2.1

For the past 3 years staff in Hyderabad, India have turned over at an explosive rate. This impacts all teams with Salesforce developers. One group has lost 12 individuals in a 3-year timeframe that have needed to be replaced. This causes issues with continuity in development resources and increased expenses in recruitment and training. It seems as if as soon as a developer becomes proficient, within 2 years they leave for another role.

Predictors to determine who may be more willing to leave:

1) Salary
2) Skills
3) Gender
4) Number of years at level
5) Organizational Change (manager change, layoffs)
6) Geographic Change (new company moving into the area)
7) Number of past jobs

2.2

This assignment started out rough due to having to absorb so much about R, find the right dataset and learn the mechanics of SVM as I build.

Observations-

- Importing the right dataset and ensuring that it is prepared in advance. I eventually decided upon utilizing the dataset with headers for ease of reading on my part (credit_card_data-headers.txt).
    - I initially selected the wrong dataset and learned about NA (missing data errors).
- I chose to utilize the read_table function to import the csv dataset. I also played with RStudio importing the credit card data under the Global Environment import dataset tool. .
- I converted the dataset into the matrix format first based on the recommendation of the homework but, subsequently determined that this conversion was more efficient based on the use of vector data in the ksvm function for classification purposes. The data.table format is a list and matricies are numeric. Since the data in the credit file was numeric the general teaching is that this is a more efficient binding mechanism. However, I learned from looking at object.size() that this seems to only be true when there are large numbers of columns of data. My data.table was smaller at 38632 bytes than the matrix at 53448 bytes.
- Changing C from 100 to 1000 did very little (really almost nothing) to change the predictions of 1 and 0s. It wasn't until I went down to 1e6 .0000001 and up to 1000000 that I saw significant differences in the prediction model. 100 up to 1000 showed the same distribution of yes and nos. When I went down to .0000001 all of the predictors went to 0s or nos. Basically the smaller number took every applicant to a no, trained for a larger margin and risks misclassifying a significant number of data points. Going larger in the value of C provided a smaller margin which lessened the risk of misclassification

and provided a more diverse set of 1 and 0s for yes and nos. In fact, there are more yes values than with the lower numbers.

- The fraction for matching the actual classification is as follows:. C=100 is 86%, C=1000000 is 63% and for C=.0000001 it is 54%. I believe that this shows that C=100 gives a pretty solid margin to minimize misclassifications. Neither significantly larger or smaller provided a greater level of accuracy.

Code:

```
# add kernlab
library(kernlab)
# Set the working directory
setwd("~/Documents/ISYE6501 Intro to Analytics Modeling")
# Read the data in
matrix1 <- read.table("credit_card_data-headers.csv",header = TRUE, sep =",")
x <- as.matrix(matrix1[,1:10])
y <- as.factor(matrix1[,11])

modelvanilla <- ksvm(x,y,type="C-svc",kernel="vanilladot",C=100,scaled=TRUE)
modelvanilla
# calculate a1…am
a <- colSums(modelvanilla@xmatrix[[1]] * modelvanilla@coef[[1]])
a
# calculate a0
a0 <- modelvanilla@b
a0
# see what the model predicts
pred <- predict(modelvanilla,matrix1[,1:10])
pred
# see what fraction of the model's predictions match the actual classification
sum(pred == matrix1[,11]) / nrow(matrix1)


#Change the value of C for vanilladot
modelCvanilla <- ksvm(x,y,type="C-svc",kernel="vanilladot",C=.0000001,scaled=TRUE)
modelCvanilla
# calculate a1…am
a <- colSums(modelCvanilla@xmatrix[[1]] * modelCvanilla@coef[[1]])
a
# calculate a0
a0 <- modelCvanilla@b
a0
# see what the model predicts
pred <- predict(modelCvanilla,matrix1[,1:10])
pred
# see what fraction of the model's predictions match the actual classification
```

```
sum(pred == matrix1[,11]) / nrow(matrix1)
```

