HW1

XXX

20 May 2019

Contents

Home Work 1 **2**

Question 2.1 . . . 2

Classification use case example . . . 2

General . . . 2

Business problem definition . . . 2

The Predictors . . . 3

Summary and conclusions . . . 3

Question 2.2.1 . . . 4

item 1 SVM model . . . 4

item 2 other kernal models . . . 6

item 3 Knn model . . . 7

Question 3.1 . . . 8

a - KNN Cross Validiation . . . 8

a - KSVM Cross Validiation . . . 9

b - KNN Train, Validiation and Test sets . . . 10

b - SVM Train, Validiation and Test sets . . . 12

Final Results . . . 13

R Code . . . 14

Read the data . . . 14

Data inspection . . . 14

Question 2.2.1 . . . 17

Question 2.2.2 . . . 22

Question 2.2.3 . . . 22

Question 3.1 . . . 24

A KNN Model . . . 24

A KSVM Cross-validation . . . 26

B KNN For Train/Validation/Test set . . . 29

B SVM For Train/Validation/Test set . . . 32

Home Work 1

Question 2.1

Question : 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.

Ans:

Classification use case example

General

The classification use case example which I provide as an answer to this question is about risk classification or risk assessment of offenders after the offender has been convicted. The model aims to help the system predict the risk that the offender introduces to the public or his community.In more general it is the risk that the offender will commit another crime. Based on these predictions, the legal system can build a set of process that will lead to decisions related to the offender:

- Selecting a Prison type or alternatives to imprisonment (such as electronic monitoring, regular probation or parole).
- probation type and probation programs . I was involved in such classification problems as part of my job as a product manager in a company that provided a full system solution (both devices and software) for electronic monitoring.

Business problem definition

When an offender is convicted, there is a need to classify its risk level as part of setting its punishment plan as well as its rehabilitation plan. The risk assessment classes or categories are usually different between country to country, but it will usually have several levels such as:

- High risk with dangerous impact on its neighborhood.
- High risk.
- Medium risk.
- Low risk

These categorical or factor levels make this problem as we learned a classification problem. A risk model first aim is to classify and predict (provide probability) to each offender in the system about his risk to the society and the probability that he may commit a crime again, or create actions that may risk others. These predictions for each offender can assist the system in providing a specific plan, treatment, that is best suitable for the offender, his community, or the society in general. The outcome can be a prison sentence, probation, parole plan or an electronic monitoring plan. This model can also be used on a decision which electronic monitoring program is the best fit for the specific offender Examples for such decisions can be :

- Should the offender be sent to prison or he may be entitled to participate in one of the alternatives to impersonate plans (e.g. electronic monitoring).
- What type of plan may fit for this offender, regular probation, electronic monitoring.
- What type of electronic monitoring plan should be selected regular home curfew (monitoring only the presence of the offender at home) , GPS monitoring program (full location monitoring , with option to set geo fencing zones).
- What type of probation officer will monitor this offender, and how often does he need to meet a probation officer.

- There are action plans that need to be taken if an offender violates his probation terms or if it violate his electronic monitoring program, these actions are usually also depends on the risk classification.

This model is not only served as a tool for decision making for the specific offender, but it is also a tool that can and be used in a different part in the legal system :

- Overview of all the offenders under the probation system for planning like budget, probation officers requirements, and other resources from legal department (e.g., based on prediction the number of offenders in each risk category the ministry of justice can plan it's budget to buy different type of electronic monitoring equipment)
- It can serve as an indication of crime status and can be tracked over years, and it can be a KPI measurement of a particular probation program or the overall Ministry of Justice.

The Predictors

The predictors for such classifiers can vary, but some of the predictors that I would choose are:

-Type of offense - This is pretty straight forward predictor, as we expect that offenders who commit murder, assault or rape will have a higher risk to the public while small offense like theft or economic offense may introduce medium to low risk. -Is this the first offense This is also a pretty obvious predictor. Unfortunately, the rate of recurring offense is pretty high. Thus multiple offense records may introduce higher risk for committing a crime again. -The age of the offenders - this is not simple predictor, but the age is a factor in setting the risk to the public. - Male or Female offender same as age, usually these two predictors should work well with other predicts and not as a stand-alone predictor. - Family and social condition. The social, economic status also has an impact about the risk and the chances to re-offense again, this predictor itself need a particular work to create a set of category levels of the socioeconomic state.

For simplicity, I have chosen the most prominent and straightforward predictors, but when developing such model other more sophisticated predictors can be researched and selected, for example, classification of the social network activity of the person and try to categorize it, or the psychological profile of the offender. While these predictors can be asses right after the conviction, based on my experience such classifier are also used during the program itself, so the model itself can be used continuously during the probation or sentence period, and more predictors can be added: - Number of program violation (example the offender missed meetings with his probation officer) - Electronic monitoring violations (example offender did not return home on time, offender violate his fencing zones condition).

Note: this is a classification case, as old history cases will be classified and labeled and based on that a model and predictions can be done.

Summary and conclusions

Some general comments and assumption :

-For all models, I have used the accuracy criteria (the ratio of the right prediction to all samples). Since this is credit application model in real-world modeling, there may be an interest to prefer or reduce the error rate to a specific direction (e.g., reduce the number of false credit application approval at the expense of false rejection), but for simplicity on this homework I stayed with the accuracy criteria only.

-I created a function that computes the Accuracy (acc_fun) it is using the base table function.

- Before any mode building, I did some minimal data exploration and verification :
  - Get the data size and type (verify it matches to the questions data)
  - Verify that no data is missing (as promised)

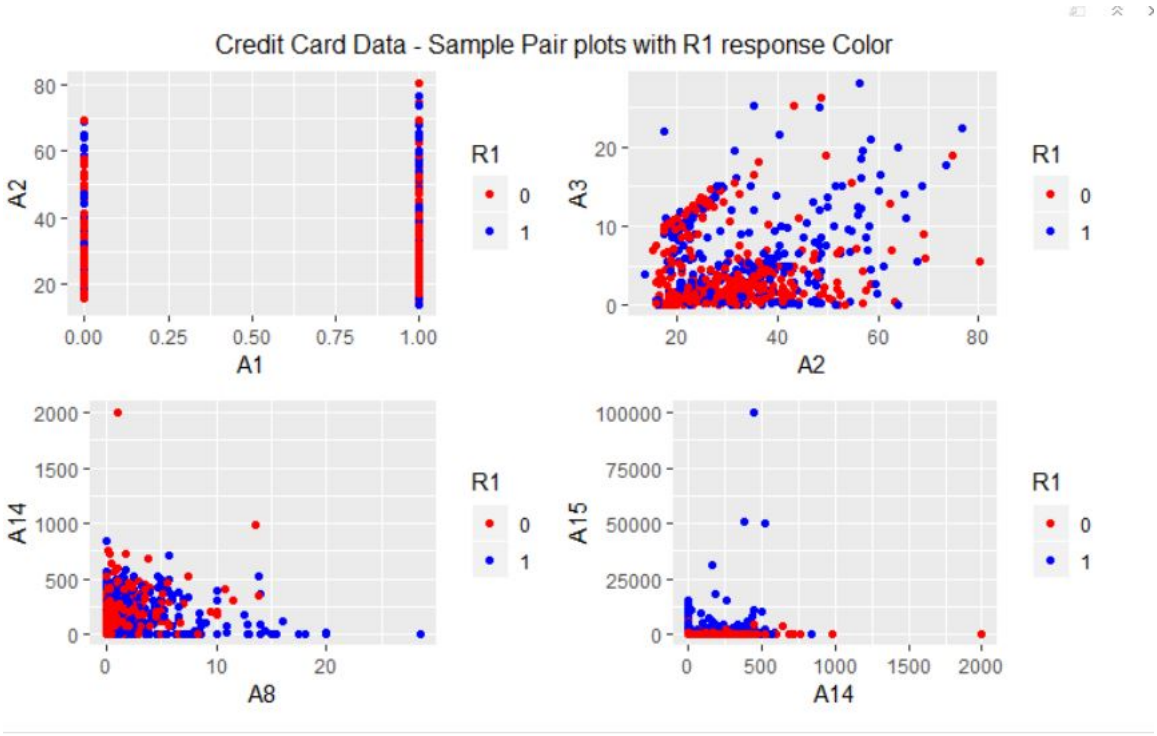


Figure 1: Credit Data pairs examples

- Since this is a multi-dimension problem I tried to plot several pairs and color them based on R1, to get some initial view about the classification problem, you can see the graph below.

The first section below contains the main results and some conclusion The second section contains the R code itself.

Question 2.2.1

item 1 SVM model

Model Name	Scaled	Train Accuracy	Parameter Tuning - C	Question
KSVM	TRUE	0.5474	1.0e-05	Question 2.2.1
KSVM	TRUE	0.5474	5.0e-05	Question 2.2.1
KSVM	TRUE	0.8379	1.0e-03	Question 2.2.1
KSVM	TRUE	0.8639	5.0e-03	Question 2.2.1
KSVM	TRUE	0.8639	1.0e-02	Question 2.2.1
KSVM	TRUE	0.8639	5.0e-02	Question 2.2.1
KSVM	TRUE	0.8639	1.0e-01	Question 2.2.1
KSVM	TRUE	0.8639	5.0e-01	Question 2.2.1
KSVM	TRUE	0.8639	1.0e+00	Question 2.2.1
KSVM	TRUE	0.8639	5.0e+00	Question 2.2.1
KSVM	TRUE	0.8639	1.0e+01	Question 2.2.1
KSVM	TRUE	0.8639	5.0e+01	Question 2.2.1
KSVM	TRUE	0.8639	1.0e+02	Question 2.2.1
KSVM	TRUE	0.8639	5.0e+02	Question 2.2.1
KSVM	TRUE	0.8623	1.0e+03	Question 2.2.1
KSVM	TRUE	0.8623	1.5e+03	Question 2.2.1
KSVM	FALSE	0.6574	1.0e-05	Question 2.2.1
KSVM	FALSE	0.6788	5.0e-05	Question 2.2.1
KSVM	FALSE	0.6926	1.0e-04	Question 2.2.1
KSVM	FALSE	0.7568	5.0e-04	Question 2.2.1
KSVM	FALSE	0.7599	1.0e-03	Question 2.2.1
KSVM	FALSE	0.7966	5.0e-03	Question 2.2.1
KSVM	FALSE	0.8333	1.0e-02	Question 2.2.1
KSVM	FALSE	0.8639	5.0e-02	Question 2.2.1
KSVM	FALSE	0.8623	1.0e-01	Question 2.2.1
KSVM	FALSE	0.8348	5.0e-01	Question 2.2.1
KSVM	FALSE	0.7079	1.0e+00	Question 2.2.1
KSVM	FALSE	0.8486	5.0e+00	Question 2.2.1
KSVM	FALSE	0.6590	1.0e+01	Question 2.2.1
KSVM	FALSE	0.7691	5.0e+01	Question 2.2.1
KSVM	FALSE	0.7217	1.0e+02	Question 2.2.1
KSVM	FALSE	0.6269	5.0e+02	Question 2.2.1
KSVM	FALSE	0.5642	1.0e+03	Question 2.2.1
KSVM	FALSE	0.6834	1.5e+03	Question 2.2.1

