Using personality factors to predict the frequency of heroin use

Ritwik Dasika, Rishab Katteri, Avery Katz, and Ishaan Nair

Rutgers University

April 19th, 2021

Abstract

The increase in heroin use has been a cause for concern for the United States of America, as we've seen the heroin overdose death rate double during 2011 to 2013. Given the scope, we want to identify a relationship between psychosocial factors (i.e. neuroticism, extraversion, openness to experience, and agreeableness, just to name a few) and the frequency of heroin use. We will implement some classification techniques tailored for multiple categories ($g \geq 2$) to classify heroin abuse based on these psychosocial factors. In order to determine the efficiencies of these classifications, we will use confusion matrices and cross validation to determine the best classification technique to use for this dataset.

Introduction

Drug abuse in general has been a serious issue in the United States of America. The recent spikes in opioid addictions and the impact that it has on communities is something that needs to be remedied. For the purposes of this project, we're going to be addressing heroin use. The inspiration for this project came from a recent report published by the Center for Disease Control and Prevention (CDC) and it was discussing the characteristics and identifying patterns of heroin users. The report established that heroin availability and use was steadily increasing, and from 2011 to 2013, the estimated annual average rate of adult users was 3.6 per 1000 for men and 1.6 per 1000 for women. Interestingly enough, the demographic that experienced the highest increase in heroin use was the 18-25 age group, as heroin use had increased by almost 108.6% between 2002-2004 to 2011-2013. Common sociological and psychosocial factors associated with heroin use were education levels, ethnicity, annual income, and use of other drugs in years past or concurrently (cannabis, heroin, ecstasy, etc.). To get a better understanding of the psychological blueprint of a heroin user, we want to identify personality characteristics and tendencies, along with parameters such as age,

gender, and education levels, that can be associated with the use of heroin in order to effectively classify and predict future heroin users.

Materials & Methods

The dataset that we will be using is from the UCI Machine Learning Repository, provided by UCI Center for Machine Learning and Intelligent Systems. Listed below are the variables that exist after the curation process:

- AGE (age group of individual)
- GENDER (male or female)
- EDU (education level)
- NSCORE (neuroticism - irrationality)
- ESCORE (extraversion - expressiveness)
- OSCORE (openness to experience)
- ASCORE (agreeableness)
- CSCORE (conscientiousness)
- IMPUL (impulsiveness)
- SS (sensation seeking/thrill seeking)
- HERO (heroin consumption)

We will be using the R programming language and RStudio to perform the classifications and analysis.

The first thing we will do is load the necessary libraries:

```
library(MASS)

library(car)

library(caret)

library(klaR)

library(heplots)

library(caret)
```

```
library(corrplot)

library(PredPsych) # 10 fold CV

library(faraway)

library(psych)

library(tidyverse)

library(pROC)

library(class)

library(e1071)

library(reprex)

library(DMwR)

library(vegan)
```

The next step is to load our data in:

```
drug = read.csv("drug-consump.csv")
```

We will then extract the necessary variables to curate it and tailor it to heroin consumption:

```
social = drug[,2:4]
categ = drug[,7:13]
use = drug[,21:24]
HERO = drug[,24]
```

Next, we need to assign numerical categories to each frequency of heroin consumption. The levels are detailed below:

CL0 - Never Used
CL1 - Used over a Decade Ago
CL2 - Used in Last Decade
CL3 - Used in Last Year
CL4 - Used in Last Month

CL5 - Used in Last Week
CL6 - Used in Last Day

The following for loop assigns numerical values to each group:

```
for(i in 1:1885){
  if(HERO[i] == "CL0"){
    HERO[i] = 0
  } else if(HERO[i] == "CL1"){
    HERO[i] = 1
  } else if(HERO[i] == "CL2"){
    HERO[i] = 2
  } else if(HERO[i] == "CL3"){
    HERO[i] = 3
  } else if(HERO[i] == "CL4"){
    HERO[i] = 4
  } else if(HERO[i] == "CL5"){
    HERO[i] = 5
  } else if(HERO[i] == "CL6"){
    HERO[i] = 6
  }
}
```

Convert the heroin using as.numeric().

```
HERO = as.numeric(HERO)
```

The final step of the curation process is to create the data frame:

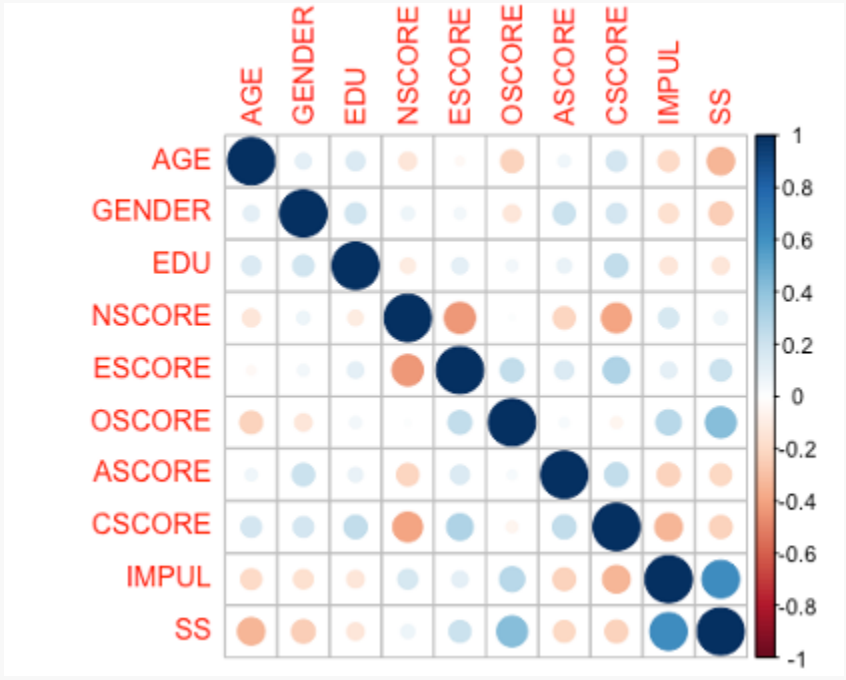
```
hero = data.frame(social, categ, HERO)
```

Now we are going to create our training and testing datasets:

```
set.seed(123)
index = sample(1:nrow(hero),round(0.75*nrow(hero)))
train = hero[index,]
test = hero[-index,]
```

We use the following R code to output a correlation plot of the correlation matrix between the predictor variables:

```
corrplot(cor(hero[,1:10]))
```



The above correlation plot shows us that there is very minimal correlation between the predictor variables except for IMPUL and SS.

Next we are going to use Box's M-test for Homogeneity of Covariance Matrices to check the assumption of homogeneity of the variance-covariance matrices.

H_0= The covariance matrices among the different groups are equal.
H_i= The covariance matrices among the different groups are not equal.

```
#Box's M test

(boxm=boxM(hero[1:10],hero$HERO))

##

##  Box's M-test for Homogeneity of Covariance Matrices

##
```

```
## data:  hero[1:10]

## Chi-Sq (approx.) = 457.73, df = 330, p-value = 3.908e-06
```

Our p-value is well under the alpha level of 0.05, which means we can reject the null hypothesis. Box's M test is very volatile and sensitive, so unless the sample sizes are unequal, we can pretty much ignore it. The only takeaway would be that the matrices are not homogenous.

Now, we are going to perform our linear discriminant analysis:

```
(lda.1 = lda(HERO~., data = train))

## Call:

## lda(HERO ~ ., data = train)

##

## Prior probabilities of groups:

##           0           1           2           3           4           5

## 0.847949081 0.039603960 0.050919378 0.032531825 0.012729844 0.007779349

##           6

## 0.008486563

##

## Group means:

##           AGE      GENDER          EDU     NSCORE      ESCORE      OSCORE

## 0  0.04152823  0.01971688  0.050525905 -0.0560435  0.02498922 -0.06534437

## 1  0.81706679 -0.05169214 -0.002557679  0.2845154 -0.35264000  0.19051018

## 2 -0.04195639 -0.16082000 -0.401630417  0.2900242 -0.14599208  0.56895597

## 3 -0.40717826 -0.16781217 -0.147683696  0.3577461 -0.06141717  0.40842587

## 4 -0.64530389 -0.21442667 -0.837240000  0.2716350 -0.24078000  0.13869889

## 5 -0.31651273 -0.30702000 -0.340178182  0.7871136 -0.66653000  0.12773091

## 6 -0.68557750 -0.24123000 -0.182171667  1.0397050 -0.69579667  0.34031417

##        ASCORE      CSCORE       IMPUL          SS

## 0  0.05710467  0.06211379 -0.06889213 -0.07046311
```

```
## 1 -0.22885804 -0.32308107  0.15037482  0.14914143
## 2 -0.28148806 -0.32336153  0.56281333  0.54398681
## 3 -0.38888935 -0.15454565  0.40351630  0.47681283
## 4 -0.46504889 -0.54999611  0.47506333  0.79067833
## 5 -0.74626273 -0.93962545  0.77080727  0.74238364
## 6 -0.96305167 -0.66808667  0.43737667  0.49802750
##
## Coefficients of linear discriminants:
##                 LD1         LD2         LD3         LD4         LD5         LD6
## AGE     0.286753946 -1.19624385  0.11467905 -0.06691943 -0.09319027  0.22786615
## GENDER -0.388533528 -0.19336753  0.35687393 -0.28168275 -0.55211228 -0.73892470
## EDU    -0.290515782 -0.06239461 -0.71849416  0.32010126  0.39575891  0.38045835
## NSCORE  0.270273519  0.00201615 -0.45633793  0.36938194 -0.17665299  0.16070188
## ESCORE -0.317201954  0.07703089  0.12038059  0.19294417 -0.09589631  0.43414336
## OSCORE  0.428776649 -0.17223760  0.30378821  0.69956837 -0.30832869 -0.47500982
## ASCORE -0.254894978 -0.11326276  0.31883912 -0.07714520  0.49589434 -0.01082464
## CSCORE -0.009827026  0.18084228  0.18164942  0.52951643 -0.28098970  0.29206735
## IMPUL   0.214834708  0.06832459  0.22775177  0.40530071  1.09232043 -0.19369418
## SS      0.349552388 -0.11849117  0.08085545 -0.66606323 -0.45933280  0.91378682
##
## Proportion of trace:
##    LD1    LD2    LD3    LD4    LD5    LD6
## 0.5777 0.3083 0.0610 0.0411 0.0077 0.0042
```

