'Choose Your Own' Project

Bank Note Authentication

Rocío Alvarez Mercé
May 07, 2019

Quick Note

Hello! This is my final "Create your Own Project" report. As I said on my MovieLens project (I'm guessing I have different peers assessing this report!), it took a little longer than expected because of work issues, so I apologize for not submitting it before.

It was written with the idea of presenting it to other people in mind, and let them be able to follow my steps, one by one. As if we were in a class or a little company meeting. That is why I use the pronoun "we" instead of "I" in every part of the report, I guess I'm used to it and I think it gives the readers a better understanding of the analysis and the steps to follow. With this quick note I wanted to emphasize that even if I use "we" in every section, I'm the only author of this report.

That being said, hope you enjoy it!

1. Introduction and Goals

For this final project, we chose the Bank Note Authentication dataset, from the UCI Machine Learning Repository, by Volker Lohweg and Helene Darksen from the University of Applied Sciences, Ostwestfalen-Lippe, Germany. This dataset was published in 2012, and helps distinguishing authentic from fake bank notes using different measured parameters.

We personally chose this dataset because we needed a fairly little dataset since our computer capacity is limited and because we're currently working on a neuroscience biomarkers project that has the exact same characteristics: 4 parameters that would help us classify our measurements into a 2 level factor (in this case **authentic** or **fake**). In other words, a binary classification problem. We wanted to use our own data but we don't have a fair amount of measurements yet, so this will be a good way to prepare ourselves for that set of data when the time comes.

The goal of this project is firstly, to understand and analyze the dataset; secondly, create different machine learning models and finally compare them, and then pick the one with the best metrics.

2. Bank Note Authentication dataset

In the Bank Note Authentication set, data was extracted from images that were taken from genuine and counterfeit banknote images. For digitization, an industrial camera usually used for print inspection was used. The final images obtained were 400 x 400 pixels gray-scale pictures, with a resolution of about 660 dpi. Image processing techniques in these cases are not only convenient in extracting useful information for analysis purposes but also saves computation time and memory space. For the Bank Note dataset, the features of the banknotes are extracted using Fast Wavelet Transforms (WT), and then analyzing the WTI (WT Images). Four different attributes were obtained:

- **Variance** of WTI: finds how each pixel varies from the neighboring pixels, and can be used in identifying sharp details such as edges.
- **Skewness** of WTI: it's a measure of symmetry, or more precisely, the lack of symmetry (if the image looks the same or not to the left and to the right of the center -reference- point).
- **Curtosis** of WTI: it's a measure of whether the data is heavy-tailed or light-tailed, relative to a normal distribution.
- **Entropy** of image (also called average information of an image): this is a quantity that measures the degree of randomness in the image.

Finally, you can find a last feature on this dataset, that classifies the banknotes in one of two categories: authentic or fake. As we stated before on the previous section of this report, one of the goals of this project is to find a suitable machine learning method to predict this classification given the four parameters described previously, for unseen data (in this case, a test set).

Packages Installation and Libraries

Listed in the code below are the packages we are going to use through this project. Run this code if you don't have them installed or if you are unsure.

```
# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr",
                                     repos = "http://cran.us.r-project.org")
if(!require(AppliedPredictiveModeling)) install.packages("AppliedPredictiveModeling",
                                     repos="http://R-Forge.R-project.org")
if(!require(rpart)) install.packages("rpart",
                                         repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest",
                                         repos = "http://cran.us.r-project.org")
```

```
library(tidyverse)
library(caret)
library(dplyr)
library(AppliedPredictiveModeling)
library(rpart)
library(randomForest)
```

3. Getting the data

The dataset can be obtained by three different ways, use whichever you prefer or it's more comfortable depending if you'd rather download the dataset or directly running a piece of code.

3.1. UCI Repository

The dataset can be downloaded directly from the website of the UCI Machine Learning Repository, using this link:

https://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_banknote_authentication.txt

Or by clicking on the "data_banknote_authentication.txt" link from the following URL:

https://archive.ics.uci.edu/ml/machine-learning-databases/00267/

NOTE 1: If you choose to download the dataset from the UCI Repository, it's **very important** that you save the dataset file in .csv format for the code to run properly.

3.2. Git Repo

Another option to get the dataset is to directly download it from my GitHub Repository, following this link:

https://github.com/rocioam/Capstone-CYO

NOTE 2: If you have chosen either the UCI or the Git Repo download options detailed above, you should run **the following** code to have your dataset ready to use (after moving the .csv file to the same location of the .R or .rmd file, whichever you are running):

```
banknote_auth <- read.csv('data_banknote_authentication.csv', header=FALSE)
colnames(banknote_auth)<- c("variance_wt","skewness_wt",
                            "curtosis_wt","entropy","authentic")
```

3.3. Running this code

If you don't want or are unable to download the dataset as indicated in options 3.1. or 3.2. detailed above, you can directly run **this piece** of code:

```
banknote_auth <- read.csv(url("https://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_banl
                          header=FALSE)

colnames(banknote_auth)<- c("variance_wt","skewness_wt","curtosis_wt","entropy","authentic")
```

This option is the easiest one in my opinion, but you need to have an active Internet connection.

4. Data Analysis

If everything went well during section 3, and you were able to successfully download your data, you should now have your **dataframe** saved in the 'banknote_auth' variable.

```
class(banknote_auth)
```

```
## [1] "data.frame"
```

If this is not the case, please go back to last section. If everything went all right, let's now analyze our data!

