hw1.1 & 1.2

Homework 1 submitted by Anubhav Rana, Kunle Lawal, Mihir Tulpule and Ali Mujtaba Lakdawala

1.1

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.

An everyday situation for which a classification model would be appropriate would be the classification of email as spam or not. In this particular case, an algorithm is used to automatically sort out electronic messages in an effort to streamline our email inboxes and generally improve the user experience while using specific email services. There are multiple predictors such an algorithm might use to classify incoming emails as spam or not – these may include:

1. Recognition of the person/entity sending the incoming email (whether said entity is a known contact/well-known organization)
2. Commonly recognized spelling or grammatical errors in the text of the received email
3. The inclusion of links to external webpages in the email (especially unsecured or potentially malicious links)
4. The presence of attached documents in said email (as well as the format of said documents, e.g. .txt, .doc, .exe)
5. The inclusion and amount of numbers (pricing, phone numbers, etc.) present in the email text

There are many more subtle factors that could be used in parsing through the text of the email (for instance, the frequency of word repetition in the email, the complexity of the language used, the presence of significant words that could be pre-marked as "red flags," etc.). A classification model would be appropriate in this instance because we are trying to categorize the emails into two distinct qualitative groups ("spam" or "not spam") based on a given number of predictors.

1.2

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.

Reading the data

```r
credit_card_data <- read.table("credit_card_data.txt", header = FALSE)
head(credit_card_data)
```

```
##   V1    V2    V3   V4 V5 V6 V7 V8  V9 V10 V11
## 1  1 30.83 0.000 1.25  1  0  1  1 202   0   1
## 2  0 58.67 4.460 3.04  1  0  6  1  43 560   1
## 3  0 24.50 0.500 1.50  1  1  0  1 280 824   1
## 4  1 27.83 1.540 3.75  1  0  5  0 100   3   1
## 5  1 20.17 5.625 1.71  1  1  0  1 120   0   1
## 6  1 32.08 4.000 2.50  1  1  0  0 360   0   1
```

```r
require("kernlab")
```

```
## Loading required package: kernlab
```

```r
credit_card_data <- as.matrix(credit_card_data)
## Applying the kvsm model with C = 100
model <- ksvm(credit_card_data[,1:10],credit_card_data[,11],
        type="C-svc",kernel="vanilladot",C=100,scaled=TRUE)
```

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

```r
#Investigating the model:
summary(model)
```

```
## Length  Class   Mode
##      1   ksvm     S4
```

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

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

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

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

```r
#Checking the model accuracy:
pred <- predict(model,credit_card_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
##  [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
```

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

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

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.

Testing other kernels

```
#Exploring other kernels for kvsm - such as rbfdot
model2 <- ksvm(credit_card_data[,1:10],credit_card_data[,11],
          type="C-svc",kernel="rbfdot",C=100,scaled=TRUE)
a <- colSums(model2@xmatrix[[1]] * model2@coef[[1]])
a0 <- -model2@b
pred2 <- predict(model2,credit_card_data[,1:10])
sum(pred2 == credit_card_data[,11])/ nrow(credit_card_data)
```

```
## [1] 0.9571865
```

`rbfdot` gives a better prediction with a 10% higher accuracy at 0.957

Exploring another kernel for ksvm `splinedot`

```
require("kernlab")
model3 <- ksvm(credit_card_data[,1:10],credit_card_data[,11],
          type="C-svc",kernel="splinedot",C=100,scaled=TRUE)
```

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

```
a <- colSums(model3@xmatrix[[1]] * model3@coef[[1]])
a0 <- -model3@b
pred3 <- predict(model3,credit_card_data[,1:10])
sum(pred3 == credit_card_data[,11])/ nrow(credit_card_data)
```

```
## [1] 0.9785933
```

`splinedot` also provides a better prediction showing that this data is better described by a more advanced model rather than a simple linear model such as vanilla dot

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).

Citation for Code Reference: http://rstudio-pubs-static.s3.amazonaws.com/24844_335efcfc09954ad99c4e05d9548ed2ad.html

We have two solutions for this problem. The first solution takes a test and training data set.

The second solution uses all the data and loops through model 1 row at a time.

```r
library(kknn)

data = read.table("credit_card_data.txt")
Sample <- sample(1:654, 196)
data.test <- data[Sample, ]
data.train <- data[-Sample, ]
dim(data)
```

```
## [1] 654  11
```

```r
dim(data.test)
```

```
## [1] 196  11
```

```r
dim(data.train)
```

