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.

From my life, when it comes to sales and selling things through websites as a business, I believe that a classification model could be very useful. For instance, if a customer meets certain criteria while browsing a website, it might be worthwhile to send them more offers, try upselling, or targeting more advertisements towards them for an eventual sale. A few predictors that might be used for this are to analyze previous buying patterns if there are any (e.g., did they make previous purchases or not), to look at how they are browsing the store website (e.g., are they spending a lot of time looking at certain products), and to look at if they are adding certain items to cart or not. If they do add items to cart and keep them there for a while, sending promotional offers including those items might help lead to a sale. Alongside the previous three predictors, other valuable predictors might be to see where they are coming to your website from. If they are coming by clicking your advertisements, then it might show that they are already interested in the product and might be more likely to purchase if given additional deals. Finally, a fifth predictor one might use is to see how many items a customer buys. If they buy a lot of items, then it might be possible to use the classification model to categorize them as a likely purchaser and include more upsells seeing that they are already willing to purchase quite a few things.

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

   Using C as 100 and with scaling set to true, the classifier equation that I obtained using the vanilladot kernel is:

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

   In order to generate this classifier, it is noted that 189 support vectors were used. Furthermore, with this SVM, I obtained a training error of 0.1360856 and the fraction of the model's predictions that were correct was 0.8639144 (percentage: 86.4%).

   The code I ran for this can be found in the appendix 2.2.1.1.

   To further investigate how the SVM was generated and what alterations to the code could result in, I also opted to run it with scaling set to false and everything else kept the same. In this instance, the fraction of correct predictions changed to 0.7217125 and the error was 0.2782875. Because the predictors have differences in orders of magnitude, it makes sense why scaling is so important in reducing error. The code for this experiment can be found in appendix 2.2.1.2.

   For more experimentation, I also wanted to see how changing the value of C could change the fraction of correct predictions. To begin, I started by setting C to 100000 as compared to 100. While I thought it would cause a change in the model's prediction accuracy, I was incorrect. The value

remained the same at 0.8639144. The code for this experiment can be found in appendix 2.2.1.3. In this case, it may have been that I did not alter the value of C enough, so I ran the same experiment again with a C value of 1000000000. Here, a more apparent change was identified. The percentage of correct predictions decreased from 86.4% to 80.2% This may be because as the value for C increases, the margin decreases. A large value of C can also cause more "correct" points to be left out. A visual representation of this would be a hyperplane being pushed too far to the right, thus causing some people with good credit to be rejected (as pushing to very high C means we are trying to allow for even less outliers).

Finally, I performed one last experiment. In this experiment, I set the value of C to 0.00000000000001 to investigate the effects of a really low value of C. The effects were immediately obvious as the fraction of the model's correct predictions decreased to 0.5474006. The cause of this stark decrease may be explained by the fact that when C is decreased, the resultant hyperplane has a larger margin. A larger margin can result in more points being misclassified, which would explain why the percentage of correct predictions decreases. The code for this experiment can be found in appendix 2.2.1.4.

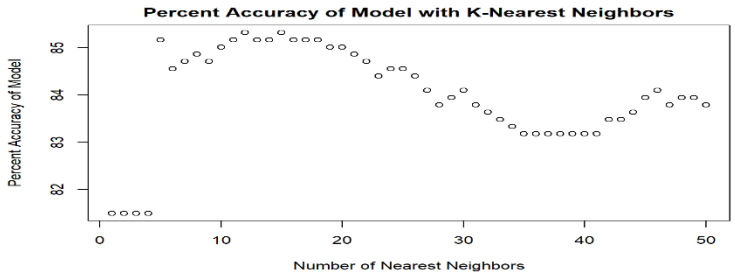
2. You are welcome, but not required, to try other (nonlinear) kernels as well; we're not covering them in this course, but they can sometimes be useful and might provide better predictions than vanilladot.

   I decided to run two different kernels. First, I ran the anovadot kernel and observed the fraction of correct predictions. The code for it can be found in appendix 2.2.2.1, and it gave a value of 0.9067278. This was higher than the vanilladot kernel. To further investigate, I also decided to run the code with scaling set to false. The resultant value for the fraction of correct predictions was 0.8042813, which is higher than the 0.7217125 obtained when the vanilladot kernel model had scaling set to false.