4.1. Knowing the data

First of all, let's take a quick look to our dataframe, what number and what kind of variables do we have?

```
dim(banknote_auth)
```

```
## [1] 1372    5
```

```
head(banknote_auth)
```

```
##   variance_wt skewness_wt curtosis_wt  entropy authentic
## 1     3.62160      8.6661     -2.8073 -0.44699         0
## 2     4.54590      8.1674     -2.4586 -1.46210         0
## 3     3.86600     -2.6383      1.9242  0.10645         0
## 4     3.45660      9.5228     -4.0112 -3.59440         0
## 5     0.32924     -4.4552      4.5718 -0.98880         0
## 6     4.36840      9.6718     -3.9606 -3.16250         0
```

It's also a good idea to look at the class of the attributes we have. Knowing the type is important because it can help you better summarize your data and prepare it for the machine learning algorithms later.

```r
sapply(banknote_auth, class)
```

```
## variance_wt skewness_wt curtosis_wt     entropy   authentic
##   "numeric"   "numeric"   "numeric"   "numeric"   "integer"
```

We can see that we have 1372 rows (observations) and 5 different column variables. As we detailed previously in the introduction, the variance, skewness, curtosis and entropy are four **numeric** parameters used for the detection of fake banknotes. The last column, **authentic** is a binary classification label (**integer** class) to indicate wheter a banknote is genuine (authentic col = 1) or fake (authentic col = 0). We are going to focus on this column since it's what we want to predict after.

4.2. Classification Levels and dataset

Let's make a summary of the statistics for the classification column:

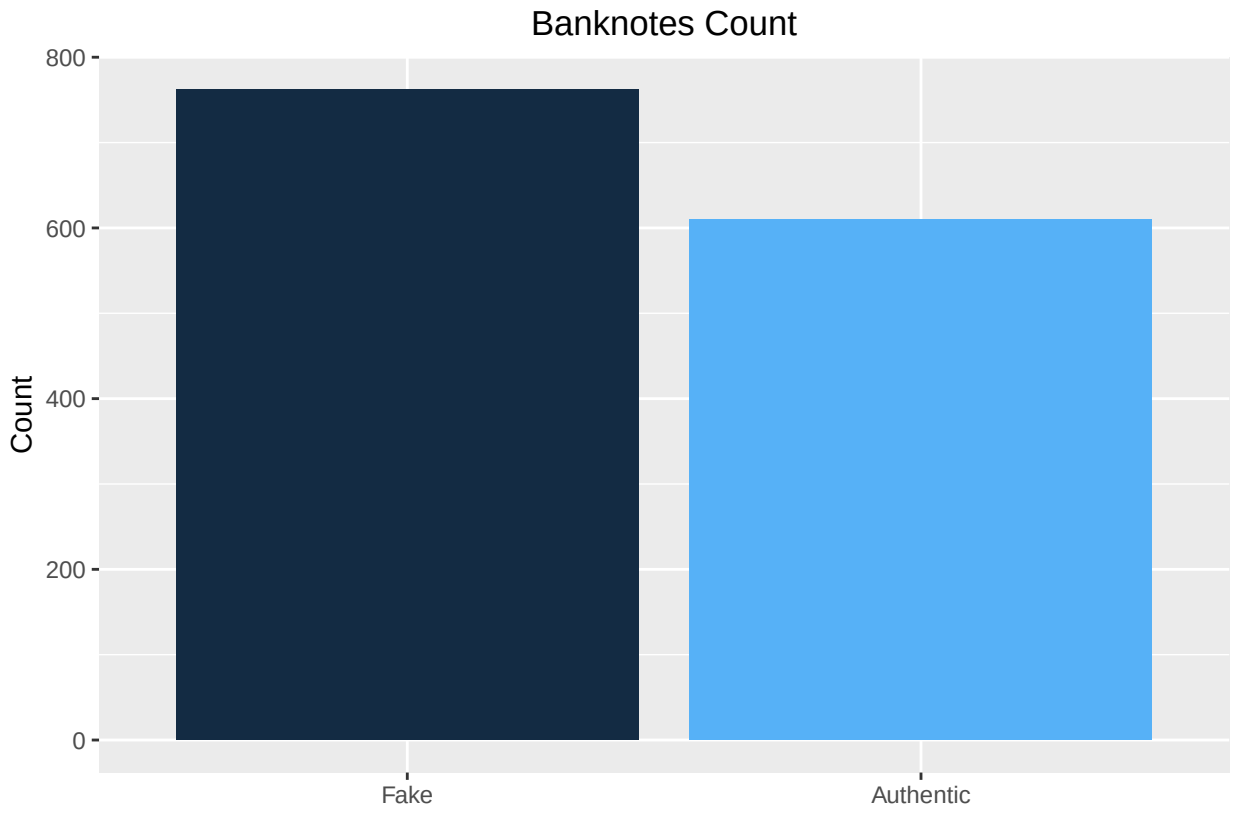
```r
summary(banknote_auth$authentic)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.4446  1.0000  1.0000
```

If we take a closer look at the **authentic** column, we can see that the mean is lower than 0.5, meaning we must have more fake observations than genuine ones.

We can plot an histogram to confirm this:

```r
library(ggplot2)
banknote_auth %>% ggplot(aes(factor(authentic),
                             fill = authentic )) + guides(fill = F) +
  geom_bar() + labs(y = "Count", x = "", title = "Banknotes Count") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_x_discrete(labels=c("Fake", "Authentic"))
```



We can see from this plot that we don't have the same number of instances for each class: we have more fake banknotes than authentic ones in our dataset, as we determined earlier by looking at the dataset summary. The difference is not very big, thus having a good number of samples for each classification.

To summarize the information we see on the previous figure, we can use the *cbind* function.

```
percentage <- prop.table(table(banknote_auth$authentic)) * 100
cbind(Count = table(banknote_auth$authentic), Percentage = percentage)
```

```
##   Count Percentage
## 0   762   55.53936
## 1   610   44.46064
```

We see that we have 762 fake banknotes and 610 authentic ones in our dataset, meaning a 55,5% and a 44.4% of the dataset respectively.

