Question1:

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:-

Problem: - Reaching Work on time

Predictors:-

1. Start time
2. Traffic in 101.
3. School traffic.
4. Elevator wait time
5. Speed

Question2.1:-

Answer:- C is 20

How:- Sum of squared coefficients should be equal to 1. Some times 1 – FUV(fraction of unexplained variance)

My program gave R^2 as .88 for C = 20. Also .68 as fraction of the model's predictions match the actual classification.

Additionally I have set the Scaled as FALSE. Since the Coefficient of V10 is too high when it is scale

See below the code and results

```
# Support Vector Machine - question 2.1
# getting the base program ready
#include libraries
library(e1071)
library(kernlab)
library(kknn)
# Read the source using read.table function
data <- read.table(
  "https://d37djvu3ytnwxt.cloudfront.net/assets/courseware/v1/39b78ff5c5c28981f009b54831d81649/asset-v1:GTx+ISYE6501x+2T2017+type@asset+block/cr


#setting up data for KSVM (X and Y values)
x <- as.matrix(data[,1:10])
y <- as.factor(data[,11])
z <- data[,2:3]
# call ksvm. Vanilladot is a simple linear kernel.
model <- ksvm(x,y,type="C-svc",kernel="vanilladot",C=20,scaled=FALSE)
print(model)
# calculate a1…am
a <- colSums(data[model@SVindex,1:10] * model@coef[[1]])
print ("Printing A")
print (a)
# calculate a0
a0 <- sum(a*data[1,1:10])
print ("Printing A0")
print(a0)
# see what the model predicts
pred <- predict(model,data[,1:10])
print("printing predition")
print(pred)
print("printing Sum of Square")
#calculate sumof squares
ss <- sum(a^2)
print (ss)
# see what fraction of the model's predictions match the actual classification
s <- sum(pred == data[,11]) / nrow(data)
print("printing fraction")
print(s)
```

```
Linear (vanilla) kernel function.

Number of Support Vectors : 189

Objective Function Value : -806.9812
Training error : 0.311927
[1] "Printing A"
           V1            V2            V3            V4            V5
 0.0374385047  0.0467830465 -0.0123965906  0.0734026630  0.9028736845
           V6            V7            V8            V9           V10
 0.0901740447  0.1010230252  0.1866312593 -0.0003248516 -0.0004888383
[1] "Printing A0"
[1] 2.696421
[1] "printing predition"
  [1] 1 1 0 1 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 0 1 1
 [38] 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0
 [75] 0 0 0 0 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 1 1 1 1 1 0 1 0 0 1 0 1
[112] 1 0 1 0 1 1 1 1 1 1 1 0 1 0 1 0 0 1 1 1 1 1 0 0 1 1 1 0 1 0 0 1 0 1 1 1 1
[149] 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1 0 0 0 0 0 1 0 1 0 1 0 1 1 1 0 1
[186] 0 1 1 1 1 1 1 1 0 1 0 1 0 0 0 1 1 1 1 0 1 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 0
[223] 0 0 1 1 0 0 0 0 0 1 1 0 1 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1
[260] 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 1 0 0 0 0 1 0 0 0
[297] 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 1 0 1 0
[334] 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 1 0 0 0 0
[371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[408] 0 0 1 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0
[445] 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0
[482] 1 1 0 0 1 1 0 0 1 0 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 0 1 0 1 1 1
[519] 1 1 0 1 1 1 1 1 0 1 1 1 0 0 1 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 1 0 1 0 0 1
[556] 1 0 1 1 1 0 0 1 1 0 1 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
[593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
[630] 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
Levels: 0 1
[1] "printing Sum of Square"
[1] 0.8774814
[1] "printing fraction"
[1] 0.6880734
```

(Showing notes – a sample of my way of figuring out the answer. Just in case my C is wrong)

First attempt (getting the base program ready – setting up matrix and figure out errors)

```
C:\Users\USWXQ7Q\Documents\Work\personal\narayanan\DataSci\Week2\data1\gtassw1.R - R ...
#setting up data for KSVM (X and Y values)
x <- as.matrix(data[,1:10])
y <- as.factor(data[,11])
# call ksvm. Vanilladot is a simple linear kernel.
model <- ksvm(x,y,type="C-svc",kernel="vanilladot",C=100,scaled=TRUE)
# calculate a1…am
a <- colSums(data[model@SVindex,1:10] * model@coef[[1]])
print("value of 'a'")
print (a)
# calculate a0
a0 <- sum(a*data[1,1:10])
print("value of 'a0'")
print (a0)
# see what the model predicts
pred <- predict(model,data[,1:10])
print("value of 'pred'")
print(pred)
# see what fraction of the model’s predictions match the actual classification
z <- sum(pred == data[,11]) / nrow(data)
print(z)
print("'model'")
print (model)
```

Results:-

```
> source("gtassw1.R")
 Setting default kernel parameters
[1] "value of 'a'"
           V1            V2            V3            V4            V5
-4.660362e-04 -1.405350e-02 -8.168866e-03  1.012922e-02  5.016095e-01
           V6            V7            V8            V9           V10
-1.403434e-03  1.291217e-03 -2.668989e-04 -2.067550e-01  5.583356e+02
[1] "value of 'a0'"
[1] -41.68294
[1] "value of '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 1 1
 [38] 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 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 0 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
[1] 0.8639144
```

```
[1] "'model'"
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
> 
```

Question2.1

Figuring out the C:-

Tried a loop for various values of C to see the impact on the predictor. No luck. No variations.