To assess the efficiency of the LDA, we will use the confusionMatrix() function:

```
fit.values = predict(lda.1, test)

confusionMatrix(fit.values$class, as.factor(test$HERO))

## Confusion Matrix and Statistics
```

```
##
##            Reference
## Prediction   0   1   2   3   4   5   6
##          0 406  11  22  19   6   5   1
##          1   0   1   0   0   0   0   0
##          2   0   0   0   0   0   0   0
##          3   0   0   0   0   0   0   0
##          4   0   0   0   0   0   0   0
##          5   0   0   0   0   0   0   0
##          6   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.8641
##                  95% CI : (0.8298, 0.8938)
##     No Information Rate : 0.862
##     P-Value [Acc > NIR] : 0.4798
##
##                   Kappa : 0.0279
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           1.00000 0.083333  0.00000  0.00000  0.00000  0.00000
## Specificity           0.01538 1.000000  1.00000  1.00000  1.00000  1.00000
## Pos Pred Value        0.86383 1.000000      NaN      NaN      NaN      NaN
## Neg Pred Value        1.00000 0.976596  0.95329  0.95966  0.98726  0.98938
## Prevalence            0.86200 0.025478  0.04671  0.04034  0.01274  0.01062
```

```
## Detection Rate         0.86200 0.002123  0.00000  0.00000  0.00000  0.00000
## Detection Prevalence   0.99788 0.002123  0.00000  0.00000  0.00000  0.00000
## Balanced Accuracy      0.50769 0.541667  0.50000  0.50000  0.50000  0.50000
##                      Class: 6
## Sensitivity          0.000000
## Specificity          1.000000
## Pos Pred Value            NaN
## Neg Pred Value       0.997877
## Prevalence           0.002123
## Detection Rate       0.000000
## Detection Prevalence 0.000000
## Balanced Accuracy    0.500000
```

Now we will perform a 10 fold cross validation to further analyze the linear discriminant:

```
LDAModel = LinearDA(Data = hero, classCol = 11, cvType = "folds", nTrainFolds = 10)
##
## Performing Linear Discriminant Analysis
##
##
## Performing k-fold Cross-validation
##
##
## ntrainTestFolds was not specified,
##  Using default value of 10 (ntrainTestFolds = 10)
##
## modelTrainFolds were not specified,
##  Using default value of 1:9
##       Predicted
## Actual   0   1   2   3   4   5   6
```

```
##      0 160   0   0   0   0   0   0
##      1   6   0   0   0   0   0   0
##      2  10   0   0   0   0   0   0
##      3   7   0   0   0   0   0   0
##      4   3   0   0   0   0   0   0
##      5   2   0   0   0   0   0   0
##      6   2   0   0   0   0   0   0
## [1] "The accuracy of discrimination was 0.85"
## [1] "Test Accuracies"
##       Predicted
## Actual   0   1   2   3   4   5   6
##      0 160   0   0   0   0   0   0
##      1   6   0   0   0   0   0   0
##      2  10   0   0   0   0   0   0
##      3   7   0   0   0   0   0   0
##      4   3   0   0   0   0   0   0
##      5   2   0   0   0   0   0   0
##      6   2   0   0   0   0   0   0
## [1] "The accuracy of discrimination was 0.84"
## k-fold LDA Analysis:
## ntrainTestFolds : 10
## modelTrainFolds 1 2 3 4 5 6 7 8 9
## nTrainFolds: 10
## Test Accuracy 0.84
## *Legend:
## ntrainTestFolds = No. of folds for training and testing dataset
## modelTrainFolds = Specific folds from the above ntrainTestFolds to use for training
## nTrainFolds = No. of folds in which to further divide Training dataset
## Test Accuracy = Mean accuracy from the Testing dataset
```

The 10-fold cross validation showed us that the linear discriminant model's accuracy on the dataset was 0.84, which is comparable to the value we got from the confusion matrix.