We see from this analysis that it's better if we convert this column to a **factor** class, so we have the two levels that we want for following analyses:

```
banknote_auth$authentic <- as.factor(banknote_auth$authentic)
```

Now our class value is a factor with two levels, 0 for fake banknotes and 1 for genuine ones.

```
class(banknote_auth$authentic)
```

```
## [1] "factor"
```

```r
levels(banknote_auth$authentic)
```

```
## [1] "0" "1"
```

4.3. Attribute analysis

Now that we have already taken a peek to the **authentic** column, we can look at and summarize the rest of the data we have in our dataset. First of all, by looking at all columns, and see what kind of parameters we have.

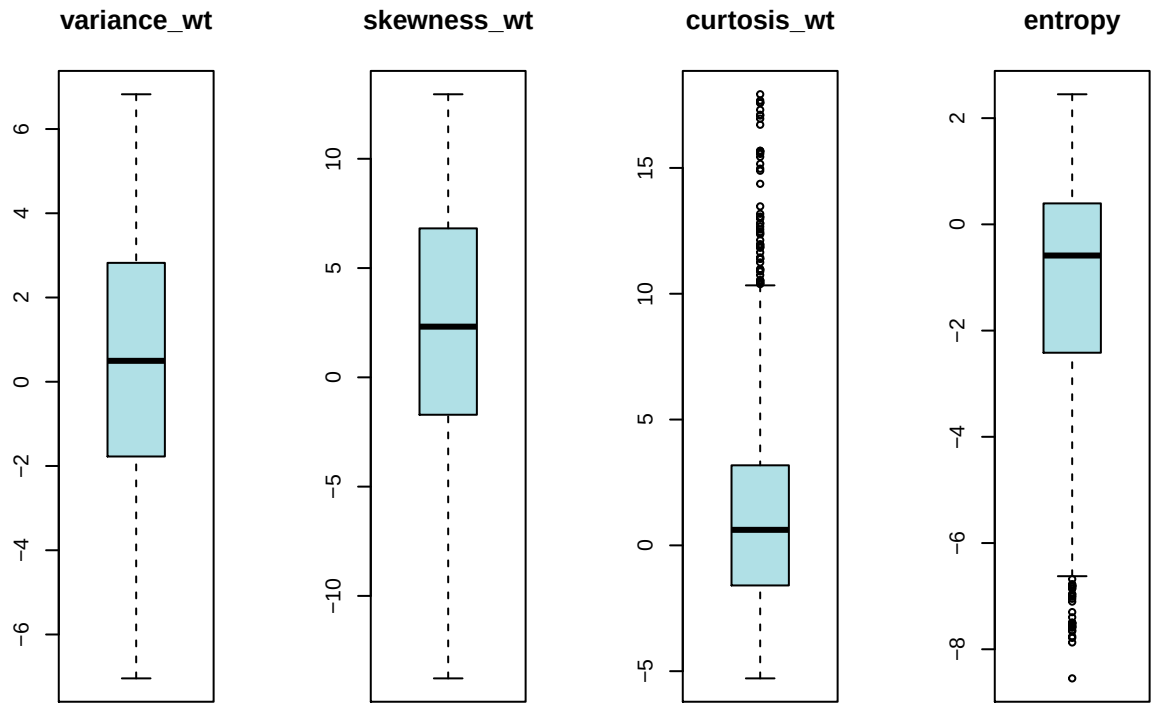
```r
summary(banknote_auth)
```

```
##   variance_wt       skewness_wt       curtosis_wt         entropy
##  Min.   :-7.0421   Min.   :-13.773   Min.   :-5.2861   Min.   :-8.5482
##  1st Qu.:-1.7730   1st Qu.: -1.708   1st Qu.:-1.5750   1st Qu.:-2.4135
##  Median : 0.4962   Median :  2.320   Median : 0.6166   Median :-0.5867
##  Mean   : 0.4337   Mean   :  1.922   Mean   : 1.3976   Mean   :-1.1917
##  3rd Qu.: 2.8215   3rd Qu.:  6.815   3rd Qu.: 3.1793   3rd Qu.: 0.3948
##  Max.   : 6.8248   Max.   : 12.952   Max.   :17.9274   Max.   : 2.4495
##  authentic
##  0:762
##  1:610
##
##
##
##
```

It's hard to analyze the data we have from this table, so we're going to use some visualization tools to make ir clearer. First, we are going to plot all four parameters in a boxplot graph:

```r
x <- banknote_auth[,1:4]
y <- banknote_auth[,5]
```

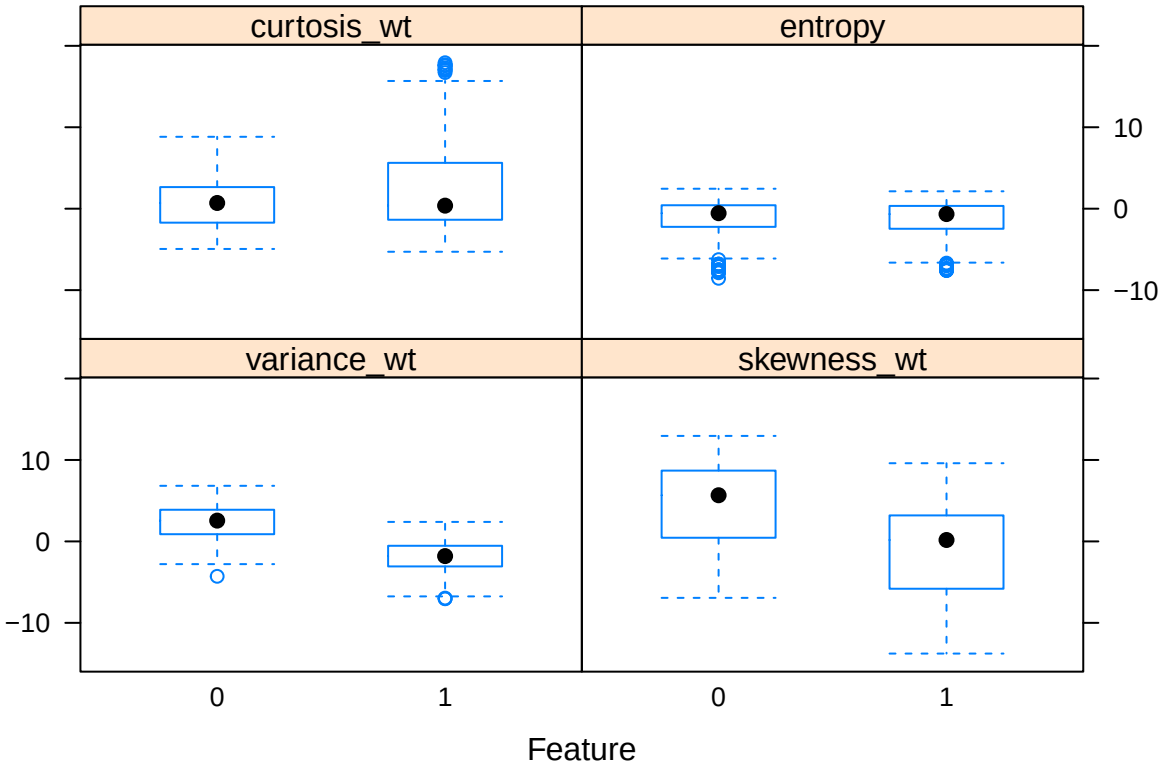
```r
par(mfrow = c(1,4))
  for(i in 1:4) {
   boxplot(x[,i], col = 'powderblue', main = names(banknote_auth)[i])
  }
```