Tried a highest value 10,000 and lowest value 5.

Here is one of the sample program for various attempts

C:\Users\USWXQ7Q\Documents\Work\personal\narayanan\DataSci\Week2\data1\gtassw1.R - R ...

```
#setting up data for KSVM (X and Y values)
x <- as.matrix(data[,1:10])
y <- as.factor(data[,11])
for (i in 1:100){
# call ksvm. Vanilladot is a simple linear kernel.
model <- ksvm(x,y,type="C-svc",kernel="vanilladot",C=i,scaled=TRUE)
# calculate a1…am
a <- colSums(data[model@SVindex,1:10] * model@coef[[1]])
# calculate a0
a0 <- sum(a*data[1,1:10])
# see what the model predicts
pred <- predict(model,data[,1:10])
print("value of 'pred'")
print(pred)
# see what fraction of the model’s predictions match the actual classification
z <- sum(pred == data[,11]) / nrow(data)

}
```

Values of C=5 and C=10,000

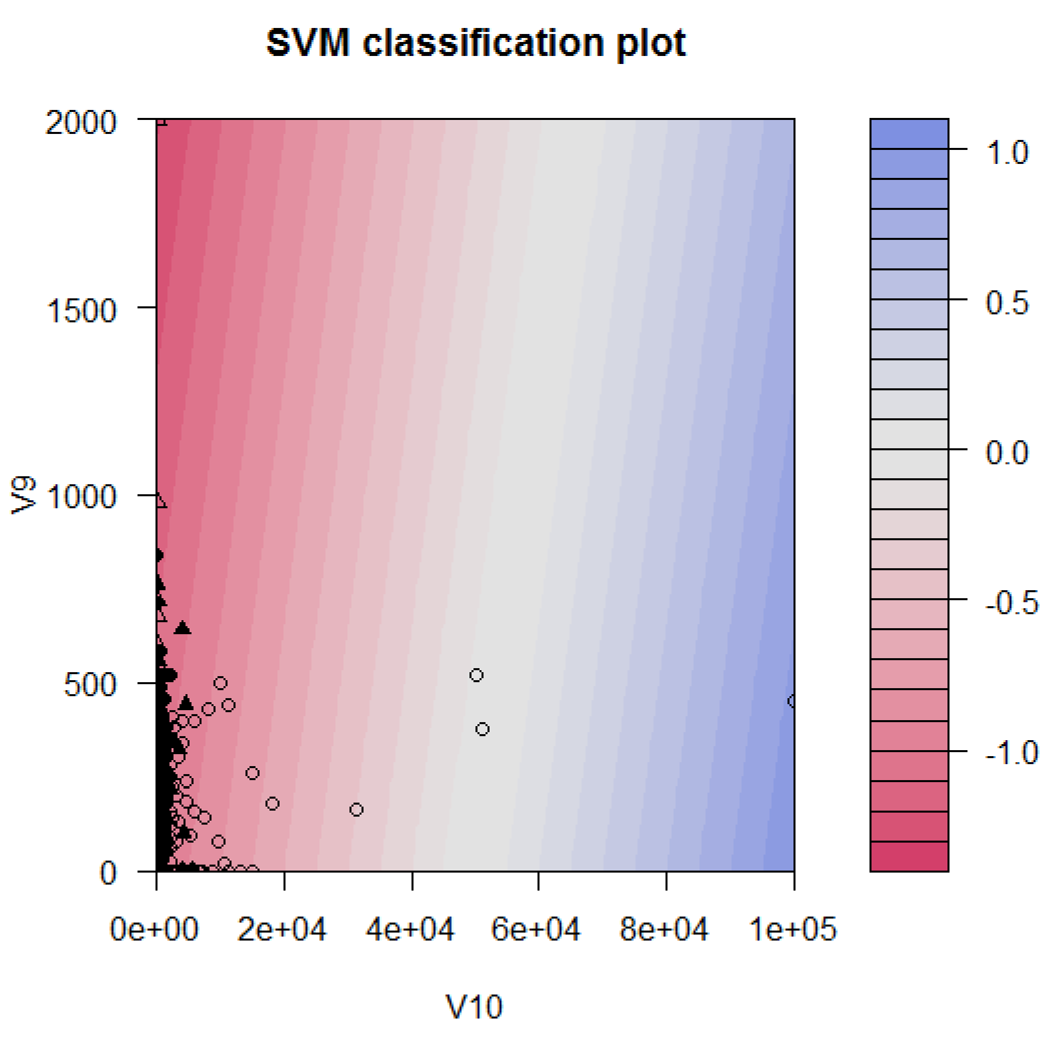
```
> source("gtassw1.R")
 Setting default kernel parameters
[1] "value of 'pred for c = 10,000'"
  [1] 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 1 0 0 0
[297] 0 0 0 0 0 0 1 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 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 0 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