Output C=100, vanilladot

```
> # add kernlab
> library(kernlab)
> # Set the working directory
> setwd("~/Documents/ISYE6501 Intro to Analytics Modeling")
> # Read the data in
> matrix1 <- read.table("credit_card_data-headers.csv",header = TRUE, sep =",")
> x <- as.matrix(matrix1[,1:10])
> y <- as.factor(matrix1[,11])
> modelvanilla <- ksvm(x,y,type="C-svc",kernel="vanilladot",C=100,scaled=TRUE)
 Setting default kernel parameters
> modelvanilla
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 100

Linear (vanilla) kernel function.

Number of Support Vectors : 189

Objective Function Value : -17887.92
Training error : 0.136086
> # calculate a1…am
> a <- colSums(modelvanilla@xmatrix[[1]] * modelvanilla@coef[[1]])
> a
           A1            A2            A3            A8            A9
-0.0010065348 -0.0011729048 -0.0016261967  0.0030064203  1.0049405641
          A10           A11           A12           A14           A15
-0.0028259432  0.0002600295 -0.0005349551 -0.0012283758  0.1063633995
> # calculate a0
> a0 <- modelvanilla@b
> a0
[1] -0.08158492
> # see what the model predicts
> pred <- predict(modelvanilla,matrix1[,1:10])
> pred
  [1] 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [39] 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
 [77] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[115] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[153] 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
[191] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
[229] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
[267] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0
[305] 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
[343] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[381] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[419] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[457] 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[495] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[533] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
[571] 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[609] 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[647] 0 0 0 0 0 0 0 0
Levels: 0 1
> # see what fraction of the model's predictions match the actual classification
> sum(pred == matrix1[,11]) / nrow(matrix1)
[1] 0.8639144
>
```

Output C = .0000001 vanilladot

```
> #Change the value of C for vanilladot
> modelCvanilla <- ksvm(x,y,type="C-svc",kernel="vanilladot",C=.0000001,scaled=TRUE)
 Setting default kernel parameters
> modelCvanilla
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1e-07

Linear (vanilla) kernel function.

Number of Support Vectors : 592

Objective Function Value : -1e-04
Training error : 0.452599
> # calculate a1…am
> a <- colSums(modelCvanilla@xmatrix[[1]] * modelCvanilla@coef[[1]])
> a
           A1            A2            A3            A8            A9
-2.159778e-07  6.865502e-06  9.372931e-06  1.802907e-05  4.127070e-05
          A10           A11           A12           A14           A15
-2.416310e-05  2.334032e-05 -1.002168e-06 -3.290839e-06  1.011249e-05
> # calculate a0
> a0 <- modelCvanilla@b
> a0
[1] 0.9999043
> # see what the model predicts
> pred <- predict(modelCvanilla,matrix1[,1:10])
> pred
  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [39] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [77] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[115] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[153] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[191] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[229] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[267] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[305] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[343] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[381] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[419] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[457] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[495] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[533] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[571] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[609] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[647] 0 0 0 0 0 0 0 0
Levels: 0 1
> # see what fraction of the model's predictions match the actual classification
> sum(pred == matrix1[,11]) / nrow(matrix1)
[1] 0.5474006
>
```

- I decided to change from a linear kernel (vanilladot) to a Gaussian kernel (rbfdot). Gaussian kernels provide linear separation for datasets with higher dimensions that are non-linear in nature. The credit card data set doesn't necessarily align with that assumption but, is a good way to look at the differences in the calculations for the kernel types. The accuracy goes up to 95% in this model.

