Assignment_1

Prashant Kubsad

5/19/2020

Question 2.1

I am working as a application developer at one of the leading Mortgage Insurance firms in the country. We insure mortgage loans to lenders like banks, credit unions when they are issuing mortgages to borrowers. We have to perform independent risk assessment on the loan application to assess the quality of the loan. We consider lot of predictors in assessing the quality of the loan. Few of the predictors are:

- Credit Score
- Borrower Income
- Number of borrowers on the loan application. Applications with 2 borrowers are more favourable compared to single borrower applications.
- DTI(Debt to Income ratio): Ratio of individual's monthly debt to his/her monthly gross income. Monthly debt includes payments like: credit card payments, auto loan payments, alimony, etc., (monthly liabilities/monthly gross income). Lower DTI ratio is favourable.
- History of prior bankruptcy on the borrower
- LTV Ratio (Loan to Value): Ratio of selling price or appraised value to loan amount. (selling price or appraised value/loan amount applied for)
- ZipCode of the property: This information helps us to identify how risky is the property with respect to flood zones, proximity to ocean shore line etc., Higher the risk, higher the premium to cover the risk upto a point. After that break point, the loan application is not favourable.

Question 2.2

2.2.1

The following code helps us determine the co-efficients of the classifier. I am going with fixed value of C=100 for now and later I have demonstrated how different values of C impact the outcome of the model. With C=100, the co efficients of the classifier (a1-a10) are :

```
coefficients of the classifier (a1-a10): -0.001006535 -0.001172905 -0.001626197 0.00300642 1.004941
-0.002825943 0.0002600295 -0.0005349551 -0.001228376 0.1063634
a0:  0.08158492
accuracy:  0.8639144
```

```
#cleaning environment and starting fresh.
rm(list = ls())
library(kernlab)
data<-read.table("../data 2.2/credit_card_data.txt", header=FALSE)
# call ksvm using Vanilladot
model <- ksvm(as.matrix(data[,1:10]),
              as.factor(data[,11]),
              type="C-svc",
              kernel="vanilladot",
              C=100,scaled=TRUE)
```

```
##  Setting default kernel parameters
# calculating a0...am
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
cat("\ncoefficients:",a[1:5])
```

```
##
## coefficients: -0.001006535 -0.001172905 -0.001626197 0.00300642 1.004941
```

```
cat("\n",a[6:10])
```

```
##
##  -0.002825943 0.0002600295 -0.0005349551 -0.001228376 0.1063634
```

```
cat("\n number of support vectors: ",model@nSV)
```

```
##
##  number of support vectors:  189
```

```
# calculate a0
a0 <- -model@b
cat("\na0: ",a0)
```

```
##
## a0:  0.08158492
```

```
# see what the model predicts
pred <- predict(model,data[,1:10])
# see what fraction of the model's predictions match the actual classification
accuracy<-sum(pred == data[,11]) / nrow(data)
cat("\naccuracy: ",accuracy)
```

```
##
## accuracy:  0.8639144
```

The equation we learnt in the lecture f(X) = $a_0 + \sum_{i=1}^{m} x_i * a_i$ will be 0 for the separator which is exactly in the center between the positive and negative hyper planes. Positive hyper plane is the boundary plane beyond which all points are positive outcomes. Similarly negative hyperplane is the boundary plane for negative outcomes. The best classifier lies in the middle of these 2 hyperplanes. So if we scale the given data, pick any row with positive result and apply the above formula, the result will be postive number (usually +1 based on scaling) , any row with negative result will be a negative number. In the console output we see there are 189 support vectors. This means that there 189 rows in the data that are tips of the support vectors which are on the hyper planes that separate the positive and negative outcomes. The parameter C represents the λ explained in the lecture 2.4. With this parameter we can adjust the trade-off we are willing to make between margin and the error.