In this part, I am Running KSVM model on the entire data set and tuning the C parameter. In general, the C parameter is a regularization that is a trade action off between the margin and classifications rate. In general :

```
-High Value of parameter C leads to a small margin.

-Small Value of parameter C lead to  Large margin.
```

This item included building a model on the entire data set, and just changing the C parameter. I have tried this twice one with Scaled option and one with not. For this version, we can see that (for the Scaled option) there is a range of C values (between 1e-3 to 5e2) that provides a similar accuracy. When C is in an extreme range (Very high or very low) we see a significant degradation in the accuracy. Also, it can be seen that when not scaling the data, the range that we get high accuracy is smaller. It also takes more time to run the mode when data is not scaled, so there is a difference between the scaled and non scaled results (in favor

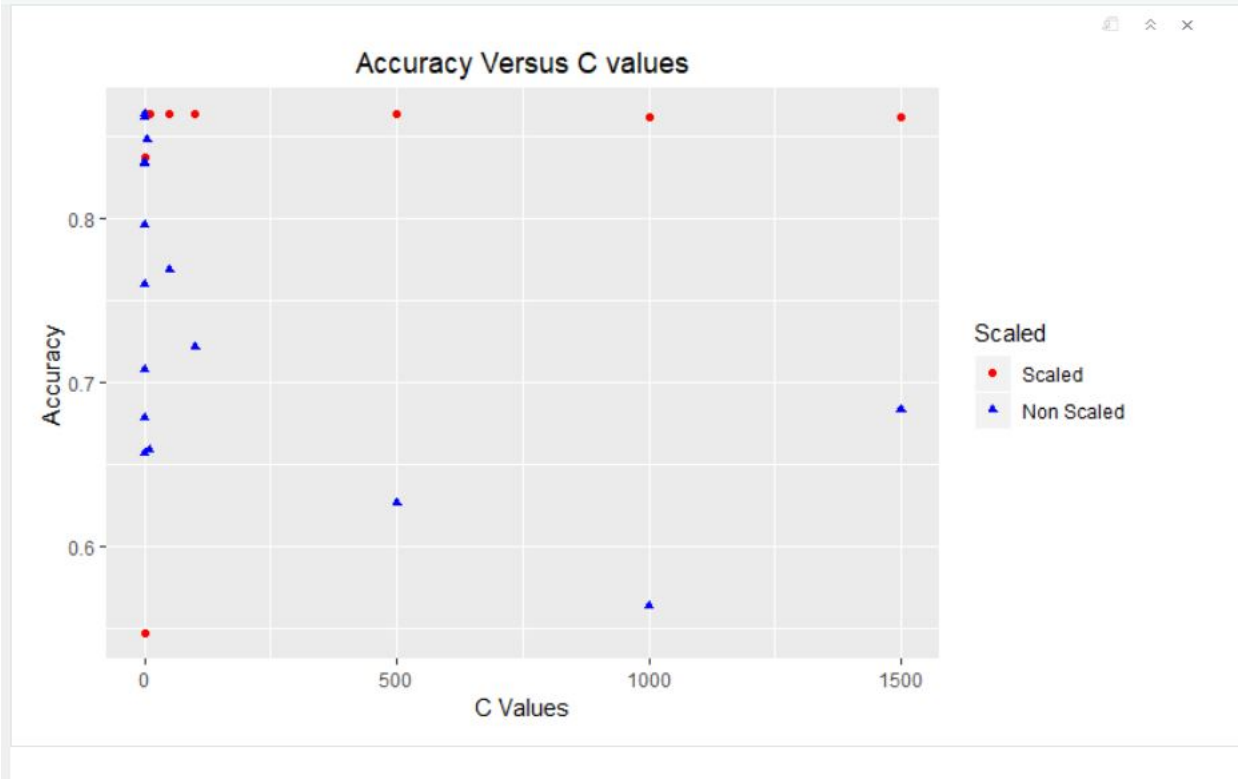


Figure 2: KSVM Model

of the scaled ones) , I was expecting a more significant influence, but it may be related that not all of the predictors are continuous (which require scaling).

For this exercise, I choose C == 0.005, rerun the model on the entire data set and calculate the coefficients.

The classifier that was chosen has the following coefficients :

A1 A2 A3

-0.003998150 -0.002928705 0.004080421

A8 A9 A10

0.051036152 0.889321257 -0.064669507

A11 A12 A14 A15

0.052625776 0.001743936 -0.014767170 0.107170370

and a0

0.06111441

The Accuracy of this model is 0.8639

At this stage, this is all that we can say about this model. Since it uses the entire data set The Accuracy is for this set, it is hard to tell how the model will act on new data, it may over-fit, as it may catch some randomness or noise related to this sample of the data only.

item 2 other kernal models

rbfdot kernel (rbfdot) - Accuracy 0.87 Polynomial kernel (degree=3) Accuracy 0.96

These are two nonlinear kernels, and I have tried them with no parameter tuning Accuracy is better (for the Polynomial much more better). Again since running on the entire data set, it is hard to say, these results may indicate it over-fit (especially the polynomial one)

item 3 Knn model

Model Name	Train Accuracy	Validation Accuracy	Test Accuracy	Parameter tunning (K)	Question
KNN	0.8150	Not Relevant	Not Relevant	1	Question 2.2.1
KNN	0.8150	Not Relevant	Not Relevant	2	Question 2.2.1
KNN	0.8150	Not Relevant	Not Relevant	3	Question 2.2.1
KNN	0.8150	Not Relevant	Not Relevant	4	Question 2.2.1
KNN	0.8517	Not Relevant	Not Relevant	5	Question 2.2.1
KNN	0.8456	Not Relevant	Not Relevant	6	Question 2.2.1
KNN	0.8471	Not Relevant	Not Relevant	7	Question 2.2.1
KNN	0.8486	Not Relevant	Not Relevant	8	Question 2.2.1
KNN	0.8471	Not Relevant	Not Relevant	9	Question 2.2.1
KNN	0.8502	Not Relevant	Not Relevant	10	Question 2.2.1
KNN	0.8517	Not Relevant	Not Relevant	11	Question 2.2.1
KNN	0.8532	Not Relevant	Not Relevant	12	Question 2.2.1
KNN	0.8517	Not Relevant	Not Relevant	13	Question 2.2.1
KNN	0.8517	Not Relevant	Not Relevant	14	Question 2.2.1
KNN	0.8532	Not Relevant	Not Relevant	15	Question 2.2.1
KNN	0.8517	Not Relevant	Not Relevant	16	Question 2.2.1
KNN	0.8517	Not Relevant	Not Relevant	17	Question 2.2.1
KNN	0.8517	Not Relevant	Not Relevant	18	Question 2.2.1
KNN	0.8502	Not Relevant	Not Relevant	19	Question 2.2.1
KNN	0.8502	Not Relevant	Not Relevant	20	Question 2.2.1
KNN	0.8486	Not Relevant	Not Relevant	21	Question 2.2.1
KNN	0.8471	Not Relevant	Not Relevant	22	Question 2.2.1
KNN	0.8440	Not Relevant	Not Relevant	23	Question 2.2.1
KNN	0.8456	Not Relevant	Not Relevant	24	Question 2.2.1
KNN	0.8456	Not Relevant	Not Relevant	25	Question 2.2.1
KNN	0.8440	Not Relevant	Not Relevant	26	Question 2.2.1
KNN	0.8410	Not Relevant	Not Relevant	27	Question 2.2.1
KNN	0.8379	Not Relevant	Not Relevant	28	Question 2.2.1
KNN	0.8394	Not Relevant	Not Relevant	29	Question 2.2.1
KNN	0.8410	Not Relevant	Not Relevant	30	Question 2.2.1
KNN	0.8379	Not Relevant	Not Relevant	31	Question 2.2.1
KNN	0.8364	Not Relevant	Not Relevant	32	Question 2.2.1
KNN	0.8349	Not Relevant	Not Relevant	33	Question 2.2.1
KNN	0.8333	Not Relevant	Not Relevant	34	Question 2.2.1
KNN	0.8318	Not Relevant	Not Relevant	35	Question 2.2.1
KNN	0.8318	Not Relevant	Not Relevant	36	Question 2.2.1
KNN	0.8318	Not Relevant	Not Relevant	37	Question 2.2.1
KNN	0.8318	Not Relevant	Not Relevant	38	Question 2.2.1
KNN	0.8318	Not Relevant	Not Relevant	39	Question 2.2.1
KNN	0.8318	Not Relevant	Not Relevant	40	Question 2.2.1
KNN	0.8318	Not Relevant	Not Relevant	41	Question 2.2.1
KNN	0.8349	Not Relevant	Not Relevant	42	Question 2.2.1
KNN	0.8349	Not Relevant	Not Relevant	43	Question 2.2.1
KNN	0.8364	Not Relevant	Not Relevant	44	Question 2.2.1
KNN	0.8394	Not Relevant	Not Relevant	45	Question 2.2.1

