Homework 1 - Week 1 - Karthick Krishna

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 am an Analyst at Mindtree Limited. Currently, I work on Marketing Analytics and Ad Analytics.I work for one of the world's largest Search Engine Platform provider on the Advertiser data and User level data.I use Python for analysis and used the Scikit Learn package in Python. Recently I worked on segregating the Customers(also known as Advertisers) into five categories based on a lot of different KPI's. I performed Regression Analysis on the following predictors to identify and classify the type of Advertiser based on lots of factors.

Few of those KPI's(predictors) are as follows,

1. The Impression gained by the advertiser using the platform
2. Click gained by the advertiser using the platform
3. Spend/Revenue of the Advertiser on the Platform
4. Cost per Click (CPC)
5. Click through Rate (CTR)

Question 2.2 (Question 1 and 2)

On an overall level "rbfdot" Kernel has performed extremly well with an accuracy of "98.471" and C(lambda) of 1000

We see a different kind of output when we split the data (70%-train and 30%-test) and it's mentioned after this piece

My approach to the given problem:

In this problem, I've used four different Kernel function types and seven different Cost function values. An iterative process of first selecting a Kernel function type followed by selecting a Cost function value, later ksvm is applied and weights, bias values, and Accuracy are calculated for a specific combination of Kernel and Cost function. This process goes on until all the combination is completed. In the end, I create a Dataframe and place all the values in it ordering on Accuracy.

Equation based on the maximum accuracy produced by the model

y = (-52.963 x A1) + (-20.619 x A2) + (-55.471 x A3) + (123.982 x A8) + (86.393 x A9) + (-93.044 x A10) + (99.034 x A11) + (-66.936 x A12) + (-75.808 x A14) + (100.064 x A15) - 0.729

My understanding of the output

The accuracy looks extremely well. Seriously, if I wasn't into Data Science or Data Analytics I'll be jumping in joy and dancing around for the accuracy that I've got. Unfortunately, I'm aware of a concept called ***Overfitting*** so I cannot enjoy the result that has been generated. The model calculated predicted accuracy on data was trained in, thus fit that data too well. Thus, we cannot be happy with this output.

Post this code I've performed the same but by splitting the data into Train and Test (70% and 30%). Kindly, view that to understand how Overfitting is handled

In [31]:
```
# Calling the needed packages
options(scipen = 999)
library(kernlab)
library(e1071)
library(data.table)
library(dplyr)

full_data <- read.table("credit_card_data-headers.txt", header = TRUE)
different_cost_values <- c(0.001,0.01,0.1,1,10,100,1000)
different_kernel_list <- c("vanilladot", "rbfdot", "anovadot", "polydot")
accuracy_value <- c()
c_value <- c()
kernel_value <- c()
weights_value <- c()
bias_value <- c()

for (i in 1:length(different_kernel_list)){
#     print(i)
    for (j in 1:length(different_cost_values)){
#     print(j)
        model <- ksvm(as.matrix(full_data[,1:10]), as.factor(full_data[,11]),
                      type= "C-svc", kernel = different_kernel_list[i],
                      C = different_cost_values[j], scaled=TRUE)
#         print(model)
        a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
#         print(a)
        a0 <- model@b
#         print(a0)
        pred <- predict(model,full_data[,1:10])
#         print(pred)
        accur <- sum(pred == full_data[,11]) / nrow(full_data)
#         print(max(accur))
        accuracy_value <- c(accuracy_value,accur)
#         print(as.data.frame(accuracy_value))
        c_value <- c(c_value, different_cost_values[j])
        kernel_value <- c(kernel_value,different_kernel_list[i])
        weights_value <- c(weights_value,a)  # the length of weights is 280 and number of unique weig
        bias_value <- c(bias_value,a0)
    }
}

weights1 <- as.data.frame(matrix(weights_value, 28, 10, byrow = T))
df <- data.frame(kernel_value, c_value, round((accuracy_value*100),3), round(bias_value,3))
names(df) <- c("Kernel","Cost Value","Accuracy", "a0")
df1 <- merge(df, round(weights1, 3), by=0)
data_to_drops <- c("Row.names")
df2 <- df1[order(df1["Accuracy"], decreasing = TRUE), !(names(df1) %in% data_to_drops)]
df2
```

```
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
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
```