> source("gtassw1.R")
 Setting default kernel parameters
[1] "value of 'pred for c = 5'"
  [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 1 0 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 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 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Conclusion:-

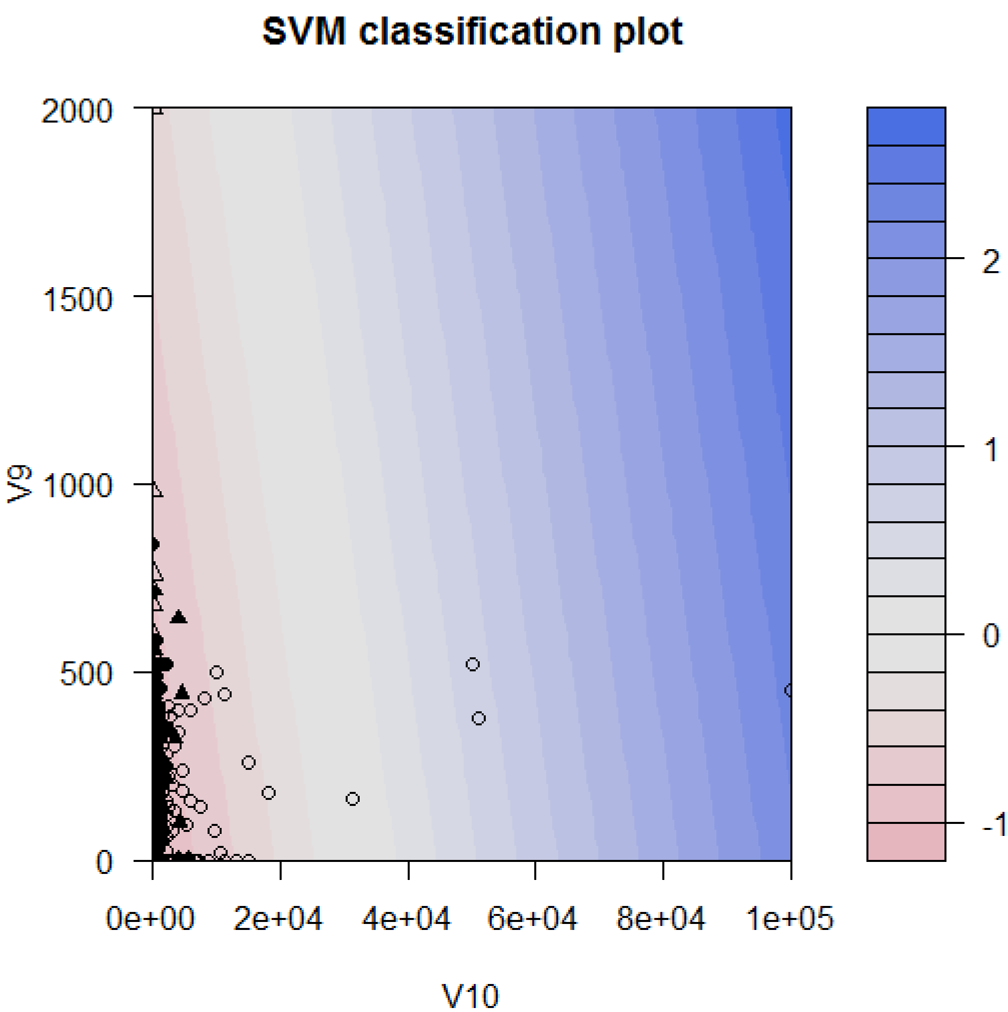
Decided to go for lowest Value C=5. Here is the reason.

