**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.

**Answer:**

I once worked in a financial firm based in Dallas Texas as an Analytics Consultant providing Analytics support to a line of business called as Post-Closing (also known as Post-Funding). To understand why a classification model will be appropriate here, I would first give an understanding of what Post-Closing means.

**Post-Closing Overview:** When a borrower want to purchase a house and he/she applies for a home loan with a bank or a money lending firm. The bank approves the loan and funds the borrower. Once the loan is funded by the bank, the bank decides to sell this loan to another firm (known as Investor) with small profit margin so that the bank selling the loan receives more money to fund more loans. This process of bank selling a funded loan to an investor is called Post-Closing.

When a bank tries to sell this loan, it has to provide all the documents to the investor and the document must be in the correct format with the correct data as expected by the investor. If there are issues with errors in the documents then these issues are called as Defects Whenever a Defect is detected, it needs to be fixed before an investor finalizes the purchase. However, there are certain defects which cannot be fixed and as a result those loans either cannot be sold or has to be sold at a heavy loss. Such loans which cannot be sold are called as ‘Scratch & Dent’ loans.

A classification model would be appropriate for the scenario mentioned above to separate regular loans with ‘Scratch & Dent’ Loans. To check if a loan would become a Scratch & Dent loan, it is dependent on the following predictors.

1. **Channel**: A loan can come through 4 channels. They are

* Wholesale – Where a bank gets bulk of loans from a wholesale realtor
* Retail – Where loan originates from a retail medium.
* Correspondent – Where loan originates from another local bank later purchased.
* Consumer Direct – Where loan originates from the borrower directly applying loan online.

For all the above channels, we get different form of documentation leading to different defects

1. **Loan Type**: A borrower can be eligible for different types of loans based on different parameters like income, job status etc. Different types of loans are USDA loans (for farmers or people in agriculture), VA loans (for veterans), FHA loans etc.
2. **Number of Defects:** As the number of defects increases, the chances of it being not fixed also increases which makes a loan fall into scratch and dent category.
3. **State:** Different states in USA have different laws for granting of loans. Most of the laws per state may be same but there may be only few laws which change causing a loan to fall in the scratch and dent category.

**Question 2.2.1**

The files credit\_card\_data.txt (without headers) and credit\_card\_data-headers.txt (with headers) contain a dataset with 654 data points, 6 continuous and 4 binary predictor variables. It has anonymized credit card applications with a binary response variable (last column) indicating if the application was positive or negative. The dataset is the “Credit Approval Data Set” from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Credit+Approval) without the categorical variables and without data points that have missing values.

1. Using the support vector machine function ksvm contained in the R package kernlab, find a good classifier for this data. Show the equation of your classifier, and how well it classifies the data points in the full data set. (Don’t worry about test/validation data yet; we’ll cover that topic soon.)

**Answer:** Given below is my analysis done in order to find a good classifier. As mentioned in the HW assignment questions, I used the KSVM function from the kernlab library. In order to calculate a linear classifier the kernel I used is vanilladot.

Here are the steps I performed to get a good classifier

**Step 1:**

Data from credit\_card\_data-headers.txt file needs to be imported into a matrix in R environment. Given below is the code to do that

*ccdata = read.table("D://MS Georgia Tech/Introduction to Analytics/HW1/credit\_card\_data-headers.txt", header = T, sep = '\t')*

**Step 2:**

Kernlab package needs to be installed and the library needs to be called. Given below is the code for it

*install.packages("kernlab")*

*library(kernlab)*

*set.seed(42)*

**Step 3:**

KSVM function is used to calculate the classifier.