	Kernel	Cost Value	Accuracy	a0	V1	V2	V3	V4	V5	V6	V7	V8	V9	
6	rbfdot	1000.000	98.471	-0.729	-52.963	-20.619	-55.471	123.982	86.393	-93.044	99.034	-66.936	-75.808	10

	Kernel	Cost Value	Accuracy	a0	V1	V2	V3	V4	V5	V6	V7	V8	V9	
5	rbfdot	100.000	95.413	-0.728	-18.784	-35.719	-8.228	55.509	51.183	-26.201	19.764	-24.521	-57.518	5
4	rbfdot	10.000	91.896	-0.439	-3.051	-17.776	3.521	27.046	32.357	-12.241	16.593	-9.436	-32.945	3
13	anovadot	100.000	90.673	-1.174	0.019	-22.498	-28.051	-2.358	2.536	-1.122	-3.115	-0.046	-16.445	1
14	anovadot	1000.000	90.673	-8.831	0.133	-29.447	-69.102	-21.960	2.657	-1.687	-5.867	-0.011	-42.511	
11	anovadot	10.000	87.309	-0.029	0.011	-8.131	-10.580	3.779	2.222	-0.359	4.490	0.002	-7.895	1
3	rbfdot	1.000	87.156	-0.416	0.391	-1.944	4.341	14.824	32.454	-7.266	19.416	-5.167	-16.303	2
10	anovadot	1.000	86.391	-0.375	0.002	-1.539	-0.900	0.698	2.050	-0.027	0.768	0.001	-2.719	1
12	vanilladot	0.010	86.391	-0.082	0.000	-0.001	0.001	0.007	0.992	-0.004	0.007	-0.001	-0.002	
16	polydot	0.010	86.391	-0.082	0.000	-0.001	0.001	0.007	0.992	-0.004	0.007	-0.001	-0.002	
17	polydot	0.100	86.391	-0.082	-0.001	-0.001	-0.001	0.003	1.004	-0.003	0.000	0.000	-0.001	
18	polydot	1.000	86.391	-0.081	-0.001	-0.001	-0.002	0.003	1.005	-0.003	0.000	-0.001	-0.001	
19	polydot	10.000	86.391	-0.082	-0.001	-0.001	-0.002	0.003	1.005	-0.003	0.000	-0.001	-0.001	
20	polydot	100.000	86.391	-0.082	-0.001	-0.001	-0.002	0.003	1.005	-0.003	0.000	-0.001	-0.001	
22	vanilladot	0.100	86.391	-0.082	-0.001	-0.001	-0.002	0.003	1.004	-0.003	0.000	0.000	-0.001	
23	vanilladot	1.000	86.391	-0.081	-0.001	-0.001	-0.002	0.003	1.005	-0.003	0.000	-0.001	-0.001	
24	vanilladot	10.000	86.391	-0.082	-0.001	-0.001	-0.002	0.003	1.005	-0.003	0.000	0.000	-0.001	
25	vanilladot	100.000	86.391	-0.082	-0.001	-0.001	-0.002	0.003	1.005	-0.003	0.000	-0.001	-0.001	
8	anovadot	0.010	86.239	-0.150	-0.008	0.024	0.025	0.223	1.811	-0.094	0.297	0.007	-0.164	
9	anovadot	0.100	86.239	-0.039	0.000	-0.155	-0.085	0.071	2.040	-0.002	0.080	0.000	-0.275	
21	polydot	1000.000	86.239	-0.070	0.000	-0.001	0.000	0.000	0.998	0.000	0.001	0.000	0.000	
26	vanilladot	1000.000	86.239	-0.070	0.000	0.001	0.001	0.001	0.999	-0.001	0.001	-0.001	0.001	
2	rbfdot	0.100	85.933	-0.498	0.439	2.596	2.904	7.351	18.092	-4.184	7.105	-0.867	-2.683	
1	vanilladot	0.001	83.792	0.223	-0.002	0.032	0.047	0.111	0.375	-0.202	0.170	-0.005	-0.025	
15	polydot	0.001	83.792	0.223	-0.002	0.032	0.047	0.111	0.375	-0.202	0.170	-0.005	-0.025	
7	anovadot	0.001	58.869	0.408	-0.005	0.073	0.090	0.179	0.413	-0.240	0.233	-0.008	-0.048	
28	rbfdot	0.010	56.728	0.379	0.194	0.860	0.963	1.831	4.127	-2.396	2.332	-0.100	-0.575	
27	rbfdot	0.001	54.740	0.939	0.019	0.086	0.098	0.183	0.413	-0.240	0.233	-0.010	-0.057	