Let's take this one step further and try the holdout method:

```
LDAmod = LinearDA(Data = hero, classCol = 11, cvType = "holdout")

##

## Performing Linear Discriminant Analysis

##

##

## Performing holdout Cross-validation

## cvFraction was not specified,

##  Using default value of 0.8 (80%) fraction for training
(cvFraction = 0.8)

##

## Proportion of Test/Train Data was :  0.2475182

##       Predicted

## Actual   0   1   2   3   4   5   6

##      0 320   1   0   0   0   0   0

##      1  12   1   0   0   0   0   0

##      2  18   0   0   0   0   0   0

##      3  13   0   0   0   0   0   0

##      4   4   0   0   0   0   0   0

##      5   3   0   0   0   0   0   0

##      6   2   0   0   0   0   0   0

## [1] "Test holdout Accuracy is  0.86"

## holdout LDA Analysis:
```

```
## cvFraction : 0.8

## Test Accuracy 0.86

## *Legend:

## cvFraction = Fraction of data to keep for training data

## Test Accuracy = Accuracy from the Testing dataset
```

We find that the holdout accuracy was 0.86, where it split the training and testing dataset. This is similar to the value we obtained from the confusion matrix, where the split was determined with 75% of the data. Here, our split was 80% of the dataset, which is why the cross validation here is slightly lower than the one obtained from the confusion matrix. In the long run, it's better to use our 10 fold cross validation to evaluate the legitimacy of our LDA.

Our LDA model accuracy was 0.84 according to the 10 fold cross validation, which is pretty solid. However, there can be another, more efficient way of classifying the observations.

Let's try running a quadratic discriminant analysis:

```
#QDA

qda.1 = qda(HERO~., data = train)

qda.1

## Call:

## qda(HERO ~ ., data = train)

##

## Prior probabilities of groups:

##           0           1           2           3           4           5

## 0.847949081 0.039603960 0.050919378 0.032531825 0.012729844 0.007779349

##           6

## 0.008486563
```

```
##
## Group means:
##           AGE      GENDER          EDU     NSCORE      ESCORE      OSCORE
## 0  0.04152823  0.01971688  0.050525905 -0.0560435  0.02498922 -0.06534437
## 1  0.81706679 -0.05169214 -0.002557679  0.2845154 -0.35264000  0.19051018
## 2 -0.04195639 -0.16082000 -0.401630417  0.2900242 -0.14599208  0.56895597
## 3 -0.40717826 -0.16781217 -0.147683696  0.3577461 -0.06141717  0.40842587
## 4 -0.64530389 -0.21442667 -0.837240000  0.2716350 -0.24078000  0.13869889
## 5 -0.31651273 -0.30702000 -0.340178182  0.7871136 -0.66653000  0.12773091
## 6 -0.68557750 -0.24123000 -0.182171667  1.0397050 -0.69579667  0.34031417
##        ASCORE      CSCORE       IMPUL          SS
## 0  0.05710467  0.06211379 -0.06889213 -0.07046311
## 1 -0.22885804 -0.32308107  0.15037482  0.14914143
## 2 -0.28148806 -0.32336153  0.56281333  0.54398681
## 3 -0.38888935 -0.15454565  0.40351630  0.47681283
## 4 -0.46504889 -0.54999611  0.47506333  0.79067833
## 5 -0.74626273 -0.93962545  0.77080727  0.74238364
## 6 -0.96305167 -0.66808667  0.43737667  0.49802750
```

To evaluate the QDA, we'll use a confusion matrix:

```
qda.fit = predict(qda.1, test)
confusionMatrix(qda.fit$class, as.factor(test$HERO))
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 395   9  22  19   5   3   1
##          1   3   3   0   0   0   0   0
```

```
##          2   2   0   0   0   0   0   0
##          3   2   0   0   0   1   1   0
##          4   1   0   0   0   0   0   0
##          5   1   0   0   0   0   1   0
##          6   2   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.8471
##                  95% CI : (0.8114, 0.8784)
##     No Information Rate : 0.862
##     P-Value [Acc > NIR] : 0.8419
##
##                   Kappa : 0.091
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.97291 0.250000 0.000000 0.000000 0.000000 0.200000
## Specificity           0.09231 0.993464 0.995546 0.991150 0.997849 0.997854
## Pos Pred Value        0.87004 0.500000 0.000000 0.000000 0.000000 0.500000
## Neg Pred Value        0.35294 0.980645 0.953092 0.959315 0.987234 0.991471
## Prevalence            0.86200 0.025478 0.046709 0.040340 0.012739 0.010616
## Detection Rate        0.83864 0.006369 0.000000 0.000000 0.000000 0.002123
## Detection Prevalence  0.96391 0.012739 0.004246 0.008493 0.002123 0.004246
## Balanced Accuracy     0.53261 0.621732 0.497773 0.495575 0.498925 0.598927
```

```
##                      Class: 6
## Sensitivity          0.000000
## Specificity          0.995745
## Pos Pred Value       0.000000
## Neg Pred Value       0.997868
## Prevalence           0.002123
## Detection Rate       0.000000
## Detection Prevalence 0.004246
## Balanced Accuracy    0.497872
```

The QDA model's accuracy was 0.8471. Although this isn't terrible either, it certainly was not an improvement over the linear discriminant analysis.

