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 financial advising, so a great problem I face weekly would be if an oncoming prospect would be a good fit for me and my firm or not. After each meeting me or anyone in my team has with a prospect, we all talk about if they seem to be a good fit for us or not, so having a model to do just that would be very useful. The predictors that come to mind to use would be – income, assets, and liabilities. I would select these three because those are the baseline topics that we talk about during our initial meetings to judge where they are at financially. For example, we probably would not want to work with someone with a lower income and lower total assets, but high liabilities. We would like to work with someone with large total assets and low liabilities though. A model to give a "work with" or "do not work with" classification would be interesting to see if it usually agrees or disagrees with what we determine personally.

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.)**

   Notes on `ksvm`
   - **You can use `scaled=TRUE` to get `ksvm` to scale the data as part of calculating a classifier.**
   - **The term λ we used in the SVM lesson to trade off the two components of correctness and margin is called C in `ksvm`. One of the challenges of this homework is to find a value of C that works well; for many values of C, almost all predictions will be "yes" or almost all predictions will be "no".**
   - **`ksvm` does not directly return the coefficients a_0 and $a_1 \ldots a_m$. Instead, you need to do the last step of the calculation yourself. Here's an example of the steps to take (assuming your data is stored in a matrix called data):[1]**

Hint: You might want to view the predictions your model makes; if C is too large or too small, they'll almost all be the same (all zero or all one) and the predictive value of the model will be poor. Even finding the right order of magnitude for C might take a little trial-and-error.

Note: If you get the error "`Error in vanilladot(length = 4, lambda = 0.5) : unused arguments (length = 4, lambda = 0.5)`", it means you need to convert `data` into matrix format:

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

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`).**

Part 1 (ksvm):

C = 0.0000001	Accuracy = 0.5474006
C = 0.000001	Accuracy = 0.5474006
C = 0.00001	Accuracy = 0.5474006
C = 0.0001	Accuracy = 0.5474006
C = 0.001	Accuracy = 0.8379205
C = 0.01	Accuracy = 0.8639144
C = 0.1	Accuracy = 0.8639144
C = 1	Accuracy = 0.8639144
C = 10	Accuracy = 0.8639144
C = 100	Accuracy = 0.8639144
C = 1000	Accuracy = 0.8623853
C = 10000	Accuracy = 0.8623853
C = 100000	Accuracy = 0.8639144
C = 1000000	Accuracy = 0.6253823
C = 10000000	Accuracy = 0.5458716

I have decided to measure the ksvm model's performance by attempting to minimize the number of misclassifications in the model while still maintaining the largest accuracy value. This was done by testing a wide range of different C-values. In terms of a higher or lower C, the higher the C-value in the ksvm model, the more weight there is towards correctness. This means that a smaller C-value maximizes the margin between the two parallel lines. Accuracy in this case is defined as the sum of the total correct classifiers/the total amount of classifiers. After hard testing a wide range of C-values, I saw that the accuracy values changed as described in the table above. I found that C = 100000 is the largest C-value that has the highest accuracy value. This means that C = 100000 provides the least number of misclassifications while upholding the highest accuracy value. The corresponding equation of this C-value is: {f(x) = 0.08054451 - 0.004117739x1 - 0.086896089x2 + 0.129715260x3 - 0.083744033x4 + 0.988381366x5 + 0.031253888x6 - 0.055666972x7 - 0.037281856x8 + 0.021940744x9 + 0.018521785x10}. When omitting 0-values, it is: {f(x) = 0.08054451 + 0.129715260x3 + 0.988381366x5}. This equation classifies at the corresponding accuracy to C = 100000 in the table, or 0.8639144.

<u>Code and Output for Part 1:</u>

```
library(kernlab)

CCdata <- read.table("credit_card_data.txt", stringsAsFactors = FALSE, header = FALSE)

CCmodel <- ksvm(V11 ~., data = CCdata, type = "C-svc", kernel = "vanilladot", C = 100000,
scaled = TRUE)

CCmodel

a <- colSums(CCmodel@xmatrix[[1]]*CCmodel@coef[[1]])
a0 <- -CCmodel@b
a
a0

pred <- predict(CCmodel,CCdata[,1:10])
pred
sum(pred == CCdata$V11)/nrow(CCdata)
```

```
>a: V1: -0.004117739, V2: -0.086896089, V3: +0.129715260, V4: -0.083744033, V5:
+0.988381366, V6: +0.031253888, V7: -0.055666972, V8: -0.037281856, V9: +0.021940744, V10:
+0.018521785x10

>a0: 0.08054451

>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 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1
 [52] 1 1 1 1 1 1 1 0 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 1 1 1 1 1 1 1
1
[103] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1
[154] 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1
```

```
[205] 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 1 1 1 1 0 0 0 0 0 0 0 0 0
0
[256] 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 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
[307] 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
[358] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
[409] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
[460] 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1
[511] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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
[562] 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
[613] 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 0 0 0 0 0 0 0 0 0 0 0

> sum(pred == CCdata$V11)/nrow(CCdata): 0.8639144
```

<u>Part 2 (KNN):</u>

The k-value that I found produced the highest accuracy is a k-value of 12. The accuracy the k-value of 12 produced is 0.853211. Instead of hard inputting different values of k and i for the knn like I did for the C-value in the ksvm, I decided to run them on a loop. The k-values I ran the loop with were 1 to 100 and the i-values were 1 to the number of points in the data set (in this case it was 654). Anything past k = 100 made the code take too long to run and I figured that 100 different k iterations of the code was more than enough to find a pretty accurate best value. I changed the maximum k-value a couple of times to see if it would produce different best k-values depending on the maximum k-value and turns out that it did. When the maximum k-value was set to 5, the best k was k = 5 with an accuracy of 0.851682, when set to 10 it was still k = 5 with the same accuracy as before, when set to 15 the best k-value changed from k = 5 to k = 12 with an accuracy of 0.853211 and it stayed that way to k = 100. So, it seems like k = 12 is the threshold that changes the optimal k-value from 5 to 12. I found it interesting that k = 12 was the optimal value because I would have thought it would take more points than 12 to best classify a point. To further investigate this, I printed out each k-values accuracy result to see if k = 12 really had the highest accuracy, or if there was something else going on. It turned out that it shared accuracy scores with different k-values, for example k = 15. This made me realize that the knn should likely be ran with the lowest k-value with the highest accuracy, even if higher k-values have the same accuracy. This makes sense in an efficiency reasoning because it is quicker to use less points to classify over a large data set if it will result in the same accuracy as a larger k-value.

<u>Code and Output for Part 2:</u>

```
library(kknn)
```

```
max_k <- 100
num_points <- nrow(CCdata)
accuracy_results <- numeric(max_k)
for (k in 1:max_k) {
        correct_predictions <- 0
        for (i in 1:num_points) {
                CCmodel_knn <- kknn(V11~., CCdata[-i,], CCdata[i,], k = k, distance = 2,
                kernel = "optimal", scale = TRUE)
                predicted_class <- round(fitted.values(CCmodel_knn))
                actual_class <- CCdata[i, 11]
                if (predicted_class == actual_class) {
                        correct_predictions <- correct_predictions + 1
                }
        }
        accuracy_results[k] <- correct_predictions / num_points
}
best_k <- which.max(accuracy_results)
best_accuracy <- accuracy_results[best_k]
print(best_k)
print(best_accuracy)
```

```
>print(best_k): 12

>print(best_accuracy): 0.853211
```