Question 2.2 (Question 1 and 2) : An Alternate Method

Reproducing the above KSVM function with a slight twist of considering 70-30 Train-Test Split for the best cost value for each of the above kernel function methods.

Using the alternate mrthod "rbfdot" Kernel has performed extremly well with an accuracy of "83.673" and C(lambda) of 1.

Equation based on the maximum accuracy produced by the model

```
y = (-0.06 x A1) + (0.819 x A2) + (3.748 x A3) + (13.569 x A8) + (26.005 x A9) + (-2.745 x A
10) + (9.356 x A11) +         (-2.473 x A12) + (-10.654 x A14) + (16.728 x A15) - 0.459
```

My understanding of the output

Although the accuracy is less compared to the output of the previous method I'm happy that I've overcome the **Overfitting** issue by splitting the data into Train and Test.

Used the following links to understand train-test split using R

1. https://cran.r-project.org/web/packages/dataPreparation/vignettes/train_test_prep.html
2. https://www.listendata.com/2015/02/splitting-data-into-training-and-test.html
3. https://stackoverflow.com/questions/17200114/how-to-split-data-into-training-testing-sets-using-sample-function

In [40]:
```
# Calling the needed packages
options(scipen = 999)
library(kernlab)
library(e1071)
library(data.table)
library(dplyr)
library(caTools)
set.seed(101)

full_data <- read.table("credit_card_data-headers.txt", header = TRUE)

sample = sample.split(full_data$R1, SplitRatio = 0.70)
X_data_train <- subset(data[,1:10], sample == TRUE)
y_data_train <- subset(data[,11], sample == TRUE)
X_data_test <- subset(data[,1:10], sample == FALSE)
y_data_test <- subset(data[,11], sample == FALSE)

different_cost_values <- c(0.001,0.01,0.1,1,10,100,1000)
different_kernel_list <- c("vanilladot", "rbfdot", "anovadot", "polydot")
accuracy_value <- c()
c_value <- c()
kernel_value <- c()
weights_value <- c()
bias_value <- c()

for (i in 1:length(different_kernel_list)){
#     print(i)
    for (j in 1:length(different_cost_values)){
#     print(j)
        model <- ksvm(as.matrix(X_data_train), as.factor(y_data_train),
                      type= "C-svc", kernel = different_kernel_list[i],
                      C = different_cost_values[j], scaled=TRUE)
#         print(model)
        a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
#         print(a)
        a0 <- model@b
#         print(a0)
        pred <- predict(model,X_data_test)
#         print(pred)
        accur <- sum(pred == y_data_test) / length(y_data_test)
#         print(max(accur))
        accuracy_value <- c(accuracy_value,accur)
#         print(as.data.frame(accuracy_value))
        c_value <- c(c_value, different_cost_values[j])
        kernel_value <- c(kernel_value,different_kernel_list[i])
        weights_value <- c(weights_value,a) # the length of weights is 280 and number of unique weigh
        bias_value <- c(bias_value,a0)
    }
}

weights1 <- as.data.frame(matrix(weights_value, 28, 10, byrow = T))
df <- data.frame(kernel_value, c_value, round((accuracy_value*100),3), round(bias_value,3))
names(df) <- c("Kernel","Cost Value","Accuracy", "a0")
df1 <- merge(df, round(weights1, 3), by=0)
data_to_drops <- c("Row.names")
df2 <- df1[order(df1["Accuracy"], decreasing = TRUE), !(names(df1) %in% data_to_drops)]
df2
```

```
 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
 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
```