Given below is my analysis

In order to get a linear classifier, I used the kernel named vanilladot (also mentioned in HW) and iterated through various values of C using the formula. Given below are the accuracies of the model for different values of c

Here I used a FOR loop to iterate different values of C from 0.1 to 100000 with an increment of 0.01 and stored the results in a vector called results

**C = 0.1 to 0.99 with increment of 0.01**

**CODE:**

*myC = seq(0.1,0.99, by=0.01)*

*results=c()*

*for(i in 1:length(myC)){*

*model <- ksvm(as.matrix(ccdata[,1:10]),as.factor(ccdata[,11]),type="C-svc",kernel="vanilladot",C= i,scaled=TRUE)*

*# calculate a1.am*

*a <- colSums(model@xmatrix[[1]] \* model@coef[[1]])*

*# calculate a0*

*a0 <- -model@b*

*# see what the model predicts*

*pred <- predict(model,ccdata[,1:10])*

*# see what fraction of the model's predictions match the actual classification*

*#results[i]=data.table(sum(pred == ccdata[,11]) / nrow(ccdata))*

*results[i]=sum(pred == ccdata[,11]) / nrow(ccdata)*

*}*

*results*

**OUTPUT:**

[1] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[12] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[23] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[34] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[45] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[56] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[67] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[78] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[89] 0.8639144 0.8639144

**C = 0 to 100 with increment of 1**

**CODE:**

*myC = seq(0,100, by=1)*

*results=c()*

*for(i in 1:length(myC)){*

*model <- ksvm(as.matrix(ccdata[,1:10]),as.factor(ccdata[,11]),type="C-svc",kernel="vanilladot",C= i,scaled=TRUE)*

*# calculate a1.am*

*a <- colSums(model@xmatrix[[1]] \* model@coef[[1]])*

*# calculate a0*

*a0 <- -model@b*

*# see what the model predicts*

*pred <- predict(model,ccdata[,1:10])*

*# see what fraction of the model's predictions match the actual classification*

*#results[i]=data.table(sum(pred == ccdata[,11]) / nrow(ccdata))*

*results[i]=sum(pred == ccdata[,11]) / nrow(ccdata)*

*}*

*results*

**OUTPUT:**

results

[1] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[12] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[23] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[34] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[45] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[56] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[67] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[78] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[89] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[100] 0.8639144 0.8639144

**C = 100 to 1000 with increment of 100**

**CODE:**

*myC = seq(100,1000, by=100)*

*results=c()*

*for(i in 1:length(myC)){*

*model <- ksvm(as.matrix(ccdata[,1:10]),as.factor(ccdata[,11]),type="C-svc",kernel="vanilladot",C= i,scaled=TRUE)*

*# calculate a1.am*

*a <- colSums(model@xmatrix[[1]] \* model@coef[[1]])*

*# calculate a0*

*a0 <- -model@b*

*# see what the model predicts*

*pred <- predict(model,ccdata[,1:10])*

*# see what fraction of the model's predictions match the actual classification*

*#results[i]=data.table(sum(pred == ccdata[,11]) / nrow(ccdata))*

*results[i]=sum(pred == ccdata[,11]) / nrow(ccdata)*

*}*

*Results*

**OUTPUT:**

results

[1] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

**C = 1000 to 100000 with increment of 1000**

**CODE:**

*myC = seq(100,1000, by=100)*

*results=c()*

*for(i in 1:length(myC)){*

*model <- ksvm(as.matrix(ccdata[,1:10]),as.factor(ccdata[,11]),type="C-svc",kernel="vanilladot",C= i,scaled=TRUE)*

*# calculate a1.am*

*a <- colSums(model@xmatrix[[1]] \* model@coef[[1]])*

*# calculate a0*

*a0 <- -model@b*

*# see what the model predicts*

*pred <- predict(model,ccdata[,1:10])*

*# see what fraction of the model's predictions match the actual classification*

*#results[i]=data.table(sum(pred == ccdata[,11]) / nrow(ccdata))*

*results[i]=sum(pred == ccdata[,11]) / nrow(ccdata)*

*}*

*Results*

**OUTPUT:**

results

[1] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[12] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[23] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[34] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[45] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[56] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[67] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[78] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[89] 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144 0.8639144

[100] 0.8639144

Based on my analysis above, I calculated the accuracy for different values of C from 0.1 to 100,000 and I find the accuracy to be constant for all the values i.e. accuracy = 0.8639144. This mean the model is approximately 86% accurate for all different values of C ranging from 0.1 to 100,000 .

