7125382	457
5000000	0.388379	0.6116208	313

One thing I learned with this exercise is to always make sure you are in the correct working directory before you start writing line of codes and always check the syntax of the lines of code, because a missing parenthesis or a symbol that is not well written could throw errors. It was also helpful to go through the Piazza forum to interact with my classmates and find solutions for problems I am encountering others may have asked about.

For this second exercise, I found the TA sessions helpful for guidance, once I got the basic formula for kknn, I was able to construct a function from that in order to iterate through the data frame

2. 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`).

```
#function to iterate to the columns of my_data

check_accuracy = function(X){
  #we start the prediction with a vector of 0s
  predicted <- rep(0,(nrow(my_data)))
  # for each row, we then estimate its response based on the other rows

  for (i in 1:nrow(my_data)){

    # we use data[-i] to remove row i of the data when finding the nearest ne
ighbors
    #V11~ means use all data but V11.
    #use scaled data

    model=kknn(V11~.,my_data[-i,],my_data[i,],k=X, scale = TRUE)
```

```r
    #since kknn will read the responses as continuous, and return the fractio
n of the k closest responses that are 1 (rather than the most common response
, 1 or 0).
    # adding 0.5 to round off to 0 (when prediction is <0.5) or 1 (when predi
ction is >0.5)
    predicted[i] <- as.integer(fitted(model)+0.5)
  }

  # calculating what fraction of the model's predictions match the actual cla
ssification
  acc = sum(predicted == my_data[,11]) / nrow(my_data)
  return(acc)
}

#
# Now call the function for values of k from 1 to 50
#

accurracy=rep(0,50) # set up a vector of 50 zeros to start
#iterate through the vector,
for (X in 1:50){
  # testing knn model with X neighbors
  accurracy[X] = check_accuracy(X)
}
#
#define which k value gives the maximum accuracy
#
which.max(accurracy)

max(accurracy)

[1] 12

[1] 0.853211
```