	Kernel	Cost Value	Accuracy	a0	V1	V2	V3	V4	V5	V6	V7	V8	V9	
3	rbfdot	1.000	83.673	-0.459	-0.060	0.819	3.748	13.569	26.005	-2.745	9.356	-2.473	-10.654	16
9	anovadot	0.100	83.673	-0.070	-0.003	-0.206	-0.084	0.090	2.039	-0.011	0.020	0.000	0.089	1
10	anovadot	1.000	83.673	-0.851	-0.044	-2.101	-0.718	0.986	2.182	-0.194	-0.322	0.010	1.407	18
12	vanilladot	0.010	83.673	-0.083	-0.002	-0.006	0.011	0.035	0.962	-0.011	0.036	-0.002	-0.005	0
16	polydot	0.010	83.673	-0.083	-0.002	-0.006	0.011	0.035	0.962	-0.011	0.036	-0.002	-0.005	0
17	polydot	0.100	83.673	-0.089	-0.002	-0.002	-0.001	0.007	1.003	-0.003	0.002	-0.001	0.000	0
18	polydot	1.000	83.673	-0.088	-0.002	-0.003	-0.002	0.010	1.006	-0.002	0.002	-0.002	0.000	0
19	polydot	10.000	83.673	-0.088	-0.002	-0.003	-0.001	0.008	1.006	-0.002	0.002	-0.002	-0.001	0
20	polydot	100.000	83.673	-0.087	-0.002	-0.003	-0.002	0.009	1.007	-0.002	0.002	-0.002	-0.001	0
21	polydot	1000.000	83.673	-0.078	0.000	0.000	0.000	0.001	0.998	-0.001	0.000	0.000	0.000	0
22	vanilladot	0.100	83.673	-0.089	-0.002	-0.002	-0.001	0.008	1.003	-0.003	0.002	-0.001	0.000	0
23	vanilladot	1.000	83.673	-0.087	-0.003	-0.003	-0.001	0.009	1.006	-0.002	0.003	-0.002	0.000	0
24	vanilladot	10.000	83.673	-0.088	-0.003	-0.003	-0.001	0.008	1.006	-0.002	0.002	-0.002	-0.001	0
25	vanilladot	100.000	83.673	-0.088	-0.002	-0.003	-0.002	0.009	1.006	-0.002	0.002	-0.002	-0.001	0
26	vanilladot	1000.000	83.673	-0.078	0.000	0.000	0.001	0.000	0.999	0.000	0.000	0.000	0.000	0
2	rbfdot	0.100	83.163	-0.495	0.247	2.651	3.040	7.128	15.311	-3.572	5.823	-0.530	-1.762	5
11	anovadot	10.000	81.633	-1.511	0.048	-6.671	-9.844	2.710	2.872	-0.803	0.373	-0.013	5.759	35
13	anovadot	100.000	81.633	-2.651	0.166	-14.771	-37.191	3.119	2.819	-1.233	1.737	0.074	3.647	56
6	rbfdot	1000.000	81.122	-1.202	-31.999	-102.940	-49.720	72.072	122.958	-37.460	46.517	-58.618	-101.104	34
5	rbfdot	100.000	80.612	-0.374	-14.415	-37.871	10.368	65.025	54.506	-23.000	18.287	-26.748	-56.384	33
4	rbfdot	10.000	80.102	-0.485	-5.983	-9.964	7.596	22.676	27.682	-5.694	3.607	-9.688	-18.861	19
14	anovadot	1000.000	80.102	-4.691	0.151	-36.112	-60.090	-10.436	3.199	-1.785	-2.353	0.057	-2.509	17
1	vanilladot	0.001	76.531	0.336	-0.002	0.038	0.061	0.111	0.283	-0.155	0.159	-0.015	-0.023	0
15	polydot	0.001	76.531	0.336	-0.002	0.038	0.061	0.111	0.283	-0.155	0.159	-0.015	-0.023	0
7	anovadot	0.001	54.592	0.541	-0.004	0.059	0.086	0.149	0.299	-0.171	0.187	-0.016	-0.032	0
27	rbfdot	0.001	54.592	0.952	0.017	0.066	0.089	0.152	0.299	-0.171	0.187	-0.022	-0.039	0
28	rbfdot	0.010	54.592	0.512	0.177	0.674	0.895	1.519	2.986	-1.709	1.872	-0.217	-0.384	0
8	anovadot	0.010	45.408	-0.235	-0.014	0.023	0.082	0.342	1.511	-0.230	0.344	-0.038	-0.091	0

