ISYE-6501
Fall 2024

Homework 1

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.

As a quality data analyst at a hospital, I have been tasked with assessing readmission rates within 30 days. The reason is because a readmission within 30 days of last discharge would signify that the care provided at the initial visit was not sufficient, or something was missed. Hospitals try to keep their "readmission within 30 days" rates at a minimum to ensure that we provide the best care during the first visit so that the patient does not have to return.

With that being said, here are 6 predictors:

- Age (Older patients tend to have more chronic diseases)
- Admission Diagnosis (The reason for admission can determine)
- Length of stay (Higher length of stay tends to signify more severe health issues)
- Pre-existing Conditions (Diseases such as diabetes can lead to other complications)
- Discharge Planning (Poor discharge instructions to patient can cause readmission)
- Socio-economic status (Lack of transport and unstable housing leads to higher difficulty to manage your health)

Question 2.2.1

The files credit_card_data.txt (without headers) and credit_card_data-headers.txt (with headers) contain a data set 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 data set is the "Credit Approval Data Set" from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/data sets/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.)

SOLUTION and ANALYSIS:

The classifier equation that I was able to gather using the scaled data is as follows:

-0.0010065348v1 - 0.0011729048v2 - 0.0016261967v3 + 0.0030064203v4 + 1.0049405641v5 - 0.0028259432v6 + 0.0002600295v7 - 0.0005349551v8 - 0.0012283758v9 + 0.1063633995v10 + 0.08158492 = 0

The training error: 0.136086 or 13.6086%
The accuracy of the classifier: 0.8639144 or 86.39144%

After running the two models with and without scaled data, it was found that the model with scaled data was more accurate than that with unscaled data. (86.39% for scaled and 72.17% for unscaled)

Please find below the code and results for ksvm function using scaled data vs unscaled data. The code input can be found in black and the code output or results can be found in blue.

SCALED DATA:

```
> library(kernlab)
> data <- read.table("credit_card_data.txt", header=FALSE)
> head(data, 10)
   V1    V2     V3    V4 V5 V6 V7 V8  V9   V10 V11
1   1 30.83  0.000 1.250  1  0  1  1 202     0   1
2   0 58.67  4.460 3.040  1  0  6  1  43   560   1
3   0 24.50  0.500 1.500  1  1  0  1 280   824   1
4   1 27.83  1.540 3.750  1  0  5  0 100     3   1
5   1 20.17  5.625 1.710  1  1  0  1 120     0   1
6   1 32.08  4.000 2.500  1  1  0  0 360     0   1
7   1 33.17  1.040 6.500  1  1  0  0 164 31285   1
8   0 22.92 11.585 0.040  1  1  0  1  80  1349   1
9   1 54.42  0.500 3.960  1  1  0  1 180   314   1
10  1 42.50  4.915 3.165  1  1  0  0  52  1442   1
> model <-
ksvm(as.matrix(data[,1:10]),as.factor(data[,11]),type="C-svc",kernel="vanilladot",C=100,scaled=TRUE)
 Setting default kernel parameters
> model
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

```
> a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
> a
          V1            V2            V3            V4            V5            V6            V7            V8 
-0.0010065348 -0.0011729048 -0.0016261967  0.0030064203  1.0049405641 -0.0028259432  
0.0002600295 -0.0005349551 
          V9           V10 
-0.0012283758  0.1063633995 
> a0<- -model@b
> a0
[1] 0.08158492
> pred <- predict(model,data[,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 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1
1
 [57] 1 1 0 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 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
[113] 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 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
1
[169] 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 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
1
[225] 1 1 1 1 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 0 1 0 0 0 0 0 0 0 0 0 0 0
0
[281] 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 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
[337] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
[393] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
[449] 0 0 0 0 0 0 0 0 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 1 1 1 1 1 1 1 1 1
1
[505] 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 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
1
[561] 1 1 1 1 0 1 1 1 1 1 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 0 0 0 0 0 0 0
0
[617] 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 0 0 0 0 0 0 0 0
Levels: 0 1
```

```
> sum(pred == data[,11]) / nrow(data)
[1] 0.8639144
```

UNSCALED DATA:

```
> library(kernlab)
> data <- read.table("credit_card_data.txt", header=FALSE)
> head(data, 10)
   V1    V2     V3    V4 V5 V6 V7 V8  V9   V10 V11
1   1 30.83  0.000 1.250  1  0  1  1 202     0   1
2   0 58.67  4.460 3.040  1  0  6  1  43   560   1
3   0 24.50  0.500 1.500  1  1  0  1 280   824   1
4   1 27.83  1.540 3.750  1  0  5  0 100     3   1
5   1 20.17  5.625 1.710  1  1  0  1 120     0   1
6   1 32.08  4.000 2.500  1  1  0  0 360     0   1
7   1 33.17  1.040 6.500  1  1  0  0 164 31285   1
8   0 22.92 11.585 0.040  1  1  0  1  80  1349   1
9   1 54.42  0.500 3.960  1  1  0  1 180   314   1
10  1 42.50  4.915 3.165  1  1  0  0  52  1442   1
> model <-
ksvm(as.matrix(data[,1:10]),as.factor(data[,11]),type="C-svc",kernel="vanilladot",C=100,scaled=FALSE)
 Setting default kernel parameters
> model
Support Vector Machine object of class "ksvm"

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

Linear (vanilla) kernel function.