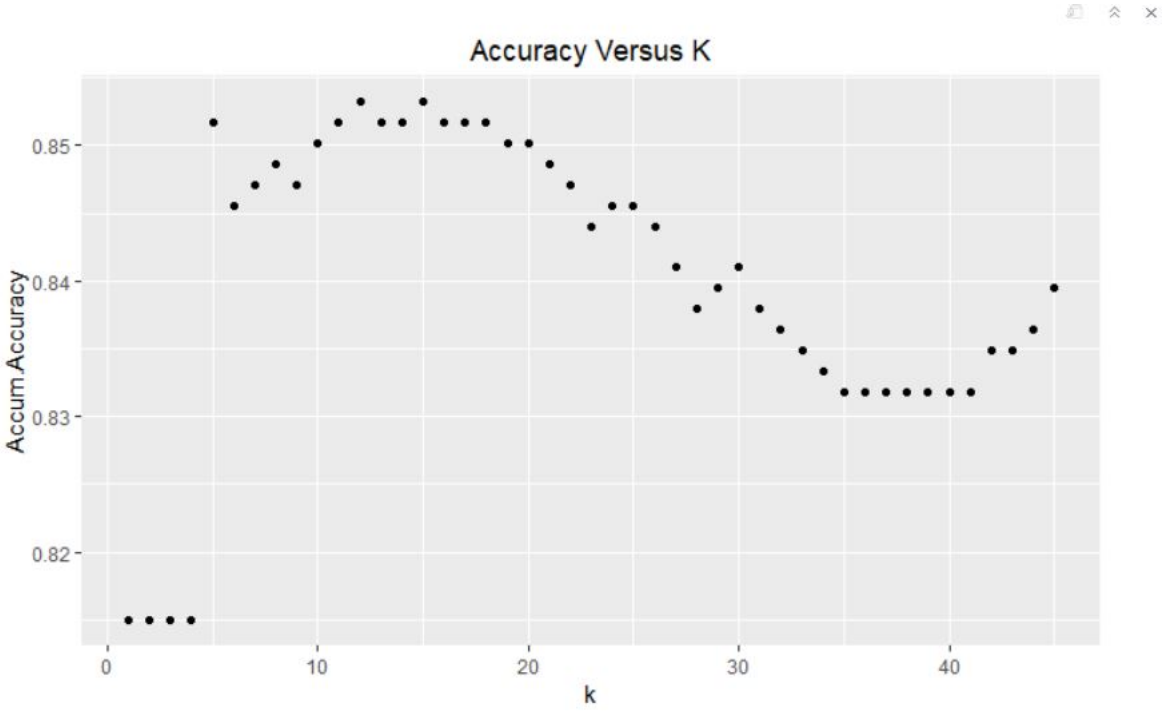
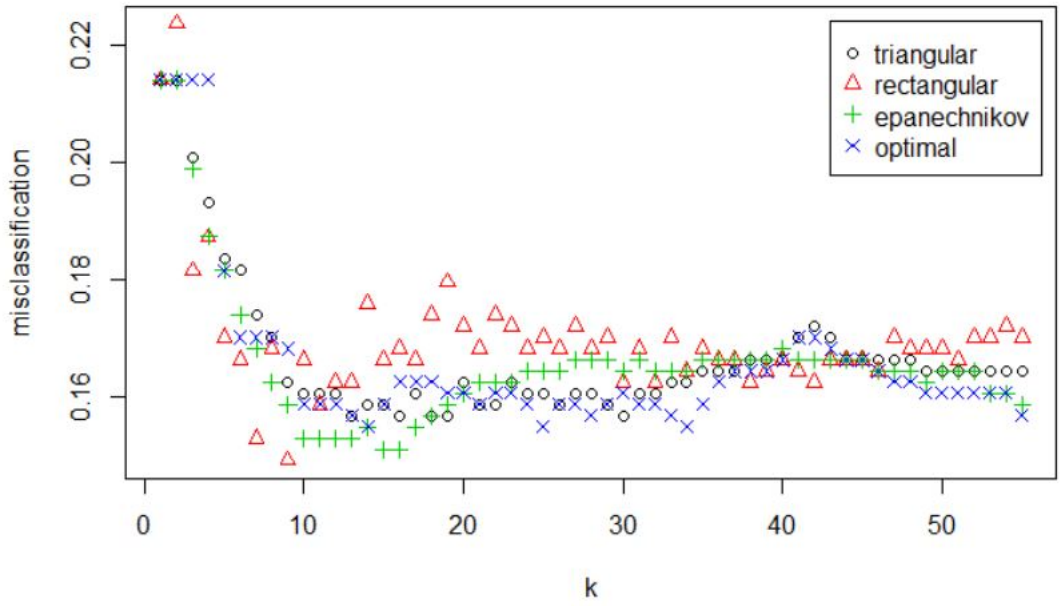


Figure 3: KNN K Versus Accuracy

These are the results for the KNN model on the entire data set. I run a search grid on K in a range from 1 to 45. The results are not significant (there is no high differences in accuracy) I choose K = 12 and got an accuracy of 0.8532. again, in this case, the model is evaluated on the entire data set, which is not recommended and the common practice, so it is hard to determine how it will perform on other data sets.

Question 3.1

a - KNN Cross Validiation



For cross-validation of KNN I used the train.kknn function which computes Cross-validation LOOCV method, this is a particular CV case where the validation is always one sample.

Note this function is has diffrent criteria for slecting the optimal K So here the graph show error and not acuuracy

The Function suggested K =9 (based on a minimum of all methods) Train Error is 0.9273423 (when using the entire train error and re building the model) Test accuracy 0.8625954 9 Here we can see good results both on the train and test sets, but as expected the train set show better accuracy (due to some randomness) compared to the test set.

a - KSVM Cross Validiation

KSVM Linear kernel Search grid on C Value Cross Validation Accuracy

Accuracy	C
0.5442208	1.0e-04
0.5820130	5.0e-04
0.8252597	1.0e-03
0.8667208	5.0e-03
0.8667208	1.0e-01
0.8667208	5.0e-01
0.8667208	1.0e+00
0.8667208	5.0e+00
0.8667208	1.0e+01
0.8667208	1.5e+01
0.8667208	2.0e+01
0.8667208	5.0e+01
0.8667208	1.0e+02
0.8667208	5.0e+02

Accuracy on the training set is 0.8666667 Accuracy on the test set when running with C = 0.003

0.8484848

Again the results are close (both on train and test)

The Model Coefficients :

```
        A1              A2                    A3
```

-0.006825324 -0.001062833 0.001577797

A8 A9 A10

0.069855984 0.643202904 -0.218218873

```
       A11             A12             A14
```

0.122200705 -0.002885020 -0.019833270

```
     A15
```

0.086796032

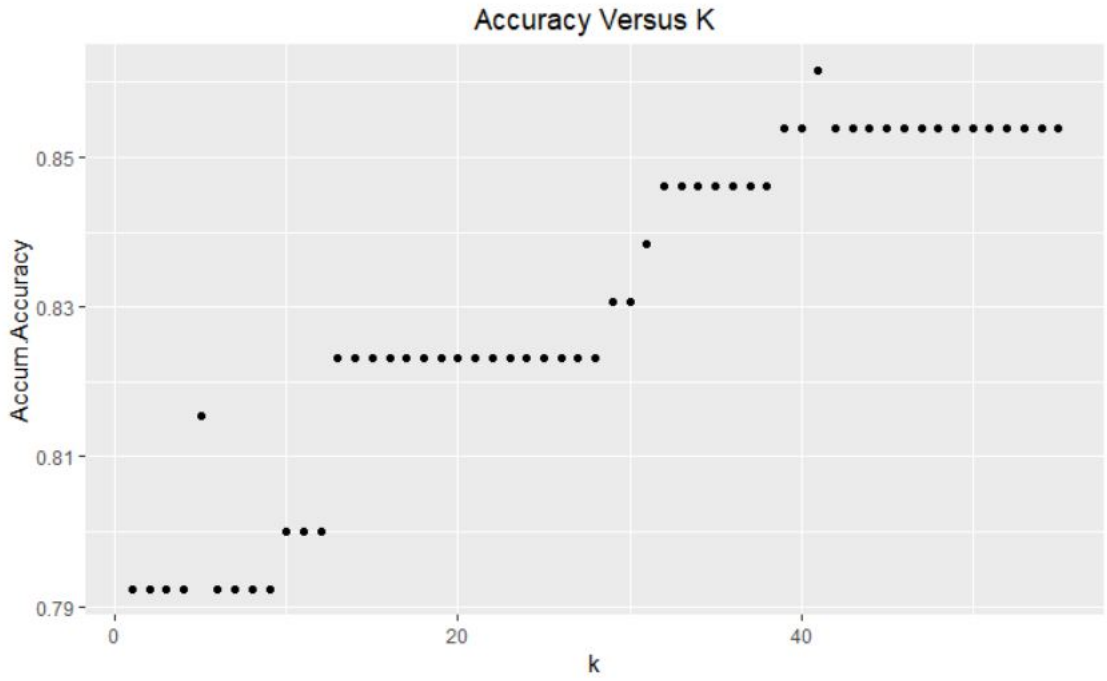
a0

-0.01859218

b - KNN Train, Validiation and Test sets

Accuracy over the validation set for different K

K	Accuracy
1	0.7923077
2	0.7923077
3	0.7923077
4	0.7923077
5	0.8153846
6	0.7923077
7	0.7923077
8	0.7923077
9	0.7923077
10	0.8000000
11	0.8000000
12	0.8000000
13	0.8230769
14	0.8230769
15	0.8230769
16	0.8230769
17	0.8230769
18	0.8230769
19	0.8230769
20	0.8230769
21	0.8230769
22	0.8230769
23	0.8230769
24	0.8230769
25	0.8230769
26	0.8230769
27	0.8230769
28	0.8230769
29	0.8307692
30	0.8307692
31	0.8384615
32	0.8461538
33	0.8461538
34	0.8461538
35	0.8461538
36	0.8461538
37	0.8461538
38	0.8461538
39	0.8538462
40	0.8538462
41	0.8615385
42	0.8538462
43	0.8538462
44	0.8538462
45	0.8538462
46	0.8538462
47	0.8538462
48	0.8538462
49	0.8538462
50	0.8538462
51	0.8538462
52	0.8538462
53	0.8538462
54	0.8538462
55	0.8538462



Selecting K = 41 Accuracy of k=41 on cross validation 0.8615385 Accuracy on the test set 0.8396947

Looking at the overall results, we see some variance in the K, but we also see that there is some decrease in the accuracy between the training to the test set This is an illustration of some of the disadvantages of the train test (or train validation test) method, in our case, the data used in training and validation caught some randomness or noise which doesn't t exit or different in the test set. so we see that the results of the training are better in the test.

b - SVM Train, Validiation and Test sets

Accuracy	C
0.4923077	1.0e-04
0.5153846	5.0e-04
0.7692308	1.0e-03
0.8538462	5.0e-03
0.8538462	1.0e-01
0.8538462	5.0e-01
0.8538462	1.0e+00
0.8538462	5.0e+00
0.8538462	1.0e+01
0.8538462	1.5e+01
0.8538462	2.0e+01
0.8538462	5.0e+01
0.8538462	1.0e+02
0.8538462	5.0e+02

C is chosen to be 0.005 Accuracy on the train set 0.8538462

Accuracy on the test set 0.8244275

Model paramaters

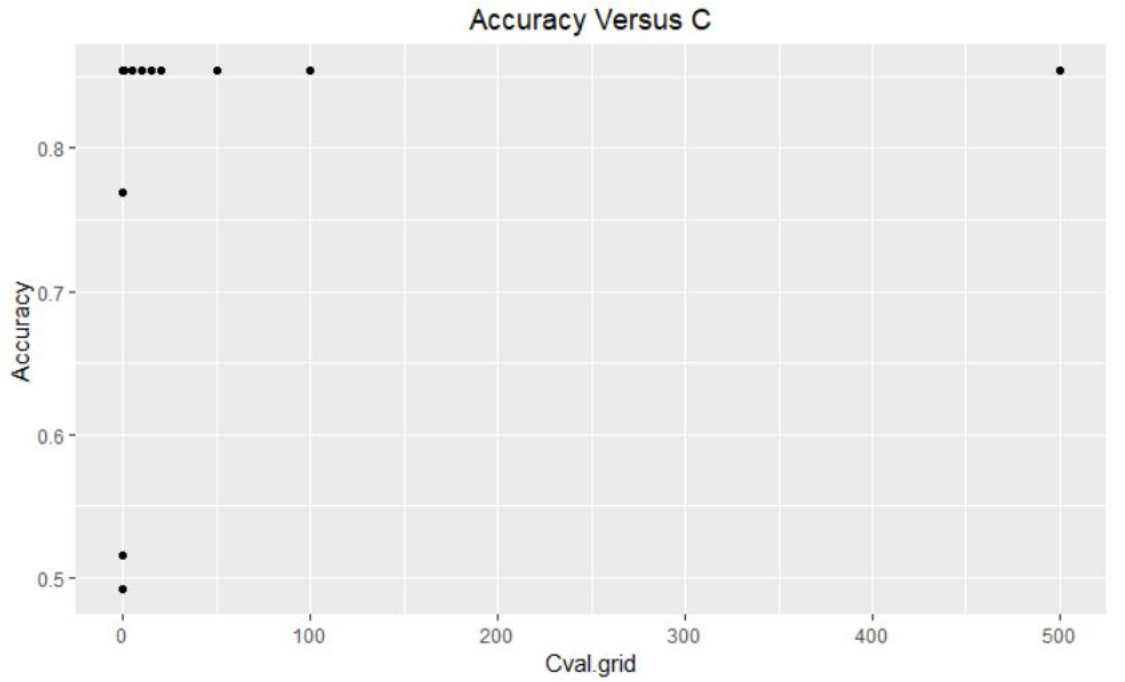


Figure 4: SVM Model Train Test Validation

```
A1            A2            A3
```