Question 2.2 (Question 3)

I'll approach the same usecase but this time I'll be using K Nearest Neighbor(KNN)

Stepwise explanation to the problem approach:

For this problem, I've used different k values to check which k value suites. This is passes on different kernel function. The knn method looks at all the data except for the i-th row of data. I've looped through different values of k to perform the same. For each kernel and each k value, I'm appending the model accuracy, k value and name of the kernel to the empty vectors that I've defined. After this step, I am joining these vectors to a dataframe and sorted (descending) by accuracy.

Using this approach, 12 and 15 seem to be good values of k, and the model accuracy for both the values of k is 85.32110 %. Since, we are using KNN I would prefer to choose k = 15 an odd number for k.

In [48]:
```
library(kernlab)
library(kknn)

full_data <- read.table("credit_card_data-headers.txt", header = TRUE)

k_list <- c(1:101)   # Checking for a correct "K-value" so I'm checking for values from 1 to 101
different_kernel_list <- c("rectangular" , "triangular", "inv", "gaussian", "rank", "optimal") # Chec
accuracy_value <- c()
k_value <- c()
kernel_value <- c()
pred <- rep(0, nrow(full_data))


for (m in 1:length(different_kernel_list)){
    for (i in 1:length(k_list)) {
        for (j in 1:nrow(full_data)) {
            kknn_model <- kknn(R1~., full_data[-j,1:11], full_data[j,1:11],
                               k = k_list[i], distance = 2, scale = TRUE,
                               kernel = different_kernel_list[m])
#             print(kknn_model)
            pred[j] <- round(fitted(kknn_model))
        }
        accur = sum(pred == full_data[,11])/length(full_data[,11])
#         prin(accur)
        accuracy_value <- c(accuracy_value, accur)
#         print(accuracy_value)
        k_value <- c(k_value, k_list[i])
#         print(k_value)
        kernel_value <- c(kernel_value, different_kernel_list[m])
#         print(kernel_value)
    }
}


df <- data.frame(kernel_value, k_value, accuracy_value*100)
names(df) <- c("Kernel","K Value","Model Accuracy")
df3 <- df[order(df["Model Accuracy"], decreasing = TRUE), ]
df3
```

