ISYE - 6501 Homework Week 1

5/21/2020

We will include the following libraries for potential usage for our analysis.

```
library(kernlab)
library(kknn)
library(caret)
library(e1071)
```

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.

I work in the Commercial Auto Insurance industry. Part of my daily job, is to evaluate insurance underwriting models to ensure profitability for our company, but also ensure competitive pricing for our customers. This type of modeling would be incredibly useful (and is currently used) in my industry. Initially, thinking about the predictors that we can utilize, many thoughts come to mind. I will focus primarily on trying to predict what premium we should charge customers. Predictors we could use would include:

- Business Experience (Various time frames, non-binary)
- Workman’s Comp Insurance (Yes/No, binary)
- Duration with Current Insurance company (Various time frames, non-binary)
- Number of At Fault Accidents in past 3 years (Various counts, non-binary)
- Number of Total Accidents in the past 3 years (Various counts, non-binary)

Question 2.2

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.

To begin the evaluation, let’s import our data from the text file. The file is saved in the working directory of this project. We will create a dataframe initially to provide a framework for further manipulation in analysis if necessary.

```
data <- read.table('credit_card_data-headers.txt', header=TRUE)
set.seed(1234)
str(data)
```

```
## 'data.frame':    654 obs. of  11 variables:
##  $ A1 : int  1 0 0 1 1 1 1 0 1 1 ...
##  $ A2 : num  30.8 58.7 24.5 27.8 20.2 ...
##  $ A3 : num  0 4.46 0.5 1.54 5.62 ...
##  $ A8 : num  1.25 3.04 1.5 3.75 1.71 ...
##  $ A9 : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ A10: int  0 0 1 0 1 1 1 1 1 1 ...
##  $ A11: int  1 6 0 5 0 0 0 0 0 0 ...
##  $ A12: int  1 1 1 0 1 0 0 1 1 0 ...
##  $ A14: int  202 43 280 100 120 360 164 80 180 52 ...
##  $ A15: int  0 560 824 3 0 0 31285 1349 314 1442 ...
##  $ R1 : int  1 1 1 1 1 1 1 1 1 1 ...
```

Upon the exploration of the dataset, we will definitely need to turn scaling on, so not to skew the effects on specific variables in the model. Scaling will essentially take the variation seen in each predictor variable, and normalize it, so that we essentially utilize a mean of zero, with unit variance. Additionally, for usage with the KSVM function, we will need to convert our x values to matrix, and our y values to factors for analysis.

As for the model itself, not only do we want to calculate the actual model itself, but we want to gather a better understanding of how the C-Values affect the accuracy of the model, and decide on which value we should utilize. To achieve this, we are creating a dataframe to capture multiple iterations of various magnitudes of C, ranging from 10^{-8} to 10^{8}. Upon the completion of this iteration, we will investigate the results of those iterations, and settle on a model that we will utilize for evaluation.

```
# Create a blank dataframe to house the iterations for values of C, as well as the Accura
  cy of the model, and a0 values.
accreview <- data.frame()

# Variable for iterations of C
for (cvar in c(.00000001,
               .0000001,
               .000001,
               .00001,
               .0001,
               .001,
               .01,
               .1,
               1,
               10,
               100,
               1000,
               10000,
               100000,
               1000000,
               10000000,
               100000000)){

# Create the Support Vector Machine Model
model <- ksvm(x = as.matrix(data[,1:10]), # Symbolic Description of the model to be fit,
   using a matrix.
              y = as.factor(data[,11]), # Response vector with one label for each row/com
  ponent of x.
              type="C-svc", # C-Classification. Used for Binary classification.
              kernel="vanilladot", # Provides a Linear SVM
              C=cvar,
              scaled=TRUE, # As noted before, scales all variables to mean and unit varia
  nce for predictions.
              kpar=list())

# Model Predictions
prediction <- predict(model,data[,1:10])

# Calculate the Prediction Accuracy
accuracy <- round(sum(prediction == data[,11])/nrow(data),3)

# Insert Model Accuracy into a dataframe for future evaluation
accreview <- rbind(accreview, data.frame(C = cvar, ACC = accuracy))
}
head(accreview,18)
```

```
##        C   ACC
## 1  1e-08 0.547
## 2  1e-07 0.547
## 3  1e-06 0.547
## 4  1e-05 0.547
## 5  1e-04 0.547
## 6  1e-03 0.838
## 7  1e-02 0.864
## 8  1e-01 0.864
## 9  1e+00 0.864
## 10 1e+01 0.864
## 11 1e+02 0.864
## 12 1e+03 0.862
## 13 1e+04 0.862
## 14 1e+05 0.864
## 15 1e+06 0.625
## 16 1e+07 0.546
## 17 1e+08 0.664
```