0.000121304 -0.008448504 0.009750556

```
  A8            A9           A10
```

0.080477561 0.659734104 -0.186881019 A11 A12 A14

0.121491624 -0.002112441 -0.005535483

A15

0.125104966

a0

0.04189554

again we see degradation in accuracy when we move to the test set either we over fit, capturing some noise or randomness which exit in the train set but not at the test set

Final Results

The table below show a summary of all the results from all questions

X	X.1	Accuracy	X.2	X.3	X.4	X.5
Model	Note	Train	Test	C	K	Sigma
SVM	Only on the Train set	0.8639	Not Relevant	0.005	Not Relevant	Not Relevant
KNN	Only on the Train set	0.853	Not Relevant	Not Relevant	12	Not Relevant
KNN	CV (LOOCV)	0.9273423	0.8625954	Not Relevant	9	Not Relevant
SVM	CV	0.8666667	0.8484848	0.003	Not Relevant	Not Relevant
KNN	Train/Test/Validation	0.8615385	0.8396947	Not Relevant	41	Not Relevant
SVM	Train/Test/Validation	0.8538462	0.8244275	0.005	Not Relevant	Not Relevant
Poly kernel	Only on training		Not Relevant	1	Not Relevant	Not Relevant
Radial kernel	Only on training	0.8715596	Not Relevant	1	Not Relevant	0.1

We can see very similar results (except some spike) As expected the cross-validation method provides the best results of estimating the model performance and reducing variance (handling new data), in general CV offers good results in term regularization (making the model more robust to the new and different type of data) Based on these results, there is no significant difference between linear SVM and KNN. There are exciting results with SVM Polynomial kernel that is worth investigating furthermore (parameter tunning)

R Code

```
library(ggplot2)
library(gridExtra)
library(kernlab)
library(caret)
library(kknn)
library(kableExtra)
library(dplyr)
library(readxl)
```

Helper function to compute accuracy

```
###################################
#Function to compute the accuracy
# The function gets two array's
# and return the accuracy
###################################
acc_fun<-function(y1,y2){

  res<-table(y1,y2)
  #Get the accuracy
  Accuracy<-((res[1,"0"]) + (res[2,"1"])) / sum(res)
  return(Accuracy)
}
```

Read the data

Loading the Credit Card data set

Data inspection

Perform some Basic Data exploratory

look at the first rows using head

```
head(Credit.Df)
```

```
##   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
```

Check dimensions (number of data points-rows), and how many observation there are (columns)

```
dim(Credit.Df)
```

```
## [1] 654  11
```

We indeed have 654 data points(rows)

We have 11 columns

as expected (10 predictors and one response)

str provide information about the variables' types

```
str(Credit.Df)
```

```
## 'data.frame':    654 obs. of  11 variables:
##  $ A1 : int  1 0 0 1 1 1 1 0 1 1 ...
##  $ A2 : num  30.8 58.7 24.5 27.8 20.2 ...
##  $ A3 : num  0 4.46 0.5 1.54 5.62 ...
##  $ A8 : num  1.25 3.04 1.5 3.75 1.71 ...
##  $ A9 : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ A10: int  0 0 1 0 1 1 1 1 1 1 ...
##  $ A11: int  1 6 0 5 0 0 0 0 0 0 ...
##  $ A12: int  1 1 1 0 1 0 0 1 1 0 ...
##  $ A14: int  202 43 280 100 120 360 164 80 180 52 ...
##  $ A15: int  0 560 824 3 0 0 31285 1349 314 1442 ...
##  $ R1 : int  1 1 1 1 1 1 1 1 1 1 ...
```

A1, A9, A10, and A12 looks like binary observations (4 observations) The rest (10) are continuous numerical The response R1 is also binary The summary provides some statistical data (Min-Max Median) This can be verified using summary

```
summary(Credit.Df)
```

```
##        A1               A2              A3               A8
##  Min.   :0.0000   Min.   :13.75   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:0.0000   1st Qu.:22.58   1st Qu.: 1.040   1st Qu.: 0.165
##  Median :1.0000   Median :28.46   Median : 2.855   Median : 1.000
##  Mean   :0.6896   Mean   :31.58   Mean   : 4.831   Mean   : 2.242
##  3rd Qu.:1.0000   3rd Qu.:38.25   3rd Qu.: 7.438   3rd Qu.: 2.615
```

```
##  Max.   :1.0000   Max.   :80.25   Max.   :28.000   Max.   :28.500
##        A9              A10              A11              A12
##  Min.   :0.0000   Min.   :0.0000   Min.   : 0.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.:0.0000
##  Median :1.0000   Median :1.0000   Median : 0.000   Median :1.0000
##  Mean   :0.5352   Mean   :0.5612   Mean   : 2.498   Mean   :0.5382
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.: 3.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :67.000   Max.   :1.0000
##       A14               A15               R1
##  Min.   :   0.00   Min.   :     0   Min.   :0.0000
##  1st Qu.:  70.75   1st Qu.:     0   1st Qu.:0.0000
##  Median : 160.00   Median :     5   Median :0.0000
##  Mean   : 180.08   Mean   :  1013   Mean   :0.4526
##  3rd Qu.: 271.00   3rd Qu.:   399   3rd Qu.:1.0000
##  Max.   :2000.00   Max.   :100000   Max.   :1.0000
```

Verify that there are no Null values

```
sum(is.na(Credit.Df))
```

```
## [1] 0
```

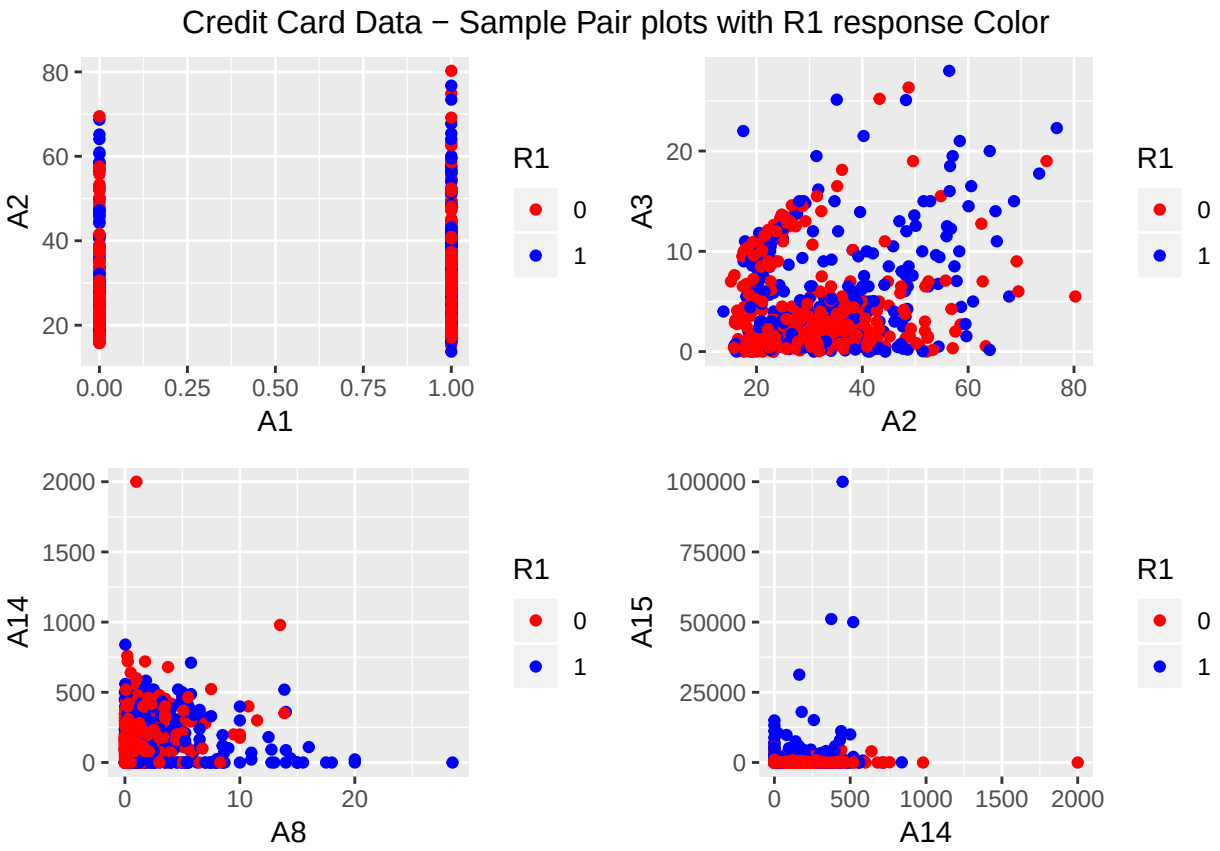
As expected there are no missing values

Convert the response to a factor variable

```
Credit.Df$R1 = as.factor(Credit.Df$R1)
```

It is a hyper-plane classification problem So for an illustration purpose, we can plot pairs of predictors and see the classification challenge (Some of the predictors are binary variables)

```
p1 = ggplot(Credit.Df, aes(x=A1, y=A2,col=R1)) + geom_point()+scale_color_manual(values=c("red", "blue"))
p2 = ggplot(Credit.Df, aes(x=A2, y=A3,col=R1)) + geom_point()+scale_color_manual(values=c("red", "blue"))
p3 = ggplot(Credit.Df, aes(x=A8, y=A14,col=R1)) + geom_point()+scale_color_manual(values=c("red", "blue"))
p4 = ggplot(Credit.Df, aes(x=A14, y=A15,col=R1)) + geom_point()+scale_color_manual(values=c("red", "blue"))

grid.arrange(p1,p2,p3,p4,top = "Credit Card Data - Sample Pair plots with R1 response Color")
```