See below the behavior of other variable with C=5 and c= 10,000. The below image helped decided to go with c=5

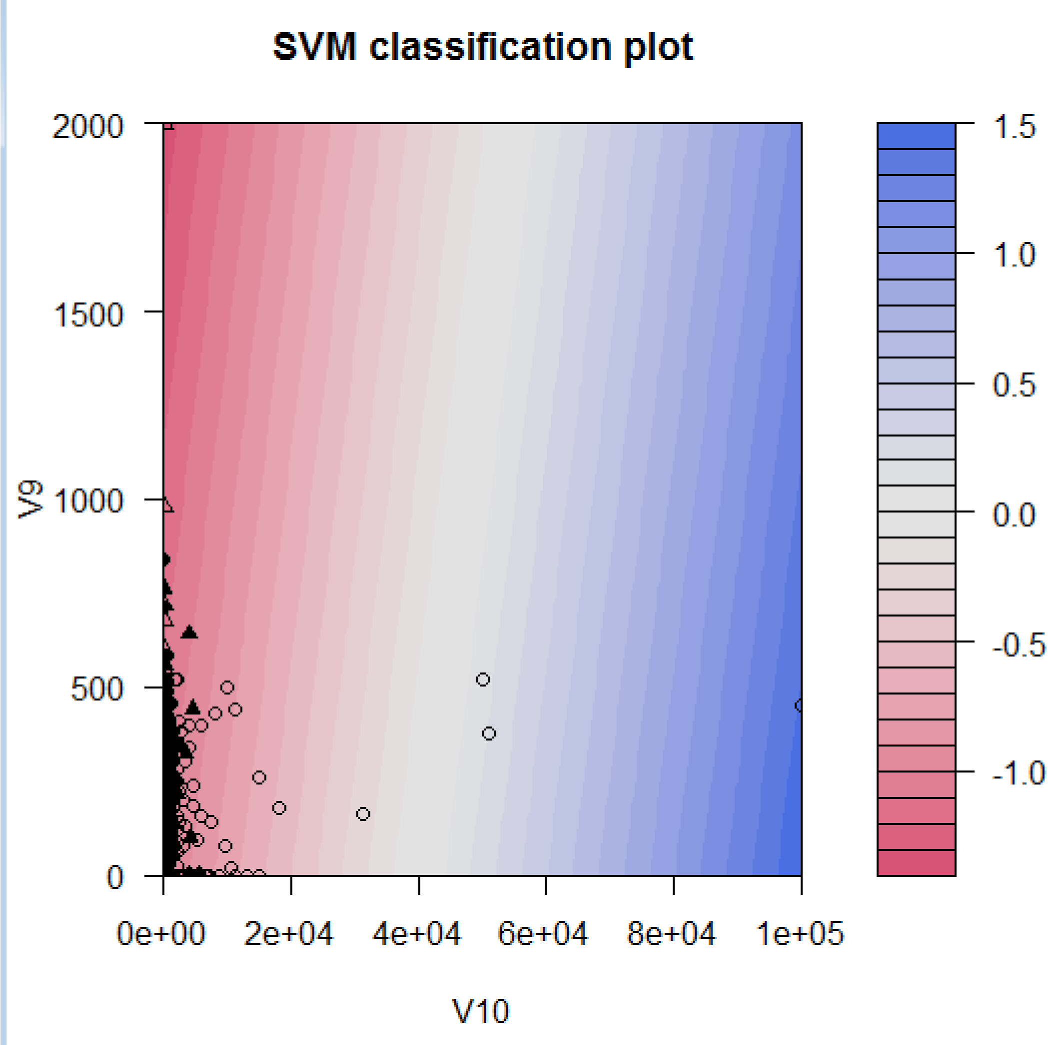
C=5

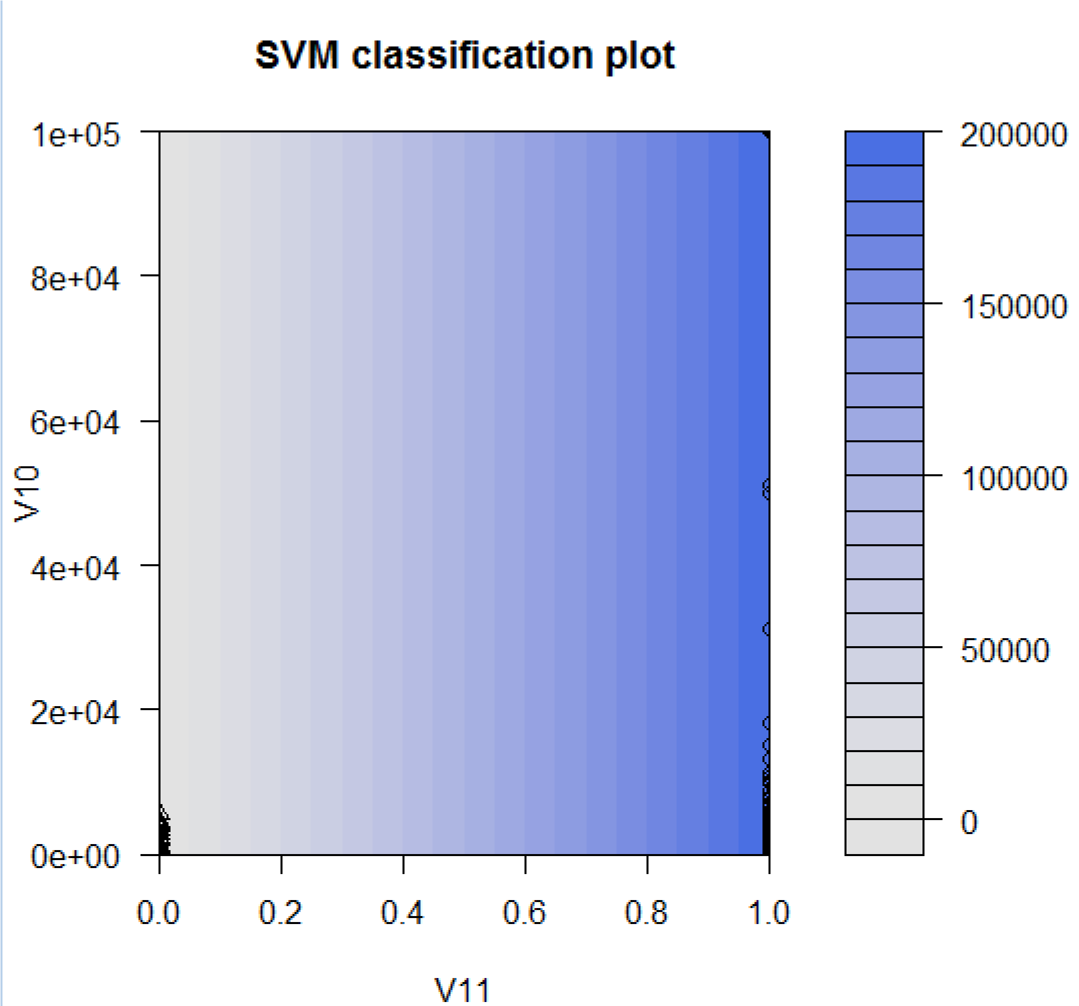

SVM classification plot
2000
1500
1000
500
0
V9
0e+00
2e+04
4e+04
6e+04
8e+04
1e+05
V10
1.0
0.5
0.0
-0.5
-1.0

C=10,000

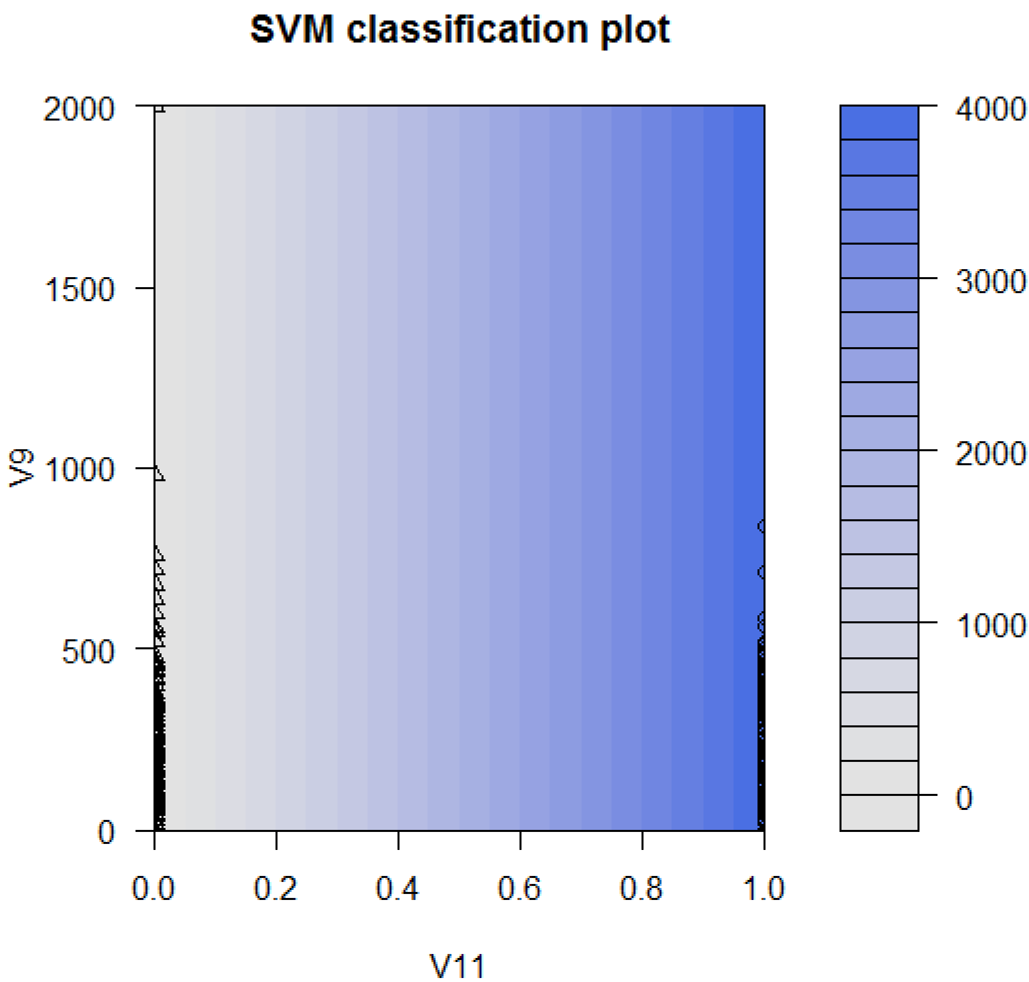


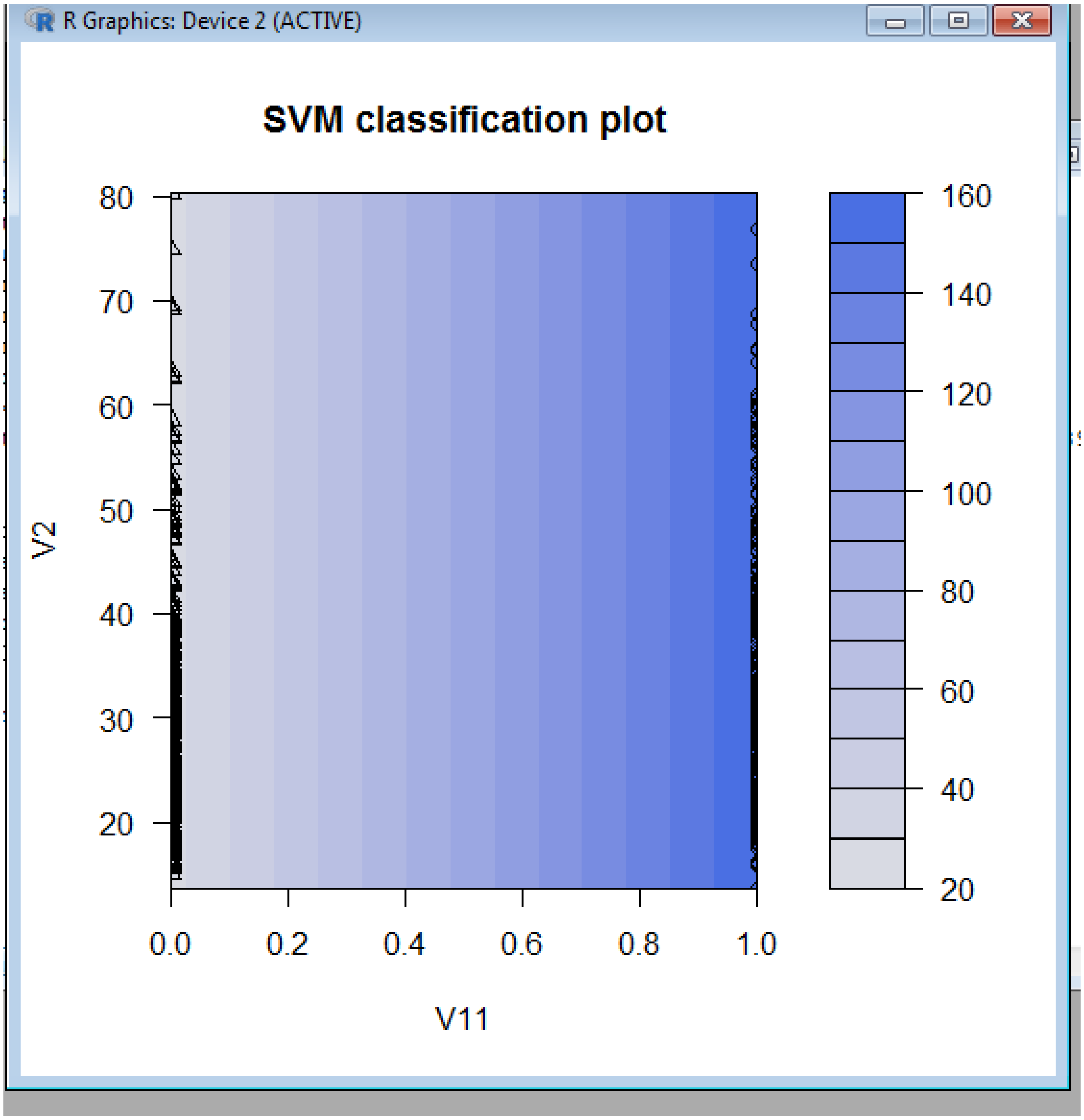
Quick reference of how C=100 looks. C=5 is much better


SVM classification plot
V9
V10
2000
1500
1000
500
0
0e+00
2e+04
4e+04
6e+04
8e+04
1e+05
1.5
1.0
0.5
0.0
-0.5
-1.0



SVM classification plot
1e+05