	Kernel	K Value	Model Accuracy
517	optimal	12	85.32110
520	optimal	15	85.32110
111	triangular	10	85.16820
510	optimal	5	85.16820
516	optimal	11	85.16820
518	optimal	13	85.16820
519	optimal	14	85.16820
521	optimal	16	85.16820
522	optimal	17	85.16820
523	optimal	18	85.16820
22	rectangular	22	85.01529
44	rectangular	44	85.01529
112	triangular	11	85.01529
113	triangular	12	85.01529
115	triangular	14	85.01529
244	inv	42	85.01529
311	gaussian	8	85.01529
426	rank	22	85.01529
448	rank	44	85.01529
515	optimal	10	85.01529
524	optimal	19	85.01529
525	optimal	20	85.01529

	Kernel	K Value	Model Accuracy
48	rectangular	48	84.86239
114	triangular	13	84.86239
116	triangular	15	84.86239
117	triangular	16	84.86239
118	triangular	17	84.86239
208	inv	6	84.86239
245	inv	43	84.86239
452	rank	48	84.86239
...	...	...	...
331	gaussian	28	82.72171
4	rectangular	4	82.56881
15	rectangular	15	82.56881
329	gaussian	26	82.56881
408	rank	4	82.56881
419	rank	15	82.56881
3	rectangular	3	82.26300
17	rectangular	17	82.26300
19	rectangular	19	82.26300
205	inv	3	82.26300
306	gaussian	3	82.26300
407	rank	3	82.26300
421	rank	17	82.26300
423	rank	19	82.26300
105	triangular	4	81.95719
1	rectangular	1	81.49847
102	triangular	1	81.49847
103	triangular	2	81.49847
203	inv	1	81.49847
204	inv	2	81.49847
304	gaussian	1	81.49847
305	gaussian	2	81.49847
405	rank	1	81.49847
506	optimal	1	81.49847
507	optimal	2	81.49847
508	optimal	3	81.49847
509	optimal	4	81.49847
104	triangular	3	80.88685
2	rectangular	2	78.59327
406	rank	2	78.59327

Question 3.1 (a)

Problem Statement: using cross-validation (do this for the k-nearest-neighbors model; SVM is optional)

Stepwise explanation to the problem approach:

For this I train using train.knn approch to find the best k value and Kernel fuction.

train.kknn - Training of kknn method via leave-one-out crossvalidation. **From the train.kknn approach, attributes(model) shows that the Accuracy is 100%, Minimal mean squared error is 10.61155 % for k = 22 and for the "inv" Kernel.**

cv.kknn - k-fold cross-validation, where k is the number number of data points (I'm using 15). Cross validates 1 row with k-1 rows. **From the cv.kknn approach, attributes(model) shows that the Model accuracy is 86.08563 % for k = 39 and for the "optimal" Kernel.**

