Capstone Project - Data Science Professional Certificate

Werner Alencar Advincula Dassuncao

2021-04-17

Contents

Introduction

Losses due to fraudulent payments have reached globally $ 28.65 billion in 2019, according to the most recent *Nilson Report* data. The United States alone accounts for over a third of the worldwide loss. These numbers are quite high and estimates for the US in 2020 are somewhere around $ 11 billion due to credit card fraud says *Julie Conroy*, research director for Aite Group's fraud and anti-money laundering practice. These fraud cases affect consumers, merchants and card issuers alike. The total of cost for credit card fraud extends far beyond the cost of the illegally purchased goods. So, being able to detect fraud before it happens is extremely important.

Access to actual financial data for research outside corporations is blocked due to privacy. For this project we will work with a data set from *The Mobile Money Payment Simulation* which was a case study based on a real company that has developed a mobile money implementation that provides mobile phone users with the ability to transfer money between themselves using the phone as a sort of electronic wallet. Edgar Alonso Lopez-Roza explains: *"The development of PaySim covers two phases. During the first phase, we modeled and implemented a MABS (Multi Agent Based Simulation) that used the schema of the real mobile money service and generated synthetic data following scenarios that were based on predictions of what could be possible when the real system starts operating. During the second phase we got access to transactional financial logs of the system and developed a new version of the simulator which uses aggregated transactional data to generate financial information more alike the original source"*.

This project aims to generate a machine learning model to predict if a transaction is fraudulent. Before we can talk about machine learning we need to explore, summarize and graph the data with the objective of learning about possible patterns and/or correlation between the variables. Secondly, we will look into the available predicting candidates (features). For this project our target variable will point if a observation is fraud or not.

Next will dive into two models for the machine learning section. The first will be Support Vector Machines (SVM) where we will deploy linear and polynomial kernels, and, the later, eXtreme Gradient Boosting (xgboost). These models will be trained using our train set, tested with the test set, and finally their performance will be "double-checked" on our final hold-out validation set.

Prepare the R environment

The packages bellow will be installed automatically, if necessary.

```
library(devtools)
library(tidyverse)
library(kableExtra)
library(gridExtra)
library(scales)
library(caret)
library(xgboost)
library(Matrix)
library(e1071)
library(clue)
library(DiagrammeR)
library(ComplexHeatmap)
```

Exploratory Data Analysis

Data availability

The PaySim dataset is publicly hosted by Kaggle through the URL: *https://www.kaggle.com/ntnu-testimon/paysim1/download*. I wrote a R script to automatically download the data, decompress the zip file and load it up to the R environment, which worked while the link did not expire. This code is available bellow in the code chunk *download the source data*. Kaggle does not provide a direct download link for this dataset, therefore the automated process to download, extract and load its data is not possible as per April 16th, 2021. This is due to the fact that the download link is dynamically created to the user connection (session) and expires shortly after clicking the download button.

```
# # KAGGLE's DOWNLOAD LINK EXPIRES SHORTLY AFTER CLICKING THE DOWNLOAD BUTTON.
# # This is an example code for automating the download, decompressing
# # and loading of data when a permanent download link is available

# # Create a temporary file
# dl <- tempfile()

# # Paste the direct download link for your connection on Kaggle
# file_direct_link <- 'https://storage.googleapis.com/kaggle-data-sets/1069/1940/bundle/archive.zip?X-G

# # Download the file to the temporary file
# download.file( file_direct_link, dl )

# # Decompress the downloaded zip file
# data <- unzip( zipfile = dl, 'PS_20174392719_1491204439457_log.csv' )

# # Read the csv file
# data <- read.csv(data)

# # remove temp file
# unlink(dl)

# Now the "data" object contains the contents from the file downloaded from Kaggle!
```

Loading the PaySim data

Alternatively, for those of you who would like to replicate the findings and results displayed here: you can load the PaySim data after its download from Kaggle and unzipping of the CSV file. Make sure to place the CSV file in the same folder you have downloaded the Rmd and R files for this project. We load the data from the file downloaded to the local machine in the code chunk named *load the data*.

```
# Load the data directly from local csv file
data <- read.csv('PS_20174392719_1491204439457_log.csv')
```

In this section we will browse the data from PaySim. Our objective is to determine what features (variables) have are relevant to predicting our desired output: is the transaction a case of 'fraud' or 'not'. The data has 6,362,620 observations of 11 variables. Usint the *str()* function we can display the data structure.

Features, data types and observation

```
## 'data.frame':    6362620 obs. of  11 variables:
##  $ step          : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ type          : chr  "PAYMENT" "PAYMENT" "TRANSFER" "CASH_OUT" ...
##  $ amount        : num  9840 1864 181 181 11668 ...
##  $ nameOrig      : chr  "C1231006815" "C1666544295" "C1305486145" "C840083671" ...
##  $ oldbalanceOrg : num  170136 21249 181 181 41554 ...
##  $ newbalanceOrig: num  160296 19385 0 0 29886 ...
##  $ nameDest      : chr  "M1979787155" "M2044282225" "C553264065" "C38997010" ...
##  $ oldbalanceDest: num  0 0 0 21182 0 ...
##  $ newbalanceDest: num  0 0 0 0 0 ...
##  $ isFraud       : int  0 0 1 1 0 0 0 0 0 0 ...
##  $ isFlaggedFraud: int  0 0 0 0 0 0 0 0 0 0 ...
```

To check if there are Not Available (NA) values on the data we can use *anyNA()*, as of R 3.1.0, which implements *any(is.na(x))* in a possibly faster way (especially for atomic vectors) since it stops after the first NA, and not wasting time checking the remainder of the data.

```
anyNA(data)
```

```
## [1] FALSE
```

There are no missing values in the dataset.

Sample row of the data

Let's continue by looking into a sample row using the *slice_sample()* on the data and explain the variables:

```
##   step    type   amount   nameOrig oldbalanceOrg newbalanceOrig   nameDest
## 1   23 CASH_IN 63105.29 C596578417          7556       70661.29 C464294308
##   oldbalanceDest newbalanceDest isFraud isFlaggedFraud
## 1              0              0       0              0
```

Description of the features

- step - maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation).
- type - CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.
- amount - amount of the transaction in local currency.
- nameOrig - customer who started the transaction.

- oldbalanceOrg - initial balance before the transaction.
- newbalanceOrig - new balance after the transaction.
- nameDest - customer who is the recipient of the transaction.
- oldbalanceDest - initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants).
- newbalanceDest - new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants).
- isFraud - This is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control or customers accounts and try to empty the funds by transferring to another account and then cashing out of the system.
- isFlaggedFraud - The business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction.

To obtain a summary statistics with mean, median, 25th and 75 quartiles, min and max values we can use the *summary()* function.