Code

```
#use a Gaussian kernel to determine the difference
modelrbfdot <- ksvm(x,y,type="C-svc",kernel="rbfdot", C = 100, scaled=TRUE)
modelrbfdot
# calculate a1…am
a <- colSums(modelrbfdot@xmatrix[[1]] * modelrbfdot@coef[[1]])
a
# calculate a0
a0 <- modelrbfdot@b
a0
```

```
# see what the model predicts
pred <- predict(modelrbfdot,matrix1[,1:10])
pred
# see what fraction of the model's predictions match the actual classification
sum(pred == matrix1[,11]) / nrow(matrix1)
```

Output C = 100 rbfdot

```
#use a Gaussian kernel to determine the difference
> modelrbfdot <- ksvm(x,y,type="C-svc",kernel="rbfdot", C = 100, scaled=TRUE)
> modelrbfdot
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 100

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0807760358405504

Number of Support Vectors : 245

Objective Function Value : -9872.054
Training error : 0.051988
> # calculate a1…am
> a <- colSums(modelrbfdot@xmatrix[[1]] * modelrbfdot@coef[[1]])
> a
       A1        A2        A3        A8        A9       A10       A11
-18.695611 -42.720954  -8.473312  60.251139  46.324897 -18.983328   4.474938
      A12       A14       A15
-23.464467 -58.124801  46.771610
> # calculate a0
> a0 <- modelrbfdot@b
> a0
[1] -0.9530265
> # see what the model predicts
> pred <- predict(modelrbfdot,matrix1[,1:10])
> pred
  [1] 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [39] 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
 [77] 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1
[115] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[153] 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[191] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1
[229] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
[267] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
```

```
[305] 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[343] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[381] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[419] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[457] 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[495] 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[533] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1
[571] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[609] 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[647] 0 0 0 0 0 0 0 0
Levels: 0 1
> # see what fraction of the model's predictions match the actual classification
> sum(pred == matrix1[,11]) / nrow(matrix1)
[1] 0.9480122
>
```

2.2.3
kknn K nearest-neighbors

- In my code I used k = 25 as the square root of 654 (total observations) for observation 250 to find the nearest neighbor value. Based on research that I did the square root is typically used and in my code this proved to be valid. The predictor is 0.00097 which if I used rounding would be 0 which is the actual R1 value of observation 250 in the credit card dataset. The ranges in data points for the various columns (A1-A10) require scaling thus scale was set to TRUE.

Code

```
# use the kknn model for k nearest neighbor
# example is just the base knn code

library(kknn)
#create a function for creating the data for prediction
i = 250
# I used k = 25 as the square root of 654
modelknn = kknn(R1~.,
            matrix1[-i,],
            matrix1[i,],
            k=25,
            distance=2,
            kernel = "optimal",
            scale = TRUE)
fitted.values(modelknn)
matrix1[i,11]
```

Output
```
> # Set the working directory
> setwd("~/Documents/ISYE6501 Intro to Analytics Modeling")
> # Read the data in
> matrix1 <- read.table("credit_card_data-headers.csv",header = TRUE, sep =",")
> library(kknn)
> #create a function for creating the data for prediction
> i = 250
> # I used k = 25 as the square root of 654
> modelknn = kknn(R1~.,
+              matrix1[-i,],
+              matrix1[i,],
+              k=25,
+              distance=2,
+              kernel = "optimal",
+              scale = TRUE)
> fitted.values(modelknn)
[1] 0.0009704286
> matrix1[i,11]
[1] 0
> View(matrix1)
>
```

- The next coding work created a loop of I to go through the entire dataset and evaluate the k nearest-neighbor for all 654 data points. The output displays the accuracy of the predictions to the actual R1 data points.