Since I have to present an equation of the classifier, I randomly choose a value of C as 100 and present my equation as shown below.

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

**Question 2.2.2**

The files credit\_card\_data.txt (without headers) and credit\_card\_data-headers.txt (with headers) contain a dataset with 654 data points, 6 continuous and 4 binary predictor variables. It has anonymized credit card applications with a binary response variable (last column) indicating if the application was positive or negative. The dataset is the “Credit Approval Data Set” from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Credit+Approval) without the categorical variables and without data points that have missing values.

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

**Answer:** Given below is my analysis for different non-linear Kernels for different values of C. I will be using different kernels "rbfdot","polydot","tanhdot","laplacedot","besseldot","anovadot" and "splinedot" in my code. The code will iterate through each kernel for a value of C. I will run this code individually for 4 values of C and provide my analysis.

**C = 1**

**CODE:**

*for(i in 1:length(myKernels)){*

*# call ksvm using kernel instead of linear*

*model <- ksvm(as.matrix(ccdata[,1:10]),as.factor(ccdata[,11]),type="C-svc",kernel=myKernels[[i]],C=1,scaled=TRUE)*

*# calculate a1.am*

*a <- colSums(model@xmatrix[[1]] \* model@coef[[1]])*

*a*

*# calculate a0*

*a0 <- -model@b*

*a0*

*# see what the model predicts*

*pred <- predict(model,ccdata[,1:10])*

*pred*

*# see what fraction of the model's predictions match the actual classification*

*results[[i]]=data.table(kernel=myKernels[[i]],accuracy=sum(pred == ccdata[,11]) / nrow(ccdata))*

*}*

*Results*

**OUTPUT:**

results

[[1]]

kernel accuracy

1: rbfdot 0.8700306

[[2]]

kernel accuracy

1: polydot 0.8639144

[[3]]

kernel accuracy

1: tanhdot 0.7217125

[[4]]

kernel accuracy

1: laplacedot 0.8639144

[[5]]

kernel accuracy

1: besseldot 0.8685015

[[6]]

kernel accuracy

1: anovadot 0.8639144

[[7]]

kernel accuracy

1: splinedot 0.9663609

**ANALYSIS:**

Looking at the output above, for value of C = 1 the kernel “splinedot” provides highest accuracy of 96.63%. The kernels ebfdot, polydot, and splinedot have a better accuracy that vanilladot for C = 1.

**C = 100**

**CODE:**

*for(i in 1:length(myKernels)){*

*# call ksvm using kernel instead of linear*

*model <- ksvm(as.matrix(ccdata[,1:10]),as.factor(ccdata[,11]),type="C-svc",kernel=myKernels[[i]],C=100,scaled=TRUE)*

*# calculate a1.am*

*a <- colSums(model@xmatrix[[1]] \* model@coef[[1]])*

*a*

*# calculate a0*

*a0 <- -model@b*

*a0*

*# see what the model predicts*

*pred <- predict(model,ccdata[,1:10])*

*pred*

*# see what fraction of the model's predictions match the actual classification*

*results[[i]]=data.table(kernel=myKernels[[i]],accuracy=sum(pred == ccdata[,11]) / nrow(ccdata))*

*}*

*Results*

**OUTPUT:**

results

[[1]]

kernel accuracy

1: rbfdot 0.9587156

[[2]]

kernel accuracy

1: polydot 0.8639144

[[3]]

kernel accuracy

1: tanhdot 0.7217125

[[4]]

kernel accuracy

1: laplacedot 1

[[5]]

kernel accuracy

1: besseldot 0.9250765

[[6]]

kernel accuracy

1: anovadot 0.9067278

[[7]]

kernel accuracy

1: splinedot 0.9785933

**ANALYSIS:**

The kernel laplacedot has the best accuracy of 100 % for C = 100. All the kernels except tanhdot has provided better accuracy than vanilladot.