Account holders

There are two kinds of account: *customer* and *merchant*. These can be identified by the first character in the *nameOrig* (initiator account) or *nameDest* (receiver account). There are no merchant accounts registered in the nameOrig column, while the nameDest column have observations both customer and merchant accounts.

Data summary

```
##       step          type               amount           nameOrig        
##  Min.   :  1.0   Length:6362620     Min.   :       0   Length:6362620    
##  1st Qu.:156.0   Class :character   1st Qu.:   13390   Class :character  
##  Median :239.0   Mode  :character   Median :   74872   Mode  :character  
##  Mean   :243.4                      Mean   :  179862                     
##  3rd Qu.:335.0                      3rd Qu.:  208721                     
##  Max.   :743.0                      Max.   :92445517                     
##  oldbalanceOrg      newbalanceOrig       nameDest          oldbalanceDest     
##  Min.   :       0   Min.   :       0   Length:6362620     Min.   :        0  
##  1st Qu.:       0   1st Qu.:       0   Class :character   1st Qu.:        0  
##  Median :   14208   Median :       0   Mode  :character   Median :   132706  
##  Mean   :  833883   Mean   :  855114                      Mean   :  1100702  
##  3rd Qu.:  107315   3rd Qu.:  144258                      3rd Qu.:   943037  
##  Max.   :59585040   Max.   :49585040                      Max.   :356015889  
##  newbalanceDest         isFraud          isFlaggedFraud  
##  Min.   :        0   Min.   :0.000000   Min.   :0.0e+00  
##  1st Qu.:        0   1st Qu.:0.000000   1st Qu.:0.0e+00  
##  Median :   214661   Median :0.000000   Median :0.0e+00  
##  Mean   :  1224996   Mean   :0.001291   Mean   :2.5e-06  
##  3rd Qu.:  1111909   3rd Qu.:0.000000   3rd Qu.:0.0e+00  
##  Max.   :356179279   Max.   :1.000000   Max.   :1.0e+00  
```

The *type* column is categorical and we will use a barplot to visualize it. On the figure ‘Frequency table of transaction types’ we can see the description of the proportion of each category.

CASH-IN is the process of increasing the balance of account by paying in cash to a merchant.
CASH-OUT is the opposite process of CASH-IN, it means to withdraw cash from a merchant which decreases the balance of the account.
DEBIT is similar process than CASH-OUT and involves sending the money from the mobile money service to a bank account.
PAYMENT is the process of paying for goods or services to merchants which decreases the balance of the account and increases the balance of the receiver.
TRANSFER is the process of sending money to another user of the service through the mobile money platform.

Distribution of transactions by type.

According to the plot *Frequency of transaction by type*, the transaction CASH-OUT is the most used, closely followed by PAYMENT. DEBIT is the least used transaction. The *Transactions per average amount* plot displays the transactions sorted by average amount, TRANSFER has the highest average, while CASH-IN and CASH-OUT have similar averages.

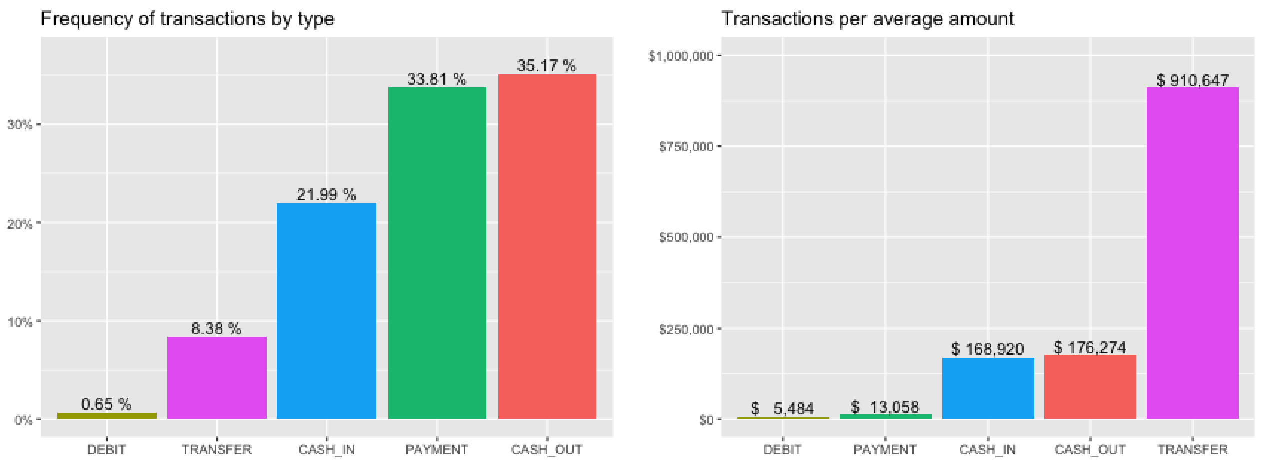


Figure 1: Distributions by transactional frequency and average amount

The plot *Boxplot of transactions by type* shows the data distribution of each transaction type side-by-side and their quick summaries, the box in the middle indicates ‘hinges’ (close to first and third quartiles) and median in the center of the box. Outliers are displayed as points above and bellow the center box.

Fraudulent transactions

We can observe the existence of five transaction categories, but only two it’s types are cases of fraud:

- TRANSFER: money is remitted to a customer, the fraudster.
- CASH-OUT: money is remitted to a merchant who pays the customer (fraudster) in cash.

Plot 4: displays the percentage of fraudulent transactions. Plot 5, 6 and 7: demonstrates the comparison between TRANSFER and CASH-OUT by average amount, number of transactions and boxplot with medium, quantiles and distribution of amounts.