Code
```
#next attempt is to create a loop
library(kknn)
setwd("~/Documents/ISYE6501 Intro to Analytics Modeling")
# Read the data in
matrix1 <- read.table("credit_card_data-headers.csv",header = TRUE, sep =",")
numberofobservations <- nrow(matrix1)
pred <- vector(length = numberofobservations)
for (i in 1:numberofobservations) {
  modelknn = kknn(R1~.,
              matrix1[-i,],
              matrix1[i,],
              k=25,
              distance=2,
              kernel = "optimal",
              scale = TRUE)
  pred[i] <- fitted.values(modelknn)
}
```

```
pred
for (i in 1:numberofobservations){
  cat("Observation:",i, "Value", matrix1[i,11], "\n")
}
```

Output (due to the length I am only displaying the first 100 values of pred and the original dataset R1 versus all 654 values)

```
 pred
[1] 0.8772352179 0.9020230096 0.4046900866 0.7259354127 0.6456467266
  [6] 0.4123371035 0.9758486082 0.6574054442 0.5019971527 0.4110426338
 [11] 0.0422411825 0.5064185931 0.6165558207 0.0050074880 0.9218822406
 [16] 0.9015827653 0.8738412110 0.8342785295 0.4641460408 0.8558864147
 [21] 0.9664722991 0.4067692401 0.9112665805 0.9115865825 0.7919095190
 [26] 0.8350175926 0.9302637892 0.9808560963 0.9272263294 0.9654664439
 [31] 0.7766476947 1.0000000000 0.9883613211 0.6759376061 0.9593711054
 [36] 0.9278902914 0.9715133525 0.8425763773 0.8654931106 0.9617598855
 [41] 0.9651308784 1.0000000000 0.8622170296 0.8646534502 1.0000000000
 [46] 0.9410883929 0.9664856697 0.9644960153 0.0000000000 0.0345335561
 [51] 0.3562141507 0.4924543156 0.5795179847 0.3912350793 0.6057088664
 [56] 0.5159164663 0.4781015681 0.4668179350 0.0792733309 1.0000000000
 [61] 0.8336863849 0.9856497682 0.6367940838 0.8727580179 0.7278336867
 [66] 0.9830145559 0.8966368705 0.9825006853 0.8979144953 0.5676625597
 [71] 0.5168899172 0.5695454686 0.6591031608 0.1033059796 0.8675193444
 [76] 0.5306724566 0.4812572501 0.9207835338 0.5977037903 0.6642462599
 [81] 0.5572637698 0.4518655482 0.5698104774 0.6528287601 0.6584903602
 [86] 0.4907860758 0.5419142474 0.5533358656 0.4842377789 0.6024450072
 [91] 0.5635780944 0.5555493136 0.4838986182 0.5089434661 0.6140148391
 [96] 0.4969212790 0.4873562946 0.8530943302 0.8692185395 0.8998025347
> for (i in 1:numberofobservations){
+   cat("Observation:",i, "Value", matrix1[i,11], "\n")
+ }
Observation: 1 Value 1
Observation: 2 Value 1
Observation: 3 Value 1
Observation: 4 Value 1
Observation: 5 Value 1
Observation: 6 Value 1
Observation: 7 Value 1
Observation: 8 Value 1
Observation: 9 Value 1
Observation: 10 Value 1
Observation: 11 Value 1
Observation: 12 Value 1
Observation: 13 Value 1
Observation: 14 Value 1
```

