Homework 1

Richard Albright
ISYE6501
Spring 2018

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 2.1

I work for a company that analyzes insider trading data for consumption by financial institutions. We want to know if mimicking the insiders trading history, is profitable over different holding periods(6 months, 1 year, 2 years) vs an index. Insiders hold different positions in each company, and can sit on the Board of Directors. They trade sometimes with a pre-written plan under the SEC 10b5-1 safe harbor provisions. There are various aspects of an insider and their trading activity that can be categorized to determine if a soecific category of insider and their trading activity is better at predicting a profitable trade over the holding period.

Question 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. 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.)
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.
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).

Answer 2.2.1

Read in the CSV

```
data <-
  read.table(
    "/Users/ralbright/Documents/ISYE6501/week1/credit_card_data-headers.txt",
    header=TRUE,
    sep="\t"
  )
```

Lets check if the data loaded.

```r
tail(data)
```

```
##     A1    A2     A3   A8 A9 A10 A11 A12 A14 A15 R1
## 649  1 40.58  3.290 3.50  0   1   0   0 400   0  0
## 650  1 21.08 10.085 1.25  0   1   0   1 260   0  0
## 651  0 22.67  0.750 2.00  0   0   2   0 200 394  0
## 652  0 25.25 13.500 2.00  0   0   1   0 200   1  0
## 653  1 17.92  0.205 0.04  0   1   0   1 280 750  0
## 654  1 35.00  3.375 8.29  0   1   0   0   0   0  0
```

Convert data to a matrix and set up our training variables for use in the ksvm function trainX is our predictors trainY is our response

```r
matrix = as.matrix(data)
trainX <- matrix[,1:10]
trainY <- matrix[,11]
```

We want step through a wide range of C values to see what values generate the best accuracy in the model lets skip every odd exponent just to hurry things along

linear model C values and their accuracy

```r
results
```

```
##        [,1]      [,2]
##  [1,] 1e-10 0.5474006
##  [2,] 1e-08 0.5474006
##  [3,] 1e-06 0.5474006
##  [4,] 1e-04 0.5474006
##  [5,] 1e-02 0.8639144
##  [6,] 1e+00 0.8639144
##  [7,] 1e+02 0.8639144
##  [8,] 1e+04 0.8623853
##  [9,] 1e+06 0.6253823
## [10,] 1e+08 0.6636086
## [11,] 1e+10 0.4923547
```

C values between 0.01 and 10000 yield the same accuracy and the best results.

Lets use the highest C value with the best results, which is $C = 10^2 = 100$

```r
cexp = 2
```

Lets run the model 1st with a C value of 100 1st

```r
model <- ksvm(
  trainX,
  trainY,
  type="C-svc",
  kernel="vanilladot",
  C=10^cexp,
  scaled=TRUE
)
```

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

```r
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
```

Lets get our a coeefficients

```r
a <- colSums(model@xmatrix[[1]] * model@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
```

```r
# a0
a0 <- model@b
a0
```

```
## [1] -0.08158492
```

The model equation for scaled variable a:

$-0.0010065348a_1$ - $0.0011729048a_2$ - $0.0016261967a_3 + 0.0030064203a_4 + 1.0049405641a_5$ - $0.0028259432a_6 + 0.0002600295a_7$ - $0.0005349551a_8$ - $0.0012283758a_9 + 0.1063633995a_{10}$ - $0.08158492 = 0$

Lets see what the model predicts

```r
predictions <- predict(model,trainX)
predictions
```

```
##   [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
##  [36] 1 1 1 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
##  [71] 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
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [141] 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 1 1 1
## [176] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [211] 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
## [246] 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
## [316] 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
## [351] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [386] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [421] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [456] 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
## [491] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [526] 1 1 1 1 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
## [596] 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 0 0
## [631] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

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

```
accuracy = sum(predictions == data[,11]) / nrow(data)
accuracy
```

```
## [1] 0.8639144
```

Answer 2.2.2

we'll use Gaussian(rbfdot) kernel this time instead

lets use exponents of 10 that match the best results of our linear model above and loop through them

```
cexpvalues <- seq(-2, 4, 1)
results = matrix(, nrow = length(cexpvalues), ncol = 2)
i=1

for (c in cexpvalues) {
  model <- ksvm(
    trainX,
    trainY,
    type="C-svc",
    kernel="rbfdot",
    C=10^c,
    scaled=TRUE
  )
  pred <- predict(model,trainX)
  accuracy = sum(pred == trainY) / nrow(data)

  # put the models cvalue in column 1
  results[i,1] = 10^c
  # put the models accuracy in column 2
  results[i,2] = accuracy
  #increment i for the next results write
  i=i+1
}
```

Gaussian model kernel C values and their accuracy

```
results
```

```
##       [,1]      [,2]
## [1,] 1e-02 0.5611621
## [2,] 1e-01 0.8593272
## [3,] 1e+00 0.8730887
## [4,] 1e+01 0.9097859
## [5,] 1e+02 0.9510703
## [6,] 1e+03 0.9801223
## [7,] 1e+04 0.9938838
```

Lets see the model summary at C=100, which we used in the linear model.

```
model <- ksvm(
  trainX,
  trainY,
  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 =  0.101980417490808
##
## Number of Support Vectors : 244
##
## Objective Function Value : -8516.101
## Training error : 0.042813
```

The training error for the Gaussian model kernel is much smaller than the linear one.

Answer 2.2.3

Lets move on to the K nearest neighbor algorithm

Run the kkn model

```
ksteps = seq(1,50)
results = matrix(, nrow = length(ksteps), ncol = 1)


for (s in ksteps) {
  predictions <- rep(0,(nrow(data)))
  for (i in 1:nrow(data)){
    model = kknn(R1~A2+A3+A8+A9+A10+A11+A12+A14+A15, data[-i,], data[i,], k=s, scale=TRUE)
    predictions[i] = as.integer(round(fitted(model),0)+0.5)
  }
  results[s] = sum(predictions == data[,11]) / nrow(data)
}
```

Here are the results of accuracies (limited to 15 after finding the best K values).

```
head(results, n=15L)
```

```
##            [,1]
##  [1,] 0.8211009
##  [2,] 0.8211009
##  [3,] 0.8211009
##  [4,] 0.8211009
##  [5,] 0.8501529
##  [6,] 0.8501529
##  [7,] 0.8532110
##  [8,] 0.8532110
##  [9,] 0.8516820
## [10,] 0.8577982
## [11,] 0.8577982
## [12,] 0.8562691
## [13,] 0.8577982
```

```
## [14,] 0.8562691
## [15,] 0.8562691
```

```r
best = max(results)
best
```

```
## [1] 0.8577982
```

The optimal K values are 10,11, and 13, with the best accuracy of 0.8577982