To make the best decision about which value we should use for C, we must first understand what the C value means. Essentially, C is a tradeoff between the training error, and flatness of the training model. As C gets larger, the final training error will decrease. Instinctively, we would suspect using a high value for C would make sense. However, the risk of increasing C too much is prevalent as we can risk losing the generalization properties of the classifier. A larger C also will take much longer to calculate our model. Our ultimate selection of C will aim for a small value. This isn't only for resource considerations, but also to keep our training error as small as we can, while ensuring the data doesn't have large fluctuations as well.

Upon reviewing the above dataframe created by our iterations, we will focus primarily on the accuracy of the model relative to a given C value. For all values where $10^{-2} < C < 10^{5}$, the accuracy for the given value for C is notably lower than the maximum accuracy of 0.864. Given the previously mentioned criteria of a lower C being desireable, we will elect to utilize where $C = .01$.

```
#Create the Support Vector Machine Model
model <- ksvm(x = as.matrix(data[,1:10]),
              y = as.factor(data[,11]),
              type="C-svc",
              kernel="vanilladot",
              C=.01,
              scaled=TRUE,
              kpar=list())
```

The coeffecients for all predictor variables are listed for a1... am.

```
# Calculate a1... am
a <- round(colSums(model@xmatrix[[1]] * model@coef[[1]]),6)
a
```

```
##        A1        A2        A3        A8        A9       A10       A11       A12
## -0.000150 -0.001482  0.001408  0.007286  0.991647 -0.004466  0.007148 -0.000547
##       A14       A15
## -0.001693  0.105482
```

The Y-Intercept(a0) for this model is:

```
# Calculate a0
a0 <- round(-model@b,6)
a0
```

```
## [1] 0.081989
```

How did the model predict each indivdual data point?

```
# Model Predictions
prediction <- predict(model,data[,1:10])
prediction
```

```
##   [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
##  [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 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 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 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 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 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 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 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 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 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 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 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 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 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 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
## Levels: 0 1
```

```
# Calculate the Prediction Accuracy
accuracy <- round(sum(prediction == data[,11])/nrow(data),3)
accuracy
```

```
## [1] 0.864
```

Breaking down the confusion matrix is crucial for proper evaluation of a model's effectiveness, as well as it's usefulness for our goals. We will create a confusion matrix to evaluate this model, and comment after the matrix in depth.

```
#Construct a Confusion Matrix for evaluation of the model
confusionMatrix(factor(data[,11], levels = c(0,1)),prediction)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 286  72
##          1  17 279
##
##                Accuracy : 0.8639
##                  95% CI : (0.8352, 0.8893)
##     No Information Rate : 0.5367
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7297
##
##  Mcnemar's Test P-Value : 1.041e-08
##
##             Sensitivity : 0.9439
##             Specificity : 0.7949
##          Pos Pred Value : 0.7989
##          Neg Pred Value : 0.9426
##              Prevalence : 0.4633
##          Detection Rate : 0.4373
##    Detection Prevalence : 0.5474
##       Balanced Accuracy : 0.8694
##
##        'Positive' Class : 0
##
```

- Prediction/Reference Table
  - The Prediction/Reference table shows true positives in the top left, where the model's predictions align with reality.
  - The Prediction/Reference table shows true negatives in the bottom right, where the model's predictions align with reality.
  - False positives occur in the bottom left, where we predict a positive, but the reality is that the observation was negative. (Type 1 Error)
  - False negatives occur in the top right, where we predict a negative, but the reality is that the observation was positive. (Type 2 Error)