Observation: 15 Value 1
Observation: 16 Value 1
Observation: 17 Value 1
Observation: 18 Value 1
Observation: 19 Value 1
Observation: 20 Value 1
Observation: 21 Value 1
Observation: 22 Value 1
Observation: 23 Value 1
Observation: 24 Value 1
Observation: 25 Value 1
Observation: 26 Value 1
Observation: 27 Value 1
Observation: 28 Value 1
Observation: 29 Value 1
Observation: 30 Value 1
Observation: 31 Value 1
Observation: 32 Value 1
Observation: 33 Value 1
Observation: 34 Value 1
Observation: 35 Value 1
Observation: 36 Value 1
Observation: 37 Value 1
Observation: 38 Value 1
Observation: 39 Value 1
Observation: 40 Value 1
Observation: 41 Value 1
Observation: 42 Value 1
Observation: 43 Value 1
Observation: 44 Value 1
Observation: 45 Value 1
Observation: 46 Value 1
Observation: 47 Value 1
Observation: 48 Value 1
Observation: 49 Value 1
Observation: 50 Value 1
Observation: 51 Value 1
Observation: 52 Value 1
Observation: 53 Value 1
Observation: 54 Value 1
Observation: 55 Value 1
Observation: 56 Value 1
Observation: 57 Value 1
Observation: 58 Value 1

```
Observation: 59 Value 1
Observation: 60 Value 1
Observation: 61 Value 1
Observation: 62 Value 1
Observation: 63 Value 1
Observation: 64 Value 1
Observation: 65 Value 1
Observation: 66 Value 1
Observation: 67 Value 1
Observation: 68 Value 1
Observation: 69 Value 1
Observation: 70 Value 1
Observation: 71 Value 0
Observation: 72 Value 0
Observation: 73 Value 0
Observation: 74 Value 0
Observation: 75 Value 0
Observation: 76 Value 0
Observation: 77 Value 0
Observation: 78 Value 0
Observation: 79 Value 0
Observation: 80 Value 0
Observation: 81 Value 0
Observation: 82 Value 0
Observation: 83 Value 0
Observation: 84 Value 0
Observation: 85 Value 0
Observation: 86 Value 0
Observation: 87 Value 0
Observation: 88 Value 0
Observation: 89 Value 0
Observation: 90 Value 0
Observation: 91 Value 0
Observation: 92 Value 0
Observation: 93 Value 0
Observation: 94 Value 0
Observation: 95 Value 0
Observation: 96 Value 0
Observation: 97 Value 0
Observation: 98 Value 0
Observation: 99 Value 0
Observation: 100 Value 0
```

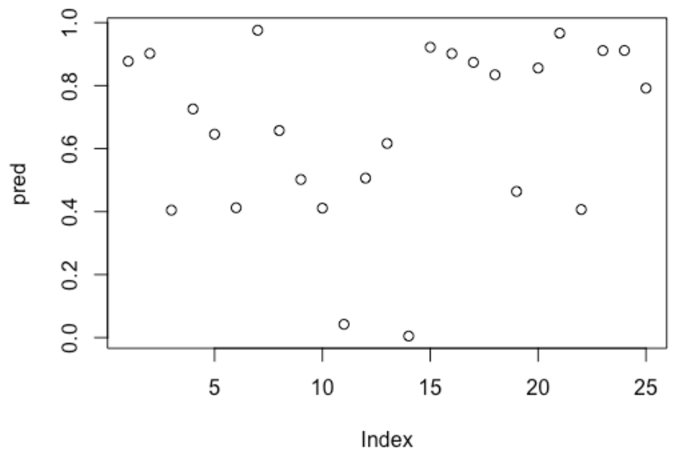
- The final coding work with kknn took values of k from 1 to 25 and plotted the resulting prediction vector.

Code

```
#next code is to evaluate k from 1 to 25 and plot the values
library(kknn)
setwd("~/Documents/ISYE6501 Intro to Analytics Modeling")
# Read the data in
matrix1 <- read.table("credit_card_data-headers.csv",header = TRUE, sep =",")
numberofobservations <- 25
pred <- vector(length = numberofobservations)
for (i in 1:numberofobservations) {
  #model.knn <- train.kknn(matrix1[-i,11]~., data=data[-i,1:10], kmax=7, scale=T)
  modelknn = kknn(R1~.,
                matrix1[-i,],
                matrix1[i,],
                k=25,
                distance=2,
                kernel = "optimal",
                scale = TRUE)
  pred[i] <- fitted.values(modelknn)
}
plot(pred)
max(pred)
min(pred)
```

Output

```
plot(pred)
> max(pred)
[1] 0.9758486
> min(pred)
[1] 0.005007488
```


pred
0.0
0.2
0.4
0.6
0.8
1.0
5
10
15
20
25
Index