Additionally, the predicted vs actual table can show us that it wasn't an improvement:

```
qda.cv = qda(HERO~., data = train, CV=TRUE)
table(train$HERO, qda.cv$class, dnn = c("Actual", "Predicted"))
##        Predicted
## Actual    0    1    2    3    4    5    6
##      0 1171    7    2   11    2    2    4
##      1   54    2    0    0    0    0    0
##      2   69    1    0    2    0    0    0
##      3   40    0    1    5    0    0    0
##      4   18    0    0    0    0    0    0
##      5   10    0    0    1    0    0    0
##      6   12    0    0    0    0    0    0
```

The issue with this dataset is that there are 7 classes to categorize the data. Multi class classification is a complex technique that will require a

more robust model. Let us use a K-nearest neighbors model to classify the data. This is a suitable algorithm for multi class classification, however, the only drawbacks to using this algorithm are that it takes more time to compute and feature importance is not something that we can determine with KNN.

The first step in implementing KNN is to normalize the data, which we will do using the following R lines:

```
#normalize data
HERO.train.norm = decostand(train, "normalize")
HERO.test.norm = decostand(test, "normalize")
```

Now we must determine the optimal k value using the following code:

```
(HERO.cv = trainControl(method = "repeatedcv", number = 10, repeats = 6))
```

```
## $method

## [1] "repeatedcv"

##

## $number

## [1] 10

##

## $repeats

## [1] 6

##

## $search

## [1] "grid"

##

## $p

## [1] 0.75

##

## $initialWindow
```

```
## NULL
##
## $horizon
## [1] 1
##
## $fixedWindow
## [1] TRUE
##
## $skip
## [1] 0
##
## $verboseIter
## [1] FALSE
##
## $returnData
## [1] TRUE
##
## $returnResamp
## [1] "final"
##
## $savePredictions
## [1] FALSE
##
## $classProbs
## [1] FALSE
##
## $summaryFunction
## function (data, lev = NULL, model = NULL)
```

```
## {
##     if (is.character(data$obs))
##         data$obs <- factor(data$obs, levels = lev)
##     postResample(data[, "pred"], data[, "obs"])
## }
## <bytecode: 0x7fed54bcc528>
## <environment: namespace:caret>
##
## $selectionFunction
## [1] "best"
##
## $preProcOptions
## $preProcOptions$thresh
## [1] 0.95
##
## $preProcOptions$ICAcomp
## [1] 3
##
## $preProcOptions$k
## [1] 5
##
## $preProcOptions$freqCut
## [1] 19
##
## $preProcOptions$uniqueCut
## [1] 10
##
## $preProcOptions$cutoff
```

```
## [1] 0.9
##
##
## $sampling
## NULL
##
## $index
## NULL
##
## $indexOut
## NULL
##
## $indexFinal
## NULL
##
## $timingSamps
## [1] 0
##
## $predictionBounds
## [1] FALSE FALSE
##
## $seeds
## [1] NA
##
## $adaptive
## $adaptive$min
## [1] 5
##
```

```
## $adaptive$alpha
## [1] 0.05
##
## $adaptive$method
## [1] "gls"
##
## $adaptive$complete
## [1] TRUE
##
##
## $trim
## [1] FALSE
##
## $allowParallel
## [1] TRUE
(train.knn = train(HERO~., data = train, method = "knn", trControl = HERO.cv))
## k-Nearest Neighbors
##
## 1414 samples
##   10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 6 times)
## Summary of sample sizes: 1273, 1272, 1272, 1273, 1273, 1272, ...
## Resampling results across tuning parameters:
##
##   k  RMSE      Rsquared    MAE
##   5  1.109335  0.01964058  0.5799095
```