**C = 1000**

**CODE:**

*for(i in 1:length(myKernels)){*

*# call ksvm using kernel instead of linear*

*model <- ksvm(as.matrix(ccdata[,1:10]),as.factor(ccdata[,11]),type="C-svc",kernel=myKernels[[i]],C=1000,scaled=TRUE)*

*# calculate a1.am*

*a <- colSums(model@xmatrix[[1]] \* model@coef[[1]])*

*a*

*# calculate a0*

*a0 <- -model@b*

*a0*

*# see what the model predicts*

*pred <- predict(model,ccdata[,1:10])*

*pred*

*# see what fraction of the model's predictions match the actual classification*

*results[[i]]=data.table(kernel=myKernels[[i]],accuracy=sum(pred == ccdata[,11]) / nrow(ccdata))*

*}*

*Results*

**OUTPUT:**

results

[[1]]

kernel accuracy

1: rbfdot 0.9847095

[[2]]

kernel accuracy

1: polydot 0.8623853

[[3]]

kernel accuracy

1: tanhdot 0.7217125

[[4]]

kernel accuracy

1: laplacedot 1

[[5]]

kernel accuracy

1: besseldot 0.9204893

[[6]]

kernel accuracy

1: anovadot 0.9067278

[[7]]

kernel accuracy

1: splinedot 0.9785933

**ANALYSIS:**

The kernel laplacedot has the best accuracy of 100 % for C = 100. All the kernels except tanhdot has provided better accuracy than vanilladot. This can also mean that the model is overfitted since we have not divided it into testing and training dataset.

**Question 2.2.3**

The files credit\_card\_data.txt (without headers) and credit\_card\_data-headers.txt (with headers) contain a dataset with 654 data points, 6 continuous and 4 binary predictor variables. It has anonymized credit card applications with a binary response variable (last column) indicating if the application was positive or negative. The dataset is the “Credit Approval Data Set” from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Credit+Approval) without the categorical variables and without data points that have missing values.

1. 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).

**Answer:**

As mentioned in the HW assignment questions, I used the KKNN function from the KKNN library.

I used the value of K as 10.

Given below are the steps I performed

**Step 1:**

KKNN package needs to be installed and the library needs to be called. Given below is the code for it

*install.packages("KKNN")*

*library(KKNN)*

**Step 2:**

In order to suggest a good value of k, so that it classifies the data point with high accuracy I created a for loop that loops through multiple values of K for all the data points in the dataset.

Given below is the code for it.

**CODE:**

*#Initiating a vector*

*results=c()*

*# Looping the value of k from 1 to 20 neighbours*

*for (j in 1:20){*

*total\_accurate\_values = 0*

*#looping through all datapoints*

*for (i in 1:654){*

*CCmodel = kknn(R1~A1+A2+A3+A8+A9+A10+A11+A12+A14+A15,*

*ccdata[-i,],*

*ccdata[i,],*

*k = j,*

*distance = 2,*

*kernel = 'optimal',*

*scale = TRUE)*

*rounded\_fitted\_value = round(fitted.values(CCmodel))*

*actual\_value = ccdata[i,11]*

*if (actual\_value == rounded\_fitted\_value) {*

*total\_accurate\_values = total\_accurate\_values+1*

*}*

*# see what fraction of the model's predictions match the actual classification*

*accuracy = total\_accurate\_values/654*

*}*

*results[j] = accuracy*

*}*

*results*

**OUTPUT:**

results

[1] 0.8149847 0.8149847 0.8149847 0.8149847 0.8516820 0.8455657 0.8470948 0.8486239 0.8470948 0.8501529 0.8516820

[12] 0.8532110 0.8516820 0.8516820 0.8532110 0.8516820 0.8516820 0.8516820 0.8501529 0.8501529

**ANALYSIS:**

Looking at the output, accuracy appears to be highest for K = 12 and K = 15 with value of 0.8532110 which is 85.32%