8e+04
6e+04
4e+04
2e+04
0e+00
V10
0.0
0.2
0.4
0.6
0.8
1.0
V11
200000
150000
100000
50000
0


SVM classification plot
V9
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0
V11
4000
3000
2000
1000
0



Assignment 2.2

KNN:- suggested Value 5.

Currently I ran for various values and picked K=5

See the details for K=5

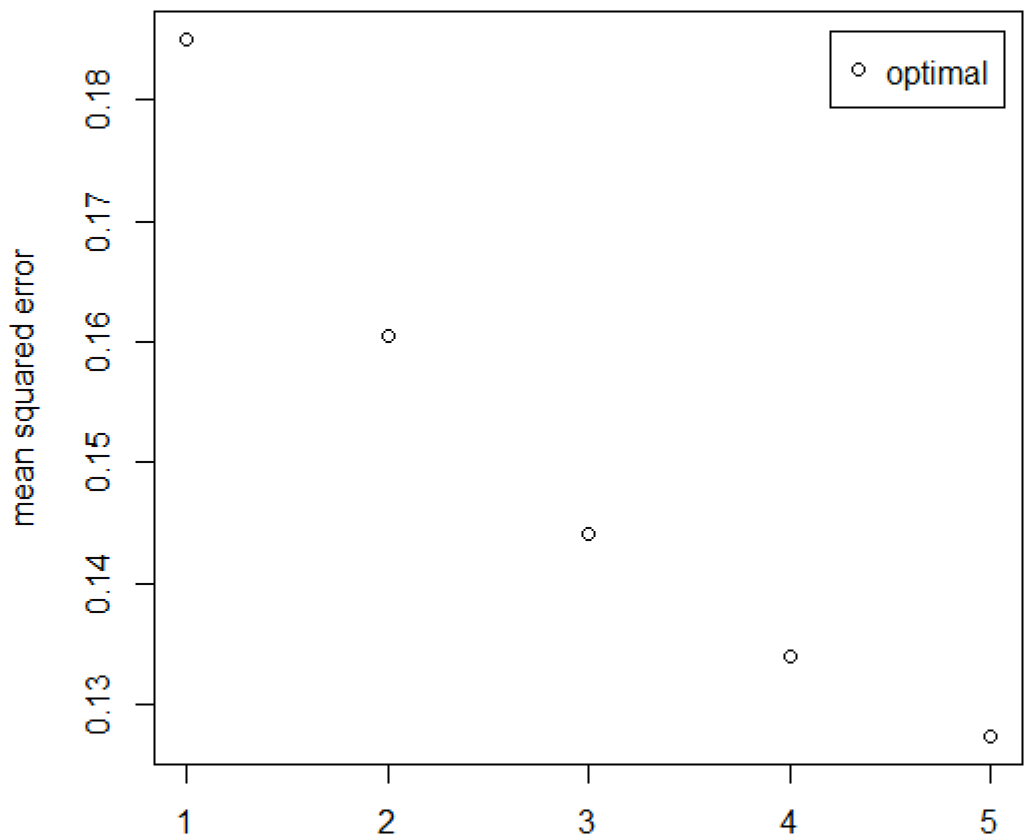
Accuracy and lowest MSE(Mean squared error) will be key to determine Kmax.

All data will be used together

C:\Users\USWXQ7Q\Documents\Work\personal\narayanan\DataSci\Week2\data1\gtassw21.R - R...