We can see on this boxplot figure the distribution of the four parameters that we have: variance, skewness, curtosis and entropy. Variance and skewness parameters are closer in values and distribution, while curtosis and entropy show different values and behaviour.

But how do these parameters change depending if the banknote is fake or authentic? To start to visualize this, we can make different plots. Let's plot again boxplot, but this time taking into account the label **authentic** (1) or **fake** (0):

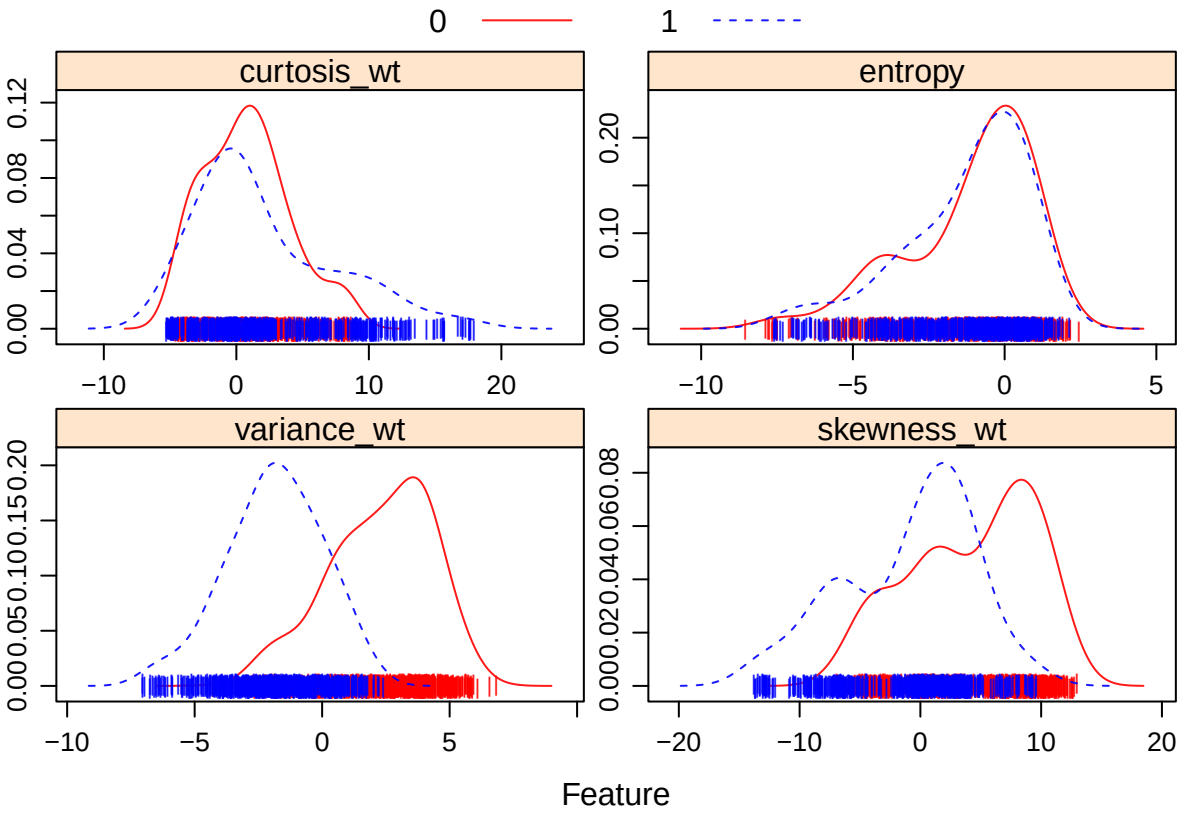
```
featurePlot(x = x, y = y, "box")
```



We can see that entropy does not vary for the two different levels, curtosis presents a small variation, but the wide errors make it difficult to actually see difference from the data. Skewness and Variance are the two parameters that present the biggest difference between authentic and fake samples, though skewness present larger errors.