Number of Support Vectors : 186

Objective Function Value : -2213.731
Training error : 0.278287
> a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
> a
         V1          V2          V3          V4          V5          V6          V7          V8
-0.0483050561 -0.0083148473 -0.0836550114  0.1751121271  1.8254844547  0.2763673361
0.0654782414 -0.1108211169
         V9         V10
-0.0047229653 -0.0007764962
> a0<- -model@b
> a0
```

```
[1] 0.5255393
> pred <- predict(model,data[,1:10])
> pred
  [1] 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1
1
 [57] 1 1 0 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 0 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 1 1 1 1
1
[113] 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 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 0 1 0 0 1 1 1 0 1 1 0 1 0 1 1
1
[169] 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1
0
[225] 1 1 0 0 0 1 0 1 0 1 0 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0
[281] 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
0
[337] 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0
0
[393] 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0
0
[449] 0 0 0 1 1 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 0 1
1
[505] 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 0 1 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 1
1
[561] 0 0 0 1 0 1 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
0
[617] 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1
Levels: 0 1
> sum(pred == data[,11]) / nrow(data)
[1] 0.7217125
```

Question 2.2.2

2. 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.

SOLUTION and ANALYSIS:

Using rbfdot instead of vanilla dot I was able to reach an accuracy of **95.1%.**

SCALED RBFDOT DATA:

```
> data <- read.table("credit_card_data.txt", header=FALSE)
> head(data, 10)
   V1    V2     V3    V4 V5 V6 V7 V8  V9   V10 V11
```

```
1   1 30.83  0.000 1.250  1  0  1  1 202    0   1
2   0 58.67  4.460 3.040  1  0  6  1  43   560   1
3   0 24.50  0.500 1.500  1  1  0  1 280   824   1
4   1 27.83  1.540 3.750  1  0  5  0 100     3   1
5   1 20.17  5.625 1.710  1  1  0  1 120     0   1
6   1 32.08  4.000 2.500  1  1  0  0 360     0   1
7   1 33.17  1.040 6.500  1  1  0  0 164 31285   1
8   0 22.92 11.585 0.040  1  1  0  1  80  1349   1
9   1 54.42  0.500 3.960  1  1  0  1 180   314   1
10  1 42.50  4.915 3.165  1  1  0  0  52  1442   1
> model <-
ksvm(as.matrix(data[,1:10]),as.factor(data[,11]),type="C-svc",kernel="rbfdot",C=100,scaled=TRUE)
> model
Support Vector Machine object of class "ksvm"

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

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0880414800035819

Number of Support Vectors : 244

Objective Function Value : -9380.79
Training error : 0.04893
> a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
> a
      V1       V2       V3       V4       V5       V6       V7       V8       V9      V10
-19.47073 -39.17016  -8.94696  57.04756  48.80274 -22.44310  10.53147 -22.95705 -59.29066  49.19924
> a0<- -model@b
> a0
[1] 0.8610477
> pred <- predict(model,data[,1:10])
> pred
  [1] 1 1 0 1 1 0 1 1 1 1 0 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 1 1 1 1 0 0 1 0 1 0 1
0
 [57] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0
0
[113] 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
1
[169] 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1
1
[225] 1 1 1 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 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
[281] 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 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
[337] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
```

```
[393] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
[449] 0 0 0 0 0 0 0 0 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 1 1 0 0 0 0 0 0 0
0
[505] 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 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
[561] 1 1 1 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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
[617] 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 0 0 0 0 0 0 0 0
Levels: 0 1
> sum(pred == data[,11]) / nrow(data)
[1] 0.9510703
```

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).

<u>SOLUTION and ANALYSIS:</u>
After running the for loop for k values 1-25 as seen in the code below, the highest accuracy was found at **k=12 with an accuracy of 85.32%.**

<u>SCALED DATA:</u>

```
> library(kknn)
> data <- read.table("credit_card_data.txt", header=FALSE)
> head(data, 10)
   V1    V2     V3    V4 V5 V6 V7 V8  V9   V10 V11
1   1 30.83  0.000 1.250  1  0  1  1 202     0   1
2   0 58.67  4.460 3.040  1  0  6  1  43   560   1
3   0 24.50  0.500 1.500  1  1  0  1 280   824   1
4   1 27.83  1.540 3.750  1  0  5  0 100     3   1
5   1 20.17  5.625 1.710  1  1  0  1 120     0   1
6   1 32.08  4.000 2.500  1  1  0  0 360     0   1
7   1 33.17  1.040 6.500  1  1  0  0 164 31285   1
8   0 22.92 11.585 0.040  1  1  0  1  80  1349   1
9   1 54.42  0.500 3.960  1  1  0  1 180   314   1
10  1 42.50  4.915 3.165  1  1  0  0  52  1442   1
> check_accuracy = function(X){
+   predicted <- rep(0,(nrow(data)))
+   for (i in 1:nrow(data)){
+     model=kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10,data[-i,],data[i,],k=X, scale = TRUE)
```

```
+     predicted[i] <- as.integer(fitted(model)+0.5)
+   }
+   accuracy = sum(predicted == data[,11]) / nrow(data)
+   return(accuracy)
+ }
> acc <- rep(0,25)
> for (X in 1:25){
+   acc[X] = check_accuracy(X)
+ }
> acc
 [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
0.8486239 0.8470948
[23] 0.8440367 0.8455657 0.8455657
> max(acc)
[1] 0.853211
> match(max(acc), acc)
[1] 12
```