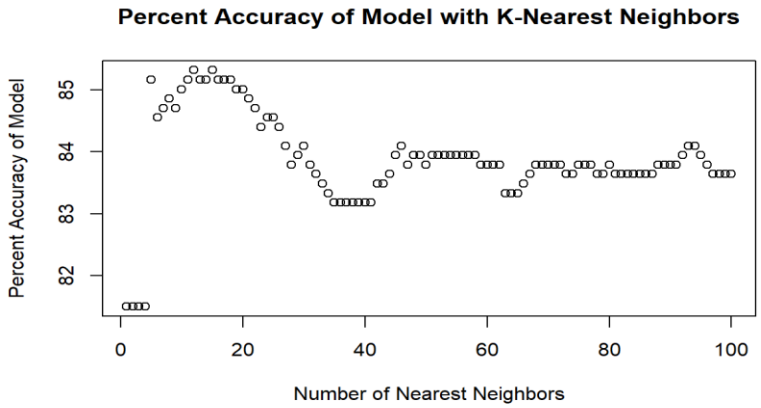
   In addition to the anovadot kernel, I also decided to investigate the rbfdot kernel. I have included the code for it in appendix 2.2.2.2. This kernel, out of all the ones that I tested, led to the highest value for the fraction of correct predictions: 0.9525994. When I turned off scaling, it also provided a surprisingly high value: 0.8975535, which is even higher than the 0.8639144 obtained with the vanilladot kernel when scaling was set to true. Therefore, out of all of the kernels used so far, I think using the rbfdot kernel would have been most accurate.

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).

   In order to discover which k values may be best to classify the data points in the full data set, I used the kknn function to create a model. With this model, I let it iterate over values of k from 1 to 50, which might be a bit excessive considering how long it took. Through this process, I identified two values that presented the highest percentage accuracy for the model. They were k=12 and k=15, which both had the same value of: 85.3211%. The full process and code are shown below in appendix 2.2.3.1. I also generated a graph, which gives a visual representation of this data, and it is shown below. As can be seen, there are two highest values (k=12 and k=15), and they are equivalent to one another:



Another interesting point of consideration is that the next highest accuracy value is 85.16820%, and it occurs a few times with varying k values, but there is a general decreasing trend in the graph. However, as there is a local peak when approaching k=50, I decided to run the code again with more k values. With k iterating from 1:100, the plot of percent accuracy was as follows:



Therefore, from this data, it seems that k=12 and k=15 still led to the highest percent accuracy for values of k from 1 to 100. This does surprise me however as I thought that increasing k would lead to increased accuracy. Yet, from the context that a very large k can cause errors by essentially always predicting the majority category, despite whatever local variation there may be, it does make sense why the ideal value for k is not extremely high. In a similar vein, when k is too small, it can be biased too easily by local exceptions and have low accuracy.

Appendix

Question 2.2.1.1

```
#Load library
library(kernlab)

#Load data
data <- read.table("C:\\Users\\User\\Downloads\\credit_card_data.txt",
                   stringsAsFactors = F, header = F)

#Create the model with scaling
model <- ksvm(as.matrix(data[,1:10]), data[,11], type="C-svc",
              kernel="vanilladot", C=100, scaled=T)

##  Setting default kernel parameters

#Calculate a1…am
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
a0 <- -model@b
a0

## [1] 0.08158492

#See what the model predicts
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
##  [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
##  [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
## [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
## [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
## [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
## [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
## [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 1 0
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
## [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
## [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 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 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
```

```
#See what fraction of the model's predictions match the actual classification
sum(pred == data[,11]) / nrow(data)
```

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

Question 2.2.1.2

```
#What happens if we create a model without scaling?
model_without_scaling <- ksvm(as.matrix(data[,1:10]), data[,11], type="C-svc",
              kernel="vanilladot", C=100, scaled=F)
```

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

```
#Calculate a1…am
a <- colSums(model_without_scaling@xmatrix[[1]] *
               model_without_scaling@coef[[1]])
a
```

```
##            V1            V2            V3            V4            V5
## -0.0483050561 -0.0083148473 -0.0836550114  0.1751121271  1.8254844547
##            V6            V7            V8            V9           V10
##  0.2763673361  0.0654782414 -0.1108211169 -0.0047229653 -0.0007764962
```

```
#Calculate a0
a0 <- -model_without_scaling@b
a0
```

```
## [1] 0.5255393
```

```
#See what the model predicts
pred_without_scaling <- predict(model_without_scaling,data[,1:10])
pred_without_scaling
```