We can further look into this with a density plot, in this case plotted again for all 4 values:

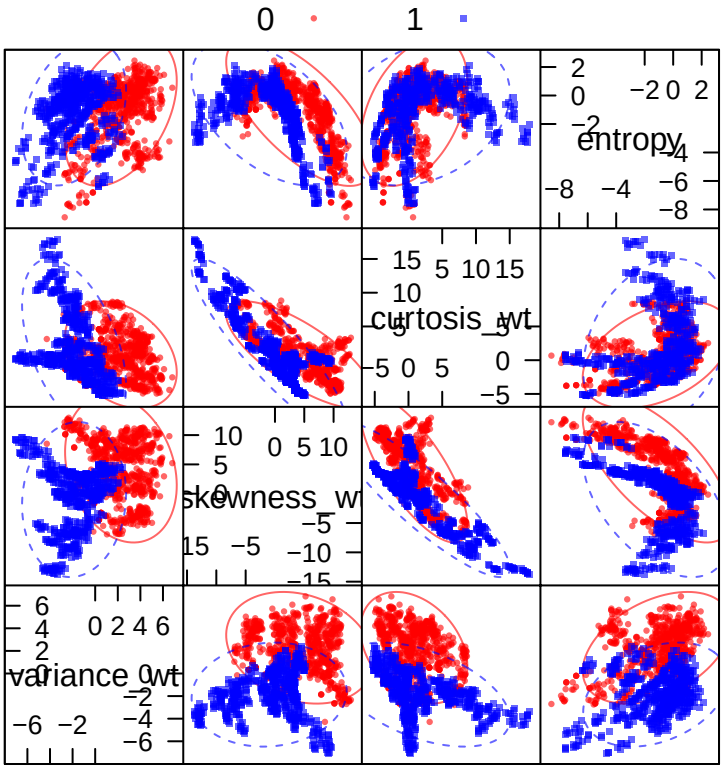
```
transparentTheme(trans = .9)
featurePlot(x = x,
            y = y,
            plot = "density",
            scales = list(x = list(relation="free"),
                          y = list(relation="free")),
            adjust = 1.5,
            pch = "|",
            layout = c(2, 2),
            auto.key = list(columns = 2))
```



We confirm what we concluded earlier: entropy presents little or no variation between the two classification levels, curtosis presents a small difference, while variance and skewness are the best parameters if we look at them one by one.

So, we have seen now, with the last two plots, that with only one parameter at one time, we can't really distinguish two levels. for our binary classification process. Let's see what happens when we plot these variables against each other, for both levels: fake (in red) and authentic (in blue). To achieve this, we use the featurePlot function as we can see on the following figure.

```
transparentTheme(set = TRUE, pchSize = 0.4, trans = .6)
featurePlot(x = banknote_auth[,1:4], y = banknote_auth[,5], "ellipse",
            auto.key = list(columns = 2))
```



Scatter Plot Matrix

We can conclude from this graph that some parameter combinations can help us classify our banknotes dataset. For example, we see a clustering process happening when we compare **variance** and **skewness**, or **variance** and **curtosis**. On the other hand, some parameters don't help us find a significative difference between counterfeit and genuine samples: for example, **curtosis** vs. **entropy**.

5. Machine Learning

5.1. Creating train and test sets

We need to create our train and test sets out of the *banknote_auth* dataframe. So, a standard way to do this is by randomly splitting the data. We will use the caret package (already loaded) and the function createDataPartition. We chose to select 20% (p = 0.2) of the data. The test set will be only used to test the performance of the model later.

```
set.seed(60)
test_index <- createDataPartition(banknote_auth$authentic, times = 1, p = 0.2, list = F)
train_set <- banknote_auth[-test_index,]
test_set <- banknote_auth[test_index,]
```

```
dimtrain <- dim(train_set)
dimtest <- dim(test_set)
dimtrain
```

```
## [1] 1097    5
```

```
dimtest
```

```
## [1] 275   5
```

We now have two sets, the **train set** with 1097 samples, the **test set** with 5 samples, and both having of course the five columns of the original dataset.

Let's check if the distribution of the variable we want to predict, **authentic** in this case, is balanced in each dataset, as we should expect:

```
checktrain <- table(train_set$authentic)/nrow(train_set)
checktest <- table(test_set$authentic)/nrow(test_set)



checktable <- matrix(c(checktrain[1],checktest[1],checktrain[2],checktest[2]),
                     ncol=2)
colnames(checktable) <- c('Fake (= 0)', 'Authentic (=1 )')
rownames(checktable) <- c('Train Set', ' Test Set')

as.table(checktable)
```

```
##           Fake (= 0) Authentic (=1 )
## Train Set  0.5551504       0.4448496
##  Test Set  0.5563636       0.4436364
```

We see that the createDataPartition function has created our train and test sets balanced. We are now going to test different machine learning algorithms as we said before, in the introduction of this report, starting with a baseline model.

5.2. Baseline Model: Logistic Regression

We have seen on the courses than Linear Regression is one of the simplest forms of Machine Learning, even if sometimes it's too rigid for some datasets. Furthermore, it serves as a baseline approach to start trying different Machine Learning algorithms to look for the best model. In this case, we are going to apply a Logistic Regression method, an extension of Linear Regression, dedicated to classification tasks. We are going to use the four predictors for this model (variance, skewness, curtosis and entropy). We use this piece of code:

```
glm_fit <- train_set %>% glm(authentic ~ variance_wt + skewness_wt
                             + curtosis_wt + entropy, data =., family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
p_hat_logit <- predict(glm_fit, newdata = test_set, type = "response")
```

To get the predictions, we use:

```
y_hat_logit <- ifelse(p_hat_logit > 0.5, 1, 0) %>% factor(levels = c(0, 1))
confusionMatrix(y_hat_logit, test_set$authentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
```

```
##          0 147   0
##          1   6 122
##
##                Accuracy : 0.9782
##                  95% CI : (0.9531, 0.992)
##     No Information Rate : 0.5564
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.956
##  Mcnemar's Test P-Value : 0.04123
##
##             Sensitivity : 0.9608
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9531
##              Prevalence : 0.5564
##          Detection Rate : 0.5345
##    Detection Prevalence : 0.5345
##       Balanced Accuracy : 0.9804
##
##        'Positive' Class : 0
##
```