Experiments with value of C:

I am running the same model for different values of C in (0.0001,10,100,1000,10000,100000). The best accuracy is seen for C = 10 or C= 100 and it starts to drop considerably around C=1000000 or C=0.0001. At these points we have moved the classifier well into positive or negative territory that results in lesser accuray.

```
# creating vector for different values of C
seqC<-c(0.0001,10,100,1000,10000,100000)
#Looping through different values of C
for(c in seqC){
  # call ksvm using Vanilladot
  model <- ksvm(as.matrix(data[,1:10]),
                as.factor(data[,11]),
```

```
                type="C-svc",
                kernel="vanilladot",
                C=c,scaled=TRUE)
  # calculating a0...am
  a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
  # calculate a0
  a0 <- -model@b
  # see what the model predicts
  pred <- predict(model,data[,1:10])
  pred
  # see what fraction of the model's predictions match the actual classification
  accuracy<-sum(pred == data[,11]) / nrow(data)
  cat("\naccuracy: ",accuracy , " for c=",c)
}
```

```
##  Setting default kernel parameters
##
## accuracy:  0.5474006  for c= 1e-04 Setting default kernel parameters
##
## accuracy:  0.8639144  for c= 10 Setting default kernel parameters
##
## accuracy:  0.8639144  for c= 100 Setting default kernel parameters
##
## accuracy:  0.8623853  for c= 1000 Setting default kernel parameters
##
## accuracy:  0.8623853  for c= 10000 Setting default kernel parameters
##
## accuracy:  0.8639144  for c= 1e+05
```

2.2.2. Experiments with different kernels.

Assuming C=100, lets run the same model with different kernel types. I have plotted accuracy v/s kernels bar plot. We can see the accuracy is better for laplace kernel followed by splinedot and rbfdot.

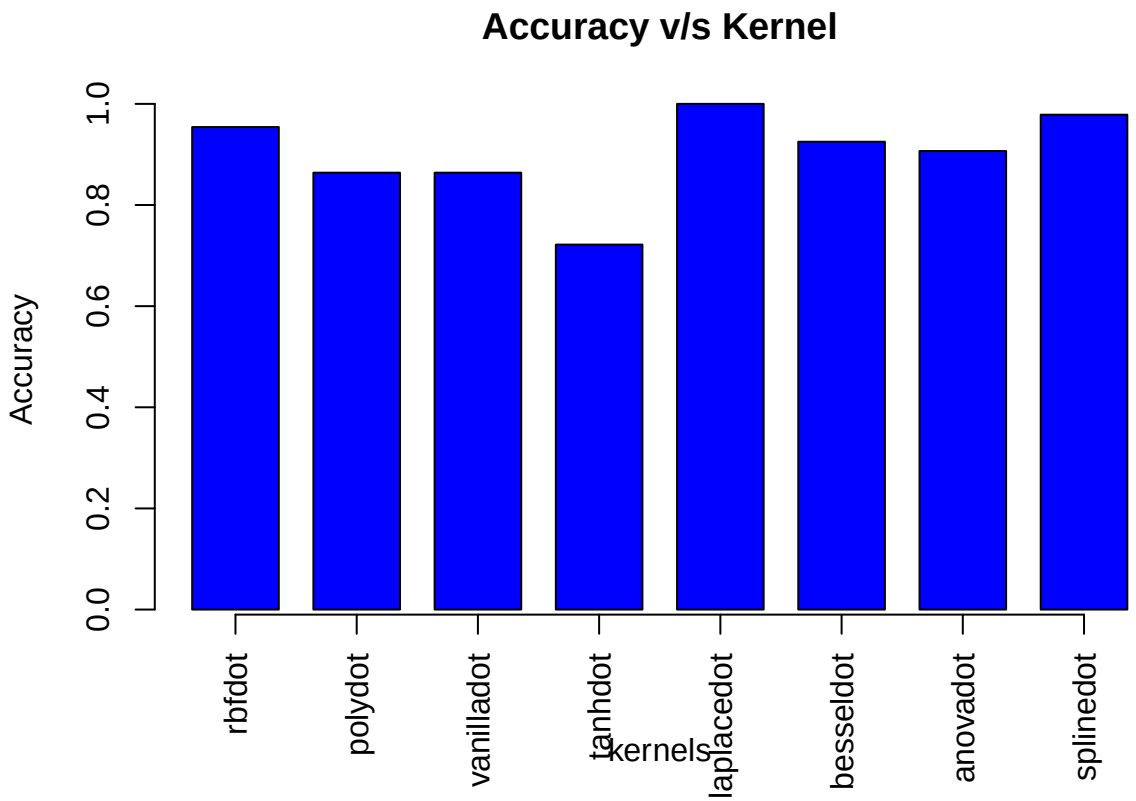
```
library(kernlab)
# creating vector for different values of C
kernelList<-c("rbfdot","polydot","vanilladot","tanhdot","laplacedot",
              "besseldot","anovadot","splinedot")
#initializing list to capture accuracies against each kernel
accuracyList <- vector()
#Looping through different values of C
for(kernel in kernelList){
  # call ksvm using Vanilladot
  model <- ksvm(as.matrix(data[,1:10]),
                as.factor(data[,11]),
                type="C-svc",
                kernel=kernel,
                C=100,
                scaled=TRUE)
  # calculating a0...am
  a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
  # calculate a0
  a0 <- -model@b
  # see what the model predicts
  pred <- predict(model,data[,1:10])
```

```
  pred
  # see what fraction of the model's predictions match the actual classification
  accuracy<-sum(pred == data[,11]) / nrow(data)
  cat("\naccuracy: ",accuracy , " for kernel=",kernel)
  accuracyList<-append(accuracyList,accuracy)
}
```

```
##
## accuracy:  0.9541284  for kernel= rbfdot Setting default kernel parameters
##
## accuracy:  0.8639144  for kernel= polydot Setting default kernel parameters
##
## accuracy:  0.8639144  for kernel= vanilladot Setting default kernel parameters
##
## accuracy:  0.7217125  for kernel= tanhdot
## accuracy:  1  for kernel= laplacedot Setting default kernel parameters
##
## accuracy:  0.9250765  for kernel= besseldot Setting default kernel parameters
##
## accuracy:  0.9067278  for kernel= anovadot Setting default kernel parameters
##
## accuracy:  0.9785933  for kernel= splinedot
```

```
barPlot <- barplot(accuracyList, space=0.4, xaxt='n',  xlab="kernels", ylab="Accuracy",
                   main="Accuracy v/s Kernel", col = "blue")
axis(1,las=2, at=barPlot,space=0.4, labels=kernelList)
```

```
## Warning in axis(1, las = 2, at = barPlot, space = 0.4, labels = kernelList):
## "space" is not a graphical parameter
```