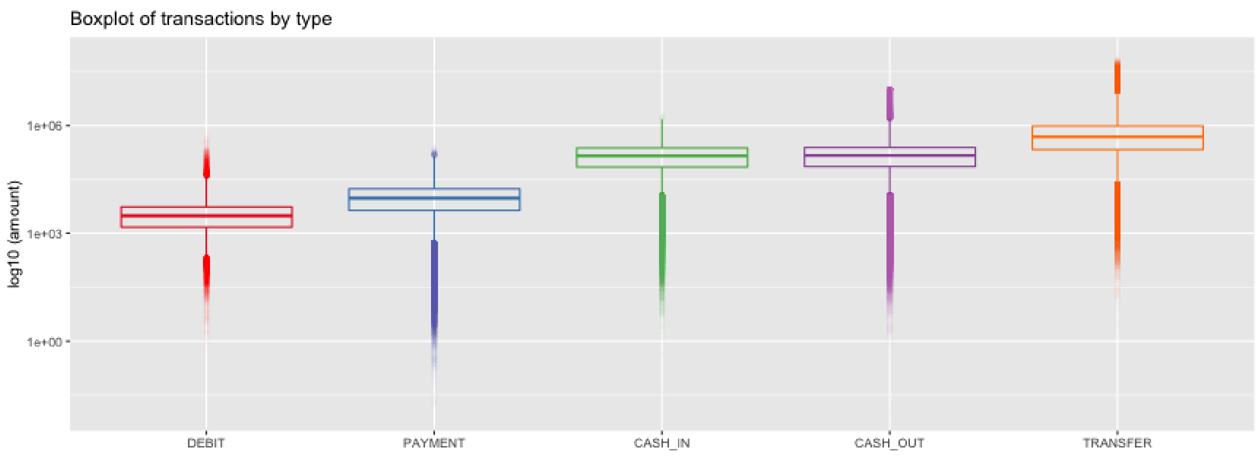
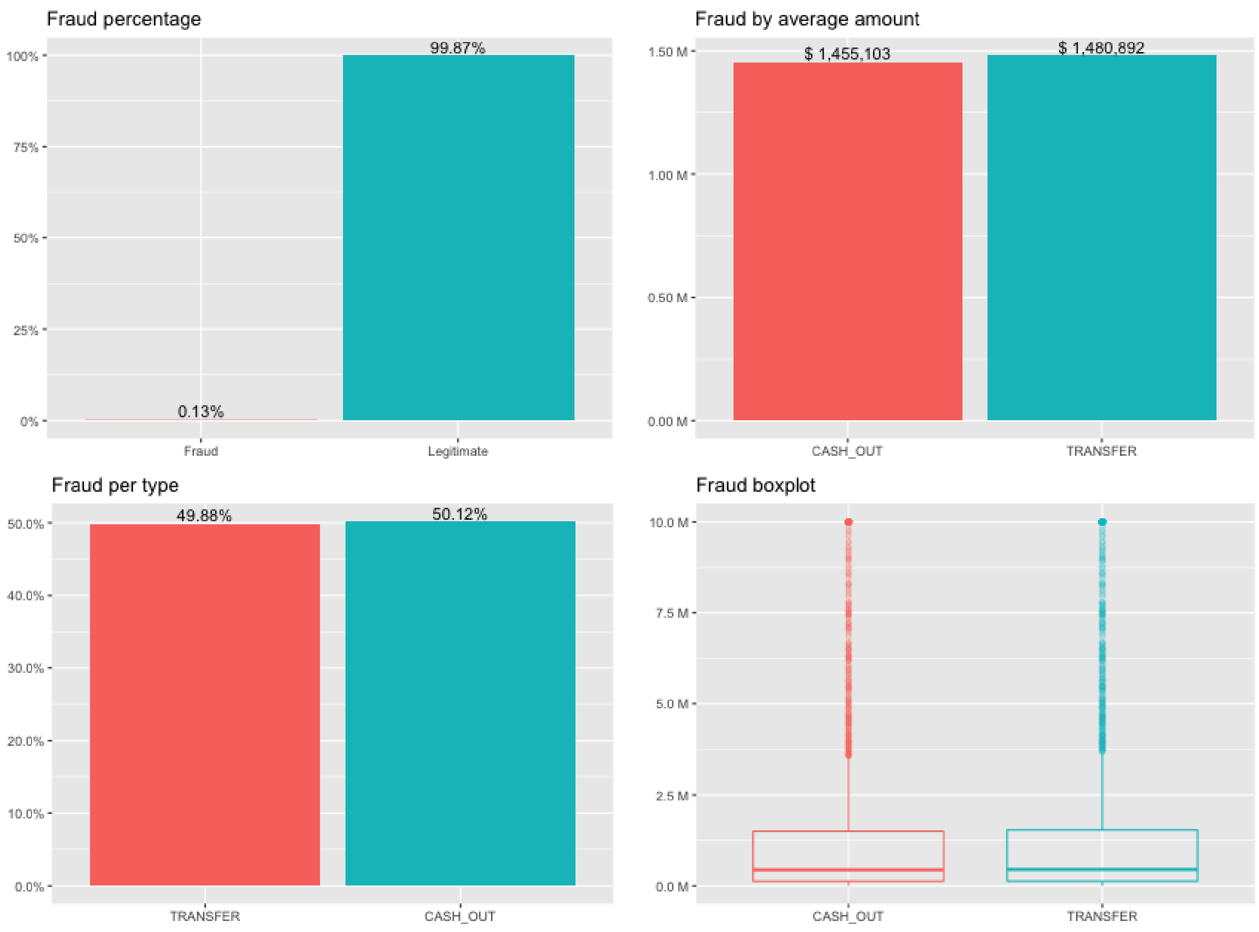


Figure 2: Box plot of transactions by type



We can observe that the percentage of fraudulent transactions is 0.13 %. It is worth noting that the number of fraud TRANSFERs is almost the same as CASH_OUT transactions. Let's look into what kind of transactions are actual fraud cases:

The total of fraudulent transactions, CASH_OUT and TRANSFER together, is 8,213 adding up to a sum of $ 12,056,415,428 with an average value per transaction of $ 1,467,967. Min and max values are, respectively, $ 0 and $ 10,000,000.

Table 2: Summary table of fraudulent transactions

type	number	average	min	max	sum
CASH_OUT	4,116	1,455,103	0	10,000,000	5,989,202,244
TRANSFER	4,097	1,480,892	63.8	10,000,000	6,067,213,184

Table 3: Summary of transactions when isFlaggedFraud is set

quantity	avg	min	max
16	4,861,598	353,874.2	10,000,000

The number of different origin accounts is 6,353,307 and the quantity of distinct receiver accounts is 2,722,362. There are no transactions between same account identifiers or same account holder. This does not match the *modus operandi* description on Kaggle: the first step is to make a TRANSFER to a fraudulent account, then execute a CASH_OUT. These two steps would have been realized by the same fraudster account (origin and destination). The data does not show the expected behavior for this fraudulent modus-operandi since there are no transaction with same agent and marked with isFraud set to 1.

When verifying if there are fraudulent transactions by the same account number for origin and destination we encounter no cases that satisfy the *modus operandi* as expected by the descriptions on the PaySim paper: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2584265/2017+IJSPM+EDL+Final+version.pdf?isAllowed=y&sequence=2

About the isFlaggedFraud feature

The column isFlaggedFraud is only set a few times in all 6 Million observations. Let's check if this column is consistently set with the isFraud column. Here we filter the data for isFraud and isFlaggedFraud equals to 1. How many observations have the isFraud and isFlaggedFraud set simmultaneously?

```
## [1] 16
```

The amount of observations with isFlaggedFraud is extremely low 16 given the total number of transactions where isFraud is set: 8,213. Eventhough we have confirmed that all observations where isFlaggedFraud is set also has isFraud variable set, the only transaction type where the isFlaggedFraud is set is: 5. So far this feature does not seem to add enough value to our model since it is extremely rare compared to the 6 Million rows of data and, nonetheless, it only occurs for TRANSFER transactions.