```
# KNN - question 2.2
# getting the base program ready
#include libraries
library(e1071)
library(kernlab)
library(kknn)
# Read the source using read.table function
data <- read.table(
  "https://d37djvu3ytnwxt.cloudfront.net/assets/courseware/v1/39b78ff5c5c28

#build Model(at present all data is together)
model <- train.kknn(V11 ~ .,data = data[,-11], kmax = 5,scale=TRUE)
print(model)
# Predictions
prediction <- predict(model, data[, -11])
print(prediction)
CM <- table(data[, 11], prediction)
print(CM)
#acurracy
accuracy <- (sum(diag(CM)))/sum(CM)
print(accuracy)
plot(model)
```



Data print

```
> source("gtassw21.R")

Call:
train.kknn(formula = V11 ~ ., data = data, kmax = 5, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1850153
Minimal mean squared error: 0.1274106
Best kernel: optimal
Best k: 5
  [1] 0.91676864 1.00000000 0.71491675 0.84353494 0.91676864 0.58386169
  [7] 1.00000000 0.91676864 0.76030358 0.47522034 0.47522034 0.58386169
 [13] 0.91676864 0.47522034 0.97459000 1.00000000 1.00000000 0.74032675
 [19] 0.76030358 0.84353494 1.00000000 0.73489359 0.91676864 1.00000000
 [25] 0.81812494 0.97459000 1.00000000 1.00000000 1.00000000 1.00000000
 [31] 0.74032675 1.00000000 1.00000000 0.91676864 1.00000000 1.00000000
 [37] 1.00000000 1.00000000 0.97459000 1.00000000 1.00000000 1.00000000
 [43] 0.97459000 1.00000000 1.00000000 1.00000000 1.00000000 1.00000000
 [49] 0.47522034 0.47522034 0.63168540 0.76030358 0.97459000 0.47522034
 [55] 0.97459000 0.73489359 0.76030358 0.65709540 0.47522034 1.00000000
 [61] 1.00000000 1.00000000 0.50063034 1.00000000 0.74032675 1.00000000
 [67] 1.00000000 1.00000000 1.00000000 0.74032675 0.23969642 0.28508325
 [73] 0.41613831 0.00000000 0.52477966 0.23969642 0.23969642 0.52477966
 [79] 0.26510641 0.44154831 0.26510641 0.02541000 0.44154831 0.49936966
 [85] 0.49936966 0.15646506 0.36831460 0.26510641 0.23969642 0.26510641
 [91] 0.41613831 0.36831460 0.08323136 0.28508325 0.18187506 0.18187506
 [97] 0.10864135 0.36831460 0.52477966 0.52477966 0.52477966 0.41613831
[103] 0.52477966 0.36831460 0.41613831 0.52477966 0.26510641 0.41613831
[109] 0.26510641 0.44154831 0.26510641 0.52477966 1.00000000 0.74032675
[115] 1.00000000 1.00000000 0.84353494 1.00000000 0.84353494 1.00000000
[121] 0.84353494 1.00000000 0.84353494 1.00000000 0.81812494 1.00000000
[127] 0.76030358 1.00000000 1.00000000 1.00000000 0.65709540 1.00000000
[133] 1.00000000 1.00000000 1.00000000 0.97459000 1.00000000 1.00000000
[139] 1.00000000 1.00000000 1.00000000 0.84353494 1.00000000 1.00000000
[145] 1.00000000 1.00000000 1.00000000 1.00000000 0.84353494 0.74032675
[151] 1.00000000 1.00000000 1.00000000 1.00000000 1.00000000 1.00000000
[157] 1.00000000 0.91676864 0.84353494 1.00000000 0.97459000 0.58386169
[163] 0.50063034 0.74032675 0.47522034 0.55845169 0.89135865 0.84353494
```