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

```
#See what fraction of the model's predictions match the actual classification
sum(pred_without_scaling == data[,11]) / nrow(data)
```

```
## [1] 0.7217125
```

Question 2.2.1.3

```
#What happens if we create a model with a value much higher than C=100?
model_with_highC <- ksvm(as.matrix(data[,1:10]), data[,11], type="C-svc",
             kernel="vanilladot", C=100000, scaled=T)
```

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

```
#Calculate a1…am
a <- colSums(model_with_highC@xmatrix[[1]] *
```

```
               model_with_highC@coef[[1]])
a

##           V1           V2           V3           V4           V5
V6
## -0.004117738 -0.086896089  0.129715260 -0.083744032  0.988381368  0.031253
888
##           V7           V8           V9          V10
## -0.055666972 -0.037281856  0.021940744  0.018521785

#Calculate a0
a0 <- -model_with_highC@b
a0

## [1] 0.08054451

#See what the model predicts
pred_with_highC <- predict(model_with_highC,data[,1:10])
pred_with_highC

##   [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
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 1 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 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
## [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 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 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 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 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
```

```
#See what fraction of the model's predictions match the actual classification
sum(pred_with_highC == data[,11]) / nrow(data)
```

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

Question 2.2.1.4

```
#What happens if we create a model with a value much lower than C=100?
model_with_lowC <- ksvm(as.matrix(data[,1:10]), data[,11], type="C-svc",
              kernel="vanilladot", C=0.00000000000001, scaled=T)
```

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

```
#Calculate a1…am
a <- colSums(model_with_lowC@xmatrix[[1]] *
               model_with_lowC@coef[[1]])
a
```

```
##            V1            V2            V3            V4            V5
## -6.479335e-14  6.730631e-13  9.364570e-13  1.803160e-12  4.127070e-12
##            V6            V7            V8            V9           V10
## -2.416310e-12  2.334032e-12 -1.403036e-13 -3.383522e-13  1.011458e-12
```

```
#Calculate a0
a0 <- -model_with_lowC@b
a0
```

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

```
#See what the model predicts
pred_with_lowC <- predict(model_with_lowC,data[,1:10])
pred_with_lowC
```

```
##   [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 0 0 0 0 0 0 0 0 0 0 0
0 0 0
##  [38] 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 0
##  [75] 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 0
## [112] 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 0
## [149] 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 0
## [186] 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 0
## [223] 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 0
```

```
## [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 0
## [297] 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
## [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 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [482] 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
## [519] 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
## [556] 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
## [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 0 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

#See what fraction of the model’s predictions match the actual classification
sum(pred_with_lowC == data[,11]) / nrow(data)

## [1] 0.5474006
```

Question 2.2.2.1

```
#Process using anovadot kernel

#Load library
library(kernlab)

#Load data
data <- read.table("C:\\Users\\User\\Downloads\\credit_card_data.txt",
                   stringsAsFactors = F, header = F)

#Create the model with scaling
model <- ksvm(as.matrix(data[,1:10]), data[,11], type="C-svc",
              kernel="anovadot", C=100, scaled=T)

##  Setting default kernel parameters

#Calculate a1…am
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
a

##            V1            V2            V3            V4            V5
V6
##   0.01883724 -22.49799706 -28.05112505  -2.35838911   2.53581364  -1.12232
```

```
794
##           V7           V8           V9          V10
##  -3.11482539  -0.04590503 -16.44537428  14.82481722

#Calculate a0
a0 <- -model@b
a0

## [1] 1.174074

#See what the model predicts
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
## [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
## [75] 1 1 0 1 0 0 1 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0 0 1 1 1 0 1 1 0 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
## [149] 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 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 0 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
## [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
## [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
## [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
## [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
## [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
## [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
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 0 0 0 1 1 0 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 0 1
## [556] 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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
## [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
```

```
#See what fraction of the model's predictions match the actual classification
sum(pred == data[,11]) / nrow(data)
```

```
## [1] 0.9067278
```

Question 2.2.2.2

```
#Process using rbfdot kernel

#Load library
library(kernlab)

#Load data
data <- read.table("C:\\Users\\User\\Downloads\\credit_card_data.txt",
                   stringsAsFactors = F, header = F)

#Create the model with scaling
model <- ksvm(as.matrix(data[,1:10]), data[,11], type="C-svc",
              kernel="rbfdot", C=100, scaled=T)

#Calculate a1…am
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
a
```

```
##         V1         V2         V3         V4         V5         V6
V7
## -19.463966 -39.430012  -9.099197  57.195017  48.694716 -22.142265  10.0401
04
##         V8         V9        V10
## -22.929251 -59.258301  48.953134
```