The table *Summary of transactions when isFlaggedFraud is set* displays a the summary of this data.

We are looking to determine if there it is possible to relate the isFlaggedFraud with other variables, such as defining possible thresholds for this feature being set and based on other columns. There are multiple transactions from the same customer names where isFlaggedFraud is not set, but these duplicates do not exist otherwise. So, the number of transactions by same user (account) can't determine if the isFlaggedFraud is set or not.

The feature *step* occurs when isFlaggedFraud is set with the following values: . These values expand over most of the range for this feature and therefore we can use it as a threshold for isFlaggedFraud being set.

Do *oldbalanceOrig*, *newbalanceOrig*, *oldbalanceDest* and *newbalanceDest* explain how isFlaggedFraud is set? The minimum value for oldbalanceDest is 0 and maximum is 0, while for oldbalanceOrig the min and max are , respectively. Therefore neither of these can be used to threshold when isFlaggedFraud is being set since their corresponding values overlap with other transactions which are not flagged. Again these are also independent from isFlaggedFraud. It is worth noting that for all flagged transactions (all of the TRANSFERs) the old and new balances are the same.

The total number of transactions where isFlaggedFraud is set is equal the number of transactions where old and new balances for origin and destination accounts, as shown on table *All cases where the isFlaggedFraud is set*.

```
## [1] "All cases where the isFlaggedFraud is set"
```

```
##    step     type     amount    nameOrig oldbalanceOrg newbalanceOrig
## 1   212 TRANSFER  4953893.1  C728984460     4953893.1      4953893.1
## 2   250 TRANSFER  1343002.1 C1100582606     1343002.1      1343002.1
## 3   279 TRANSFER   536624.4 C1035541766      536624.4       536624.4
## 4   387 TRANSFER  4892193.1  C908544136     4892193.1      4892193.1
## 5   425 TRANSFER 10000000.0  C689608084    19585040.4     19585040.4
## 6   425 TRANSFER  9585040.4  C452586515    19585040.4     19585040.4
## 7   554 TRANSFER  3576297.1  C193696150     3576297.1      3576297.1
## 8   586 TRANSFER   353874.2 C1684585475      353874.2       353874.2
## 9   617 TRANSFER  2542664.3  C786455622     2542664.3      2542664.3
## 10  646 TRANSFER 10000000.0   C19004745    10399045.1     10399045.1
## 11  646 TRANSFER   399045.1  C724693370    10399045.1     10399045.1
## 12  671 TRANSFER  3441041.5  C917414431     3441041.5      3441041.5
## 13  702 TRANSFER  3171085.6 C1892216157     3171085.6      3171085.6
## 14  730 TRANSFER 10000000.0 C2140038573    17316255.1     17316255.1
## 15  730 TRANSFER  7316255.0 C1869569059    17316255.1     17316255.1
## 16  741 TRANSFER  5674547.9  C992223106     5674547.9      5674547.9
##       nameDest oldbalanceDest newbalanceDest isFraud isFlaggedFraud
## 1   C639921569              0              0       1              1
## 2  C1147517658              0              0       1              1
## 3  C1100697970              0              0       1              1
## 4   C891140444              0              0       1              1
## 5  C1392803603              0              0       1              1
## 6  C1109166882              0              0       1              1
## 7   C484597480              0              0       1              1
## 8  C1770418982              0              0       1              1
## 9   C661958277              0              0       1              1
## 10 C1806199534              0              0       1              1
## 11 C1909486199              0              0       1              1
## 12 C1082139865              0              0       1              1
## 13 C1308068787              0              0       1              1
## 14 C1395467927              0              0       1              1
## 15 C1861208726              0              0       1              1
## 16 C1366804249              0              0       1              1
```

After these observations we can conclude that the *isFlaggedFraud* feature does not seem be set according to a set logic and only 16 times total. We can consider this variable not relevant and remove it from our prediction model.

Variable name consistency

From the structure of the data, we rename one of the features with a typo so that it matches the name structure of the other features and increase readability: "oldbalanceOrg" to "oldbalanceOrig".

```
# For readability and consistency, we will correct the
# column name "oldbalanceOrg" to "oldbalanceOrig"
data <- data %>% rename(oldbalanceOrig = oldbalanceOrg)
```

In this section we will organize the data according to the findings from the previous section. We will keep the data related only to the fraudulent transactions: TRANSFER and CASH_OUT. We create a object with features X for analysis and our target value Y. Since we only have two types of transactions we will use 0's for TRANSFERs and 1's for CASH_OUTs.

```r
# create a X data object with
X <- data %>% filter( (type == 'TRANSFER') | (type == 'CASH_OUT') )

# grab the target variable 'isFraud' from the filtered dataframe X
Y <- X$isFraud
# remove 'isFraud' from X
X <- X[,!(names(X) %in% 'isFraud')]

# remove the columns that are not relevant according to data exploration
drop <- c('nameOrig', 'nameDest', 'isFlaggedFraud')
X <- X[,!(names(X) %in% drop)]

# Encode binary values for TRANSFER and CASH_OUT
X <- X %>% mutate(type = str_replace_all(type, c('TRANSFER' = '0', 'CASH_OUT' = '1')))

# Convert the type to integer
X$type <- as.integer(X$type)

# Remove from memory
rm(data)

# Create two new features(columns) to help train the models
X <- X %>% mutate(balance_error_orig = newbalanceOrig + amount - oldbalanceOrig,
                  balance_error_dest = oldbalanceDest + amount - newbalanceDest)
```

Looking into to the balance of the destination accounts:

We have seen that there are various transactions with zero balances on the destination account. The percentages of these transactions are quite different. For non-fraudulent transactions the percentage is 0.06 % while for fraudulent transactions we have 49.56 %. The later percentage is substantially higher indicating that it could be a strong indicator of fraud. Instead, improve the fraud detection we highlight this by replacing the value 0 with -1 in the columns "oldbalanceDest" and "newbalanceDest". The old account balance will not be used with a statistic since it could mask this indicator of fraud and make fraudulent transactions appear genuine.

Prior and post transaction balances in the origin accounts

In the case of the origin accounts we observe the presence of several cases of zero balances before and after the transactions. For legitimate transactions we have 47.37 % and for fraudulent this number is much smaller 0.3 %. The proportion of fraudulent transactions considering the origin account balances is much smaller.

Creating new variables

On this dataset we have information about the balance for the origin and destination accounts. We will leverage this data by creating two new features: *balance_error_orig* and *balance_error_dest*, since the balance values might help us to identify fraudulent transactions. The calculation for these new features is: 'new balance' + 'amount' - 'old balance' = 'balance_error'.

