Project 1

Marcus McKenzie

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.

Given our current environment, a classification model to classify whether someone has a virus and may be unaware(asymptomatic) could prove to be helpful in slowing the spread of a virus. Where the person lives, the person's age, how many people that person comes into contact with everyday, where they have recently traveled to and whether or not they have previously contracted the virus could all be some possible predictors as to whether or not a person may be carrying a virus.

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. (Don't worry about test/validation data yet; we'll cover that topic soon.)

Load the support vector machine library:

```
library(kernlab) #ksvm library
```

Load in the data from provided data files:

```
#load data
data <- read.table("credit_card_data.txt", header=FALSE, stringsAsFactors = FALSE)
#Documents/OMSCS/Analytics Modeling/Assignments/Assignment1/
head(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
```

Create the ksvm model:

```
#svm models

#linear kernel
model <- ksvm(as.matrix(data[,1:10]),as.factor(data[,11]), type="C-svc", kernel="vanilladot",  C=100, s
```

```
##  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
```

Calculate coefficients a1, .. an:

```
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
```

Calculate a0:

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

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

Calculate predictions:

```
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
##  [38] 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 1 1 0
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 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 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 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
## [223] 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 0 0 0 0 0 1
## [260] 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 1 0 0 0
## [297] 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 1 0
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
## [445] 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 1 1 1 1 1 1 1
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [519] 1 1 1 1 1 1 1 1 1 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
## [556] 1 1 1 1 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 0 0 1 0 0 0 0
## [593] 0 0 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 0
## [630] 0 0 0 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
```

Calculate accuracy of predictions:

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

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

Part 1 Analysis:

The equation of the classifier can be determined by the a coefficients calculated above. The equation for this particular classifier would be

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

This equation from the ksvm model classifies the data fairly accurately as is represented by the 86% accuracy, which is calculated above.

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.

```
library(kernlab) #ksvm library
#load data
data <- read.table("credit_card_data.txt", header=FALSE, stringsAsFactors = FALSE)
#Documents/OMSCS/Analytics Modeling/Assignments/Assignment1/
head(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
```

Change kernel type:

```
#polynomial kernel
#model <- ksvm(as.matrix(data[,1:10]),as.factor(data[,11]), type="C-svc", kernel="polydot",  C=100, sca

#gaussian kernel
model <- ksvm(as.matrix(data[,1:10]),as.factor(data[,11]), type="C-svc", kernel="rbfdot",  C=100, scale

#laplacian kernel
#model <- ksvm(as.matrix(data[,1:10]),as.factor(data[,11]), type="C-svc", kernel="laplacedot",  C=100,

model
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0976236950921117
##
## Number of Support Vectors : 242
##
## Objective Function Value : -8775.541
## Training error : 0.045872
```

```
#calculate coefficients a1, .. an
a <- colSums((model@xmatrix[[1]]) * model@coef[[1]])
a
```

```
##         V1         V2         V3         V4         V5         V6         V7
## -18.763589 -35.596229  -8.170341  55.509921  51.204604 -26.280034  19.961741
##         V8         V9        V10
## -24.520213 -57.410992  53.900127
```

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

```
## [1] 0.7242642
```

```
#prediction
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
##  [38] 1 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 0 0 0 0
##  [75] 0 0 0 1 0 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
## [112] 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
## [149] 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 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 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
## [223] 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 0 0 0 0 1
## [260] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [297] 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
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [445] 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 1 1 1 1 1 1 1
## [482] 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 0 0 0 0 0 0 0
## [519] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [556] 1 1 1 1 1 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 1 0 1 0 0
## [593] 0 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 0
## [630] 0 0 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
```

```
#fraction of pred that matches actual
sum(pred == data[,11]) / nrow(data)
```

```
## [1] 0.9541284
```

Part 2 Analysis:

The same process as in part 1 was repeated for this part. However, in this case the kernels were changed from a linear kernel to various other kernels. From the accuracy value shown above, 95%, it is clear that the gaussian kernel provides better predictions than the linear kernel.

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).

Load kknn library:

```
library(kknn) #k nearest neighbor library
```

Load data:

```
data <- read.table("credit_card_data.txt", header=FALSE, stringsAsFactors = FALSE)
#Documents/OMSCS/Analytics Modeling/Assignments/Assignment1/
head(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
```

KNN Model function:

```
chk_acc = function(X){

  pr<- rep(0,(nrow(data)))

  for (i in 1:nrow(data)){

    #pr <- knn(iris_train,iris_test,cl=iris_target_category,k=13, scale=TRUE)
    model=kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10,data[-i,],data[i,],k=X, scale=TRUE)

    #round to integer value
    r = 0.5
    pr[i] <- as.integer(fitted(model)+r)

  }

  #correct predictions
  acc = sum(pr == data[,11]) / nrow(data)
  return(acc)

}
```

For loop for each k -neighbors:

```
accuracy <- rep(0, 20)
for (X in 1:20){

  accuracy[X] = chk_acc(X) # test with X neighbors

}
```

Accuracy:

```
accuracy
```

```
##  [1] 0.8149847 0.8149847 0.8149847 0.8149847 0.8516820 0.8455657 0.8470948
##  [8] 0.8486239 0.8470948 0.8501529 0.8516820 0.8532110 0.8516820 0.8516820
## [15] 0.8532110 0.8516820 0.8516820 0.8516820 0.8501529 0.8501529
```

```
accuracy <- as.matrix(accuracy * 100)
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
```

Part 3 Analysis:

The classifier function for the k-nearest neighbor model classifies the data fairly accurately as is represented by the accuracy for each k-value lying between 80% and 85%, which is calculated above.

The k-value with the highest accuracy appears to be 12, with an accuracy of 85.32%. This is relatively similar to the accuracy calculated for the support vector machine, from part 1.