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.

In my free time, I am typically playing video games. Although I do not play as much I did when I was younger, I love playing League of Legends and ToonTown. Sadly, I find it very difficult to explore new titles and find games that look like something I would enjoy; and if I do find something interesting or recommended to me, I end up getting bored with it rather quickly. Although Steam does a fantastic job, I think it would be more effective to recommend games based on a numeric rating on how likely a person is to enjoy a game (rather than just categorizing into "recommended/ not recommended"). Predictors that can be used are:

1. Genre Preference – What genres do they mostly own? Do they play them a lot?

2. Time played (individual titles and/or genre)

3. Graphics/Art style – More enjoyment from cartoony style? Or realistic?

4. Social Interaction Level – Do they mostly play single player or multiplayer?

5. Storytelling/Themes (Do I mainly play story games/are there common themes?)

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.)

Calling libraries

```
#housekeeping
library(pacman)
pacman::p_load(rio, kernlab, kknn, rmarkdown, pandoc, knitr)
```

Importing the dataset using Rio and using head to preview the data to ensure it was read in properly.

```
#import dataset (without header)
data <- (import("D:/Users/Marcus/Desktop/grad school/FALL 2024/Analytic Modeling/hw 1/credit_card_data.txt"))
```

```
#previewing data
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
```

I am now building the model. I kept the C value at 100 since changing it to random number between 0-1000 had no effect on the accuracy of the model. Going to higher numbers caused the accuracy to drop. Therefore, I left it at C=100 since that seems to be the best.

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

##  Setting default kernel parameters
```

The next section finds the coefficients for $a_1x_1+\dots a_nx_n + a_0 = 0$, given the parameters we set for the model.

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

## [1] 0.08158492
```

The coefficients are:

$-0.0010065348x_1 + -0.0011729048x_2 + -0.0016261967x_3 + 0.0030064203x_4 + 1.0049405641x_5 + -0.0028259432x_6 + 0.0002600295x_7 + -0.0005349551x_8 + -0.0012283758x_9 + 0.1063633995x_{10} + 0.08158492 = 0$

The next section will build the prediction model and compare the prediction to the actual classification

```
#prediction model
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 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
```

```
#model prediction vs actual classification
sum(pred == data[,11]) / nrow(data)
```

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

The model gives us a prediction accuracy of 0.8639144 (margin of error: 0.1360856). In other words, it is saying that its results should be 86% accurate.

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.

This is a modified version of the code provided. By creating a list called other_kernels, filled with the names of the kernel arguments, I can create a loop that quickly tests various kernels for our prediction model.

```
#housekeeping
library(pacman)
pacman::p_load(rio, kernlab, kknn)

#import dataset (without header)
data <- (import("D:/Users/Marcus/Desktop/grad school/FALL 2024/Analytic Modeling/hw 1/credit_card_data.txt"))
```

```r
#previewing data
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
```

```r
#call ksvm
other_kern <-
list("rbfdot","laplacedot","polydot","tanhdot","besseldot","anovadot","spline
dot")
for (k in other_kern) {
  model <- ksvm(as.matrix(data[,1:10]), as.factor(data[,11]),
                type="C-svc",
                kernel=k,
                C=100,
                scaled=TRUE)

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

  # accuracy
  acc = sum(pred == data[,11]) / nrow(data)
  print(paste0(k, "=", acc))

}
```

```
## [1] "rbfdot=0.954128440366973"
## [1] "laplacedot=1"
##  Setting default kernel parameters
## [1] "polydot=0.863914373088685"
##  Setting default kernel parameters
## [1] "tanhdot=0.7217125382263"
##  Setting default kernel parameters
## [1] "besseldot=0.925076452599388"
##  Setting default kernel parameters
## [1] "anovadot=0.906727828746177"
##  Setting default kernel parameters
## [1] "splinedot=0.978593272171254"
```

These are the accuracies for each kernel; laplacedot has the highest accuracy.

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).

```r
#housekeeping
library(pacman)
pacman::p_load(rio, kernlab, kknn)

#import dataset (without header)
data <- (import("D:/Users/Marcus/Desktop/grad school/FALL 2024/Analytic
Modeling/hw 1/credit_card_data.txt"))

#previewing data
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
```

```r
acc_knn = function(X){
  pred <- rep(0,(nrow(data))) # initialize vector of 0s

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

    # data[-i] means remove row i of the data when finding NN.
    # using scaled data

    model=kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10,data[-i,],data[i,],k=X,
scale = TRUE)

    #for rounding to 0 or 1
    pred[i] <- as.integer(fitted(model)+0.5)
  }

    #calculation for accuracy
    acc = sum(pred == data[,11]) / nrow(data)
    return(acc)
}


#initalize vector of 30 0s
knn_test <- rep(0,30)
for (X in 1:30){
  knn_test[X] = acc_knn(X) #tests knn with x neighbors

}

knn_test
```

```
##  [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 0.8486239
## [22] 0.8470948 0.8440367 0.8455657 0.8455657 0.8440367 0.8409786 0.8379205
## [29] 0.8394495 0.8409786
```

Using kNN shows that at k = 12 and k = 15 are the most accurate.