Question 2.2.1

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 will cover that topic soon.)

```
Credit.Df<-read.table("credit_card_data-headers.txt", header = TRUE, sep = "", dec = ".")
Credit.Df$R1<-as.factor(Credit.Df$R1)
```

Building the first model with default parameters

```
svp = ksvm(R1 ~ ., data = Credit.Df, type="C-svc",scaled = TRUE,kernel = "vanilladot",kpar=list())
svp
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 190
##
## Objective Function Value : -179.385
## Training error : 0.136086
```

As required - Estimating the results on the full set

```
y<-predict(svp,newdata=Credit.Df[,1:10])
Accuracy<-acc_fun(y,Credit.Df$R1)
Accuracy
```

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

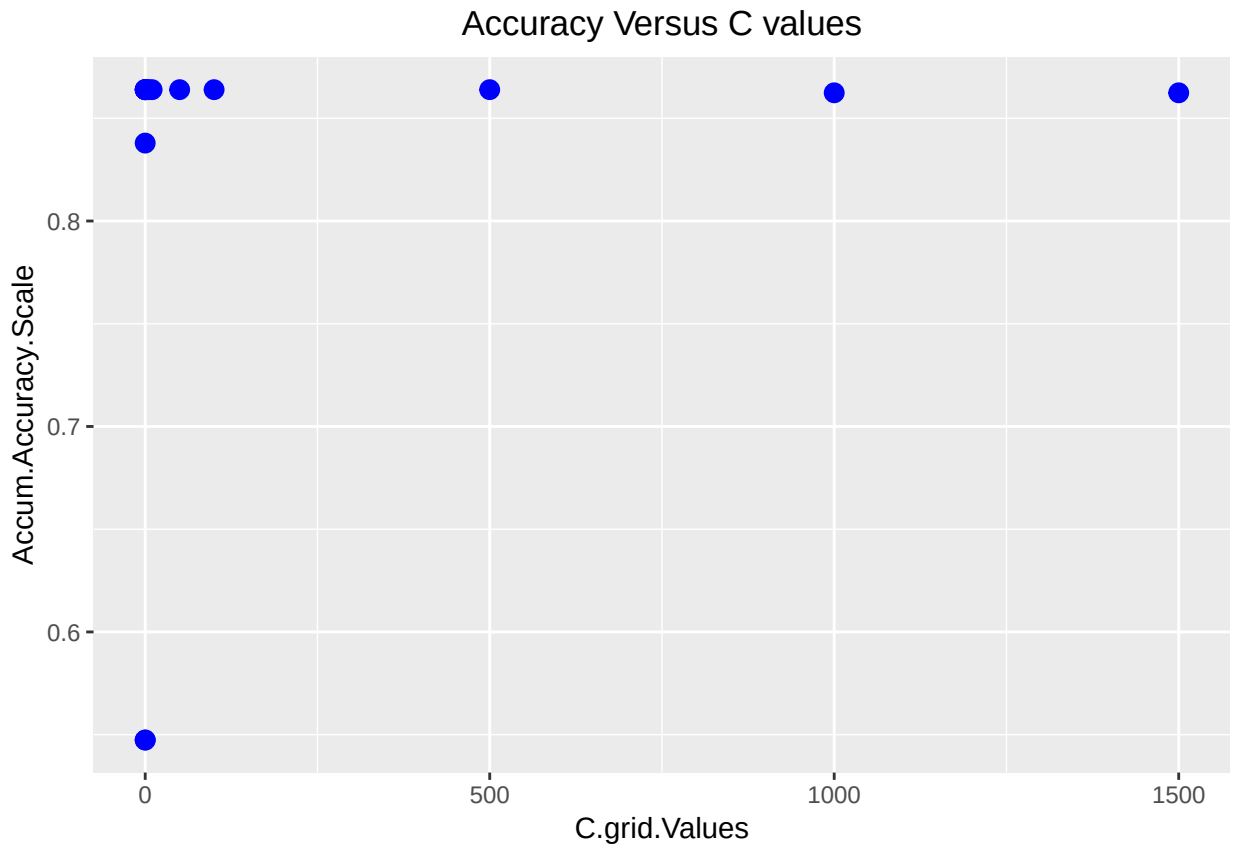
Since in this section, we are requested to work on the entire set I will do a simple grid search on the C parameter, and check the accuracy It is just as an illustration because C is a regulation parameter And it makes more sense to test it on a test data set (not data that was used for training)

```
Accum.Accuracy<-c()
#set the boundary of the grid search
C.grid.Values = c(0.00001,0.00005,0.001,0.005,0.01,0.05,0.1,0.5,1,5 , 10,50,100,500,1000,1500)
x <- model.matrix(R1~.,Credit.Df )[,-1]
y<-as.factor(Credit.Df$R1)
#Create a for loop over the grid search boundaries
for (Cval in  C.grid.Values){
  svp = ksvm(as.matrix(x),as.factor(y), type="C-svc",scaled = TRUE,kernel =  "vanilladot",C=Cval,kpar=li
  #preform prediction
  ypredict<-predict(svp,newdata=x)
  #Calculate Accuracy
  Accuracy<-acc_fun(y,ypredict)

  Accum.Accuracy<-c(Accum.Accuracy,Accuracy)
}

Accum.Accuracy
```

```
##  [1] 0.5474006 0.5474006 0.8379205 0.8639144 0.8639144 0.8639144 0.8639144
##  [8] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144
## [15] 0.8623853 0.8623853
```

Plot a Graph of Accuracy Versus C

```
Accum.Accuracy.Scale<-Accum.Accuracy
DF<-data.frame(C.grid.Values,Accum.Accuracy.Scale)
ggplot(aes(x=C.grid.Values,y=Accum.Accuracy.Scale),data=DF)+geom_point(fill="blue", color ="blue" ,size=
```




```
MaxIndex<-which.max(Accum.Accuracy)
C.grid.Values[MaxIndex]
```

```
## [1] 0.005
```

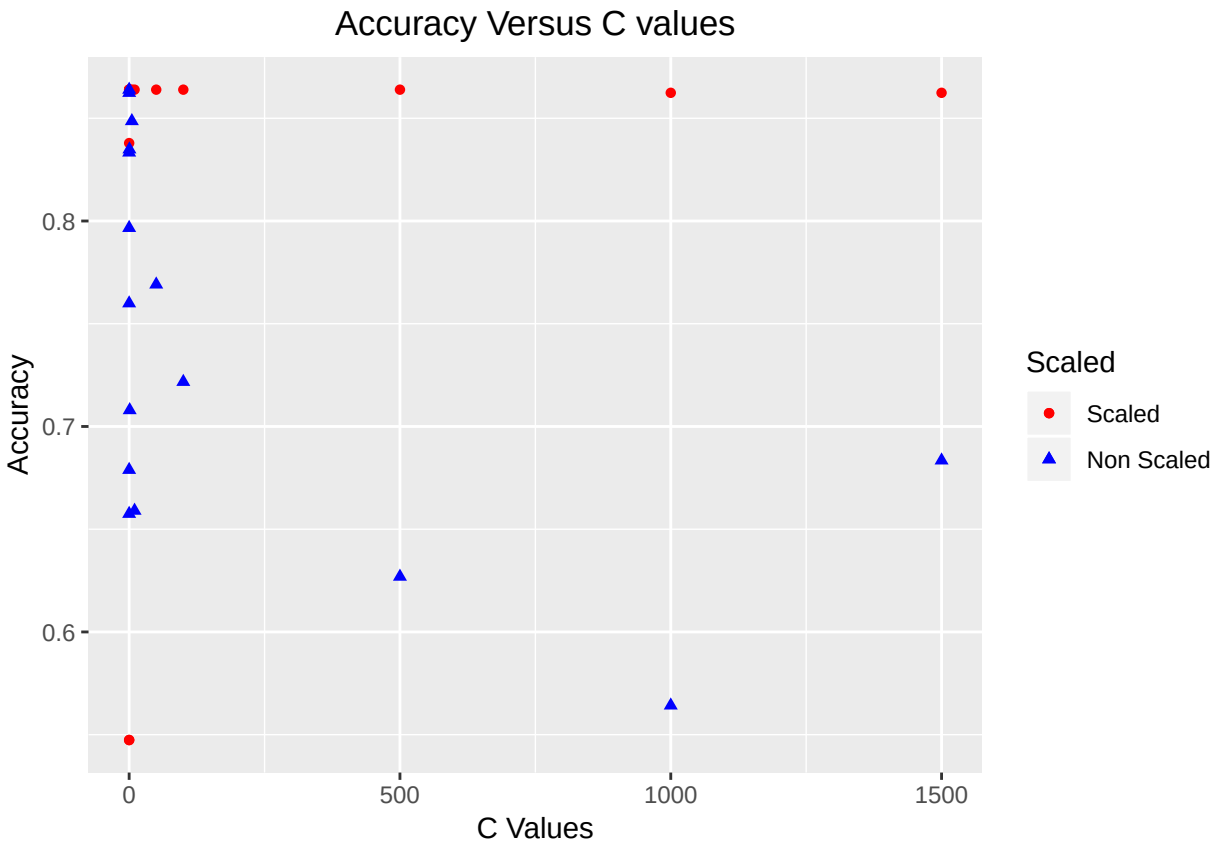
Lets see the impact when we don't use scaling

```
Accum.Accuracy<-c()
#set the boundry of the grid search
C.grid.Values = c(0.00001,0.00005,0.001,0.005,0.01,0.05,0.1,0.5,1,5 , 10,50,100,500,1000,1500)
x <- model.matrix(R1~.,Credit.Df )[,-1]
y<-as.factor(Credit.Df$R1)
#Create a for loop over the grid search boundries
for (Cval in  C.grid.Values){
  svp = ksvm(as.matrix(x),as.factor(y), type="C-svc",scaled = FALSE,kernel =  "vanilladot",C=Cval,kpar=
  #preform prediction
  ypredict<-predict(svp,newdata=x)
  #Calculate accuracy
  Accuracy<-acc_fun(y,ypredict)


  Accum.Accuracy<-c(Accum.Accuracy,Accuracy)
}

Accum.Accuracy
```

```
##  [1] 0.6574924 0.6788991 0.7599388 0.7966361 0.8333333 0.8639144 0.8623853
##  [8] 0.8348624 0.7079511 0.8486239 0.6590214 0.7691131 0.7217125 0.6269113
## [15] 0.5642202 0.6834862
```

