Coursework (CM3111)
Big Data Analytics

Philip Tsvetanov
1408561

December 19, 2017

1 Data Exploration

1.1 Data Choice

In modern days, there is a new field of research that focuses on finding a problem before it even has happened - it's way easier to prevent bad human traits than to fix their results. Traits that are different for each person - like susceptance to sickness, either physical (Diabetes, allergies, cancer) or mental (depression, suicide, radical ideologies) - might easily diagnosed in the future, which would warn the people to stay alert at any potential signs. Millions of lives can be saved and a lot of pain can be evaded if we use such scientific methods. One of the things that will make that possible is working with Big Data. In my coursework I am using a data set from a poll where the participants were asked questions that relate to three major topics: personal information, personality traits and drug consumption.

I downloaded my data set from the UCI Machine Learning Repository on 30.10.2017. Link to the data set: http://archive.ics.uci.edu/ml/datasets/Drug+consumption+%28quantified%29

1.2 Technology-Platform

I chose not to use and Big Data tools, because my data set consisted of just less than 2,000 rows, so I found that working with standard tools like R and RStudio to be sufficient for the computation that were needed for this task.

1.3 Problem Statement Data Exploration

This data set consists of 1885 respondents that got surveyed for their drug usage. Note that this does not include only illegal substances, but also more trivial substances like Alcohol, Nicotine and Caffeine consumption. The aim is to predict weather a person is a user or a non-user of a said drug - a classification problem. The authors provided several different problems in the data set description that can be tackled, and I have chosen to predict weather a certain participant is a cannabis consumer or not. Cannabis is perhaps the most popular illegal drug in our society and it's been target of great public debate over the past years. Some countries and states within the USA have gone so far to allow its recreational or medical usage. I have seen and heard of many young people to engage in Cannabis consumption during their break at work and this could be quite dangerous for the employee themselves, the other employees, the clients and the image of the company, if an incident was to occur. A tool that can accurately predict whether a person is likely to be a cannabis user or not could be used by companies to evaluate potential Cannabis users that are interviewing for a job and reconsider them in cases where the job requires working in risk-rich environments. Of course, if such a tool was developed, it can be used to predict other types of drug usage as well.

```r
#setting up working directory
getwd()

## [1] "D:/My Documents/Computer Science 3/BigData"

setwd("D:/My Documents/Computer Science 3/BigData/")

#reding file
df = read.csv('drug_consumption.data', header=T, na.strings= c(""))
```

Time to explore the data

```r
cat("Number of columns: ", ncol(df))

## Number of columns:  32

cat("Number of columns: ", nrow(df))

## Number of columns:  1884
```

The data set consists of 1885 rows and 32 columns.

```r
names(df)

##  [1] "X1"         "X0.49788"   "X0.48246"   "X.0.05921" "X0.96082"
##  [6] "X0.12600"   "X0.31287"   "X.0.57545" "X.0.58331" "X.0.91699"
## [11] "X.0.00665" "X.0.21712" "X.1.18084" "CL5"        "CL2"
## [16] "CL0"        "CL2.1"      "CL6"        "CL0.1"     "CL5.1"
## [21] "CL0.2"      "CL0.3"      "CL0.4"      "CL0.5"     "CL0.6"
## [26] "CL0.7"      "CL0.8"      "CL0.9"      "CL0.10"    "CL2.2"
## [31] "CL0.11"     "CL0.12"
```

The columns are named that way, because the data has went through a normalization process, but each attribute is explained in the data set's webpage.

As we continue exploring the data we have to start pre-processing it, so we can understand it better, so the next paragraph could be considered as part of both *Data Exploration* and *Preprocessing* subsections.
First thing we want to process are the features It will be hard to work with their default labels.

```r
names(df) =c('ID','Age','Gender', 'Education', 'Country', 'Ethnicity', 'Nscore', 'Escore',
         'Oscore', 'Ascore', 'Cscore', 'Impulsiveness', 'Sensation','Alcohol', 'Amphet', 'Amyl',
         'Benzos', 'Caffeine', 'Cannabis', 'Chocolate', 'Cocaine', 'Crack', 'Ecstasy', 'Heroin',
          'Ketamine', 'Legals', 'LSD', 'Meth', 'Mushrooms', 'Nicotine', 'Semer', 'VSA')
```

Now that the attributes are labeled we can have a look at their values:

```r
head(df)

##   ID      Age   Gender Education  Country Ethnicity   Nscore   Escore
## 1  2 -0.07854 -0.48246   1.98437  0.96082  -0.31685 -0.67825  1.93886
## 2  3  0.49788 -0.48246  -0.05921  0.96082  -0.31685 -0.46725  0.80523
## 3  4 -0.95197  0.48246   1.16365  0.96082  -0.31685 -0.14882 -0.80615
## 4  5  0.49788  0.48246   1.98437  0.96082  -0.31685  0.73545 -1.63340
## 5  6  2.59171  0.48246  -1.22751  0.24923  -0.31685 -0.67825 -0.30033
## 6  7  1.09449 -0.48246   1.16365 -0.57009  -0.31685 -0.46725 -1.09207
```

```
##     Oscore   Ascore   Cscore Impulsiveness Sensation Alcohol Amphet Amyl
## 1  1.43533  0.76096 -0.14277      -0.71126  -0.21575     CL5    CL2  CL2
## 2 -0.84732 -1.62090 -1.01450      -1.37983   0.40148     CL6    CL0  CL0
## 3 -0.01928  0.59042  0.58489      -1.37983  -1.18084     CL4    CL0  CL0
## 4 -0.45174 -0.30172  1.30612      -0.21712  -0.21575     CL4    CL1  CL1
## 5 -1.55521  2.03972  1.63088      -1.37983  -1.54858     CL2    CL0  CL0
## 6 -0.45174 -0.30172  0.93949      -0.21712   0.07987     CL6    CL0  CL0
##   Benzos Caffeine Cannabis Chocolate Cocaine Crack Ecstasy Heroin Ketamine
## 1    CL0      CL6      CL4       CL6     CL3   CL0     CL4    CL0      CL2
## 2    CL0      CL6      CL3       CL4     CL0   CL0     CL0    CL0      CL0
## 3    CL3      CL5      CL2       CL4     CL2   CL0     CL0    CL0      CL2
## 4    CL0      CL6      CL3       CL6     CL0   CL0     CL1    CL0      CL0
## 5    CL0      CL6      CL0       CL4     CL0   CL0     CL0    CL0      CL0
## 6    CL0      CL6      CL1       CL5     CL0   CL0     CL0    CL0      CL0
##   Legals LSD Meth Mushrooms Nicotine Semer VSA
## 1    CL0 CL2  CL3       CL0      CL4   CL0 CL0
## 2    CL0 CL0  CL0       CL1      CL0   CL0 CL0
## 3    CL0 CL0  CL0       CL0      CL2   CL0 CL0
## 4    CL1 CL0  CL0       CL2      CL2   CL0 CL0
## 5    CL0 CL0  CL0       CL0      CL6   CL0 CL0
## 6    CL0 CL0  CL0       CL0      CL6   CL0 CL0
```

