Introduction to Analytics Modeling Homework 1

Code

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: Working as a pharmacy technician I realize many situations in which a classification model can be used. The most common situation would be to answer whether a customer can receive his/her medication on that same day. A few predictors we use are: if the prescription is clear of errors(dosage, doctor, etc), we have the medication and the necesary quantity in stock and if the customer has decided on a method of payment.

Question 2.2: 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.)

Hide

```
library(kernlab) #used for ksvm
library(kknn) #used for knn
```

Hide

```
#load the data
cc_data <- read.table("credit_card_data.txt", header=F, stringsAsFactors = F)
#see the first few rows
head(cc_data)
```

	V1 <int>	V2 <dbl>	V3 <dbl>	V4 <dbl>	V5 <int>	V6 <int>	V7 <int>	V8 <int>	V9 <int>
1	1	30.83	0.000	1.25	1	0	1	1	202
2	0	58.67	4.460	3.04	1	0	6	1	43
3	0	24.50	0.500	1.50	1	1	0	1	280
4	1	27.83	1.540	3.75	1	0	5	0	100
5	1	20.17	5.625	1.71	1	1	0	1	120
6	1	32.08	4.000	2.50	1	1	0	0	360

6 rows | 1-10 of 11 columns

Hide

```
#set up model
cc_model <- ksvm(as.matrix(cc_data[,1:10]),as.factor(cc_data[,11]), C=100, scaled = T, kernel="v
anilladot", type = "C-svc")
```

```
 Setting default kernel parameters
```

Hide

```
#take a look at it
cc_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
```

After trying multiple values of C I saw no significant difference on the error. Therefore, I decided to keep the value of C at 100.

Hide

```
#calculating the coefficients (a1..am)
a <- colSums(cc_model@xmatrix[[1]] * cc_model@coef[[1]] )
#print out a
a
```

```
           V1            V2            V3            V4            V5            V6
-0.0010065348 -0.0011729048 -0.0016261967  0.0030064203  1.0049405641 -0.0028259432
           V7            V8            V9           V10
 0.0002600295 -0.0005349551 -0.0012283758  0.1063633995
```

Hide

```
#calculate a0
a0 <- -cc_model@b
#print a0
a0
```

```
[1] 0.08158492
```

The classifier's equation is then: -0.00100653481057611v1 - 0.00117290480611665v2 - 0.00162619672236963v3 + 0.0030064202649194v4 + 1.00494056410556v5 - 0.00282594323043472v6 + 0.000260029507016313v7 - 0.000534955143494997v8 - 0.00122837582291523v9 + 0.106363399527188v10 + 0.081584921659538v0 = 0

Hide

```
#lets see what the model predicts
pred <- predict(cc_model, cc_data[,1:10])
pred
```

```
  [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
 [44] 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 0 1 1 1 1 1 1 1 1
 [87] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[130] 1 1 1 1 1 1 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
[173] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[216] 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 0 0 0 0 0 0 0 0 0 0
[259] 1 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 1 0 0 0 0 0
[302] 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 1 0 0 0 0 0 0 0 0
[345] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[388] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[431] 0 0 0 0 0 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 1 1 1 1 1 1
[474] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[517] 1 1 1 1 1 1 1 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
[560] 1 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 1 0 0 0 0 0 0 0 0 0 0
[603] 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 0
[646] 0 0 0 0 0 0 0 0 0 0
Levels: 0 1
```

Hide

```
#lets observe the percentage of the model's correct predictions.
sum(pred == cc_data[,11]) / nrow(cc_data) * 100
```

```
[1] 86.39144
```

The model's accuracy is 86.39%.

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

Hide

```
acc_chk = function(Z){
  pred<- rep(0,(nrow(cc_data)))

  for (i in 1:nrow(cc_data)){
  #ensure it doesn't use i itself
    knn_model=kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10,cc_data[-i,],cc_data[i,],k=Z, scale = T)
    pred[i] <- as.integer(fitted(knn_model)+0.5) #for rounding
  }

  acc = sum(pred == cc_data[,11]) / nrow(cc_data)
  return(acc)
}
```

Hide

```
test_vec <- rep(0,30) # 30 zeroes vector for accuracy test (knn values ranging from 1 to 30)
for (Z in 1:30){
  test_vec[Z] = acc_chk(Z)
}
```