Question 2.2.3

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.

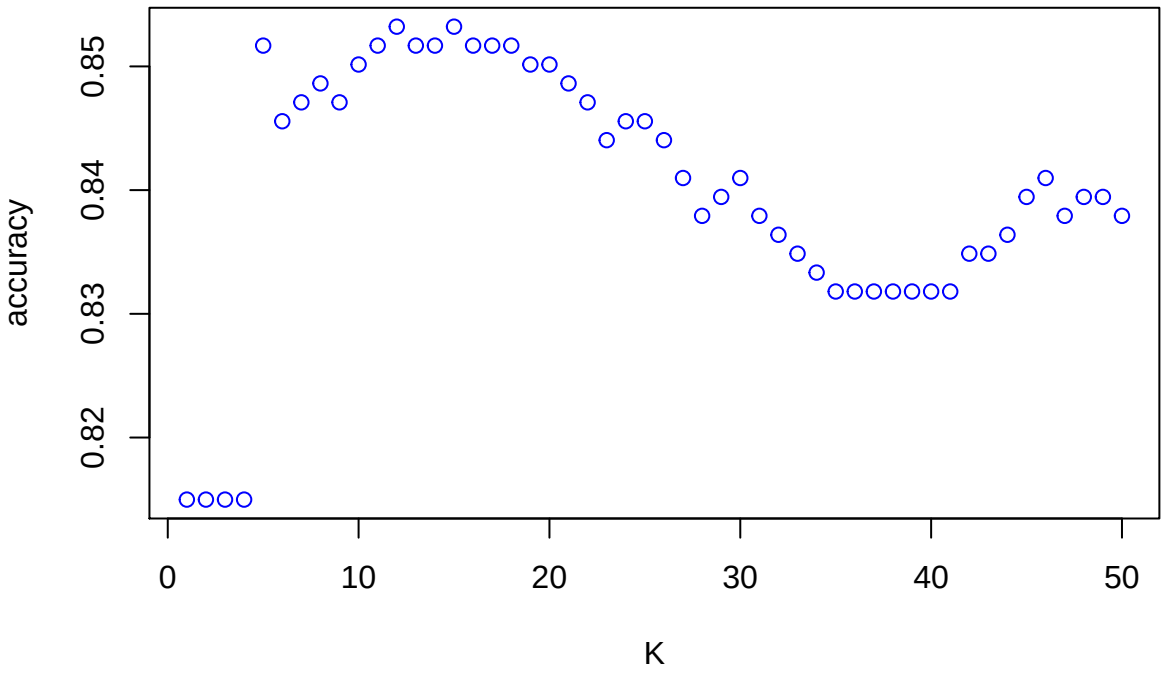
The approach I have employed is to loop over 654 rows of the data and each time using a row(i) to validate the accuracy of the model. This is essentially leave one out cross validation (LOOCV) technique. Essentially we are taking out 1 row and creating a model with the rest. After that we are testin the model's accuracy with that 1 row we left out. I have repeated the process for values of K=0:50. During the process I have captured the accuracy for each value of K and plotted accuracy graph. Based on this logic, i see the best accuracy was observerd when **K= 12 when Distance=2**

```
#cleaning environment and starting fresh.
rm(list = ls())
library(kknn)
# read file into data vector
data<- read.table("../data 2.2/credit_card_data.txt", header=FALSE)
#creating list of K values from 1 to 50.
kList = seq(1,50)
#initializing vector to store accuracies for each value of k
accuracyList <- vector()
#looping for each K
for(k in kList){

  #initializing counter of correct predictions to 0
  countCorrectPrediction=0
  for(i in 1:nrow(data)){
    trainingSet<-data[-i,]
    validationRow<-data[i,]
    model=kknn(formula=V11~.,
               train=trainingSet,
               test = validationRow,
               scale=TRUE,
               kernel = "optimal",
               k=k,
               distance=2)
    #rounding of the predicted values to 0 or 1
    prediction=round(model$fitted.values)
    if(prediction==validationRow[,11]){
      countCorrectPrediction = countCorrectPrediction+1
    }

  }
  accuracy = countCorrectPrediction/nrow(data)
  cat("K: ",k,"\tNumber of correct predictions: ",countCorrectPrediction,
      "\taccuracy = ",accuracy,"\n")
  accuracyList<-append(accuracyList,accuracy)
}
```

```
## K:  1    Number of correct predictions:  533     accuracy =  0.8149847
## K:  2    Number of correct predictions:  533     accuracy =  0.8149847
## K:  3    Number of correct predictions:  533     accuracy =  0.8149847
## K:  4    Number of correct predictions:  533     accuracy =  0.8149847
## K:  5    Number of correct predictions:  557     accuracy =  0.851682
## K:  6    Number of correct predictions:  553     accuracy =  0.8455657
## K:  7    Number of correct predictions:  554     accuracy =  0.8470948
```

```
## K:  8   Number of correct predictions:  555    accuracy =  0.8486239
## K:  9   Number of correct predictions:  554    accuracy =  0.8470948
## K:  10   Number of correct predictions:  556    accuracy =  0.8501529
## K:  11   Number of correct predictions:  557    accuracy =  0.851682
## K:  12   Number of correct predictions:  558    accuracy =  0.853211
## K:  13   Number of correct predictions:  557    accuracy =  0.851682
## K:  14   Number of correct predictions:  557    accuracy =  0.851682
## K:  15   Number of correct predictions:  558    accuracy =  0.853211
## K:  16   Number of correct predictions:  557    accuracy =  0.851682
## K:  17   Number of correct predictions:  557    accuracy =  0.851682
## K:  18   Number of correct predictions:  557    accuracy =  0.851682
## K:  19   Number of correct predictions:  556    accuracy =  0.8501529
## K:  20   Number of correct predictions:  556    accuracy =  0.8501529
## K:  21   Number of correct predictions:  555    accuracy =  0.8486239
## K:  22   Number of correct predictions:  554    accuracy =  0.8470948
## K:  23   Number of correct predictions:  552    accuracy =  0.8440367
## K:  24   Number of correct predictions:  553    accuracy =  0.8455657
## K:  25   Number of correct predictions:  553    accuracy =  0.8455657
## K:  26   Number of correct predictions:  552    accuracy =  0.8440367
## K:  27   Number of correct predictions:  550    accuracy =  0.8409786
## K:  28   Number of correct predictions:  548    accuracy =  0.8379205
## K:  29   Number of correct predictions:  549    accuracy =  0.8394495
## K:  30   Number of correct predictions:  550    accuracy =  0.8409786
## K:  31   Number of correct predictions:  548    accuracy =  0.8379205
## K:  32   Number of correct predictions:  547    accuracy =  0.8363914
## K:  33   Number of correct predictions:  546    accuracy =  0.8348624
## K:  34   Number of correct predictions:  545    accuracy =  0.8333333
## K:  35   Number of correct predictions:  544    accuracy =  0.8318043
## K:  36   Number of correct predictions:  544    accuracy =  0.8318043
## K:  37   Number of correct predictions:  544    accuracy =  0.8318043
## K:  38   Number of correct predictions:  544    accuracy =  0.8318043
## K:  39   Number of correct predictions:  544    accuracy =  0.8318043
## K:  40   Number of correct predictions:  544    accuracy =  0.8318043
## K:  41   Number of correct predictions:  544    accuracy =  0.8318043
## K:  42   Number of correct predictions:  546    accuracy =  0.8348624
## K:  43   Number of correct predictions:  546    accuracy =  0.8348624
## K:  44   Number of correct predictions:  547    accuracy =  0.8363914
## K:  45   Number of correct predictions:  549    accuracy =  0.8394495
## K:  46   Number of correct predictions:  550    accuracy =  0.8409786
## K:  47   Number of correct predictions:  548    accuracy =  0.8379205
## K:  48   Number of correct predictions:  549    accuracy =  0.8394495
## K:  49   Number of correct predictions:  549    accuracy =  0.8394495
## K:  50   Number of correct predictions:  548    accuracy =  0.8379205
```

```
plot(kList,accuracyList,col="blue",xlab="K", ylab="accuracy")
```