The first thing we see are the different types of attributes. As I mentioned earlier there are three major types of attributes: The first one is *Personal information* and the attributes that represent that are Age, Gender, Education, Ethnicity.
Then we have *Personal traits*, which are based on NEO-FFI-R model (neuroticism, extraversion, openness to experience, agreeableness, and conscientiousness). In that order, the attributes that represent these are Nscore, Escore, Oscore, Ascore, Cscore. Aside from this we also have BIS-11 (impulsivity), and ImpSS (sensation seeking), which correspond to "Impulsiveness" and "Sensation" in the data set.
The last group are the drug consumptions themselves, with the label of each attribute describing the drug that is being evaluated. Some of the labels are shortened for improving work efficiency while developing this report, such as:

- "Amyl" refers to *Amyl nitrite consumption*
- "Legals" refers to *legal highs* - over-the-counter drugs
- "Meth" refers to *Methadone*
- "VSA" refers to *Volatile substances*

When we look at the first two groups, we see that the data is normalized and it's not clear what each value represents. However, all of that information can be found on the data set's web page (linked above). Here is an example:

- 3. Gender (Real) is gender of participant:
- Value Meaning Cases Fraction
- 0.48246 Female 942 49.97%
- -0.48246 Male 943 50.03%

We can see that in the *Gender* attribute the value *0.48246* represents a female participant and *0.48246* represent a male.

When we consider the third group - the drug consumption - we see that the data is in form of factors:

- 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

With "CL0" being the lowest possible usage and "CL6" - the highest.

1.4 Pre-processing

Now that we know how the data in this set looks like, we can look at the label distribution of the Cannabis attribute.

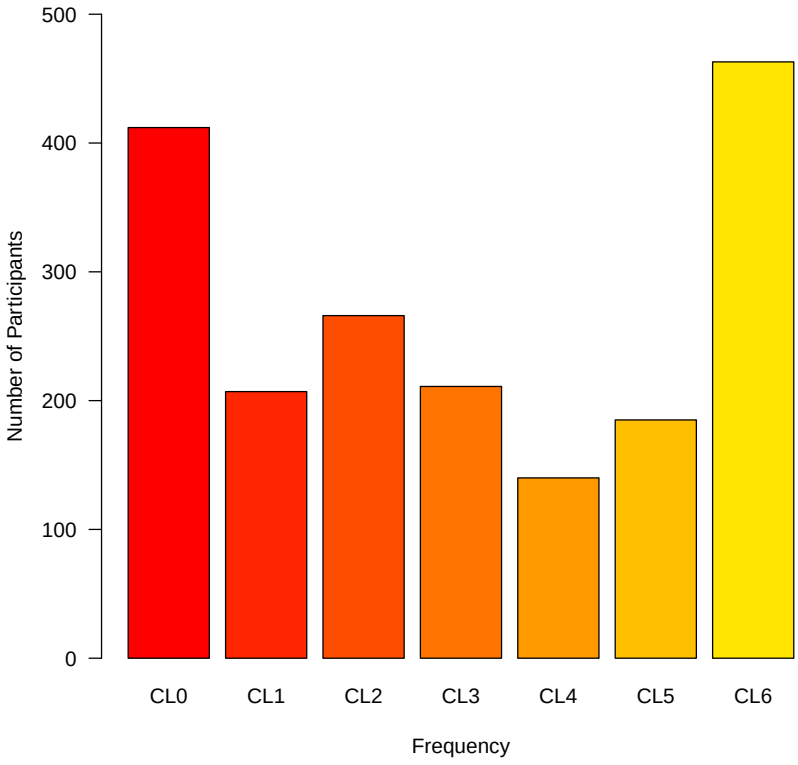

Figure 1: Cannabis use

Figure1 gives us a rough idea how often the participants consume Cannabis. As we can see, the distribution of this attribute is more or less even, with CL0 and CL6 slightly above the rest, but quite even, which means the data is not skewed, which means that it is a favourable candidate for prediction. However, the suggested problem is to do a binary classification, with the two suggested cases being "Non-User" (CL0 and CL1) and "User" (CL2 - CL6). That means we have to reassign the 7 different levels into two new ones:

```
levels(df$Cannabis) = list("NonU" = c("CL0", "CL1"),
                           "User" = c("CL2","CL3","CL4", "CL5","CL6"))
```

And once we do that, the diagram looks like this:

```
barplot(table(df$Cannabis), xlab = "Frequency", ylab = "Number of Participants", las = 1, ylim=c(0,1400), col = ra
```