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

```
## [1] 0.869335
```

```
#See what the model predicts
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 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 1 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 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
## [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
## [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
## [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
## [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
## [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
## [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 1 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
## [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 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
```

```
#See what fraction of the model's predictions match the actual classification
sum(pred == data[,11]) / nrow(data)
```

```
## [1] 0.9510703
```

Question 2.2.3.1

```
#Load library
library(kknn)

#Load data
data <- read.table("C:\\Users\\User\\Downloads\\credit_card_data.txt",
                   stringsAsFactors = F, header = F)

#Build a function to check accuracy of K-nearest-neighbors
checking_accuracy <- function(A){

  #Create a vector with all zeroes
  prediction <- rep(0, (nrow(data)))

  #For each row, use the other rows to guess its response
  for (i in 1:nrow(data)){
    #data[-i,] enables us to remove row i when finding nearest neighbours
```

```
    #Create the model
    knn_model = kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10, data[-i,], data[i,],
                     k=A, scale = T)

    #Rounds to 0 if fitted value less than 0.5 and to 1 if greater than 0.5
    prediction[i] <- as.integer(fitted(knn_model) + 0.5)
  }

  #Now we calculate the fraction of correct predictions
  fraction_correct = sum(prediction == data[,11]) / nrow(data)
  return(fraction_correct)
}

#Create an empty vector
acc = rep(0,50)
#Add in the values of Knn with "A" number of neighbors
for (A in 1:50){
  acc[A] = checking_accuracy(A)
}

#Accuracy as percentage
acc <- acc * 100
acc
```

```
##  [1] 81.49847 81.49847 81.49847 81.49847 85.16820 84.55657 84.70948 84.862
39
##  [9] 84.70948 85.01529 85.16820 85.32110 85.16820 85.16820 85.32110 85.168
20
## [17] 85.16820 85.16820 85.01529 85.01529 84.86239 84.70948 84.40367 84.556
57
## [25] 84.55657 84.40367 84.09786 83.79205 83.94495 84.09786 83.79205 83.639
14
## [33] 83.48624 83.33333 83.18043 83.18043 83.18043 83.18043 83.18043 83.180
43
## [41] 83.18043 83.48624 83.48624 83.63914 83.94495 84.09786 83.79205 83.944
95
## [49] 83.94495 83.79205
```

```
#Plot accuracy
plot(acc, main="Percent Accuracy of Model with K-Nearest Neighbors", xlab="Nu
mber of Nearest Neighbors", ylab="Percent Accuracy of Model")
```

```
#Analyze the data
max(acc)
```

```
## [1] 85.3211
```

```
which.max(acc)
```

```
## [1] 12
```

Exact values of the percent accuracy of k from 1 to 100:

```
  [1] 81.49847 81.49847 81.49847 81.49847 85.16820 84.55657 84.70948 84.86239
  [9] 84.70948 85.01529 85.16820 85.32110 85.16820 85.16820 85.32110 85.16820
 [17] 85.16820 85.16820 85.01529 85.01529 84.86239 84.70948 84.40367 84.55657
 [25] 84.55657 84.40367 84.09786 83.79205 83.94495 84.09786 83.79205 83.63914
 [33] 83.48624 83.33333 83.18043 83.18043 83.18043 83.18043 83.18043 83.18043
 [41] 83.18043 83.48624 83.48624 83.63914 83.94495 84.09786 83.79205 83.94495
 [49] 83.94495 83.79205 83.94495 83.94495 83.94495 83.94495 83.94495 83.94495
 [57] 83.94495 83.94495 83.79205 83.79205 83.79205 83.79205 83.33333 83.33333
 [65] 83.33333 83.48624 83.63914 83.79205 83.79205 83.79205 83.79205 83.79205
 [73] 83.63914 83.63914 83.79205 83.79205 83.79205 83.63914 83.63914 83.79205
 [81] 83.63914 83.63914 83.63914 83.63914 83.63914 83.63914 83.63914 83.79205
 [89] 83.79205 83.79205 83.79205 83.94495 84.09786 84.09786 83.94495 83.79205
 [97] 83.63914 83.63914 83.63914 83.63914
```