Question 3.1

3.1.a

As part of solution to Question 2.2.3 we have implemented LOOCV, for this question i want to use cv.kknn function to do k-fold cross validation on the data set. In k-fold cross validation, we split the data into k number of buckets. We use one bucket as validation set and rest of the buckets as training set. We repeat this process iteratively to make sure every bucket has been used as validation bucket once. cv.kknn() function does this logic for us when parameter *'kcv'* is passed. kcv parameter indicates the number of buckets I want to split my data set into.

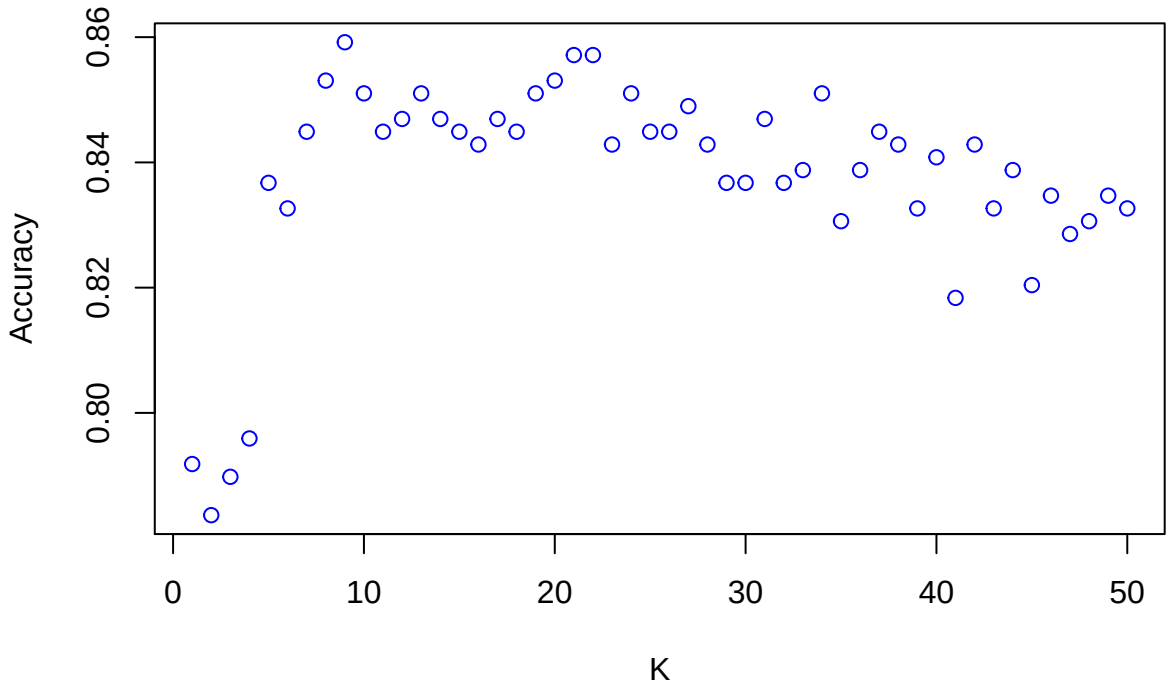
I am splitting overall data into 2 sets - trianing set and testing set. K-fold cross validation is done only on the training set to determine the best parameters for the model. Once we identify the best model, we can test its accuracy on testing set. I am randomizing the rows we pick in for validation set and testing set. If we set the number of partitions for *k-fold cross validation = 10, we find the best K classifier as 9.*

```
#cleaning environment and starting fresh.
rm(list = ls())
set.seed(1)
library(kknn)
# read file into data vector
data<- read.table("../data 2.2/credit_card_data.txt", header=FALSE)

#randomly selecting 75% data for training and 25% for testing.
randomTrainingRows<-sample(1:nrow(data),round(0.75*nrow(data)))
trainingData<-data[randomTrainingRows,]
testingData<-data[-randomTrainingRows,]
#creating list of K values from 1 to 50.
kList = seq(1,50)
#initializing frame to store accuracies for each value of k
accuracyList <- data.frame(k=integer(),accuracy=double())
#looping for each K
  for(k in kList){
    model_cv_kknn<-cv.kknn(formula=V11~.,
```

```
                          data=trainingData,
                          kcv=10,
                          k=k,
                          kernel="optimal",
                          scale=TRUE,
                          distance=2)

    model_vector <- as.data.frame(model_cv_knn)
    accuracy<-sum(round(model_vector[,2])==trainingData[,11])/nrow(trainingData)
    accuracyList[nrow(accuracyList) + 1,] = list(k,accuracy)
  }

bestK<-accuracyList[which.max(accuracyList$accuracy),]$k
bestAccuracy<-accuracyList[which.max(accuracyList$accuracy),]$accuracy
cat("\nbest k= ",bestK," with accuracy=",bestAccuracy)
```

```
##
## best k=  9  with accuracy= 0.8591837
```

```
plot(accuracyList$k,accuracyList$accuracy,col="blue",xlab="K", ylab="Accuracy")
```




```
# Using the best value K determined above, applying the prediction on testing set
modelTesting<-kknn (formula=V11~.,
                train=trainingData,
                test = testingData,
                scale=TRUE,
                kernel = "optimal",
                k=bestK,
                distance=2)
prediction=round(modelTesting$fitted.values)
accuracyTestingSet=sum(prediction==testingData[,11])/nrow(testingData)
cat("\naccuracy of the model on testing set: ",accuracyTestingSet)
```

```
##
## accuracy of the model on testing set:  0.8658537
```