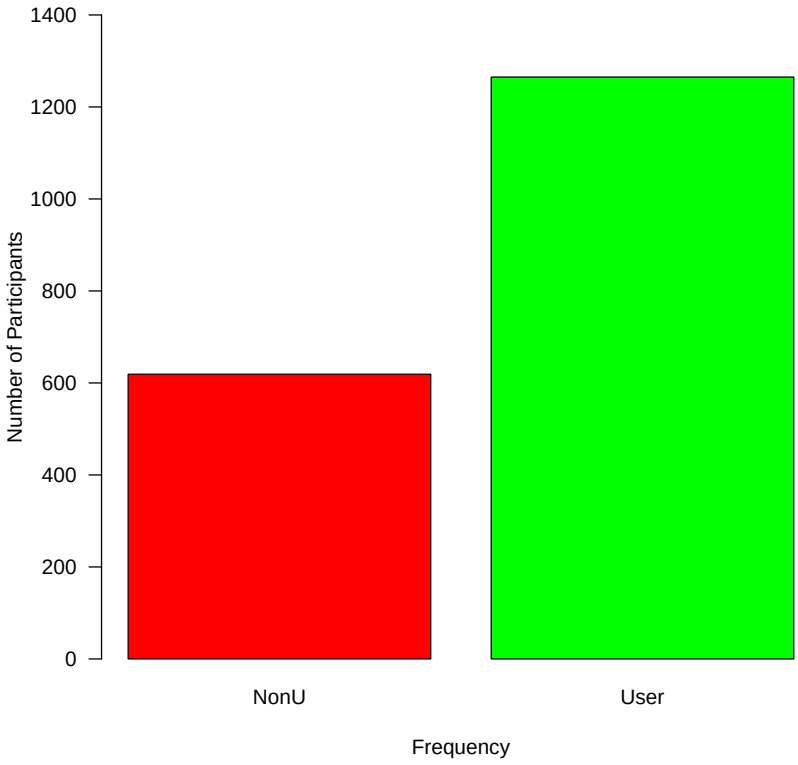

Figure 2: Cannabis use - after level reassignment

Taking *Figure2* into consideration, the two levels of the predictor are reasonably well-balanced with the non-users being about twice as less as the users. This will hopefully contribute in producing good results in our model.

The next step is to see if there are any null values in our data:

```
naCounts = sapply(df,function(x) sum(is.na(x)))
naCounts

##            ID           Age        Gender     Education       Country
##             0             0             0             0             0
##     Ethnicity        Nscore        Escore        Oscore        Ascore
##             0             0             0             0             0
##        Cscore Impulsiveness     Sensation       Alcohol        Amphet
```

```
##             0             0             0             0             0
##          Amyl        Benzos      Caffeine      Cannabis     Chocolate
##             0             0             0             0             0
##       Cocaine         Crack       Ecstasy        Heroin      Ketamine
##             0             0             0             0             0
##        Legals           LSD          Meth     Mushrooms      Nicotine
##             0             0             0             0             0
##         Semer           VSA
##             0             0
```

There aren't any so N/A values are not an issue for this data set.

Since we are interested primarily in Cannabis, it would be ideal to drop some features that do not correlate to our predictor. The *Semeron* feature is a fictitious drug that was added to the survey in order to identify "over-claimers" - people that would claim they used all of the drugs on the list for one reason or another.

```
summary(df$Semer)

##  CL0  CL1  CL2  CL3  CL4
## 1876    2    3    2    1

#We can check which these rows are
semerRows = which(!df$Semer == 'CL0')
cat("Rows where Semeron is not equals to CL0: ", semerRows)

## Rows where Semeron is not equals to CL0:  727 817 1516 1533 1698 1769 1806 1823
```

As we can see it's only 8 participants that have answered with anything else other than CLO - "Never used". This is a negligible number and we can easily exclude the from the data set, because we don't know if the data in these rows is correct. If a person submitted a false answer to this question, they might be "over-claimers".

```
df = df[-semerRows,]
df$Semer = NULL
```

We also do not need the participant's ID for our machine-learning task.

```
df$ID = NULL
```

Now we should set the rest of the categorical attributes as "factors"

```
#Making sure R knows our attributes are factors (R sees them as chars otherwise)
df$Cannabis = as.factor(df$Cannabis)
df$Chocolate = as.factor(df$Chocolate)
df$Caffeine = as.factor(df$Caffeine)
df$Nicotine = as.factor(df$Nicotine)
df$Alcohol = as.factor(df$Alcohol)
df$Amphet = as.factor(df$Amphet)
df$Amyl = as.factor(df$Amyl)
df$Benzos = as.factor(df$Benzos)
df$Cocaine = as.factor(df$Cocaine)
```

```
df$Crack = as.factor(df$Crack)
df$Ecstasy = as.factor(df$Ecstasy)
df$Heroin = as.factor(df$Heroin)
df$Ketamine = as.factor(df$Ketamine)
df$Legals = as.factor(df$Legals)
df$LSD = as.factor(df$LSD)
df$Meth = as.factor(df$Meth)
df$Mushrooms = as.factor(df$Mushrooms)
df$Nicotine = as.factor(df$Nicotine)
df$VSA = as.factor(df$VSA)
```

Now that we're done with the pre-processing, it's a good idea to back up our dataframe, in case we make some changes to it that need to be reverted in the future.

```
#backing up df
df_backup = df
```

2 Modelling/Classification

We start by installing randomForest in our Environment. Then we load it.

```
#We have to install package the first time we use it
#install.packages('e1071', dependencies=TRUE)
require(randomForest)

## Loading required package: randomForest
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

2.1 Dividing the data

```
#get a seed for the sampling
set.seed(3)
#sample the data
trainingRowIndex = sample(1:nrow(df), 0.7*nrow(df))
#create train and test sets
train = df[trainingRowIndex,]
test = df[-trainingRowIndex,]
```

I use random sampling which ensures we get random and robust results in our *train* and *test* datasets. This way we make sure the results are not biased by the order of the data in the data set. We can see if the entries have distributet evenly among the two data sets:

```
table(train$Cannabis)

##
## NonU User
##  432  881
```

```
table(test$Cannabis)

##
## NonU User
##  187  376
```

As we can see there are roughly twice as many Cannabis users as non-users in both sets, so the distribution is even.

2.2 Build a model using training set

My work is based on a classification problem, so I decided to use the model that deals with classification tasks the best - the *randomForest*. However, we need to ensure we use the right amount of trees in our forest by measuring the accuracy of the model when the forest has a different amount of trees.

```
#Variable that holds the number of trees
ntrees = 100
#Dataframe that stores accuracies at the corresponding number of trees
results = data.frame(NTrees=as.numeric(), Accuracy=as.numeric())

#For-loop
for (i in 1:10){
  fit = randomForest(train[,-18], train[,18],
    xtest=test[,-18], ytest=test[,18],
    ntree=ntrees, proximity=TRUE)

  #Test the RF model for this run
  preds = levels(train[,18])[fit$test$predicted]

  #Calculate accuracy
  auc = (sum(preds ==test[,18])/nrow(test))*100
  results = rbind(results, data.frame(NTrees=ntrees,Accuracy=auc))

  #Increment trees
  ntrees = ntrees + 100
}# end for loop
```

And here's the graph:

```r
#Plot the results
#We need to load ggplot library the first time using it
library(ggplot2)
ggplot(results, aes(x=NTrees)) +
geom_line(aes(y = Accuracy)) +
scale_x_continuous(name = "NTrees", breaks = seq(100,1000,by=100)) +
scale_y_continuous(name = "PredictionAccuracy")
```

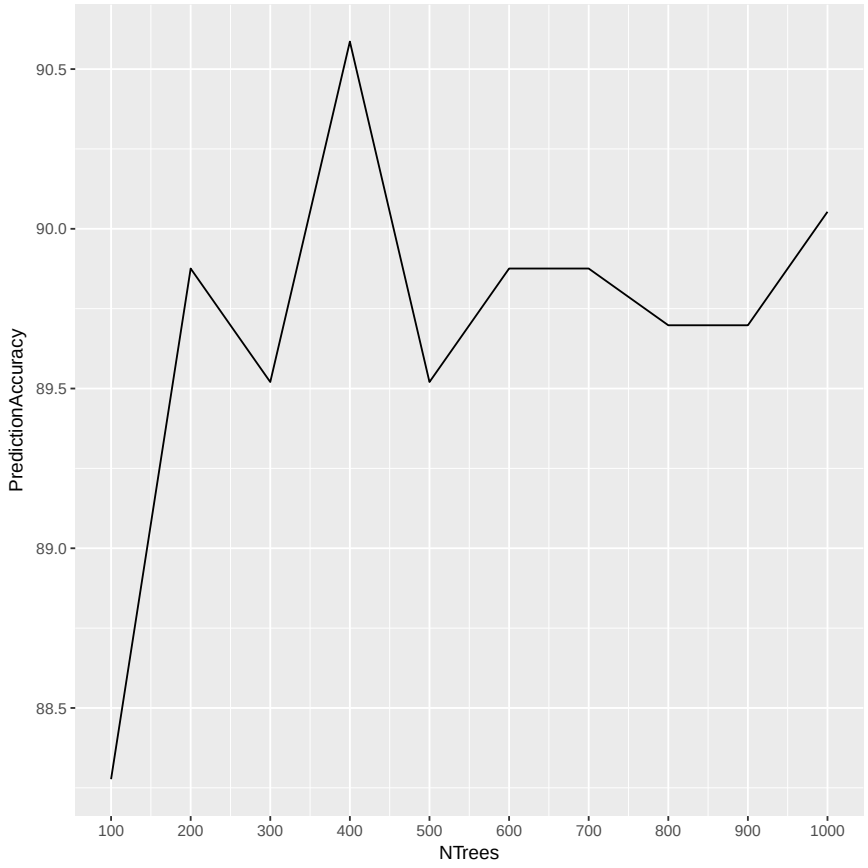

Figure 3: Difference in accuracy with ntree 100-1000

As we can see in *Figure3* the best results are produced when *ntree = 400*, so that is what I'm going to use in my model.

```r
fit = randomForest(as.factor(Cannabis) ~., data=train, ntree=400)
#print out the results
fit

##
## Call:
##  randomForest(formula = as.factor(Cannabis) ~ ., data = train,      ntree = 400)
##                Type of random forest: classification
##                      Number of trees: 400
## No. of variables tried at each split: 5
##
```

```
##         OOB estimate of  error rate: 11.81%
## Confusion matrix:
##      NonU User class.error
## NonU  345   87  0.20138889
## User   68  813  0.07718502
```

We build our model and run the "fit" command which gives us a useful summary of the forest. It has used 400 trees as we requested and 5 variables at each split of the tree. The out-of-bag error rate is about 12%, which is quite good for our first try. The command also gives us the confusion matrix - we will focus on that later.

2.3 Test and evaluate your model

Here we test our model with the test set.

```
#Making the prediction and storing it into a vector
probabilities = predict(fit, newdata = test[,-18], type="prob")[,2]
#calculating accuracy
test_accuracy = mean(probabilities==test$Cannabis)
#Printing out accuracy
cat("Test accuracy: ", test_accuracy)

## Test accuracy:  0
```

The test accuracy we receive after we complete the predictions is almost 90%, which is a satisfying result for out first model. We can get a better idea about our results with the Area Under Curve (AUC) number.

```
#Installing ROCR package if it's the first time it's used
#install.packages("ROCR")
require(ROCR)

## Loading required package: ROCR
## Loading required package: gplots
##
## Attaching package: 'gplots'
## The following object is masked from 'package:stats':
##
##     lowess

#Calculating accuracy
predictions = prediction(probabilities, test$Cannabis)

#Calculating Area under curve
auc = performance(predictions, measure="auc")
auc = auc@y.values[[1]]
cat("Area Under Curve: ", auc)

## Area Under Curve:  0.9530877
```

We get a result of about 0.95, which is a only a few False Positive results among our predictions, however lets visialize this.

```
performance = performance(predictions, measure = "tpr", x.measure = "fpr")
plot(performance)
```

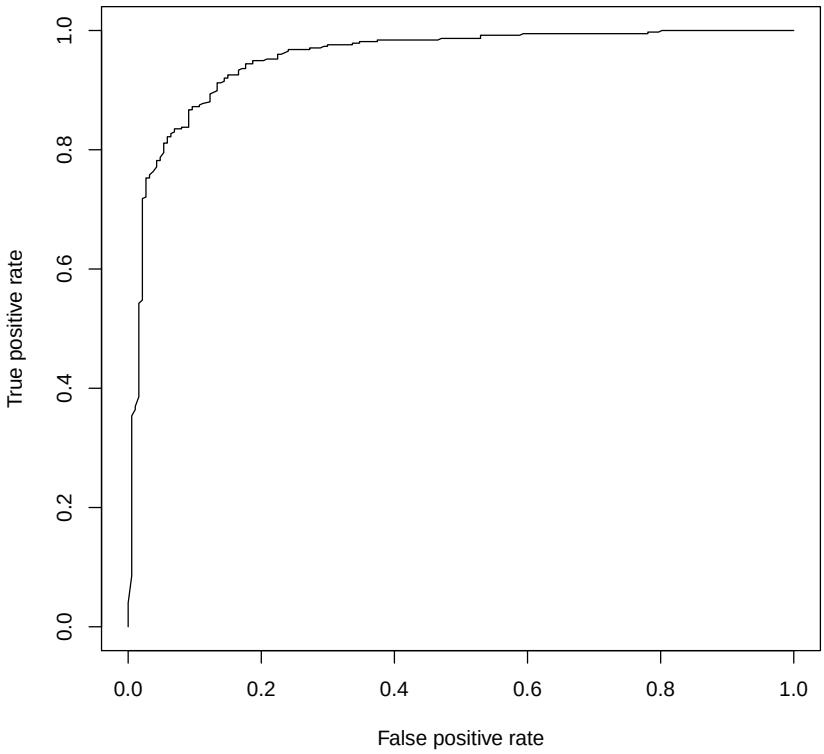

Figure 4: Area Under Curve Plot: True Pos. vs False Pos.

As we see on *Figure4* the amount of False Positives is quite small. Although AUC is not the most important calcultion of the accuracy, it is important when we are dealing with healthcare - a false prediction might lead to a wrong treatment, etc. And in this case - if this was model was to be implemented as a method of predicting potential drug addicts, we wouldn't want to diagnose a healthy person with a potential addiction.

2.4 Report and discuss results

The first model achieved almost 90% accuracy on it's first run. This is a good result which might be caused by the steps we took to ensure a good fit: we are using randomForest, which deals quite well with classification problems, we calculated the optimal number of trees, we removed variables that have no significance to the accuracy - "Semeron" and "ID".

3 Improving Performance

In this section I am attempting to improve the accuracy of the model by implementing different techniques.

3.1 Exploring different mtry valules

randomForest takes an argument "mtry" - that represents the number of variables taken into consideration on each split of the tree. Changing this number might yield better results. To do this, we test the model's accuracy with the different mtry values it can take - they vary from the square root of the number of attributes (in this case, 5) to the number of attributes minus one (in our case, 29).
For each model, we save the out-of-bag error and the test error *(Test error = 1 - Accuracy)* into vectors and plot them on a graph to see which value of mtry produces the best result.