- Accuracy is calculated by $\frac{TruePositives+TrueNegatives}{TruePositives+TrueNegatives+FalsePositives+FalseNegatives}$
  - $\frac{286+279}{286+72+17+279} = \frac{565}{654} = .8639$
- We are 95% confidence that the true estimated accuracy of this model lies between .8352 and .8893.
- Our No Information rate is .5367. This essentially means how well we would predict the outcomes of this model without knowing anything other than the given distributions of the classes we are trying to predict. Our accuracy is a good bit higher than this value.
- Sensitivity refers to essentially our True Positive rate. Put another way, 94.39% of our true positives were accurately predicted.
- Specificity on the other hand refers to 1- False Positive Rate, or basically the % of people that didn't get grouped into false positives.

Given the data we are investigating is related specifically to credit card approvals, our goal for the company would be to mitigate losses, and make profitable lending decisions, with lower risk of defaulting, and a higher profitability for the company. To achieve this, we would want to minimze false positives, or specifically cases where a customer does not qualify, but we approve them. If our strategy is profitability (and not growth), we want to improve the specificity rate as much as possible to further qualify our model to be considered ideal.

---

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.

To investigate further, I will explore two nonlinear kernels. The Splinedot Kernel, and the Radial Basis Kernel (Gaussian). There is no basis in selecting these kernels other than furthering my knowledge to the the application of this data, and modeling in general.

```
#Create the Support Vector Machine Model
model <- ksvm(x = as.matrix(data[,1:10]),
              y = as.factor(data[,11]),
              type="C-svc",
              kernel="splinedot",
              C=.01,
              scaled=TRUE,
              kpar=list())

# Calculate a1... am
a <- round(colSums(model@xmatrix[[1]] * model@coef[[1]]),6)

# Calculate a0
a0 <- round(-model@b,6)

# Model Predictions
prediction <- predict(model,data[,1:10])

# Calculate the Prediction Accuracy
accuracy <- round(sum(prediction == data[,11])/nrow(data),3)

#Construct a Confusion Matrix for evaluation of the model
confusionMatrix(factor(data[,11], levels = c(0,1)),prediction)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 329  29
##          1  95 201
##
##                Accuracy : 0.8104
##                  95% CI : (0.7782, 0.8397)
##     No Information Rate : 0.6483
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6098
##
##  Mcnemar's Test P-Value : 5.309e-09
##
##             Sensitivity : 0.7759
##             Specificity : 0.8739
##          Pos Pred Value : 0.9190
##          Neg Pred Value : 0.6791
##              Prevalence : 0.6483
##          Detection Rate : 0.5031
##    Detection Prevalence : 0.5474
##       Balanced Accuracy : 0.8249
##
##        'Positive' Class : 0
##
```

The Splinedot Model is not as accurate, and has a higher no-information rate as well, reducing the usefulness of the model from the linear version. The sensitvity in particular took a large hit comparatively speaking, as we're now only successfully predicting 77.6% of true positives in this model, although there was a slight improvement in the specificity of the model's projections.

```
#Create the Support Vector Machine Model
model <- ksvm(x = as.matrix(data[,1:10]),
              y = as.factor(data[,11]),
              type="C-svc",
              kernel="rbfdot",
              C=.01,
              scaled=TRUE,
              kpar=list())

# Calculate a1... am
a <- round(colSums(model@xmatrix[[1]] * model@coef[[1]]),6)

# Calculate a0
a0 <- round(-model@b,6)

# Model Predictions
prediction <- predict(model,data[,1:10])

# Calculate the Prediction Accuracy
accuracy <- round(sum(prediction == data[,11])/nrow(data),3)

#Construct a Confusion Matrix for evaluation of the model
confusionMatrix(factor(data[,11], levels = c(0,1)),prediction)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 358   0
##          1 296   0
##
##                Accuracy : 0.5474
##                  95% CI : (0.5083, 0.586)
##     No Information Rate : 1
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.5474
##             Specificity :     NA
##          Pos Pred Value :     NA
##          Neg Pred Value :     NA
##              Prevalence : 1.0000
##          Detection Rate : 0.5474
##    Detection Prevalence : 0.5474
##       Balanced Accuracy :     NA
##
##        'Positive' Class : 0
##
```

The Accuracy in particular of the Radial Basis Kernal Gaussian model is significantly worse, with No Information Rate classification at 100%.