Plot a Graph of Accuracy Versus C of both scaled and non scaled options

```
Accum.Accuracy.Non.Scale<-Accum.Accuracy
DF<-mutate(DF,Accum.Accuracy.Non.Scale = Accum.Accuracy.Non.Scale)
#ggplot(aes(x=C.grid.Values,y=Accum.Accuracy),data=DF)+geom_point()+ggtitle("Accuracy #Versus C values"




cols    <- c( "c1" = "Red", "c2" = "Blue" )
shapes  <- c("s1" = 16, "s2" = 17)

p1<-ggplot(DF, aes(x=C.grid.Values)) +
    geom_point(aes(y = Accum.Accuracy.Scale,color = "c1", shape = "s1")) +
    geom_point(aes(y = Accum.Accuracy.Non.Scale,color = "c2", shape = "s2"))
p1<-p1+ggtitle("Accuracy Versus C values")+theme(plot.title = element_text(hjust = 0.5))

p1 <- p1 + scale_color_manual(name = "Scaled",
                              breaks = c("c1", "c2"),
                              values = cols,
                              labels = c("Scaled", "Non Scaled"))
p1 <- p1 + scale_shape_manual(name = "Scaled",
                             breaks = c("s1", "s2"),
                             values = shapes,
                             labels = c("Scaled", "Non Scaled"))

p1<-p1+labs(title="Accuracy Versus C values", y="Accuracy", x="C Values")
p1 <- p1
p1
```



So we choose the scale model with C == 0.005 Calculate the linear coefficients of the selected model

```
svp.best = ksvm(as.matrix(x),as.factor(y), type="C-svc",scaled = TRUE,kernel =  "vanilladot",C=.005,kpa
a <- colSums(svp.best@xmatrix[[1]] * svp.best@coef[[1]])
a
```

```
##           A1           A2           A3           A8           A9
## -0.003998150 -0.002928705  0.004080421  0.051036152  0.889321257
##          A10          A11          A12          A14          A15
## -0.064669507  0.052625776  0.001743936 -0.014767170  0.107170370
```

Calculate a0

```
a0<--svp.best@b
a0
```

```
## [1] 0.06111441
```

Predict and calculate the accuracy of the chosen model

```
x <- model.matrix(R1~.,Credit.Df )[,-1]
y<-as.factor(Credit.Df$R1)
ypredict<-predict(svp.best,newdata=x)
Accuracy<-acc_fun(y,ypredict)
Accuracy
```

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

Question 2.2.2

2. You are welcome, but not required, to try other (nonlinear) kernels as well; we are not covering them in this course, but they can sometimes be useful and might provide better predictions than vanilladot.

We can try a Radial kernel

```
x <- model.matrix(R1~.,Credit.Df )[,-1]
y<-as.factor(Credit.Df$R1)
svp.rad = ksvm(x,y, type="C-svc",scaled = TRUE,kernel =  "rbfdot")

ypredict<-predict(svp.rad,newdata=x)
Accuracy<-acc_fun(y,ypredict)
Accuracy
```

```
## [1] 0.8715596
```

The Radial Basis is very popular kernel and as we can see it get a better accuracy even without running the parameters

Another kernel is Polydot (polynomial)

```
x <- model.matrix(R1~.,Credit.Df )[,-1]
y<-as.factor(Credit.Df$R1)
svp.poly = ksvm(x,y, type="C-svc",scaled = TRUE,kernel =  "polydot",kpar=list(degree = 3))

ypredict<-predict(svp.poly,newdata=x)
Accuracy<-acc_fun(y,ypredict)
Accuracy
```

```
## [1] 0.9663609
```

With Polynomial kernel we get even better accuracy. We need to remember that we are training on the entire data set So we are probably over fitting

Question 2.2.3

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. Don't forget to scale the data (scale=TRUE in kknn).

```
Credit.Df<-read.table("credit_card_data-headers.txt", header = TRUE, sep = "", dec = ".")




Accum.Accuracy<-c()
for (x in 1:45){
  pred<-rep(0,(nrow(Credit.Df)))
  for (i in 1:nrow(Credit.Df)){
    knn.mod <-kknn(R1~.-R1,train = Credit.Df[-i,],test = Credit.Df[i,],k=x, scale = TRUE)
```

```
    pred[i] <- as.integer(fitted(knn.mod)+0.5)
  }
  #Calculate accuracy for this K


  Accuracy<-acc_fun(as.factor(Credit.Df$R1),as.factor(pred))
  #Get the accurecy

  Accum.Accuracy<-c(Accum.Accuracy,Accuracy)

}

Accum.Accuracy
```

```
##  [1] 0.8149847 0.8149847 0.8149847 0.8149847 0.8516820 0.8455657 0.8470948
##  [8] 0.8486239 0.8470948 0.8501529 0.8516820 0.8532110 0.8516820 0.8516820
## [15] 0.8532110 0.8516820 0.8516820 0.8516820 0.8501529 0.8501529 0.8486239
## [22] 0.8470948 0.8440367 0.8455657 0.8455657 0.8440367 0.8409786 0.8379205
## [29] 0.8394495 0.8409786 0.8379205 0.8363914 0.8348624 0.8333333 0.8318043
## [36] 0.8318043 0.8318043 0.8318043 0.8318043 0.8318043 0.8318043 0.8348624
## [43] 0.8348624 0.8363914 0.8394495
```

```
max(Accum.Accuracy)
```

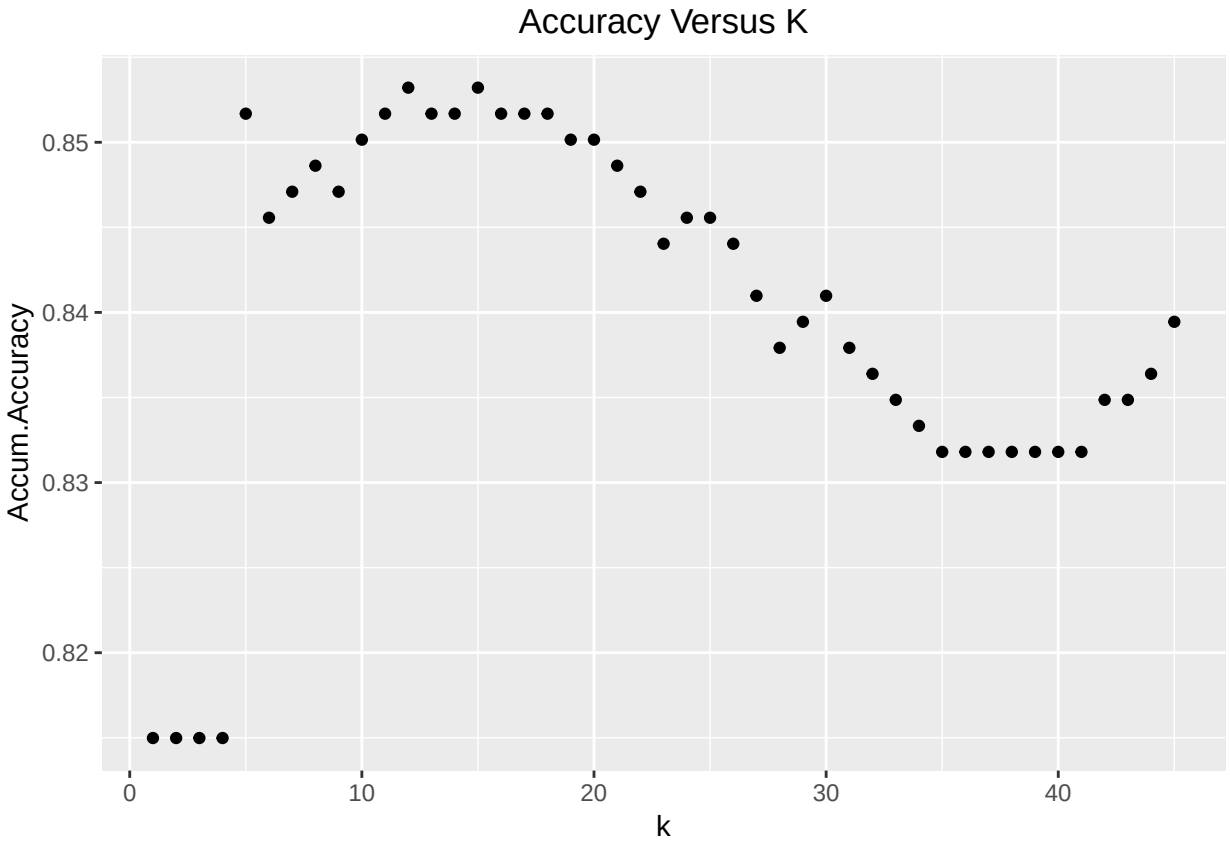
```
## [1] 0.853211
```

```
which.max(Accum.Accuracy)
```

```
## [1] 12
```

So the Optimal Accuracy is 0.853, and the Optimal K is 12 (on the entire data set)

```
DF<-data.frame(Accum.Accuracy,k=1:45)
ggplot(data=DF,aes(x=k,y=Accum.Accuracy))+geom_point()+ggtitle("Accuracy Versus K")+theme(plot.title = e
```



Question 3.1

A KNN Model

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

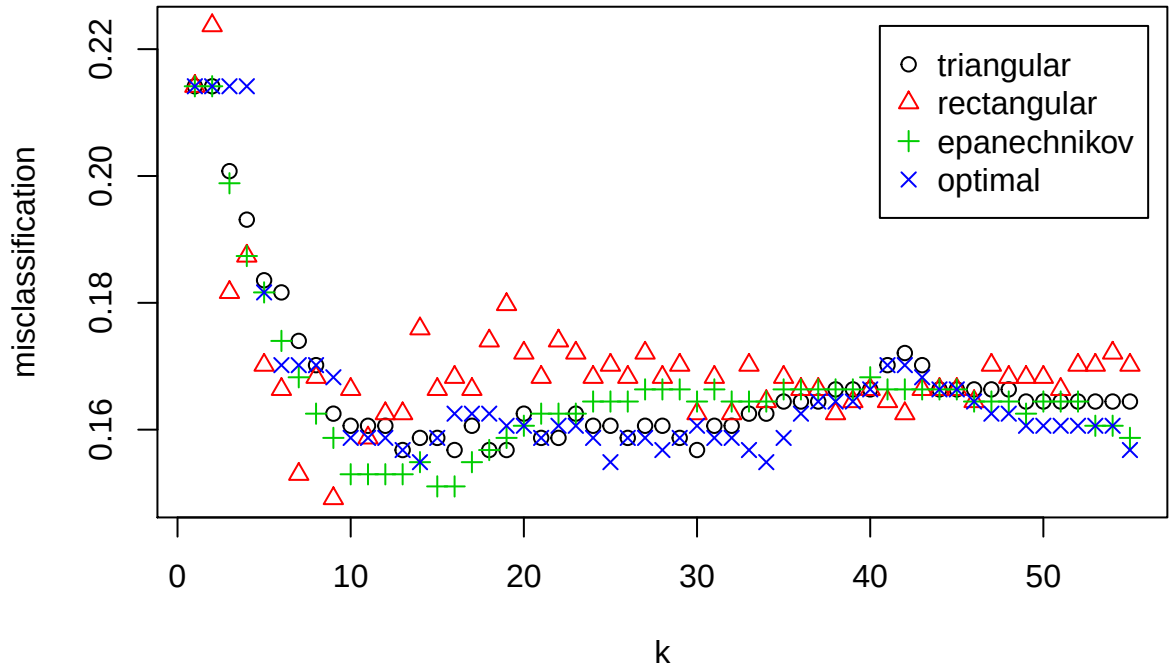
Item a

```
Credit.Df<-read.table("credit_card_data-headers.txt", header = TRUE, sep = "", dec = ".")
Credit.Df$R1<-as.factor(Credit.Df$R1)
```