```
#Creating vectors where error rates will be stored
oobError = double(25)
testError = double(25)
medianError = double(25)
#mtry = 0

#For each mtry fit a new model and measure test and oob error rates
for(mtry in round(sqrt(ncol(test))):(ncol(test))-1)
{
  fit = randomForest(as.factor(Cannabis)~., data=train, mtry=mtry, ntree=400)
  oobError[mtry-5] = fit$err.rate[,1][400]
  probs = predict(fit, newdata=test[-18])
  testError[mtry-5] = with(test, mean(probs!=test$Cannabis))
  medianError[mtry-5] =  (oobError[mtry-5]+testError[mtry-5])/2
}
```

Once we loop through all the different possibilities, we create a matrix of our results and plot it using the library ggplot2.

```
mtryNum = c(5:29)
#creating data frame
testData = data.frame(as.vector(testError), as.vector(oobError),
                      as.vector(medianError), as.vector(mtryNum))

ggplot(testData, aes(testData[,4])) +
  geom_line(aes(y = testData[,1], colour = "testError")) +
  geom_line(aes(y = testData[,3], colour = "meanError") ) +
  geom_line(aes(y = testData[,2], colour = "oobError") ) +
  scale_x_continuous(name = "mtry", breaks = seq(5,29,by=2)) +
  scale_y_continuous(name = "Error %")
```

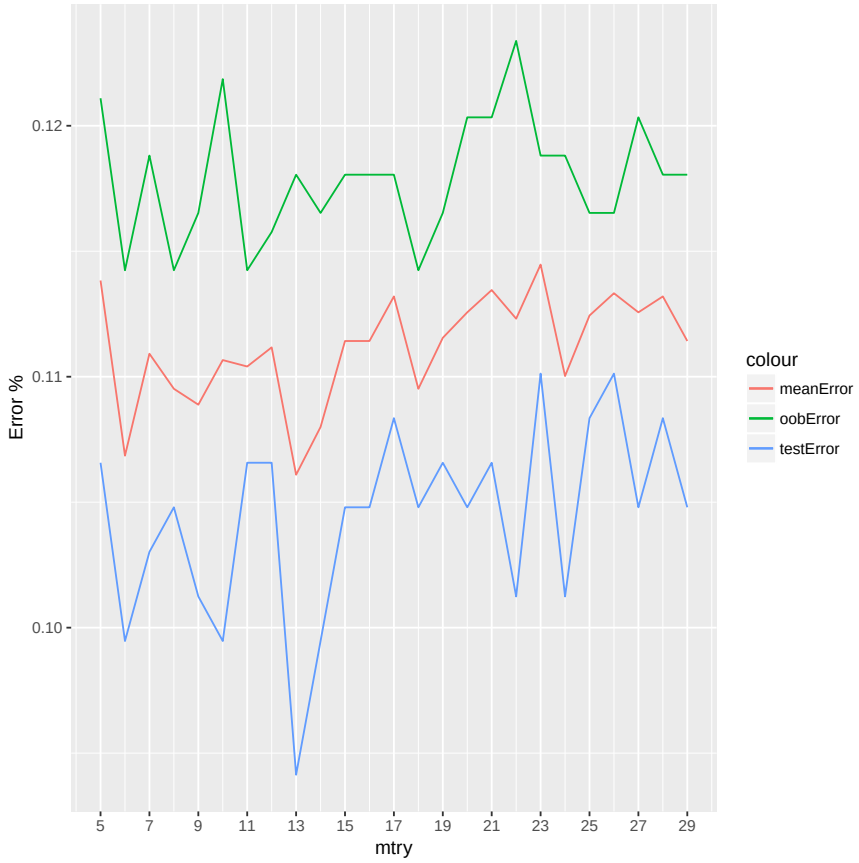

Figure 5: Plot of the test and oob error rates per mtry

The results that *Figure5* produces slightly differ on each every model, but I have noticed that when $mtry = 6$ and $mtry = 12$ produce good results. That is why I decided to test out how it compares against the default value of mtry that randomForest picks (mtry = 5).

In order to achieve consistency, we will run each test ten times and take the average results of the ten runs. We do this by creating a data frame where we would store the oob error and the test error for each of the values of mtry that we are testing.