Cross validation in SVM model

We can perform the k-fold cross validation with KSVM model as well by passing *'cross'* paramter. Lets pass the same cross = 10 as we did in KKNN and we see the accuracy in the r output. Both models KKNN and KSVM are showing almost same level of accuracies when run with 10 fold cross validation. As we saw in 2.2.1, 'laplacedot' kernel had best accuracy,lets use that kernel for various values of C. I am storing the models created for each value of C in a vector. Once we determine the best model, am using that model to test accuracy on the testing set.

```
#cleaning environment and starting fresh.
rm(list = ls())
set.seed(1)
library(kernlab)
data<-read.table("../data 2.2/credit_card_data.txt", header=FALSE)
randomTrainingRows<-sample(1:nrow(data),round(0.75*nrow(data)))
trainingData<-data[randomTrainingRows,]
testingData<-data[-randomTrainingRows,]

#initializing a empty vector to store model for each iteration.
modelsList<-list()
# creating vector for different values of C
seqC<-c(10,100,1000)
#Looping through different values of C
for(c in seqC){
# call ksvm using laplacedot kernel.
model <- ksvm(as.matrix(trainingData[,1:10]),
              as.factor(trainingData[,11]),
              type="C-svc",
              kernel="laplacedot",
              C=c,
              scaled=TRUE,
              cross=10)
modelsList<-append(modelsList,model)

# calculating a0...am
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
cat("\ncoefficients:",a[1:5])
cat("\n",a[6:10])
# calculate a0
a0 <- -model@b
cat("\na0: ",a0)
# see what the model predicts
pred <- predict(model,trainingData[,1:10])
# see what fraction of the model's predictions match the actual classification
accuracy<-sum(pred == trainingData[,11]) / nrow(trainingData)
cat("\naccuracy: ",accuracy, "for c=", c)
}
```

```
##
## coefficients: -3.04535 -8.921485 0.5436491 10.92367 50.68629
##  -16.83767 17.08152 -0.1097594 -10.79068 29.73279
## a0:  0.3362978
```

```
## accuracy:  0.944898 for c= 10
## coefficients: -0.8399426 -11.21904 -4.347539 13.22218 44.72737
##  -12.61039 16.316 -0.8226873 -14.04055 32.61956
## a0:  0.3921122
## accuracy:  1 for c= 100
## coefficients: -1.051888 -11.86799 -5.165957 14.04697 48.13727
##  -13.63494 16.98206 -0.9221714 -15.11607 34.60816
## a0:  0.3748356
## accuracy:  1 for c= 1000
```

```
# Models with c=100 and 1000 are resulting in same accuracy,
# lets take model with c=100 and apply it on the testing set.
modelC100=modelsList[[2]]
predTesting <- predict(modelC100,testingData[,1:10])
accuracy<-sum(predTesting == testingData[,11]) / nrow(testingData)
cat("\naccuracy: ",accuracy, "for testing data set")
```

```
##
## accuracy:  0.8658537 for testing data set
```