For cross-validation of KNN I used the train.kknn function which computes Cross-validation LOOCV method, this is a particular CV case where the validation is always one sample. 1. Split the data to train and test set (80% - 20%) 2. Tune the K parameters on the training data using cross-validation (LOOCV) 3 Train the model with the chosen K on the entire train set (check accuracy) 4. Check accuracy on the test set

```
#kfolds<-10
set.seed(2)
  #split the data to train and test sets
train.index<-sample(1:nrow(Credit.Df),0.8*nrow(Credit.Df))
train<-Credit.Df[train.index,]
test<-Credit.Df[-train.index,]
```

```
fit.knn <- train.kknn(R1 ~ .-R1, train, ks=c(1:55), kernel =
    c("triangular", "rectangular", "epanechnikov", "optimal"), distance = 1,scale = TRUE)
plot(fit.knn)
```




```
fit.knn
```

```
##
## Call:
## train.kknn(formula = R1 ~ . - R1, data = train, ks = c(1:55),     distance = 1, kernel = c("triangula
##
## Type of response variable: nominal
## Minimal misclassification: 0.1491396
## Best kernel: rectangular
## Best k: 9
```

The chosen K is 9

use the best mode and first train it on the entire train set calculate the accuracy and then test it on test set

```
knn.mod <-kknn(R1~.-R1,train = train,test = train,k=9, scale = TRUE)
#knn.predict<-predict(knn.mod,test)
Accuracy<-acc_fun(fitted(knn.mod), train$R1)
#Get the accurecy

Accuracy
```

```
## [1] 0.9273423
```

```
knn.mod <-kknn(R1~.-R1,train = train,test = test,k=9, scale = TRUE)
#knn.predict<-predict(knn.mod,test)
Accuracy<-acc_fun(fitted(knn.mod), test$R1)

Accuracy
```

```
## [1] 0.8625954
```

A KSVM Cross-validation

Though this function has a cross-validation build in
I have decided to build for the learning exercise I have decided to do this manually I will do the cross-validation to tune the C parameter The steps are as follow : 1. Split the data into 80% (train+cross validation) 20% (test) 2. shuffle the train data 3 build 10 folds of the train data 4 create a grid search for C (between .5 and 15) 5. for each C do cross-validation, average the accuracy 6. compare all the results and choose the best C (according to the accuracy) 7. train the model with the chosen parameter on the full train set 8. run the model on the test set - check the accuracy

Item a

```
Credit.Df<-read.table("credit_card_data-headers.txt", header = TRUE, sep = "", dec = ".")
Credit.Df$R1<-as.factor(Credit.Df$R1)

set.seed(2256)
#Split the data to train and test sets
train.index<-base::sample(1:nrow(Credit.Df),0.85*nrow(Credit.Df))
train<-Credit.Df[train.index,]
test<-Credit.Df[-train.index,]

#Shuffle the train data
train<-train[sample(nrow(train)),]
```

```
Accum.Accuracy<-c()
Cval.grid<-c(0.0001,0.0005,.001,0.005,0.1,0.5,1,5 , 10,15,20,50,100,500)
#Cval.grid<-seq(1:15)

kfolds<-10

#Create 10 equally size folds indexs
folds <- cut(seq(1,nrow(train)),breaks=kfolds,labels=FALSE)
Accum.Accuracy<-c()
for (Cvalue in Cval.grid){


  folds.Accum.Accuracy<-c()
  for(i in 1:kfolds){
    #Segement
    KfoldtestIndexes <- which(folds==i,arr.ind=TRUE)
    KfoldtestData <- train[KfoldtestIndexes, ]
    KfoldtrainData <- train[-KfoldtestIndexes, ]
```

```
    #convert the training and test data into matrix
    x <- model.matrix(R1~.,KfoldtrainData )[,-1]
    y<-as.factor(KfoldtrainData$R1)
    xtest<-model.matrix(R1~.,KfoldtestData)[,-1]
    ytest<-as.factor(KfoldtestData$R1)
    #build the model
    svp.Kfold = ksvm(x,y, type="C-svc",scaled = TRUE,kernel = "vanilladot",C=Cvalue,kpar=list())

    #predict
    ypredict<-predict(svp.Kfold,xtest)

    #Get the accurecy
    Accuracy<-acc_fun(ytest,as.factor(ypredict))


    folds.Accum.Accuracy<<-c(folds.Accum.Accuracy,Accuracy)
  }
Accum.Accuracy<-c(Accum.Accuracy,mean(folds.Accum.Accuracy))

}
print(max(Accum.Accuracy))
```

```
## [1] 0.8667208
```

```
MaxIndex<-which.max(Accum.Accuracy)
Cval.grid[MaxIndex]
```

```
## [1] 0.005
```

Now we can fine tune around C = 0.005

```
Accum.Accuracy<-c()
Cval.grid<-c(0.00001,0.0005,0.001,0.003,0.004,0.005,.006,.007,0.008,1)
#Cval.grid<-seq(1:15)




Accum.Accuracy<-c()
for (Cvalue in Cval.grid){
folds.Accum.Accuracy<-c()


  for(i in 1:kfolds){
    #Segement
    KfoldtestIndexes <- which(folds==i,arr.ind=TRUE)
    KfoldtestData <- train[KfoldtestIndexes, ]
    KfoldtrainData <- train[-KfoldtestIndexes, ]

    #convert the training and test data into matrix
    x <- model.matrix(R1~.,KfoldtrainData )[,-1]
    y<-as.factor(KfoldtrainData$R1)
```

```
    xtest<-model.matrix(R1~.,KfoldtestData)[,-1]
    ytest<-as.factor(KfoldtestData$R1)
    #build the model
    svp.Kfold = ksvm(x,y, type="C-svc",scaled = TRUE,kpar=list(),kernel = "vanilladot",C=Cvalue)

    #predict
    ypredict<-predict(svp.Kfold,xtest)
     #Get the accurecy
    Accuracy<-acc_fun(ytest,as.factor(ypredict))


    folds.Accum.Accuracy<<-c(folds.Accum.Accuracy,Accuracy)
  }
Accum.Accuracy<-c(Accum.Accuracy,mean(folds.Accum.Accuracy))

}
print(max(Accum.Accuracy))
```

```
## [1] 0.8667208
```

```
MaxIndex<-which.max(Accum.Accuracy)
Cval.grid[MaxIndex]
```

```
## [1] 0.003
```

No major difference Lets build the model on the entire train set with C=0.003 and check it on the test set

```
x <- model.matrix(R1~.,train)[,-1]
y<-(train$R1)
svp.Best =  ksvm(x,y, type="C-svc",scaled = TRUE,kernel ="vanilladot",C=0.003,kpar=list())

#xtrain<-model.matrix(R1~.,train)[,-1]
#ytrain<-as.factor(train$R1)
ypredict<-predict(svp.Best,x,type = "response")

Accuracy<-acc_fun(y,as.factor(ypredict))
#Get the accurecy
Accuracy
```

```
## [1] 0.8666667
```

```
#xtest<-model.matrix(R1~.,test)[,-1]
xtest<-model.matrix(R1~.,test[,-1])
ytest<-(test$R1)
ypredict<-predict(svp.Best,xtest)
#Get the accurecy
Accuracy<-acc_fun(ytest,as.factor(ypredict))

Accuracy
```

```
## [1] 0.8484848
```

Calculate the linear coefficients of the selected model:

```
a <- colSums(svp.Best@xmatrix[[1]] * svp.Best@coef[[1]])
a
```

```
##           A1           A2           A3           A8           A9
## -0.006825324 -0.001062833  0.001577797  0.069855984  0.643202904
##          A10          A11          A12          A14          A15
## -0.218218873  0.122200705 -0.002885020 -0.019833270  0.086796032
```

Calculate a0

```
a0<--svp.Best@b
a0
```

```
## [1] -0.01859218
```

B KNN For Train/Validation/Test set

In this section, I will use the following steps Split the data into : – 60 % train set – 20% validation set (choose K) – 20% test - test the model The Split will be done Randomly (in the second section I will use the rotation method)

```
Credit.Df<-read.table("credit_card_data-headers.txt", header = TRUE, sep = "", dec = ".")
Credit.Df$R1<-as.factor(Credit.Df$R1)
```

```
set.seed(1232)
#split the data to train and test  sets
train.index<-sample(1:nrow(Credit.Df),0.8*nrow(Credit.Df))
traintmp<-Credit.Df[train.index,]
test<-Credit.Df[-train.index,]
#split the train data into train and validation set
set.seed(1232)
valid.index<-sample(1:nrow(traintmp),0.2*nrow(Credit.Df))
train<-traintmp[-valid.index,]
valid<-traintmp[valid.index,]
```

Check dimensions

```
dim(train)
```

```
## [1] 393  11
```

```
dim(test)
```

```
## [1] 131  11
```

```
dim(valid)
```

```
## [1] 130  11
```