```
#data frame for storing error rates
mtryTests = data.frame(type=as.factor(c(1,2,1,2,1,2)),
```

```
                      errorRate=c(0.0,0.0,0.0,0.0,0.0,0.0),
                      mtry=as.factor(c(5,5,6,6,12,12)))
#variables for storing average error rates for each mtry
oobError = 0.0
testError = 0.0
set.seed(3)
#mtry = 5
#Redistributing data sets
trainingRowIndex = sample(1:nrow(df), 0.8*nrow(df))
train = df[trainingRowIndex,]
test = df[-trainingRowIndex,]
#Fitting
for(i in 1:10){
  fit = randomForest(as.factor(Cannabis)~., data=train, mtry=5, ntree=400)
  oobError = oobError + fit$err.rate[,1][400]
  probs = predict(fit, newdata=test[-18])
  testError = testError + with(test, mean(probs!=test$Cannabis))
}

#When we're done we average out the error rates from the 10 tests and we save them
mtryTests$errorRate[1] = oobError/10
mtryTests$errorRate[2] = testError/10
#And then we clear the variables
oobError = 0.0
testError = 0.0

#mtry = 6
set.seed(3)
trainingRowIndex = sample(1:nrow(df), 0.8*nrow(df))
train = df[trainingRowIndex,]
test = df[-trainingRowIndex,]

for(i in 1:10){
  fit = randomForest(as.factor(Cannabis)~., data=train, mtry=6, ntree=400)
  oobError = oobError + fit$err.rate[,1][400]
  probs = predict(fit, newdata=test[-18])
  testError = testError + with(test, mean(probs!=test$Cannabis))
}

mtryTests$errorRate[3] = oobError/10
mtryTests$errorRate[4] = testError/10
oobError = 0.0
testError = 0.0

#mtry = 12
set.seed(3)
trainingRowIndex = sample(1:nrow(df), 0.8*nrow(df))
train = df[trainingRowIndex,]
test = df[-trainingRowIndex,]

for(i in 1:10){
  fit = randomForest(as.factor(Cannabis)~., data=train, mtry=12, ntree=400)
  oobError = oobError + fit$err.rate[,1][400]
```

```
  probs = predict(fit, newdata=test[-18])
  testError = testError + with(test, mean(probs!=test$Cannabis))
}

mtryTests$errorRate[5] = oobError/10
mtryTests$errorRate[6] = testError/10
```

Once done, we plot the results on a graph:

```
#Needed fot rescaling
library(scales)
ggplot(data=mtryTests, aes(x=mtry, y=errorRate, fill=factor(type))) +
  geom_bar(stat="identity", position="dodge") +
  scale_fill_discrete(name="Error rates",
                      breaks=c(1, 2),
                      labels=c("OOB Error", "Test Error")) +
  xlab("mtry")+ylab("Error rate") +
  scale_y_continuous(limits=c(0.075,0.125),oob = rescale_none) +
  theme_minimal()
```

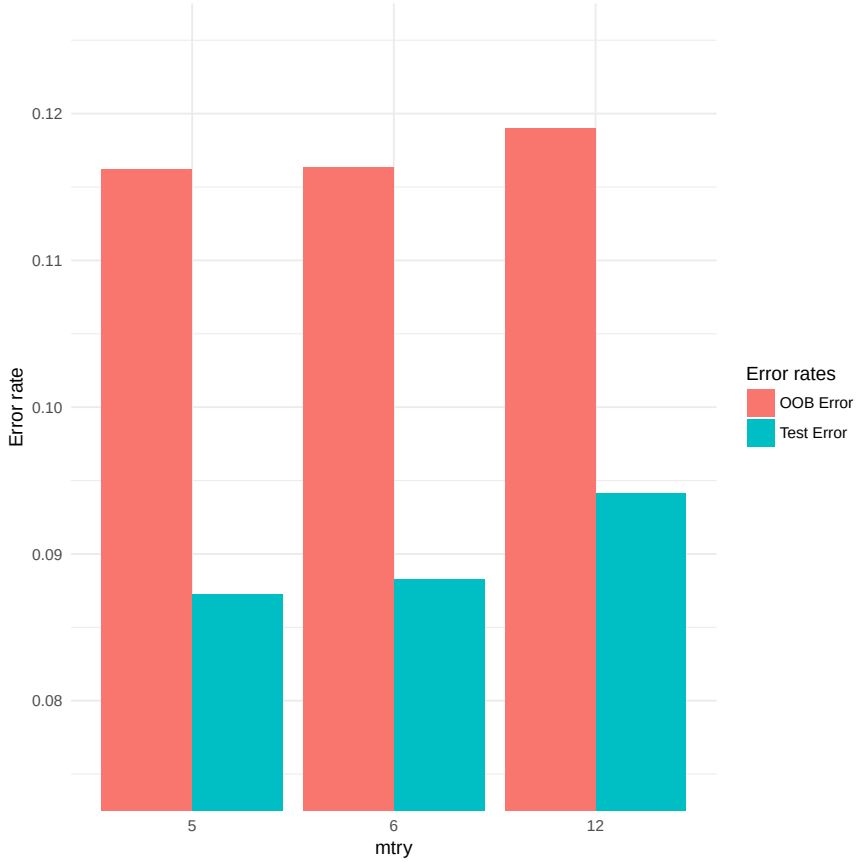

Figure 6: Accuracy differency between mtry = 5 and mtry = 6

And as we see on *Figure6*, the results don't vary by much, but *mtry = 5* and *mtry = 6* definitely provde better accuracy than *mtry = 12*. *mtry = 6* sometimes provides smaller oob error rate, but

alwas has higher test error rate than $mtry = 5$. This is why we will continue to use $mtry = 5$.

3.2 Changing the training/testing set ratios

The next variable we will explore is the data set ratios of the training and testing sets. Depending on the data we have, a smaller or bigger training set can improve the accuracy of the model. We do this by changing the training and testing set ratios when we sample the data from the df data set and then measuring the oob and test error rates. We will look at three possibilities: a 75/25 distribution, a 80/20 and a 90/10.

```
#Vector for storing accuracy
divisionTests = double(3)
set.seed(3)
#deviding 0.75 train/0.25 test
trainingRowIndex = sample(1:nrow(df), 0.75*nrow(df))
train = df[trainingRowIndex,]
test = df[-trainingRowIndex,]
#Fitting
fit = randomForest(as.factor(Cannabis)~., data=train, mtry=6, ntree=400)
probabilities = predict(fit, test[,-18])
#calculating accuracy and storing result
divisionTests[1] = mean(probabilities==test$Cannabis)

#deviding 0.8 train/0.2 test
set.seed(3)
trainingRowIndex = sample(1:nrow(df), 0.8*nrow(df))
train = df[trainingRowIndex,]
test = df[-trainingRowIndex,]
fit = randomForest(as.factor(Cannabis)~., data=train, mtry=6, ntree=400)
probabilities = predict(fit, test[,-18])
divisionTests[2] = mean(probabilities==test$Cannabis)