3.1.b KKNN Model

Lets split our data set into 3 sub sets. I am going with 60% Training, 20% Validation and 20% Testing set. First lets take straight up first 60% rows into training bucket, next 20% into validation bucket and remainig rows into testing bucket. I have also done a code below to point out the difference that can come out if we just change the way we are picking the rows when we split the data.

When we run KKNN model using trainingSet and validationSet we get best *K as 10*. If we run this model agianst the testing set, we get accuracy of *K=10, 91.17%*

```
#cleaning environment and starting fresh.
rm(list = ls())
library(kknn)
#setting seed for consistent results
set.seed(1)
# read file into data vector
data_3_1_b<- read.table("../data 2.2/credit_card_data.txt", header=FALSE)
#splitting the data into 3 sets of 60%,20%,20%.
trainingDataSet <- data_3_1_b[1:392,]
validationDataSet <- data_3_1_b[393:523,]
testingDataSet<- data_3_1_b[523:624,]

#creating list of K values from 1 to 50.
kList = seq(1,50)
#initializing vector to store accuracies for each value of k
accuracyList <- data.frame(k=integer(),accuracy=double())
#looping for each K
for(k in kList){
  #initializing counter of correct predictions to 0
  model=kknn(formula=V11~.,
             train=trainingDataSet,
             test = validationDataSet,
             scale=TRUE,
             kernel = "optimal",
             k=k,
             distance=2)
```

```
  #rounding of the predicted values to 0 or 1
  prediction<-round(model$fitted.values)
  accuracy=sum(prediction==validationDataSet[,11])/nrow(validationDataSet)
  accuracyList[nrow(accuracyList) + 1,] = list(k,accuracy)
}

bestK<-accuracyList[which.max(accuracyList$accuracy),]$k
bestAccuracy<-accuracyList[which.max(accuracyList$accuracy),]$accuracy
cat("\nOn the validation set, best k= ",bestK," with accuracy=",bestAccuracy)
```

```
##
## On the validation set, best k=  10  with accuracy= 0.8549618
```

```
#running the mode on the tesitng set with best value of K = 5
modelTestingSet=kknn(formula=V11~.,
                     train=trainingDataSet,
                     test = testingDataSet,
                     scale=TRUE,
                     kernel = "optimal",
                     k=bestK,
                     distance=2)
#rounding of the predicted values to 0 or 1
predictionTestingSet<-round(modelTestingSet$fitted.values)
accuracyTestingSet <- sum(predictionTestingSet == testingDataSet[,11])/nrow(testingDataSet)
cat("\naccuracy on the testing set: ",accuracyTestingSet,"\n")
```

```
##
## accuracy on the testing set:  0.9117647
```