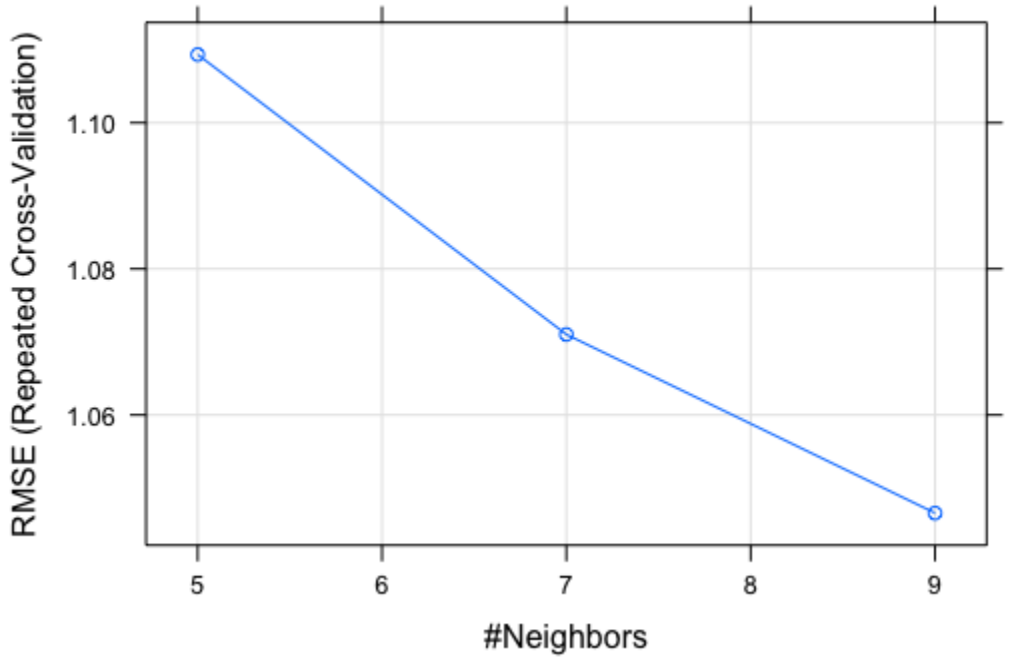
```
##    7  1.071024  0.03184394  0.5758392

##    9  1.046570  0.04447961  0.5670666

##

## RMSE was used to select the optimal model using the smallest value.

## The final value used for the model was k = 9.
```



The output shows us that k=9 returned the most optimal KNN model, according to the root mean squared errors. You can see that the Rsquared value also increased as k increased and mean absolute error decreased accordingly. The plot above also shows us that the RMSE decreases as we increase the number of neighbors (Use the `plot(train.knn)` function to get your plot). The function train() tunes the knn model and performs a resampling based performance measure on the model, which is how it was able to determine the optimal number of neighbors, or k.

Now we will create our KNN model, setting k=9.

```
HERO.knn = knn(HERO.train.norm, HERO.test.norm, cl = lab, k=9, prob=TRUE)
```

```
summary(HERO.knn)
```

```
##   0   1   2   3   4   5   6
## 419   0  19  15   2   1  15
```

To determine the efficiency of this model, we will once again use the confusionMatrix() function:

```
confusionMatrix(HERO.knn, as.factor(test[,11]))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 406  12   1   0   0   0   0
##          1   0   0   0   0   0   0   0
##          2   0   0   9   7   2   1   0
##          3   0   0   7   4   3   1   0
##          4   0   0   1   0   0   1   0
##          5   0   0   1   0   0   0   0
##          6   0   0   3   8   1   2   1
##
## Overall Statistics
##
##                Accuracy : 0.8917
##                  95% CI : (0.8601, 0.9183)
##     No Information Rate : 0.862
##     P-Value [Acc > NIR] : 0.03256
##
##                   Kappa : 0.5289
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000  0.00000  0.40909 0.210526 0.000000 0.000000
## Specificity            0.8000  1.00000  0.97773 0.975664 0.995699 0.997854
## Pos Pred Value         0.9690      NaN  0.47368 0.266667 0.000000 0.000000
## Neg Pred Value         1.0000  0.97452  0.97124 0.967105 0.987207 0.989362
## Prevalence             0.8620  0.02548  0.04671 0.040340 0.012739 0.010616
## Detection Rate         0.8620  0.00000  0.01911 0.008493 0.000000 0.000000
## Detection Prevalence   0.8896  0.00000  0.04034 0.031847 0.004246 0.002123
## Balanced Accuracy      0.9000  0.50000  0.69341 0.593095 0.497849 0.498927
##                      Class: 6
## Sensitivity          1.000000
## Specificity          0.970213
## Pos Pred Value       0.066667
## Neg Pred Value       1.000000
## Prevalence           0.002123
## Detection Rate       0.002123
## Detection Prevalence 0.031847
## Balanced Accuracy    0.985106
```

We get an accuracy of 0.8917 when using a KNN model. From the accuracy, we can see that KNN is a better classifier for this dataset as opposed to a linear or quadratic discriminant analysis. In addition, the KNN model has a sensitivity of 1.0 for class 6, which means it predicts the individuals who have used heroin in the last day with full accuracy, and has a specificity of 0.970 for class 6, which means it predicts the

individuals who have not used heroin in the last day with about 97% accuracy, all based on the predictor variables. On the flip side, it is able to predict class 0 true positive individuals with full accuracy (people who have never used heroin) and it is able to identify true negative individuals who have used heroin with 80% accuracy (20% being false positives for heroin use).