Visualizations

Since our goal is to identify the fraud cases and we have learned from the data that only TRANSFER and CASH_OUT transactions are possible cases, we will filter the data to include only these observations. Next we will create visualizations and try to 'see' a difference between legitimate and fraudulent transactions. Starting with the transaction amount, we have the plot *Dispersion of transactions over amount.* We can observe that the dispersion over amount is not conclusive given that the same values occur both in legitimate and fraudulent transactions. Worth noting the for fraud cases the amount seems restricted evenly between transfers and cash_outs in the lower range. On the other hand for legitimate transactions, cash_outs operations have very small amounts compared to the legitimate transfers.

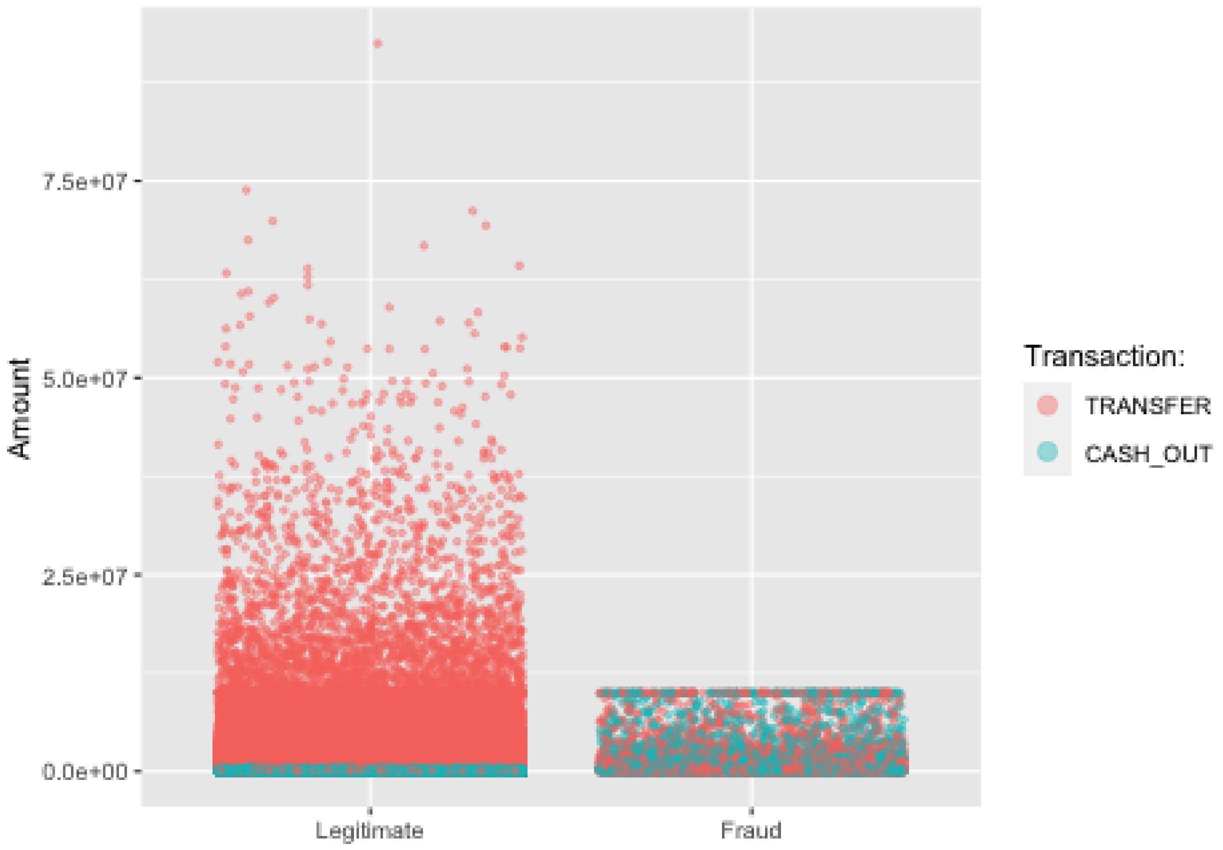


Figure 3: Dispersion of transactions over amount

Moving on to the next variable: step. As observed in the data exploration, the value of the step feature represents the number of hours since the start of the 30 day simulation. It is quite remarkable how the transactions types are homogeneously dispersed for fraudulent ones. The legitimate observations seem to display a pattern of bands(lines) over time.

Now we will plot using the two new computed features balance error for origin and destination accounts. Observing the first feature on plot *Balance error origin account* we can observe a dispersion similar to what we saw when plotting over the transaction amount.

The plot *Balance error on destination account* displays a very interesting dispersion, we can clearly see that for legitimate transactions the values tend to be negative, and for fraudulent transactions the values are gravitating towards positive side of the scale. Note that for Fraud transactions the green dots (CASH_OUT) tend to stay in the 0 error margin, while legitimate transactions are much more disperse.

The difference between fraudulent and legitimate transactions can be also be observed by the correlation heatmap. Heatmaps are great to analyse data because it translates the variation of the correlation values