Hide

```
knn_accuracy <- as.matrix(test_vec * 100) #see accuracy as percentage
knn_accuracy #print out knn values and percentage of accuracy
```

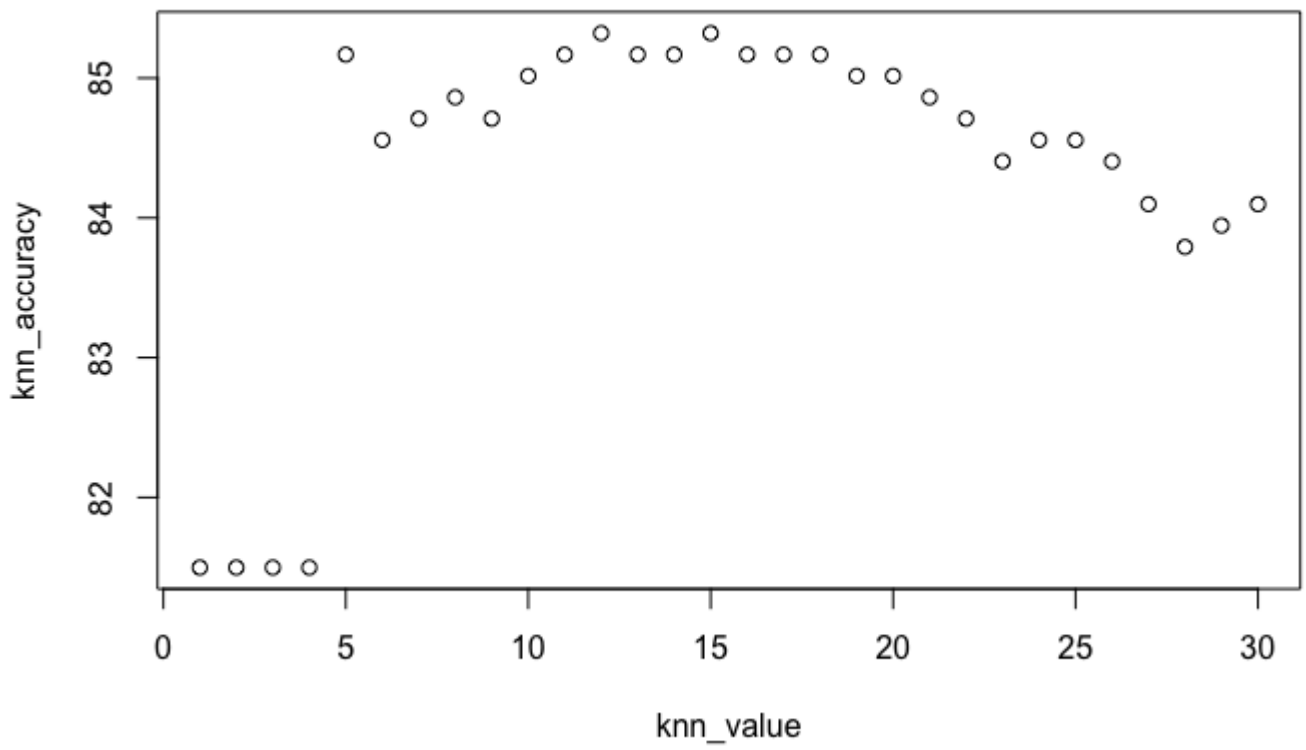
```
         [,1]
 [1,] 81.49847
 [2,] 81.49847
 [3,] 81.49847
 [4,] 81.49847
 [5,] 85.16820
 [6,] 84.55657
 [7,] 84.70948
 [8,] 84.86239
 [9,] 84.70948
[10,] 85.01529
[11,] 85.16820
[12,] 85.32110
[13,] 85.16820
[14,] 85.16820
[15,] 85.32110
[16,] 85.16820
[17,] 85.16820
[18,] 85.16820
[19,] 85.01529
[20,] 85.01529
[21,] 84.86239
[22,] 84.70948
[23,] 84.40367
[24,] 84.55657
[25,] 84.55657
[26,] 84.40367
[27,] 84.09786
[28,] 83.79205
[29,] 83.94495
[30,] 84.09786
```

Hide

```
knn_value <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30)
```

Hide

```
plot(knn_value,knn_accuracy)#observe accuracies per knn value
```



Hide

```
max(knn_accuracy)
```

```
[1] 85.3211
```

The knn value that best classifies the data points is 12 with an accuracy of 85.3211%.