#deviding 0.9 train/0.1 test
set.seed(3)
trainingRowIndex = sample(1:nrow(df), 0.9*nrow(df))
train = df[trainingRowIndex,]
test = df[-trainingRowIndex,]
fit = randomForest(as.factor(Cannabis)~., data=train, mtry=6, ntree=400)
probabilities = predict(fit, test[,-18])
divisionTests[3] = mean(probabilities==test$Cannabis)
```

We plot the results:

```
mtryNum = c(5:29)
#creating data frame
division = data.frame(Results=rep(divisionTests), Ratios=c("0.75","0.8","0.9"))
ggplot(data=division, aes(x=Ratios, y=Results)) +
  geom_bar(stat="identity", fill="steelblue") +
  scale_y_continuous(limits=c(0.84,0.94),oob = rescale_none) +
  theme_minimal()
```

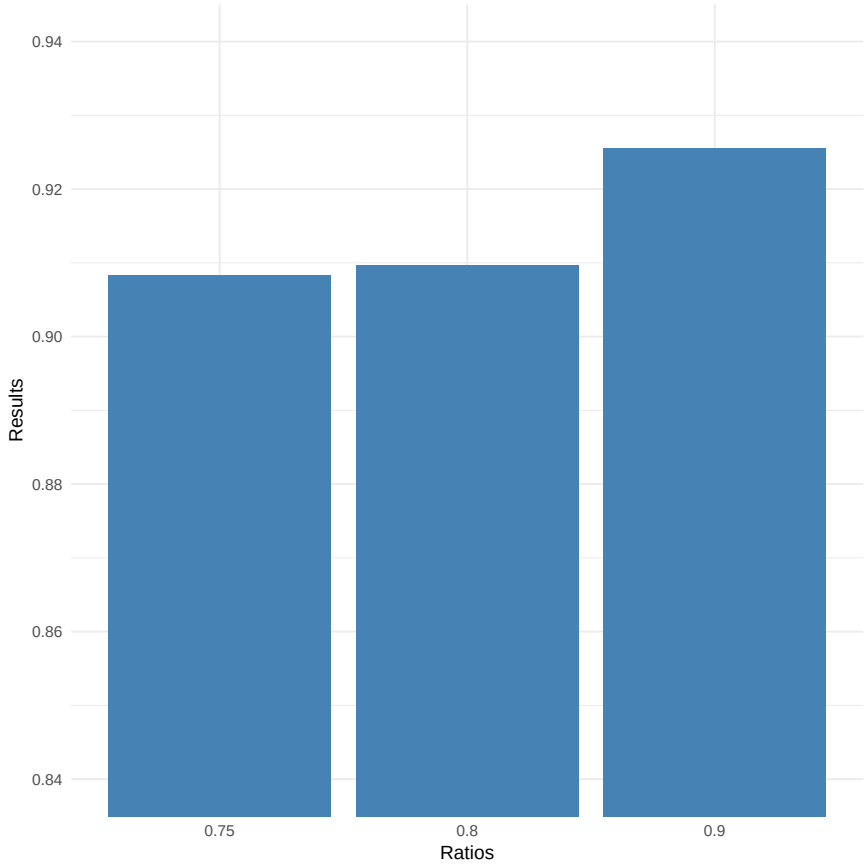

Figure 7: Accuracy per different test/train set ratios

As we can see on *Figure7* the 80/20 distribution provides the best results and that is what we will be using to improve our model.

3.3 Feature Selection

We can improve our model by picking the best features for our model. Since I have build a model with a good accuracy by using all the features, now we we attempt to get better results by removing some of the least important features. We can see a ranking of important features by creating a fit and calling the function *varImpPlot*.

```
#Create get a clean model
set.seed(3)
trainingRowIndex = sample(1:nrow(df), 0.8*nrow(df))
train = df[trainingRowIndex,]
```

```
test = df[-trainingRowIndex,]
fit = randomForest(as.factor(Cannabis)~., data=train, importance = TRUE, mtry=6, ntree=400)
```

The function will produce the following graph:

```
varImpPlot(fit)
```

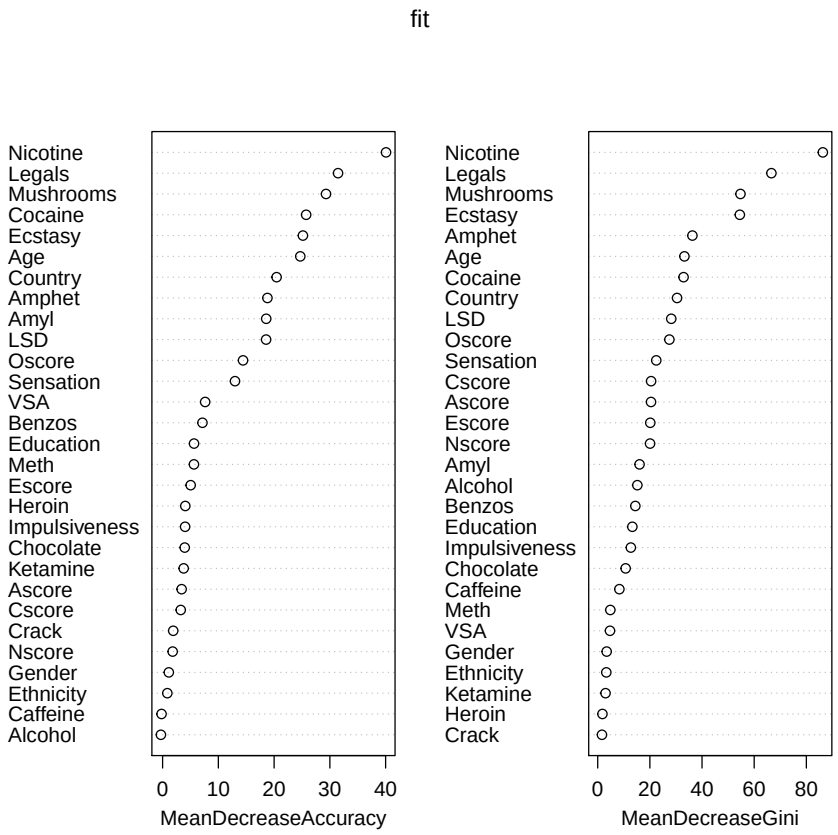

Figure 8: Variable Importance of the model

As we can see on *Figure8* there are two different types of variable importance that randomForest takes into account. The left represents how much the accuracy of the model drops when each of the attributes is not used and the right one - the gini index.
I notice that on the left graph Caffeine is the only variable that has a significantly lower importance, but on the right both Crack and Heroin have relatevly low values. That is why we will remove each of those attributes from the model and see if we will receive an increase in performance.
I do this in a similar fashion as the previous technique - by running the model 10 times and measuring the error rates of each run. Then we store them in a data frame and plot them on a graph.