We can see that we have an accuracy of 97.8%. This dataset is fairly simple, that's why we already have a very good accuracy, but we can still see that the algorithm misplaced 6 different samples. Let's see if we can improve this. For that purpose, we are going to create a table to put the results we get for each model we test:

```
acc_baseline <- confusionMatrix(y_hat_logit, test_set$authentic)$overall[1]
accuracy_results <- data_frame(Method = "Baseline Model", Accuracy = acc_baseline)

accuracy_results
```

```
## # A tibble: 1 x 2
##   Method         Accuracy
##   <chr>             <dbl>
## 1 Baseline Model    0.978
```

Note on entropy parameter

One thing that's interesting to note here, is that from the data analysis part, we have seen that entropy didn't give us much information to classify the banknotes. Just for fun and exploring, we are going to run again the code, without using **entropy** as a predictor, to see how it performs. Here it is:

```
glm_fit2 <- train_set %>% glm(authentic ~ variance_wt + skewness_wt +
                                curtosis_wt, data =., family = "binomial")

p_hat_logit2 <- predict(glm_fit2, newdata = test_set, type = "response")

y_hat_logit2 <- ifelse(p_hat_logit2 > 0.5, 1, 0) %>% factor(levels = c(0, 1))
confusionMatrix(y_hat_logit2, test_set$authentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction   0   1
##          0 147   0
##          1   6 122
##
##                Accuracy : 0.9782
##                  95% CI : (0.9531, 0.992)
##     No Information Rate : 0.5564
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.956
##  Mcnemar's Test P-Value : 0.04123
##
##             Sensitivity : 0.9608
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9531
##              Prevalence : 0.5564
##          Detection Rate : 0.5345
##    Detection Prevalence : 0.5345
##       Balanced Accuracy : 0.9804
##
##        'Positive' Class : 0
##
```

We see that we obtain the same results either taking or not into account the **entropy** parameter, but if we take one of the others predictors out of the equation we perform worse (we are not going to show it here not to make this report too long, but if we just eliminate curtosis from the model for example, we have a worse accuracy). So, it would be interesting to see if **entropy** is really an useful parameter, or just as the data analysis and this code showed us, it doesn't improve our classification goal.

5.3. Classification/Decision Trees

How can we draw a decision tree from this data? Is the **entropy** behaviour going to be reflected in it? Let's find out. With this code, we split the predictors into different regions, and then, using the observations in a particular region, a prediction is made. For this part

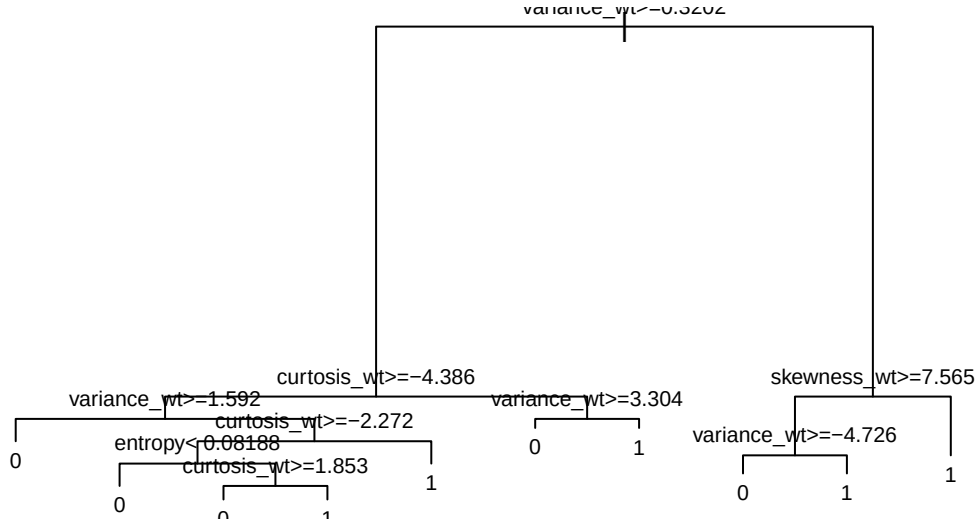
```
fit_tree <- rpart(banknote_auth$authentic~., data = banknote_auth)
fit_tree
```

```
## n= 1372
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1372 610 0 (0.555393586 0.444606414)
##    2) variance_wt>=0.320165 715  77 0 (0.892307692 0.107692308)
##      4) curtosis_wt>=-4.38605 673  45 0 (0.933135215 0.066864785)
##        8) variance_wt>=1.5922 489   3 0 (0.993865031 0.006134969) *
##        9) variance_wt< 1.5922 184  42 0 (0.771739130 0.228260870)
##         18) curtosis_wt>=-2.2722 157  18 0 (0.885350318 0.114649682)
##           36) entropy< 0.081882 120   1 0 (0.991666667 0.008333333) *
##           37) entropy>=0.081882 37  17 0 (0.540540541 0.459459459)
##             74) curtosis_wt>=1.85305 18   0 0 (1.000000000 0.000000000) *
##             75) curtosis_wt< 1.85305 19   2 1 (0.105263158 0.894736842) *
##         19) curtosis_wt< -2.2722 27   3 1 (0.111111111 0.888888889) *
```

```
##      5) curtosis_wt< -4.38605 42  10 1 (0.238095238 0.761904762)
##       10) variance_wt>=3.30405 10   0 0 (1.000000000 0.000000000) *
##       11) variance_wt< 3.30405 32   0 1 (0.000000000 1.000000000) *
##    3) variance_wt< 0.320165 657 124 1 (0.188736682 0.811263318)
##      6) skewness_wt>=7.5653 105  20 0 (0.809523810 0.190476190)
##       12) variance_wt>=-4.726 85   0 0 (1.000000000 0.000000000) *
##       13) variance_wt< -4.726 20   0 1 (0.000000000 1.000000000) *
##      7) skewness_wt< 7.5653 552  39 1 (0.070652174 0.929347826) *
```