Results

10-fold CV LDA:

```
k-fold LDA Analysis:

## ntrainTestFolds : 10

## modelTrainFolds 1 2 3 4 5 6 7 8 9

## nTrainFolds: 10

## Test Accuracy 0.84
```

LDA Confusion Matrix:

```
          Reference

## Prediction   0   1   2   3   4   5   6

##          0 406  11  22  19   6   5   1

##          1   0   1   0   0   0   0   0

##          2   0   0   0   0   0   0   0

##          3   0   0   0   0   0   0   0

##          4   0   0   0   0   0   0   0

##          5   0   0   0   0   0   0   0

##          6   0   0   0   0   0   0   0

##

## Overall Statistics

##

##                Accuracy : 0.8641
```

```
##                  95% CI : (0.8298, 0.8938)
##     No Information Rate : 0.862
##     P-Value [Acc > NIR] : 0.4798
##
##                   Kappa : 0.0279
```

QDA Confusion Matrix:

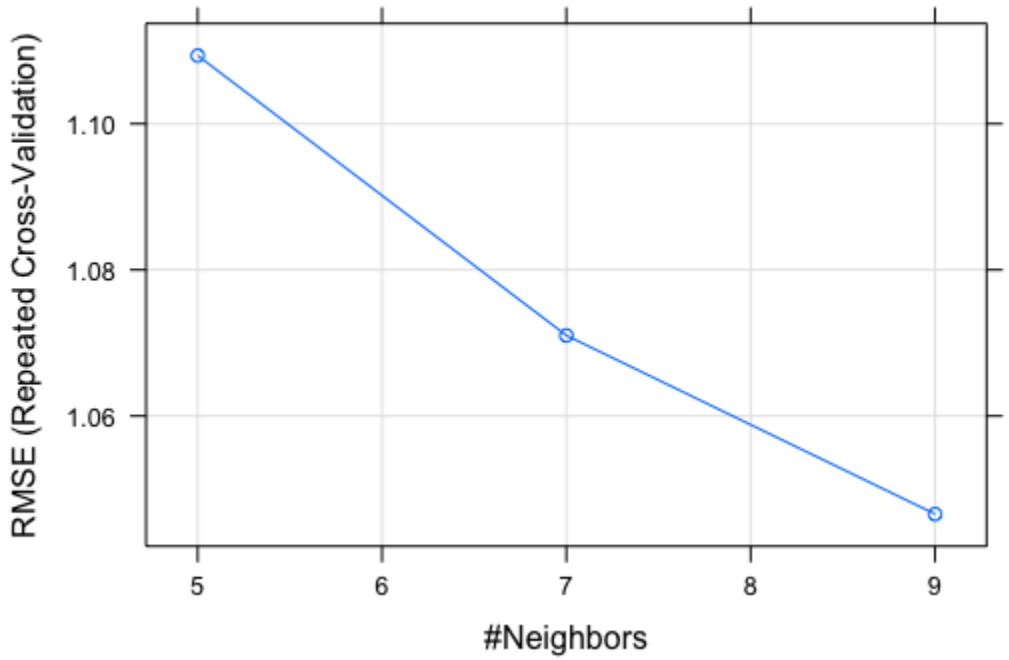
```
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 395   9  22  19   5   3   1
##          1   3   3   0   0   0   0   0
##          2   2   0   0   0   0   0   0
##          3   2   0   0   0   1   1   0
##          4   1   0   0   0   0   0   0
##          5   1   0   0   0   0   1   0
##          6   2   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.8471
##                  95% CI : (0.8114, 0.8784)
##     No Information Rate : 0.862
##     P-Value [Acc > NIR] : 0.8419
##
##                   Kappa : 0.091
```

KNN Confusion Matrix:

```
##           Reference
## Prediction   0   1   2   3   4   5   6
```

```
##          0 406  12   1   0   0   0   0
##          1   0   0   0   0   0   0   0
##          2   0   0   9   7   2   1   0
##          3   0   0   7   4   3   1   0
##          4   0   0   1   0   0   1   0
##          5   0   0   1   0   0   0   0
##          6   0   0   3   8   1   2   1
##
## Overall Statistics
##
##                Accuracy : 0.8917
##                  95% CI : (0.8601, 0.9183)
##     No Information Rate : 0.862
##     P-Value [Acc > NIR] : 0.03256
##
##                   Kappa : 0.5289
```

K vs RMSE Plot:



Discussion

We will examine the pros and cons of each of the techniques we used in this experiment.

LDA

The pros of using LDA would be that it is a very simple, fast, and portable algorithm which beats some algorithms, like logistic regression, and is most typically used when its assumptions are met. The cons, however, is that it requires a normality assumption on the features and predictors, which to us, is the number of classes we were testing the dataset on. It also might not be the best fit in terms of modelling for specific categorical variables.