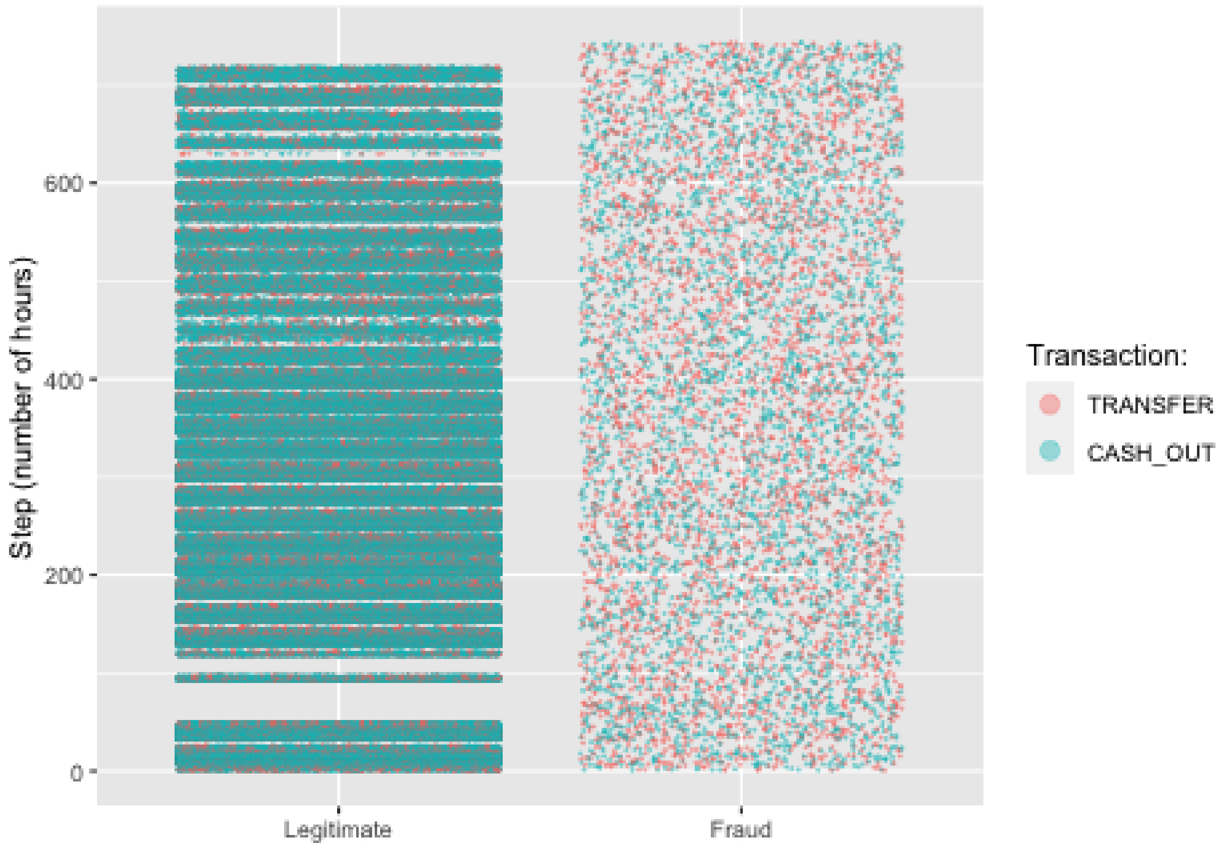


Figure 4: Dispersion of transactions over step (time)