Now lets see the same above model re-done but with one change, the way we split the data into trainig,validation and testing set. Instead of taking straight up first 60%, 20% and remaining 20%, if we randomize the way we pick rows to put into each bucket, we see best K value changes to 5. It is interesting that small change in one of the input parameters can change the classifier. This make is paramount that we try our best to randomize the data and reduce bias.

```
#cleaning environment and starting fresh.
rm(list = ls())
library(kknn)
#setting seed for consistent results
set.seed(1)
# read file into data vector
data_3_1_b<- read.table("../data 2.2/credit_card_data.txt", header=FALSE)
#splitting the data into 3 sets of 60%,20%,20% randomising the selection of the data.
ss <- sample(1:3,size=nrow(data_3_1_b),replace=TRUE,prob=c(0.6,0.2,0.2))
trainingDataSet <- data_3_1_b[ss==1,]
validationDataSet <- data_3_1_b[ss==2,]
testingDataSet <- data_3_1_b[ss==3,]

#creating list of K values from 1 to 50.
kList = seq(1,50)
#initializing vector to store accuracies for each value of k
accuracyList <- data.frame(k=integer(),accuracy=double())
#looping for each K
for(k in kList){
  #initializing counter of correct predictions to 0
  model=kknn(formula=V11~.,
```

```
              train=trainingDataSet,
              test = validationDataSet,
              scale=TRUE,
              kernel = "optimal",
              k=k,
              distance=2)
  #rounding of the predicted values to 0 or 1
  prediction<-round(model$fitted.values)
  accuracy=sum(prediction==validationDataSet[,11])/nrow(validationDataSet)
  accuracyList[nrow(accuracyList) + 1,] = list(k,accuracy)
}

bestK<-accuracyList[which.max(accuracyList$accuracy),]$k
bestAccuracy<-accuracyList[which.max(accuracyList$accuracy),]$accuracy
cat("\nOn the validation set, best k= ",bestK," with accuracy=",bestAccuracy)
```

```
##
## On the validation set, best k=  5  with accuracy= 0.8473282
```

```
#running the mode on the tesitng set with best value of K = 5
modelTestingSet=kknn(formula=V11~.,
                     train=trainingDataSet,
                     test = testingDataSet,
                     scale=TRUE,
                     kernel = "optimal",
                     k=bestK,
                     distance=2)
#rounding of the predicted values to 0 or 1
predictionTestingSet<-round(modelTestingSet$fitted.values)
accuracyTestingSet <- sum(predictionTestingSet == testingDataSet[,11])/nrow(testingDataSet)
cat("\naccuracy on the testing set: ",accuracyTestingSet,"\n")
```

```
##
## accuracy on the testing set:  0.8139535
```

3.1.b KSVM Model

Lets run same split on KSVM model. We see that the accuracy is same for C=10,100,1000. Lets take C=100 and run the prediction on the testing set and see that accuracy is around 81.39%

```
#cleaning environment and starting fresh.
rm(list = ls())
library(kknn)
#setting seed for consistent results
set.seed(1)
# read file into data vector
data_3_1_b_ksvm<- read.table("../data 2.2/credit_card_data.txt", header=FALSE)
#splitting the data into 3 sets of 60%,20%,20% randomising the selection of the data.
ss <- sample(1:3,size=nrow(data_3_1_b_ksvm),replace=TRUE,prob=c(0.6,0.2,0.2))
trainingDataSet <- data_3_1_b_ksvm[ss==1,]
validationDataSet <- data_3_1_b_ksvm[ss==2,]
testingDataSet <- data_3_1_b_ksvm[ss==3,]

seqC<-c(10,100,1000)
#Looping through different values of C
```

```
for(c in seqC){
  # call ksvm using Vanilladot
  model <- ksvm(as.matrix(trainingDataSet[,1:10]),
                as.factor(trainingDataSet[,11]),
                type="C-svc",
                kernel="vanilladot",
                C=c,
                scaled=TRUE)
  # calculating a0...am
  a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
  # calculate a0
  a0 <- -model@b
  # see what the model predicts
  pred <- predict(model,validationDataSet[,1:10])
  pred
  # see what fraction of the model's predictions match the actual classification
  accuracy<-sum(pred == validationDataSet[,11]) / nrow(validationDataSet)
  cat("\naccuracy: ",accuracy , " for c=",c)
}
```

```
##  Setting default kernel parameters
##
## accuracy:  0.8473282  for c= 10 Setting default kernel parameters
##
## accuracy:  0.8473282  for c= 100 Setting default kernel parameters
##
## accuracy:  0.8473282  for c= 1000
```

```
#running the model on the tesitng set with best value of C = 100
cat("\n Since all the values returned same accuracy, lets apply C=100 on testing set")
```

```
##
##  Since all the values returned same accuracy, lets apply C=100 on testing set
```

```
modelTestingKsvm <- ksvm(as.matrix(trainingDataSet[,1:10]),
                         as.factor(trainingDataSet[,11]),
                         type="C-svc",
                         kernel="vanilladot",
                         C=100,scaled=TRUE)
```

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

```
pred <- predict(modelTestingKsvm,testingDataSet[,1:10])
# see what fraction of the model's predictions match the actual classification
accuracyTestingKsvm<-sum(pred == testingDataSet[,11]) / nrow(testingDataSet)
cat("\naccuracy: ",accuracyTestingKsvm , " on testing dataset")
```

```
##
## accuracy:  0.8139535  on testing dataset
```