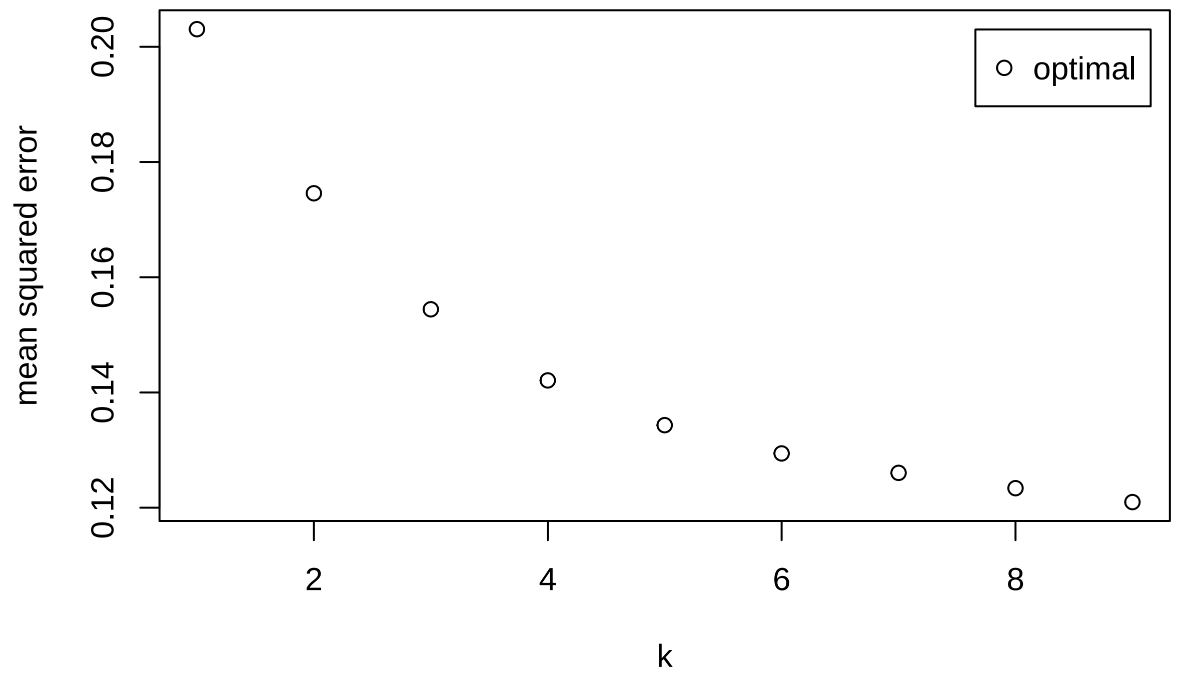
```
## [1] 458  11
```

We set kmax = 9 as it minimizes the mean squared error shown by the plot:

```r
model4 <- train.kknn(V11 ~ ., data = data.train, kmax = 9)
model4
```

```
##
## Call:
## train.kknn(formula = V11 ~ ., data = data.train, kmax = 9)
##
## Type of response variable: continuous
## minimal mean absolute error: 0.2028566
## Minimal mean squared error: 0.1209746
## Best kernel: optimal
## Best k: 9
```

```r
plot(model4)
```



In the following loop we determine the optimum threshold by looping through all the possible thresholds from 0.05 - 0.85 and checking the accuracy for each one.

```
for (t in seq(0.4, 0.85, 0.05)){
  prediction <- predict(model4, data.test[, -11])
#The conditional statement categorizes the prediction
#It also converts the variable to discrete.
  t_prediction = ifelse(prediction > t, 1 ,0)
  CM <- table(data.test[, 11], t_prediction)
  print(CM)

  accuracy <- (sum(diag(CM)))/sum(CM)
  print(accuracy)
  print(t)
  print(strrep("-", 10))

}
```

```
##    t_prediction
##      0  1
##   0 79 27
##   1  5 85
## [1] 0.8367347
## [1] 0.4
## [1] "----------"
##    t_prediction
##      0  1
##   0 81 25
##   1  5 85
## [1] 0.8469388
## [1] 0.45
## [1] "----------"
##    t_prediction
##      0  1
##   0 86 20
##   1  8 82
## [1] 0.8571429
## [1] 0.5
## [1] "----------"
##    t_prediction
##      0  1
##   0 86 20
##   1 13 77
## [1] 0.8316327
## [1] 0.55
## [1] "----------"
##    t_prediction
##      0  1
##   0 88 18
##   1 19 71
## [1] 0.8112245
## [1] 0.6
## [1] "----------"
##    t_prediction
##      0  1
##   0 89 17
##   1 24 66
## [1] 0.7908163
## [1] 0.65
## [1] "----------"
##    t_prediction
##      0  1
##   0 93 13
##   1 29 61
## [1] 0.7857143
## [1] 0.7
## [1] "----------"
##    t_prediction
##      0  1
##   0 96 10
##   1 34 56
## [1] 0.7755102
## [1] 0.75
## [1] "----------"
##    t_prediction
##      0  1
##   0 97  9
##   1 42 48
## [1] 0.7397959
## [1] 0.8
## [1] "----------"
##    t_prediction
##      0  1
##   0 98  8
##   1 47 43
## [1] 0.7193878
## [1] 0.85
## [1] "----------"
```

We also used just 1 point in our test set (and the rest in the training) and evaluated the model for thresholds ranging from 0.4 to 0.7.

```
#This is a similar loop to the the one in the previous solution.
for (t in seq(0.4, 0.7, 0.05)){
  for (i in 1:nrow(data)){
    data.test = data[i,]
    data.train = data[-i,]

    model <- train.kknn(V11 ~ ., data = data.train, kmax = 9)

    prediction <- predict(model, data.test[, -11])


    t_prediction <- ifelse(prediction < t, 0, 1)

    if (t_prediction == data.test[, 11]){
      data$accuracy[i] = 100
    }
    else{
      data$accuracy[i] = 0
    }

  }

  total_accuracy = sum(data$accuracy)
  print(total_accuracy)

  final_accuracy = total_accuracy / (654*100)
  print(final_accuracy)
  print(t)
  print(strrep('--', 10))
}
```

```
## [1] 56000
## [1] 0.8562691
## [1] 0.4
## [1] "--------------------"
## [1] 61300
## [1] 0.9373089
## [1] 0.45
## [1] "--------------------"
## [1] 56900
## [1] 0.8700306
## [1] 0.5
## [1] "--------------------"
## [1] 59300
## [1] 0.9067278
## [1] 0.55
## [1] "--------------------"
## [1] 56600
## [1] 0.8654434
## [1] 0.6
## [1] "--------------------"
## [1] 56800
## [1] 0.8685015
## [1] 0.65
## [1] "--------------------"
## [1] 53900
## [1] 0.824159
## [1] 0.7
## [1] "--------------------"
```