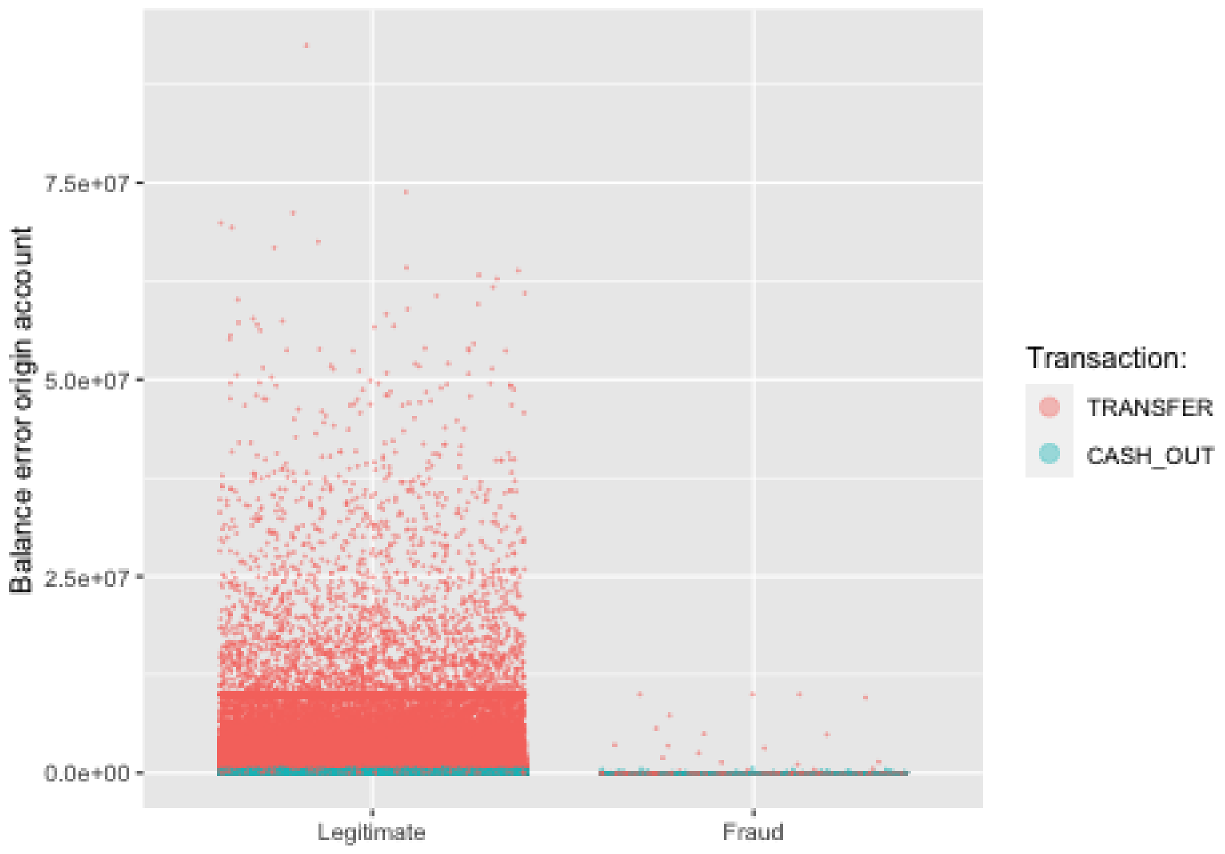


Figure 5: Balance error origin account

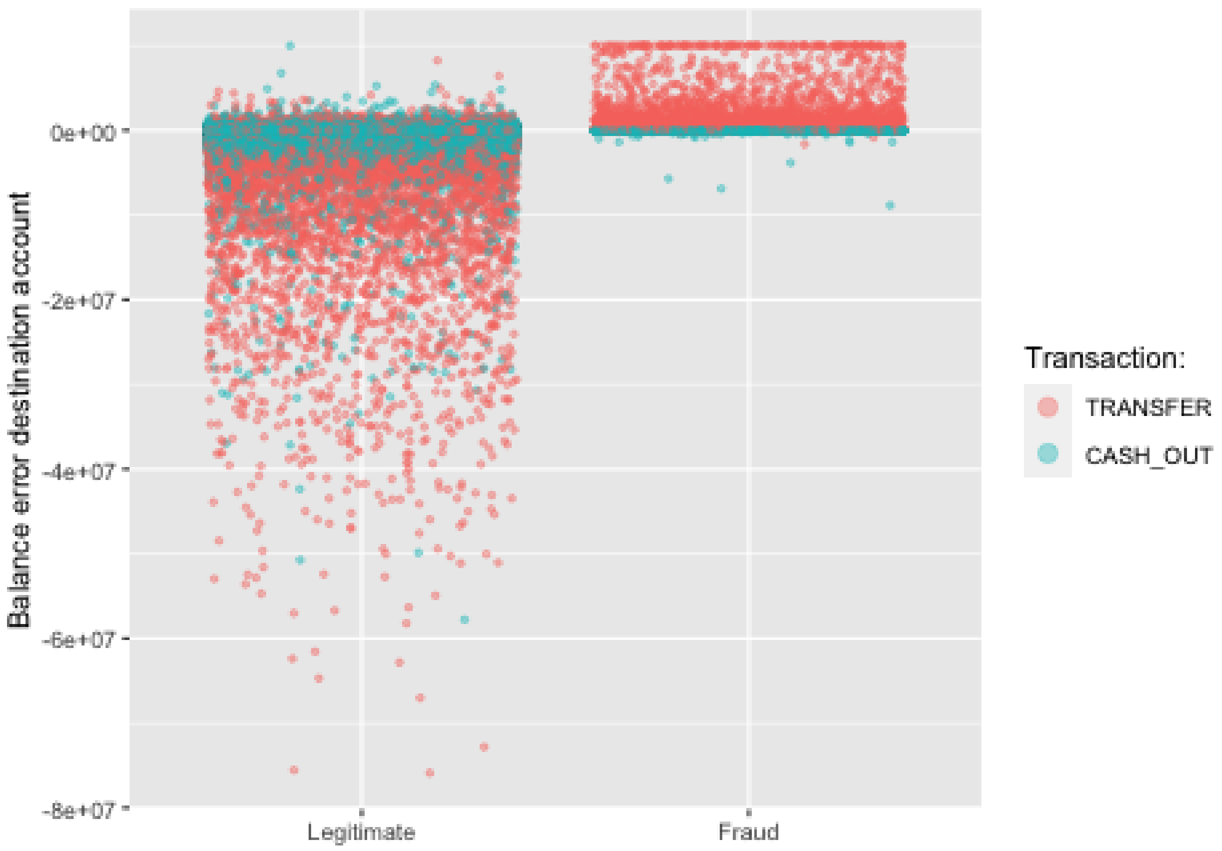


Figure 6: Balance error destination account

between the features. We will use a cluster heat map where the features are the rows and columns of a matrix. The higher values will receive a ‘hot’ color and lower values a ‘cold’ color. This hopefully will aid us in determining what features to focus on our machine learning model.

The package ComplexHeatmap provides great options to create customizeable heatmaps with many options. The figure *Heatmaps of correlations of transactions* display the level of correlation among the features. A correlation exposes the measure of dependence between two quantities, the most broadly used is the Pearson correlation. A correlation coefficient of 1 is a perfect direct linear relationship, and -1 represents a perfect inverse correlation. The closer the coeficient value is to either -1 or 1 the stronger the correlation between the features. We can observe this on the aforementioned heat map.

```
# compute the correlations ignoring the NA's: 'complete.obs'
correlation_nonfraud = cor(Xnonfraud[,names(Xnonfraud) != 'step'], use = 'complete.obs')
correlation_fraud = cor(Xfraud[,names(Xfraud) != 'step'], use = 'complete.obs')
```

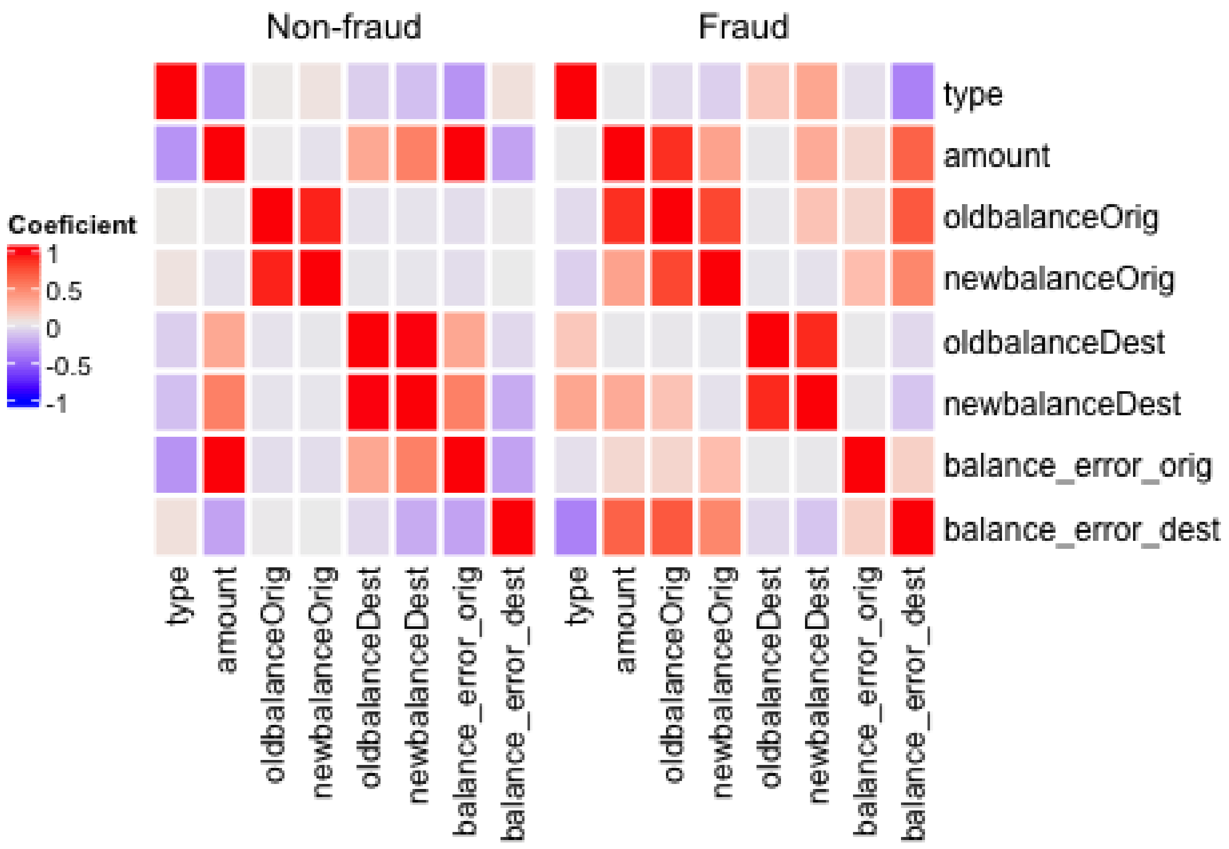


Figure 7: Heatmaps of correlation of transactions

The comparison of the heatmaps allows us to observe a stronger correlation between *balance_error_dest* and particularly the features *amount*, *oldbalanceOrig*, *newbalanceDest* on the Fraud cases!

Methodology

Data preparation

We will use the same observations from the data on both machine learning techniques to make predictions. Considering the fact that Support Vector Machines and eXtreme Gradient Boosting algorithms *look at* and *process* the data differently we will adjust the train set to fit each individual model's requirements so that we can achieve our goal. We will create a X, dataframe, that contains all features to be used during the training, testing and validation of the algorithm and Y, numeric vector, with the legend for all of these observations.