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).

Methodology for answering this question:

I first wanted to make sure to be able to review various values of K for the number of variables selected in this model. To achieve this, I did as follows:

1. Created a for loop, iterating from k = 2 through 20. I elected 20 as the max number arbitrarily.
2. Created a second for loop, to iterate through each row, creating a prediction with the model.
3. Constructed the model using a one left out method, where training has all but 1 row, and validation is a singular row.

4. I then summed the predictions of all rows to estimate the accuracy of the model.
5. Finally, I inserted each K value, and it’s requiste accuracy into a dataframe for further analysis.

```
# Construct an empty dataframe
knnaccreview <- data.frame()

# Assigns a list of 654 zeros to populate knnpredictions for insertion during the modelin
  g process
knnpredictions <- rep(0,nrow(data))

# Start of the two for loops to iterate through various values of K, and then each row of
  data.
for (kvar in c(2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)){
for (i in 1:nrow(data)){

knnmodel <- kknn(R1~., # Response variable, utilizing all other columns for modeling
                 data[-i,], # -i selects all values other than a single value
                 data[i,], # i selects one specific value
                 k=kvar, # changed to kvar to allow the loop iteration to test various k
   values
                 scale = TRUE # The data must be scaled
                 )
knnpredictions[i] <- as.integer(fitted(knnmodel)+0.5) # Populates each row through iterat
  ion with pred value, rounding as needed
knnaccuracy <- sum(knnpredictions == data[,11]) / nrow(data)} # Calculates the accuracy o
  f all predictions
knnaccreview <- rbind(knnaccreview, data.frame(K = kvar, ACC = knnaccuracy))} # Inserts a
  ll results into a DF for review.
```

To determine the best K value, we will compare the accuracy of each value of K generated by our model iteration.

```
knnaccreview
```

```
##     K       ACC
## 1   2 0.8149847
## 2   3 0.8149847
## 3   4 0.8149847
## 4   5 0.8516820
## 5   6 0.8455657
## 6   7 0.8470948
## 7   8 0.8486239
## 8   9 0.8470948
## 9  10 0.8501529
## 10 11 0.8516820
## 11 12 0.8532110
## 12 13 0.8516820
## 13 14 0.8516820
## 14 15 0.8532110
## 15 16 0.8516820
## 16 17 0.8516820
## 17 18 0.8516820
## 18 19 0.8501529
## 19 20 0.8501529
```

```
knnaccreview[which.max(knnaccreview$ACC),]
```

```
##     K      ACC
## 11 12 0.853211
```

A K-Value of 12 or 15 would be our best selection for this model, based on the accuracy of the predictions given.

Question 3.1a

Using the same data set (credit_card_data.txt or credit_card_data-headers.txt) as

in Question 2.2, use the ksvm or kknn function to find a good classifier:

(a) using cross-validation (do this for the k-nearest-neighbors model; SVM is optional)

To complete this question, I will need to break the data into various datasets for training, validation, and testing. For Question 3.1a, we will only need a training and testing dataset.

```
set.seed(1) # Setting Seed for Random Generation consistency
knnsample <- sample(1:nrow(data),as.integer(0.7*nrow(data))) # Creating a random sample o
  f the our dataset (70%)
knntrain <- data[knnsample,] # Creates training dataset, inclusive of 70% of the total da
  taset
knntest <- data[-knnsample,] # Creates testing dataset, inclusive of the other 30% of the
  total dataset
nrow(knntrain) # Number of Rows for Training
```

```
## [1] 457
```

```
nrow(knntest) # Number of Rows for Testing
```

```
## [1] 197
```