In [67]:
```
library(kernlab)
library(kknn)
library(data.table)
library(e1071)
library(caTools)
require(caTools)
set.seed(101)

full_data <- read.table("credit_card_data-headers.txt", header = TRUE)
kernels = c("rectangular", "triangular", "epanechnikov", "biweight", "triweight", "cos", "inv", "gaus

cat("-------------Start of training kknn method-------------\n")

train_model <- train.kknn(R1~., full_data, kmax = 101, distance = 2, kernel = kernels, scale = TRUE)
train_model
best_train_kernel <- train_model$best.parameters$kernel # We are finding the best kernel factor
best_train_kvalue <- train_model$best.parameters$k # We are finding the best kvalue for the model

kknn_train_model <- train.kknn(R1~., full_data, ks=best_train_kvalue, distance = 2,
                               kernel = best_train_kernel, scale = TRUE)
# print(kknn_train_model)
pred <- round(predict(kknn_train_model, full_data[,1:10]))
# print(pred)
train_model_accuracy <- sum(pred == full_data[,11])/length(full_data[,11])
# print(train_model_accuracy)
cat("\nBest Kernel is:",best_train_kernel,", Best K Value is:",best_train_kvalue,", and the accuracy
    train_model_accuracy*100,"%\n")



cat("\n-------------Start of cross-validation for kknn method-------------\n")

kvals <- c(1:101)
model_accuracy <- c()

for (i in 1:length(kernels)){
    for (j in 1:length(kvals)){
        cv_model <- cv.kknn(R1~., data= full_data, kcv = 15, scale = TRUE,
                            kernel = kernels[i], k = kvals[j])
        cv_model <- data.table(cv_model[[1]])
#         print(cv_model)
        cv_model_accuracy <- sum(round(cv_model$yhat) == full_data$R1)/length(full_data$R1)
#         print(cv_model_accuracy)
        model_accuracy <- c(model_accuracy, cv_model_accuracy)
#         print(model_accuracy)
    }
    cat("\nFor", kernels[i], "kernel, model accuracy is:", max(model_accuracy)*100,
        "% and corresponding k-value is:", kvals[which.max(model_accuracy[1:length(kvals)])])
}
```

```
-------------Start of training kknn method-------------


Call:
train.kknn(formula = R1 ~ ., data = full_data, kmax = 101, distance = 2,     kernel = kernels, scale
= TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1850153
Minimal mean squared error: 0.1061155
Best kernel: inv
Best k: 22


Best Kernel is: inv , Best K Value is: 22 , and the accuracy is: 100 %

-------------End of training the kknn method-------------

-------------Start of cross-validation for kknn method-------------

For rectangular kernel, model accuracy is: 84.86239 % and corresponding k-value is: 39
For triangular kernel, model accuracy is: 85.3211 % and corresponding k-value is: 39
For epanechnikov kernel, model accuracy is: 85.3211 % and corresponding k-value is: 39
For biweight kernel, model accuracy is: 85.47401 % and corresponding k-value is: 39
```

```
For triweight kernel, model accuracy is: 85.93272 % and corresponding k-value is: 39
For cos kernel, model accuracy is: 86.08563 % and corresponding k-value is: 39
For inv kernel, model accuracy is: 86.08563 % and corresponding k-value is: 39
For gaussian kernel, model accuracy is: 86.08563 % and corresponding k-value is: 39
For optimal kernel, model accuracy is: 86.08563 % and corresponding k-value is: 39

-------------End of the cross-validation for kknn method-------------
```

Question 3.1 (b)

For this question, I split the data into 3 parts into train-valid-test (70-15-15).

My understanding from the output:

The TEST ACCURACY is maximum for "85.75740%" for "vanilladot" Kernel function and a Cost value of "0.01" which is higher than the TRAIN ACCURACY. So, now when I see the output of **"anovadot" Kernel function** with a **Cost value of "1000"** has a better output of **"86.74699" TEST ACCURACY that is less than the TRAIN ACCURACY of "91.90372"** as the final output.