```
#Creating data frame
tests = data.frame(type=as.factor(c(1,2,1,2,1,2,1,2)),
                        errorRate=c(0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0),
                attributes=as.factor(c("Base","Base","Caffeine","Caffeine","Crack","Crack","Heroin","Her

#Our clean model - the benchmark
```

```
oobError = 0.0
testError = 0.0
#Distributing the train and test sets
set.seed(3)
trainingRowIndex = sample(1:nrow(df), 0.8*nrow(df))
train = df[trainingRowIndex,]
test = df[-trainingRowIndex,]

#Running the test 10 times
for(i in 1:10){
  fit = randomForest(as.factor(Cannabis)~., data=train, mtry=5, ntree=400)
  oobError = oobError + fit$err.rate[,1][400]
  probs = predict(fit, newdata=test[-18])
  testError = testError + with(test, mean(probs!=test$Cannabis))
}

#saving the average error rates in the matrix
tests$errorRate[1] = oobError/10
tests$errorRate[2] = testError/10
oobError = 0
testError = 0

#Model without Caffeine
df$Caffeine = NULL
set.seed(3)
trainingRowIndex = sample(1:nrow(df), 0.8*nrow(df))
train = df[trainingRowIndex,]
test = df[-trainingRowIndex,]
for(i in 1:10){
  fit = randomForest(as.factor(Cannabis)~., data=train, mtry=6, ntree=400)
  oobError = oobError + fit$err.rate[,1][400]
  probs = predict(fit, newdata=test[-17])
  testError = testError + with(test, mean(probs!=test$Cannabis))
}

tests$errorRate[3] = oobError/10
tests$errorRate[4] = testError/10
oobError = 0
testError = 0

#Without Crack
#Loading clean data
df = df_backup
#Dropping Crack
df$Crack = NULL
set.seed(3)
trainingRowIndex = sample(1:nrow(df), 0.8*nrow(df))
train = df[trainingRowIndex,]
test = df[-trainingRowIndex,]
for(i in 1:10){
  fit = randomForest(as.factor(Cannabis)~., data=train, mtry=12, ntree=400)
  oobError = oobError + fit$err.rate[,1][400]
  probs = predict(fit, newdata=test[-18])
```

```
  testError = testError + with(test, mean(probs!=test$Cannabis))
}

tests$errorRate[5] = oobError/10
tests$errorRate[6] = testError/10

#Without Heroin
#Loading clean data
df = df_backup
#Dropping Heroin
df$Heroin = NULL
set.seed(3)
trainingRowIndex = sample(1:nrow(df), 0.8*nrow(df))
train = df[trainingRowIndex,]
test = df[-trainingRowIndex,]
for(i in 1:10){
  fit = randomForest(as.factor(Cannabis)~., data=train, mtry=12, ntree=400)
  oobError = oobError + fit$err.rate[,1][400]
  probs = predict(fit, newdata=test[-18])
  testError = testError + with(test, mean(probs!=test$Cannabis))
}

tests$errorRate[7] = oobError/10
tests$errorRate[8] = testError/10
```

Now we plot the results:

```
ggplot(data=tests, aes(x=attributes, y=errorRate, fill=factor(type))) +
  geom_bar(stat="identity", position="dodge") +
  scale_fill_discrete(name="Error rates",
                      breaks=c(1, 2),
                      labels=c("OOB Error", "Test Error")) +
  xlab("Attributes")+ylab("Error rate") +
  scale_y_continuous(limits=c(0.075,0.2),oob = rescale_none) +
  theme_minimal()
```

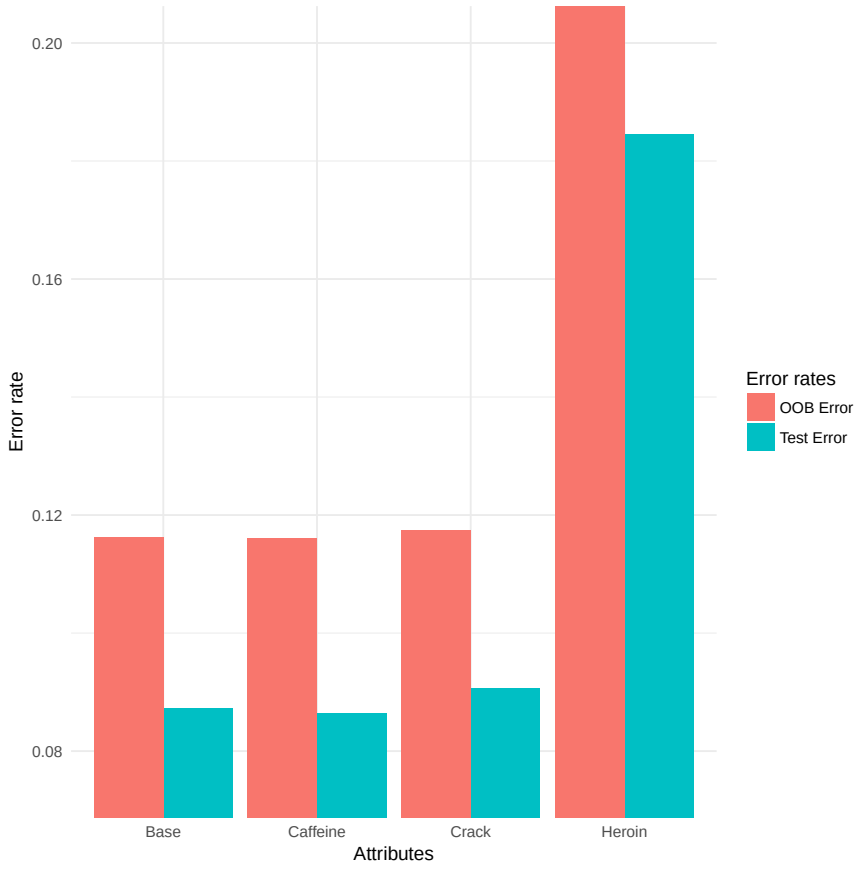

Figure 9: Accuracy differency when Caffeine, Crack and Heroin are removed

Looking at *Figure8*, we see straight away that taking Heroin out of the model produces the worst results. The other three models have similar error rates, but removing Caffeine could be slightly benefitial to the model.