We can visually see where the splits were made using this code:

```
plot(fit_tree)
text(fit_tree, cex = 0.7)
```



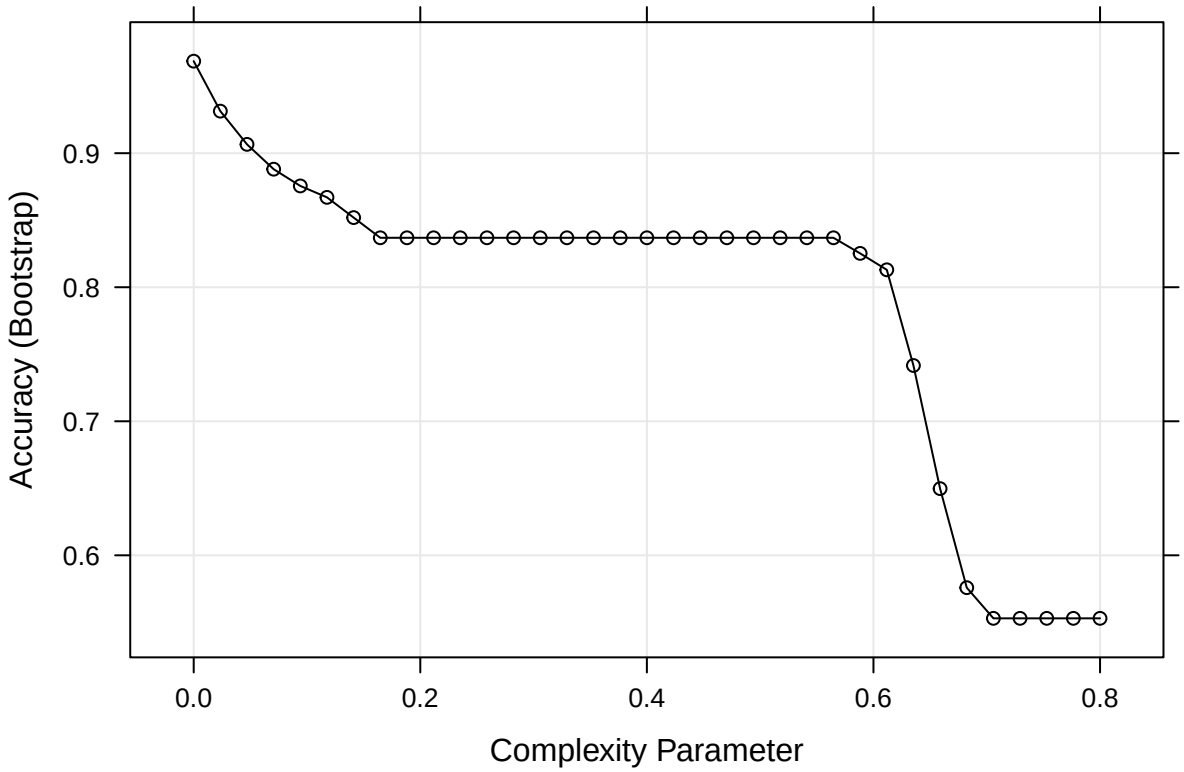
We see that, as we would think by looking at our previous data analysis, that **variance** is a main predictor when it comes to distinguish between counterfeit and genuine banknotes. Then, both **curtosis** and **skewness** are important when we want to make this classification. Finally, as we also pointed before, **entropy** appears at the end, as a predictor with a minor weight than the other three.

Now, as a second method, we are going to see how well a Decision Tree algorithm performs on the training data. We also try to find the best complexity parameter *cp* using cross-validation. This may not be the best for this kind of simple datasets, but for a more exploratory objective we are going to do it.

```
xrpart <- train_set[, 1:4]
yrpart <- train_set$authentic
train_rpart <- train(xrpart, yrpart, method = "rpart",
                     tuneGrid = data.frame(cp = seq (0,0.8,len = 35)))
```

Now we can plot the accuracy obtained for each *cp* value:

```r
plot(train_rpart)
```



We see that the maximum accuracy (thus, the best fit) is achieved with $cp = 0$. We suppose this happens because the tree is heavily pruned beforehand because of the nature of the data.

To see how well it performed on our data, we use the following code:

```r
acc_tree <- confusionMatrix(predict(train_rpart, test_set), test_set$authentic)$overall["Accuracy"]
confusionMatrix(predict(train_rpart, test_set), test_set$authentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 151   3
##          1   2 119
##
##                Accuracy : 0.9818
##                  95% CI : (0.9581, 0.9941)
##     No Information Rate : 0.5564
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9631
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9869
##             Specificity : 0.9754
##          Pos Pred Value : 0.9805
```

```
##          Neg Pred Value : 0.9835
##              Prevalence : 0.5564
##          Detection Rate : 0.5491
##    Detection Prevalence : 0.5600
##       Balanced Accuracy : 0.9812
##
##        'Positive' Class : 0
##
```

```
acc_tree
```

```
##  Accuracy
## 0.9818182
```

We obtain an accuracy of about 98%. This accuracy is better than the one reached by our baseline method, we now have less misplaced predictions (less false positives and false negatives):

```
accuracy_results <- bind_rows(accuracy_results,
                          data_frame(Method = "Decision Tree",
                                     Accuracy = acc_tree))
accuracy_results
```

```
## # A tibble: 2 x 2
##   Method         Accuracy
##   <chr>             <dbl>
## 1 Baseline Model    0.978
## 2 Decision Tree     0.982
```