QDA

For QDA, the pros are that it is more flexible to use than LDA because for LDA, we have to estimate fewer parameters. Additionally, QDA assumes that each class has its own covariance matrix, which is different from LDA. When our K classes have different covariance matrices, LDA suffers from high bias and QDA might be a better choice in that regard. So, essentially, it becomes a standoff between bias and variance. Thus, it is crucial to test the underlying assumptions of LDA and QDA on any data set we use and see which method would be better suited for how we want to analyze the data.

However, the biggest con we have is the curse of dimensionality. The higher the dimension of the data set, or the more predictor variables we have, the more parameters we have to estimate. This, as a result, would lead to high variance, which could further complicate our experimental analysis, so we have to be careful when we decide to use QDA over LDA.

KNN

For KNN classification, or k-nearest neighbors, the pros are that it is the simplest algorithm and is highly accurate when compared to LDA or QDA, making it easy to interpret, understand, and implement. It also doesn't make any assumptions about the data, meaning it can be used for a good majority of problems that we encounter. KNN is mostly used for regression and classification tasks, which is what we are implementing here, unlike some other supervised learning algorithms.

However, the cons are that KNN stores most of all the data, meaning that the mode requires a lot of memory and could be computationally expensive. Large datasets, as a result, might cause predictions to take a long time. KNN could also be very sensitive to the scale of the dataset and can be thrown off by irrelevant features or predictor variables fairly easily in comparison to other models. It makes sense since it is a classification

technique, not specifically a model building function, so it would most likely contribute more to multicollinearity as well as overfit to the dataset instead of adjusting for new test data in the chance we record some more.

Overall

After explaining each of the pros and cons of LDA, QDA, and KNN, we can then assess which would be the best in terms of our own modelling purposes. Seeing that we have multiple classes for each of the substance usage for heroin, we decided that a classification technique would suit the dataset the best. Our sensitivity analysis produced results that we were satisfied with, as it predicts with very high accuracy for whether a person used or did not use heroin in the past day or so. With linear discriminant analysis, we see that it boasts an 86.17% accuracy, which is also satisfactory in terms of the scope of our dataset analysis, but it does not analyze the data as well as KNN does, which is why we opted to choose KNN over LDA. When we conducted our QDA, it boasted an 84.71% accuracy, which is not as good as our LDA results. Thus, QDA and LDA would be ruled out and KNN would make the most sense to proceed with in our analysis. It makes sense as we are not bringing in any more test data and are just analyzing what is in front of us, the dataset itself. Since we are not going to include any more data, it makes more sense to implement KNN to best describe the dataset.

After the conclusion of this project, we were able to create a classification model that utilized the various personality and psychosocial factors available in the dataset, indicating that these variables are able to predict heroin use to some extent. Essentially, it lines up with the studies and analysis conducted by the CDC, which point out the strong correlations between the factors, such as age, gender, education levels, annual income, etc., and heroin use and overdoses.

Acknowledgements

Class material provided by Professor Mardekian, Rutgers University. The textbook used was Applied Multivariate Statistical Analysis, 6th Edition, by Richard Arnold Johnson and Dean W. Wichern. Dataset provided by UCI Center for Machine Learning and Intelligent Systems.

Literature Cited

CDC:
https://www.cdc.gov/media/releases/2015/p0707-heroin-epidemic.html

https://www.cdc.gov/mmwr/preview/mmwrhtml/mm6426a3.htm

Dataset:
https://archive.ics.uci.edu/ml/datasets/Drug+consumption+%28quantified%29#

Linear vs. Quadratic Discriminant Analysis - Comparison of Algorithms - (thatdatatho.com)

What is a KNN (K-Nearest Neighbors)? | Unite.AI

Pros and Cons of popular Supervised Learning Algorithms | by Sumaiya Sande | Analytics Vidhya | Medium

Contributions

All group members contributed to the project equally, from data curation to coding to the write up, to ensure the timely completion and delivery of the project. Ritwik was primarily responsible for allocating tasks, coding, and submitting the project on time, Rishab was responsible for

interpreting the results, Avery was responsible for analyzing the LDA, and Ishaan was responsible for discussing the cross validations. All group members participated in the review of the project.

3 Exam Questions

1. How would non-equal feature covariance matrices change the decision boundary created by the LDA? (Short Answer)

   Answer: Non-equal feature covariance matrices will change the decision boundary created by the LDA from a linear boundary to a quadratic boundary, effectively changing the LDA to a QDA.

2. The goal of cross validation is prediction or to determine how the results of a statistical model will perform to an independent dataset. (True/False)

   Answer: True

3. Sensitivity = (# of true _____)/(# of true _____ + # of false _____)
   Choices:
   a) Positives, negatives, positives
   b) Positives, positives, positives
   c) Positives, positives, negatives
   d) Negatives, negatives, negatives
   Answer: c