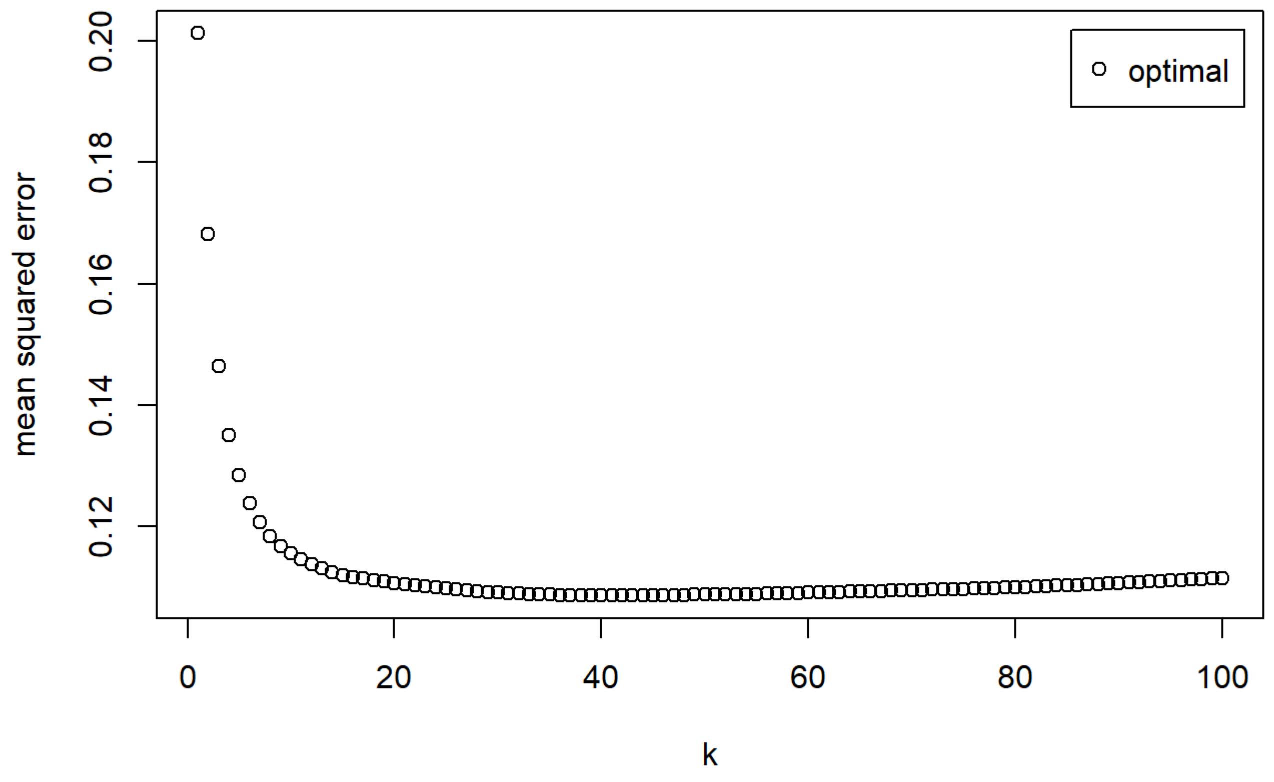
```
nrow(knntrain)+nrow(knntest) == nrow(data) # Boolean check to confirm that the number of
   rows match the total dataset.
```

```
## [1] TRUE
```

```
k_max <- 100 #Maximum number of K values to evaluate
```

Train.kknn utilizes a leave-one out method cross validation. We elect to train all variables with our response variable, scaling our data as well. This dataset used is the training dataset, inclusive of 70% of our full dataset, selected as a random sample.

```
knncrossmodel <- train.kknn(R1~., knntrain, kmax = k_max, scale = TRUE)
plot(knncrossmodel)
```



```
knncrossmodel
```

```
##
## Call:
## train.kknn(formula = R1 ~ ., data = knntrain, kmax = k_max, scale = TRUE)
##
## Type of response variable: continuous
## minimal mean absolute error: 0.1955315
## Minimal mean squared error: 0.1085533
## Best kernel: optimal
## Best k: 41
```

Upon reviewing both outputs, we can see that the mean squared error of the training model is reduced greatly near the k=40 to k=50 mark, and increases past that point. The output from the model confirms that our minimum MSE is .1085533, with an the optimal kernal, and best k of 41.