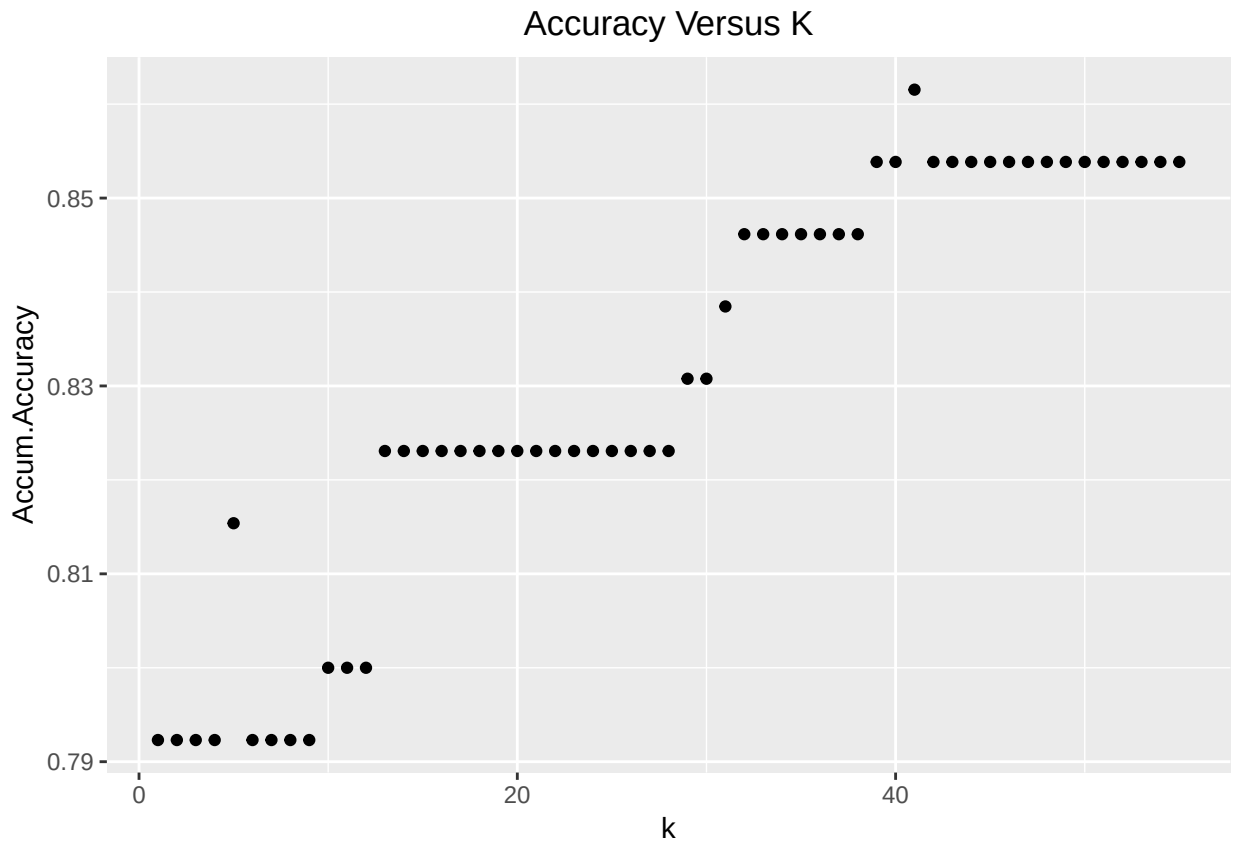
Iterate and build model on train set and test it on the validation set

```
Accum.Accuracy<-c()

for (x in 1:55){

  knn.mod <-kknn::kknn(R1~.-R1,train = train,test = valid,k=x, scale = TRUE)


  #Calculate accuracy for this K
  Accuracy<-acc_fun(as.factor(valid$R1),as.factor(fitted(knn.mod)))

  Accum.Accuracy<-c(Accum.Accuracy,Accuracy)

}


Accum.Accuracy
```

```
##  [1] 0.7923077 0.7923077 0.7923077 0.7923077 0.8153846 0.7923077 0.7923077
##  [8] 0.7923077 0.7923077 0.8000000 0.8000000 0.8000000 0.8230769 0.8230769
## [15] 0.8230769 0.8230769 0.8230769 0.8230769 0.8230769 0.8230769 0.8230769
## [22] 0.8230769 0.8230769 0.8230769 0.8230769 0.8230769 0.8230769 0.8230769
## [29] 0.8307692 0.8307692 0.8384615 0.8461538 0.8461538 0.8461538 0.8461538
## [36] 0.8461538 0.8461538 0.8461538 0.8538462 0.8538462 0.8615385 0.8538462
## [43] 0.8538462 0.8538462 0.8538462 0.8538462 0.8538462 0.8538462 0.8538462
## [50] 0.8538462 0.8538462 0.8538462 0.8538462 0.8538462 0.8538462
```

Plot a graph of K Versus accuracy

```
DF<-data.frame(Accum.Accuracy,k=1:55)
ggplot(data=DF,aes(x=k,y=Accum.Accuracy))+geom_point()+ggtitle("Accuracy Versus K")+theme(plot.title = e
```



Max Accuracy

```
max(Accum.Accuracy)
```

```
## [1] 0.8615385
```

K for Max accuracy

```
which.max(Accum.Accuracy)
```

```
## [1] 41
```

Test the model on the test set

```
knn.best <-kknn(R1~.-R1,train = train,test = test,k=41, scale = TRUE)
Accuracy<-acc_fun(as.factor(test$R1),as.factor(fitted(knn.best)))

Accuracy
```

```
## [1] 0.8396947
```

B SVM For Train/Validation/Test set

For the second option of this question I will use an SVM kernel,

Split the data into : – 60 % train set – 20% validation set (choose K) – 20% test - test the model The Split will be done Randomly

```
Credit.Df<-read.table("credit_card_data-headers.txt", header = TRUE, sep = "", dec = ".")
Credit.Df$R1<-as.factor(Credit.Df$R1)
```

```
set.seed(1232)
#split the data to train and test  sets
train.index<-sample(1:nrow(Credit.Df),0.8*nrow(Credit.Df))
traintmp<-Credit.Df[train.index,]
test<-Credit.Df[-train.index,]
#split the train data into train and validation set
set.seed(1232)
valid.index<-sample(1:nrow(traintmp),0.2*nrow(Credit.Df))
train<-traintmp[-valid.index,]
valid<-traintmp[valid.index,]
```

Check dimensions

```
dim(train)
```

```
## [1] 393  11
```

```
dim(test)
```

```
## [1] 131  11
```

```
dim(valid)
```

```
## [1] 130  11
```

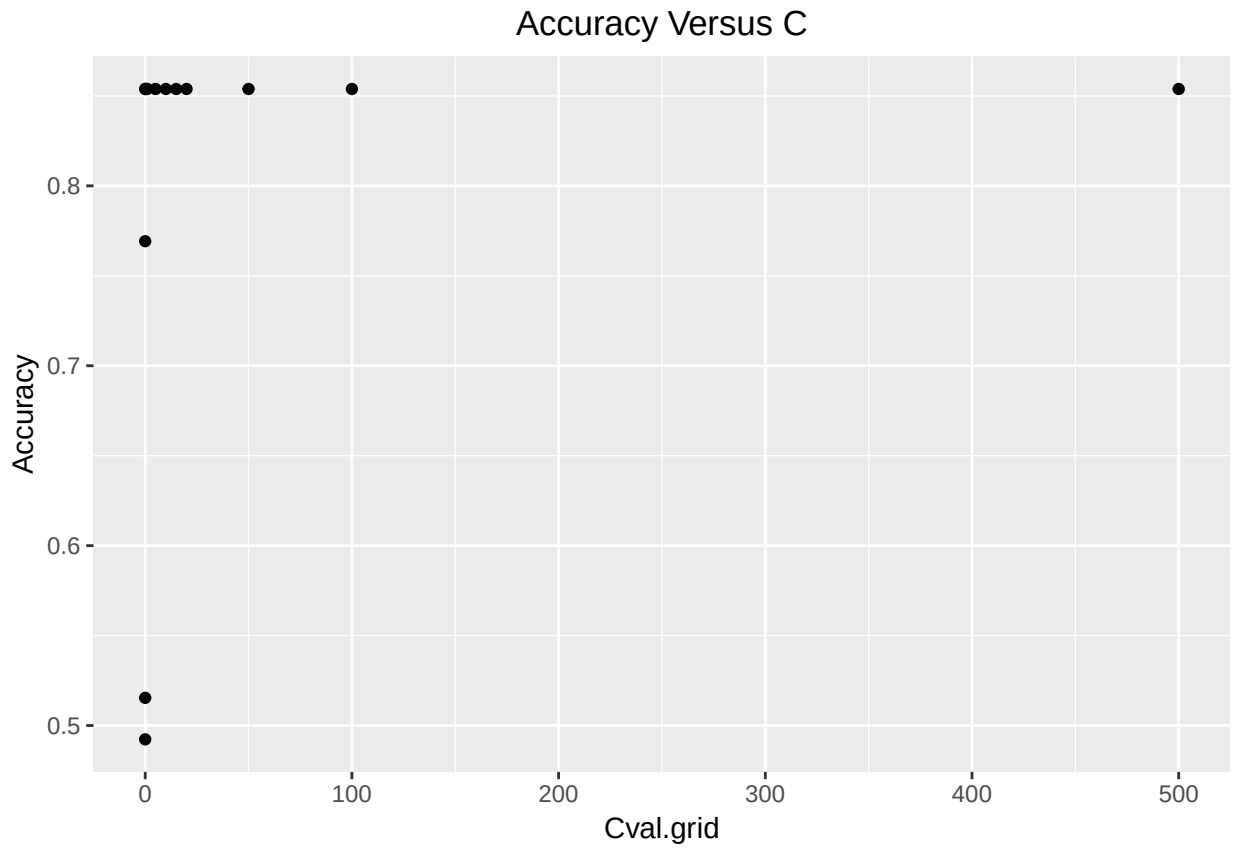
```
x <- model.matrix(R1~.,train )[,-1]

y<-as.factor(train$R1)

xvalid<-model.matrix(R1~.,valid )[,-1]
yvalid<-as.factor(valid$R1)

Accum.Accuracy<-c()
Cval.grid<-c(0.0001,0.0005,.001,0.005,0.1,0.5,1,5 , 10,15,20,50,100,500)

for (Cval in Cval.grid) {
  svp.lin = ksvm(x,y, type="C-svc",scaled = TRUE,kernel =  "vanilladot",kpar=list(),C=Cval)
  ypredict<-predict(svp.lin,newdata=xvalid)
  #Calculate accuracy for this K
  Accuracy<-acc_fun(ypredict,yvalid)
  #Get the accurecy
```

```
  Accum.Accuracy<-c(Accum.Accuracy,Accuracy)

}

Accum.Accuracy
```

```
##  [1] 0.4923077 0.5153846 0.7692308 0.8538462 0.8538462 0.8538462 0.8538462
##  [8] 0.8538462 0.8538462 0.8538462 0.8538462 0.8538462 0.8538462 0.8538462
```

Plot a graph of C Versus accuracy

```
DF<-data.frame(Accuracy=Accum.Accuracy)
ggplot(data=DF,aes(x=Cval.grid,y=Accuracy))+geom_point()+ggtitle("Accuracy Versus C")+theme(plot.title =
```



Max Accuracy

```
max(Accum.Accuracy)
```

```
## [1] 0.8538462
```

C for Max accuracy

```
Cval.grid[which.max(Accum.Accuracy)]
```

```
## [1] 0.005
```

Let's check accuracy on the test set for the chosen model

```
#Train on the entire train set degree
svp.lin = ksvm(x,y, type="C-svc",scaled = TRUE,kernel =  "vanilladot",kpar=list(),C=0.005)

#convert to Matrix
xtest<-model.matrix(R1~.,test )[,-1]
ytest<-as.factor(test$R1)

#Predict
ypredict<-predict(svp.lin,newdata=xtest)
#Calculate accuracy for this K
Accuracy<-acc_fun(ypredict,ytest)


Accuracy
```

```
## [1] 0.8244275
```

Calculate the linear coefficients of the selected model:

```
a <- colSums(svp.lin@xmatrix[[1]] * svp.lin@coef[[1]])
a
```

```
##           A1           A2           A3           A8           A9
##  0.008729903 -0.002994442  0.017272759  0.097587004  0.777673793
##          A10          A11          A12          A14          A15
## -0.078749715  0.112634223  0.006058410 -0.014303467  0.126418269
```

Calculate a0

```
a0<--svp.lin@b
a0
```

```
## [1] -0.0127517
```