In [88]:

```
library(kernlab)
library(e1071)
library(caTools)
require(caTools)
set.seed(101)

full_data <- read.table("credit_card_data-headers.txt", header = TRUE)
kernel_list = c("vanilladot", "rbfdot", "anovadot", "polydot")

train_test_split <- c(0.7,0.15,0.15)           #(Train-Validation-Test) split

cost_value <- c()
kernel_val <- c()
train_frac_list <- c()
validate_frac_list <- c()
test_frac_list <- c()
Train_Accuracy <- c()
Validation_Accuracy <- c()
Test_Accuracy <- c()

for (k1 in 1:length(kernel_list)){

#Spliting the Data into train, validation and test
    Training_split <- train_test_split[1]
    Validation_split <- train_test_split[2]
    Test_split <- train_test_split[3]

    Training_SampleSize <- floor(Training_split * nrow(full_data))
#     print(Training_SampleSize)
    Validation_SampleSize <- floor(Validation_split * nrow(full_data))
#     print(Validation_SampleSize)
    Test_SampleSize <- floor(Test_split * nrow(full_data))
#     print(Test_SampleSize)

    Training_index <- sort(sample(seq_len(nrow(full_data)), size=Training_SampleSize))
    NotTraining_index <- setdiff(seq_len(nrow(full_data)), Training_index)
    Validation_index <- sort(sample(indicesNotTraining, size=Validation_SampleSize))
    Test_index <- setdiff(NotTraining_index, Validation_index)

# Training Data
    dfTraining <- full_data[Training_index, ]
    x_dfTraining <- dfTraining[,1:10]
    y_dfTraining <- dfTraining[,11]

# Validation Data
    dfValidation <- full_data[Validation_index, ]
    x_dfValidation <- dfValidation[,1:10]
    y_dfValidation <- dfValidation[,11]

# Test Data
    dfTest <- full_data[Test_index, ]
    x_dfTest <- dfTest[,1:10]
    y_dfTest <- dfTest[,11]


    cost_values <- c(0.001,0.01,0.1,1,10,100,1000)
    train_accuracy <- c()
    prediction_train <- c()
    train_model <- c()

    for (i in 1:length(cost_values)){
        ksvm_model <- ksvm(as.matrix(x_dfTraining), as.factor(y_dfTraining),
                           type = "C-svc", kernel = kernel_list[k1],
                           C = cost_values[i], scaled = TRUE)
        a <- colSums(ksvm_model@xmatrix[[1]]*ksvm_model@coef[[1]])
        a0 <- ksvm_model@b
        accuracy <- sum(predict(ksvm_model,x_dfTraining) == y_dfTraining)/length(y_dfTraining)
        train_accuracy <- c(train_accuracy,accuracy)
        train_model <- c(train_model, ksvm_model)
        prediction_train <- c(prediction_train, predict(ksvm_model,x_dfTraining))
    }

    model <- train_model[which.max(train_accuracy[1:length(cost_values)])]
    C_val = cost_values[which.max(train_accuracy[1:length(cost_values)])]
    train_accuracy <- max(train_accuracy[1:length(cost_values)])
    validation_predict <- predict(model[[1]],x_dfValidation)
```

```
    validation_accuracy <- sum(validation_predict == y_dfValidation)/length(y_dfValidation)
    test_predict <- predict(model[[1]],x_dfTest)
    test_accuracy <- sum(test_predict == y_dfTest)/length(y_dfTest)
    Train_Accuracy <- c(Train_Accuracy, train_accuracy)
    Validation_Accuracy <- c(Validation_Accuracy, validation_accuracy)
    Test_Accuracy <- c(Test_Accuracy, test_accuracy)
    cost_value <- c(cost_value, C_val)
    train_frac_list <- c(train_frac_list, Training_split*100)
    validate_frac_list <- c(validate_frac_list, Validation_split*100)
    test_frac_list <- c(test_frac_list, Test_split*100)
    kernel_val <- c(kernel_val, kernel_list[k1])
}

df <- data.frame(kernel_val, cost_value, train_frac_list, validate_frac_list,
                 test_frac_list, Train_Accuracy*100,
                 Validation_Accuracy*100, Test_Accuracy*100)
names(df) <- c("Kernel","Cost Value","Train Split %","Validate Split %","Test Split %",
               "Train Accuracy %",
               "Validate Accuracy %","Test Accuracy %")

df4 <- df[order(df["Test Accuracy %"], decreasing = TRUE), ]
df4
```

```
 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
 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
```

	Kernel	Cost Value	Train Split %	Validate Split %	Test Split %	Train Accuracy %	Validate Accuracy %	Test Accuracy %
1	vanilladot	0.01	70	15	15	85.12035	85.71429	88.75740
3	anovadot	1000.00	70	15	15	91.90372	89.79592	86.74699
4	polydot	0.01	70	15	15	86.87090	83.67347	84.30233
2	rbfdot	1000.00	70	15	15	99.56236	91.83673	72.61905