Next, we'd like to check the accuracy of the training model. The list of accuracies at the end will tell us how accurate each value of k was in utilization in the model. While we may see better accuracy at some points with lower k values, keep in mind, that the optimal situation minimizes the mean square error, which is not always reflected in the accuracies listed below.

```
accuracy_k <- rep(0,k_max) # Creates an array with 0's for all values up to the k_max.

for (a in 1:k_max){ # Iterates through each k value, and calculates the accuracy for each
  k value.

  predictedk <- as.integer(fitted(knncrossmodel)[[a]][1:nrow(knntrain)]+0.5)
  accuracy_k[a] <- sum(predictedk == knntrain$R1) / nrow(knntrain)
}
accuracy_k # Gives us the all of the accuracies
```

```
##   [1] 0.7986871 0.7986871 0.7986871 0.7986871 0.8424508 0.8336980 0.8402626
##   [8] 0.8424508 0.8446389 0.8446389 0.8468271 0.8490153 0.8468271 0.8446389
##  [15] 0.8446389 0.8446389 0.8424508 0.8424508 0.8424508 0.8380744 0.8380744
##  [22] 0.8358862 0.8358862 0.8358862 0.8380744 0.8336980 0.8315098 0.8336980
##  [29] 0.8293217 0.8315098 0.8315098 0.8315098 0.8293217 0.8249453 0.8271335
##  [36] 0.8293217 0.8271335 0.8315098 0.8315098 0.8336980 0.8315098 0.8336980
##  [43] 0.8336980 0.8336980 0.8336980 0.8336980 0.8380744 0.8380744 0.8380744
##  [50] 0.8380744 0.8380744 0.8380744 0.8380744 0.8358862 0.8358862 0.8358862
##  [57] 0.8402626 0.8424508 0.8402626 0.8380744 0.8402626 0.8402626 0.8380744
##  [64] 0.8380744 0.8402626 0.8380744 0.8380744 0.8380744 0.8380744 0.8358862
##  [71] 0.8358862 0.8358862 0.8358862 0.8358862 0.8358862 0.8380744 0.8380744
##  [78] 0.8380744 0.8380744 0.8380744 0.8380744 0.8358862 0.8358862 0.8380744
##  [85] 0.8380744 0.8402626 0.8402626 0.8402626 0.8402626 0.8402626 0.8402626
##  [92] 0.8402626 0.8402626 0.8402626 0.8402626 0.8402626 0.8402626 0.8402626
##  [99] 0.8402626 0.8424508
```

Now that we have completed the training model, as well as the optimized values, we will repeat the process for final model selection on our testing dataset.

```
knncrossmodeltest <- train.kknn(R1~., knntest, kmax = k_max, scale = TRUE)
knncrossmodeltest
```

```
##
## Call:
## train.kknn(formula = R1 ~ ., data = knntest, kmax = k_max, scale = TRUE)
##
## Type of response variable: continuous
## minimal mean absolute error: 0.2351918
## Minimal mean squared error: 0.1198174
## Best kernel: optimal
## Best k: 39
```

```
accuracy_ktest <- rep(0,k_max)

for (b in 1:k_max){

  predictedktest <- as.integer(fitted(knncrossmodeltest)[[b]][1:nrow(knntest)]+0.5)
  accuracy_ktest[b] <- sum(predictedktest == knntest$R1) / nrow(knntest)
}
```

```
accuracy_ktest
```

```
##   [1] 0.7360406 0.7360406 0.7360406 0.7360406 0.7868020 0.7868020 0.7868020
##   [8] 0.7817259 0.7868020 0.7868020 0.7868020 0.8020305 0.8020305 0.8020305
##  [15] 0.8071066 0.8071066 0.8071066 0.8121827 0.8172589 0.8172589 0.8223350
##  [22] 0.8223350 0.8274112 0.8274112 0.8274112 0.8324873 0.8375635 0.8426396
##  [29] 0.8426396 0.8375635 0.8426396 0.8375635 0.8426396 0.8426396 0.8426396
##  [36] 0.8426396 0.8426396 0.8426396 0.8426396 0.8426396 0.8426396 0.8426396
##  [43] 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635
##  [50] 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635
##  [57] 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635
##  [64] 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635
##  [71] 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635
##  [78] 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635
##  [85] 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635
##  [92] 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635 0.8375635
##  [99] 0.8375635 0.8375635
```