5.4. Random Forest

The third model we are going to test is the Random Forest approach: the goal is to improve prediction performance and reduce instabiliy by averaging multiple decision trees, a forest of trees constructed with randomness. We use our training data this way:

```
train_rf <- randomForest(train_set$authentic~., data = train_set)
confusionMatrix(predict(train_rf, test_set), test_set$authentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 152   0
##          1   1 122
##
##                Accuracy : 0.9964
##                  95% CI : (0.9799, 0.9999)
##     No Information Rate : 0.5564
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9926
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9935
##             Specificity : 1.0000
```

```
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9919
##              Prevalence : 0.5564
##          Detection Rate : 0.5527
##    Detection Prevalence : 0.5527
##       Balanced Accuracy : 0.9967
##
##        'Positive' Class : 0
##
```

```
acc_rf <- confusionMatrix(predict(train_rf, test_set), test_set$authentic)$overall[1]
```

We see that we have improved our accuracy by 1%, thus now reaching 99%. We can apply the "Rborist" method to try and get a better accuracy, but we won't do it on this report because of computing capacity. Updating the table, we get:

```
accuracy_results <- bind_rows(accuracy_results,
                          data_frame(Method = "Random Forest",
                                     Accuracy = acc_rf))
accuracy_results
```

```
## # A tibble: 3 x 2
##   Method         Accuracy
##   <chr>             <dbl>
## 1 Baseline Model    0.978
## 2 Decision Tree     0.982
## 3 Random Forest     0.996
```

5.5. K-nearest Neighbours (KNN)

To try to get an even higher accuracy for this dataset, we are now going to use the KNN approach. Using this model, for any point for which we want to estimate the conditional probability, we are going to look at the k-nearest points and then take an average of these points. To use this method we are going to run the following piece of code:

```
knn_fit <- knn3(train_set$authentic ~., data = train_set, k = 5)

y_hat_knn <- predict(knn_fit, test_set, type = "class")
confusionMatrix(data = y_hat_knn, reference = test_set$authentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 153   0
##          1   0 122
##
##                Accuracy : 1
##                  95% CI : (0.9867, 1)
##     No Information Rate : 0.5564
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
```

```
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5564
##          Detection Rate : 0.5564
##    Detection Prevalence : 0.5564
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

Using a value of k = 5, we get 100% accuracy, beating of course all previous models used. We add it to the accuracy results table.

```
acc_knn <- confusionMatrix(y_hat_knn, test_set$authentic)$overall[1]
accuracy_results <- bind_rows(accuracy_results,
                          data_frame(Method = "KNN",
                                     Accuracy = acc_knn))

accuracy_results
```

```
## # A tibble: 4 x 2
##   Method         Accuracy
##   <chr>             <dbl>
## 1 Baseline Model    0.978
## 2 Decision Tree     0.982
## 3 Random Forest     0.996
## 4 KNN               1
```

What happens if instead of taking all predictors, as we noticed before, we only take the most important ones and we leave **entropy** out? Last time, it didn't have any improvement in the final accuracy. Let's try this for the KNN model.

```
knn_fit2 <- train_set %>% knn3(authentic ~ variance_wt + skewness_wt + curtosis_wt, data =.)

y_hat_knn2 <- predict(knn_fit2, test_set, type = "class")
confusionMatrix(data = y_hat_knn2, reference = test_set$authentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 150   1
##          1   3 121
##
##                Accuracy : 0.9855
##                  95% CI : (0.9632, 0.996)
##     No Information Rate : 0.5564
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9706
##  Mcnemar's Test P-Value : 0.6171
```

```
##
##             Sensitivity : 0.9804
##             Specificity : 0.9918
##          Pos Pred Value : 0.9934
##          Neg Pred Value : 0.9758
##              Prevalence : 0.5564
##          Detection Rate : 0.5455
##    Detection Prevalence : 0.5491
##       Balanced Accuracy : 0.9861
##
##        'Positive' Class : 0
##
```

In this case, we see that **entropy** is an important predictor to take into account, because without it, we just showed that our accuracy decreases to 98%. Still, using KNN methods with only three predictors gives us a better accuracy than using the baseline model or a decision tree method with all four predictors!

6. Conclusions

The objective of this report was getting to know the Banknote Authentication dataset from the UCI Repository, play a little with the variables and the parameters we had to get to know the data, and then explore different Machine Learning models and see how they perform, and if we could get a good accuracy.

We went through the package installation, data download and analysis on the first sections, and then tried different approaches to get a high accuracy value, starting from the simplest approach possible: the baseline Approach (here, it was Logistic Regression), to others as Decision Tree, Random Forest and K-Nearest Neighbors.

We found that with the baseline approach we obtained an accuracy of around 97%, which is not bad at all, due to the fairly simple nature of this dataset. Then, we compared it to other models, achieving 98% and 99% for Decision Tree and Random Forest models respectively. Finally, we got 100% accuracy using the KNN algorithm, with k set to 5.

We also remarked, both during the data analysis and the Machine Learning sections, that the **entropy** predictor was the less imporant one: we got a better accuracy for the KNN method with only 3 predictors (**variance**, **skewness** and **curtosis**), compared to the Baseline Model or the Decision Tree model, that performed worse with all 4 predictors. Still, **entropy** it's an important predictor if our goal is to achieve the best accuracy possible.

7. Personal Comments (and more conclusions)

Some personal comments I wanted to write about this project: I liked very much doing it, playing with the data and the graphs, to explore and put into this project all the hard work done on the past few months. When I started with this whole Data Science course (the 9 of them!), I didn't even know how to install RStudio, and now I'm so proud that I am able to write a whole report, do some analysis on the data and test it on different Machine Learning models, and even more, try to improve its accuracy.

I achieved 100% accuracy because, as I noted before, the banknote authentication set is fairly simple, but this won't probably happen in my future classification algorithm that I am going to develop for my neuroscience project during the next few months (as I mentioned in the introduction), because it won't be data as neat and classifiable as this one, but going through different models was fun and enlightening, and has given me a very wide approach to all that I can do in the future.

Thank you all and I hope you enjoyed this report!

GitHub repo: https://github.com/rocioam/Capstone-CYO