```
[637] 0.00000000 0.00000000 0.15646506 0.15646506 0.23969642 0.00000000
[643] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
[649] 0.25967325 0.00000000 0.00000000 0.00000000 0.02541000 0.00000000
   prediction
      0 0.0254099991601482 0.0832313551821549 0.108641354342303
  0 236                 13                 10                 5
  1   0                  0                  0                 0
   prediction
    0.156465060461443 0.181875059621592 0.239696415643598 0.259673248873949
  0                16                 5                 8                15
  1                 0                 0                 0                 0
   prediction
    0.265106414803747 0.285083248034097 0.342904604056104 0.368314603216252
  0                10                 4                 1                 6
  1                 0                 0                 0                 0
   prediction
    0.416138309335393 0.441548308495541 0.475220336322304 0.499369664517548
  0                 7                 6                 0                 2
  1                 0                 0                18                 0
   prediction
    0.500630335482452 0.524779663677696 0.558451691504459 0.583861690664607
  0                 0                14                 0                 0
  1                 3                 0                 5                 8
   prediction
    0.631685396783748 0.657095395943896 0.714916751965903 0.734893585196253
  0                 0                 0                 0                 0
  1                 3                 8                 3                 3
   prediction
    0.740326751126051 0.760303584356402 0.818124940378408 0.843534939538557
  0                 0                 0                 0                 0
  1                18                14                 7                22
   prediction
    0.891358645657697 0.916768644817845 0.974590000839852   1
  0                 0                 0                 0   0
  1                 4                20                19 141
```

Accuracy

[1] 0.3608563

Question 3:-

KNN

Determining best KNN before cross validation

Best KNN = 20. As KNN increased success rate increased. See below the program and evidence

```
# KNN - question 2.2
# getting the base program ready
#include libraries
library(e1071)
library(kernlab)
library(kknn)
# Read the source using read.table function
data <- read.table("https://d37djvu3ytnwxt.cloudfront.net/assets/courseware/v1/39b78ff5c5c28981f009b54831d81649/asset-v1:GTx+
gc <- data[,11]
gc.subset <- data[,-11]
t1 <- 1:450
t2<-  451:553
t3 <- 534:654
train <- data[t1,]
validate <- data[t2,]
test <- data[t3,]
train.gc <- train[,-11]
validate.gc <- validate[,-11]
test.gc <- test[,-11]

train.def <- train[,11]
validate.def <- validate[,11]
test.def <- test[,11]

library(class)
#build Model
#start to train
knn.1 <-  knn(train.gc, test.gc, train.def, k=1)
knn.5 <-  knn(train.gc, test.gc, train.def, k=5)
knn.20 <- knn(train.gc, test.gc, train.def, k=20)

#calculate proportion of correct classification
v01 <- 100 * sum(test.def == knn.1)/100  # For knn = 1
print ("for KNN=1")
print (v01)

v05 <- 100 * sum(test.def == knn.5)/100  # For knn = 5
print ("for KNN=5")
print (v05)
```

```
v05 <- 100 * sum(test.def == knn.5)/100  # For knn = 5
print ("for KNN=5")
print (v05)

v20 <- 100 * sum(test.def == knn.20)/100  # For knn = 20
print ("for KNN=20")
print (v20)

#success rate
table(knn.1 ,test.def)
table(knn.5 ,test.def)
table(knn.20 ,test.def)
```

Here is the success rate

```
> source("gtassw31.R")
[1] "for KNN=1"
[1] 73
[1] "for KNN=5"
[1] 78
[1] "for KNN=20"
[1] 76
> table(knn.1 ,test.def)
     test.def
knn.1  0  1
    0 53 20
    1 28 20
> table(knn.5 ,test.def)
     test.def
knn.5  0  1
    0 57 19
    1 24 21
> table(knn.20 ,test.def)
      test.def
knn.20  0  1
     0 51 15
     1 30 25
> 
```

Program for cross validation

```
# Matrix to store predictions
p.cv <- matrix(NA, n.train, length(klist))
# Prepare the folds
s <- split(sample(n.train),rep(1:nfolds,length=n.train))
# Cross-validation
for (i in seq(nfolds)) {
            p.cv[s[[i]],] <- knn(klist,train.gc[-s[[i]],,drop=FALSE], train.def[-s[[i]],drop=FALSE], train.gc[s[[i]],,drop=FALSE])
      }
```