Question 3.1b

Using the same data set (credit_card_data.txt or credit_card_data-headers.txt) as

in Question 2.2, use the ksvm or kknn function to find a good classifier:

(b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the #### other is optional).

```
set.seed(2)
svmsample <- sample(1:nrow(data),size = round(nrow(data)*.6), replace = FALSE) # Creating
  a random sample of the our dataset (60%)
svmtrain <- data[svmsample,] # Creates training dataset, inclusive of 60% of the total da
  taset
svmvaldata <- data[-svmsample,] # Assign the remaining 40% of data to a new dataset for s
  plitting
svmvalsample <- sample(1:nrow(svmvaldata),size = round(nrow(svmvaldata)*.5), replace = FA
  LSE) # Creates a random sampleof the remaining 40%, splitting to two 20% parts.
svmvalid <- svmvaldata[svmvalsample,] # Assigns 20% of remaining full dataset to validati
  on
svmtest <- svmvaldata[-svmvalsample,] # Assigns 20% of remaining full dataset to test
nrow(svmtrain)
```

```
## [1] 392
```

```
nrow(svmvalid)
```

```
## [1] 131
```

```
nrow(svmtest)
```

```
## [1] 131
```

```
nrow(svmtrain) + nrow(svmtest) + nrow(svmvalid) == nrow(data) # Boolean confirmation that
  the full dataset matches the 3 test datasets.
```

```
## [1] TRUE
```

This analysis will begin with the creation an SVM model for each of the various kernels utilized in KSVM models:

- RBFDot
- Polydot
- Tanhdot
- Vanilladot
- Laplacedot
- Besseldot
- Anovadot
- Splinedot

As part of the loop constructed below, we will then calculate all predictions, as well as the sum of the predictions, with the goal of calculating the accuracy of each model. The accuracy will be given in the form of a “Model Score”.

Finally, this information will be summarized in a dataframe.

Note, we have assumed $C = 100$ for all iterations, as well as scaled our data.

```
kernel_eval <- data.frame()
kernels <- c("rbfdot", "polydot", "tanhdot", "vanilladot", "laplacedot", "besseldot", "an
  ovadot", "splinedot")
for (kern in kernels){
svmmodel <- ksvm(x = as.matrix(svmtrain[,1:10]), y = as.factor(svmtrain[,11]), type="C-sv
  c", kernel= kern, C=.100, scaled=TRUE, kpar=list()) #Ran through the training dataset
svmprediction <- predict(svmmodel,svmvalid[,1:10]) # Ran through the validation dataset
pred_accuracy <- sum(svmprediction == svmvalid[,11])/nrow(svmvalid) # Ran through the val
  idation dataset
kernel_eval <- rbind(kernel_eval, data.frame(Kernel = kern, Model_Score = pred_accuracy))
}
kernel_eval
```

```
##        Kernel Model_Score
## 1      rbfdot   0.5190840
## 2     polydot   0.8702290
## 3     tanhdot   0.8167939
## 4  vanilladot   0.8702290
## 5  laplacedot   0.7938931
## 6   besseldot   0.8549618
## 7    anovadot   0.8702290
## 8   splinedot   0.8320611
```

Our preferred model as summarized in the table above will be the polydot kernel. To ensure easy access to this model, I will recreate the model variables associated with the loop above specific to polydot.

```
polydotmodel <- ksvm(x = as.matrix(svmtrain[,1:10]), y = as.factor(svmtrain[,11]), type=
  "C-svc", kernel= "polydot", C=.100, scaled=TRUE, kpar=list()) # Training Dataset
polydotprediction <- predict(svmmodel,svmvalid[,1:10]) # Validation Dataset
polydot_accuracy <- sum(svmprediction == svmvalid[,11])/nrow(svmvalid) # Validation Datas
  et
```

Finally, to assess our chosen model's accuracy, we will run our polydot model through our testing dataset for our final accuracy evaluation.

```
svmmodeltest <- predict(polydotmodel, svmtest[,1:10]) #Testing Dataset
chosen_model_accuracy <- sum(svmmodeltest == svmtest[,11])/nrow(svmtest) #Testing Dataset
chosen_model_accuracy
```

```
## [1] 0.8625954
```