The distribution of classes in the dataset is extremely important. There is an extreme skew on the data, meaning that from our two possible classes (fraud and not-fraud) over 99% of the data belongs to the majority class (not-fraud). For the SVM we will use a downSample() approach to create proportionally distributed sets from both classes and enhance performance (by reducing the number of fraud transactions) by reducing the computational cost. On the other hand the XGBoost by nature is much more suited to handle imbalanced data sets. More details about the models in the following sections.

Validation data

We will select 10% of the data to create our validation set. This set will be used to verify the performance of our final model on completely unknown data. For this we will use the function createDataPartition() from the *Caret* package. We define which feature will be the *target feature*, in our case, the *isFraud* variable. The following parameters are *time* which determines the number of partitions to create; *proportion* will be our desired ratio: 10%, the *list* parameter set to FALSE determines that we want our outcome as a matrix with number of rows equal to *floor(p * length(y))* and 'times' columns. The output of createDataPartition is a logical vector for the validation set which we use to filter the whole dataset. The remaining set with 90% of the data can be filtered out by using its inverse index.

Train and Test data

Next, we will split the remaining 90 % of the PaySim data in *Train* and *Test* sets. These two sets will be used to train and test our model, the split will be respectively 80% / 20%. Also, using createDataPartition() with proportion set to 0.2.

We will explore two different approaches to solving our problem: *Support Vector Machines* from *package e1071* and *eXtreme Gradient Boosting* from the *XGBoost package*. Once we are satisfied with the performance of the models we will compare them them on our final hold-out-set: validation set.

Machine learning algorithms used for classification usually assume that there is a equal number of examples for each class. A great challenge with this dataset is the severe imbalance of its data. This means that the distribution of the observations is skewed towards the 'not fraud' (legitimate transaction) class. The number of fraudulent transactions is minuscule (NA) compared to immense number of legitimate transactions (NA). The ratio is about NA %.

The 3 main steps:

1. We will train (or learn) our models using the train data and provide its corresponding legends: *Fraud* (y == 1) and *not-Fraud* (y == 0). The positive class will be y == 1.
2. Next we test our model, providing the test data (from X) and comparing our model's predictions with the corresponding legends (actual legends) from Y. Here we have a chance to go back to step 1 (model training) and adjust parameters to try to achieve better prediction results. Once the results are satisfactory, we will move on to step 3. Since our model is making predictions on a binary classification problem, our prediction accuracy will be used to compare the models. If the results are not satisfactory on step 2 we return to step one with different parameters.
3. Finally we check the models performance on our final hold-out-set (validation set) to do a additional prediction test. Note that the validation data was not used during training of either model.

Table 4: Output of the downSample function

isFraud	n
0	8213
1	8213

Machine Learning Models

Support Vector Machines

Support Vector Machines from package 'e1071' is one of the most robust prediction methods and are based on statistical learning frameworks. Given a set of training examples labeled as one of two categories (fraud or not-fraud), it maps these examples as points in space so that the gap between these two categories is maximized. The new examples (test set) are then mapped to that same space and their categories are predicted based on what side of the gap between the categories the points fall.

In SVM, a data point is viewed as a p-dimensional vector (p is the number of features), the objective is to separate the data points using a (p -1)-dimensional hyperplane, in other words, be able to determine what class a *new* data point belongs to without having the label for the its class (the value of our *isFraud* variable).

Our first order of business will be to use the downSample() function from the Caret package. This *down-sampling* randomly samples the majority class (isFraud == 0) to be the same size as the second class (isFraud == 1), in this case the minority class. This greatly reduces the total number of observations, but is less computationally expensive. The number of examples from the output of the downSample() function can be seen on table *Output of the downSample function.* We can see that there is an equivalent number of observations for each class.

We will use to types of SVM kernels: Linear and Polynomial. The training will be done using the train_set and the testing the models accuracy using the test set. After separating the train and test set, we create our model by defining the target features and the predicting features in the formula = "Class ~ ." The *Class* is our desired output and "~." means use all features available ('amount', 'type', 'newbalanceOrig', 'oldbalanceOrig').

Our SVM model with linear kernel achieved this performance:

```
# Confusion matrix for the SVM linear model on test set
CM_SVM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1378  155
##          1  101 1324
##
##                Accuracy : 0.9135
##                  95% CI : (0.9027, 0.9233)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8269
##
##  Mcnemar's Test P-Value : 0.0009247
##
```

Table 5: SVM Results

model	kernel	data	accuracy	sensitivity	specificity
SVM	linear	test_set	0.9134550	0.8951995	0.8951995
SVM	polynomial	test_set	0.9675456	0.9729547	0.9729547
SVM	linear	validation_set	0.9069343	0.8819951	0.8819951
SVM	polynomial	validation_set	0.9628954	0.9708029	0.9708029

```
##             Sensitivity : 0.8952
##             Specificity : 0.9317
##          Pos Pred Value : 0.9291
##          Neg Pred Value : 0.8989
##              Prevalence : 0.5000
##          Detection Rate : 0.4476
##    Detection Prevalence : 0.4817
##       Balanced Accuracy : 0.9135
##
##        'Positive' Class : 1
##
```

SVM Results

The table *SVM Results* shows the results of the Linear and Polynomial SVM kernels when making predictions on the test set as well as on the validation set. The SVM kernels performed quite well in both test and validation sets, but the *Polynomial* kernel performed much better overall with accuracy of 0.9628954 on the validation set. The linear kernel performed with accuracy of 0.9069343. The table also includes information on the **sensitivity** (also known as Recall) which measures the ratio of actual positive cases that were correctly predicted and the **specificity** which displays the proportion of actual negative cases which were correctly predicted.

Confusion matrix for the predictions for the validation set:

```
# Confusion matrix for the SVM polynomial model on test set
CM_SVM_val_poly
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 785  24
##          1  37 798
##
##                Accuracy : 0.9629
##                  95% CI : (0.9526, 0.9715)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9258
##
##  Mcnemar's Test P-Value : 0.1244
##
```

```
##             Sensitivity : 0.9708
##             Specificity : 0.9550
##          Pos Pred Value : 0.9557
##          Neg Pred Value : 0.9703
##              Prevalence : 0.5000
##          Detection Rate : 0.4854
##    Detection Prevalence : 0.5079
##       Balanced Accuracy : 0.9629
##
##        'Positive' Class : 1
##
```

eXtreme Gradient Boosting (XGBoost)

XGBoost is an implementation of the Gradient Boosted Decision Trees algorithm. To solve our problem of classifying if a observation is fraud or not, XGBoost creates a set (an ensemble) of weak decision trees and combines these to train a strong learner model. We start the cycle by taking an existing model and computing the errors for every observation in the data. Next a new model is created to predict these errors. We add predictions from this error-predicting model to the "strong learner model". In order to make a prediction, we take all predictions from the previous models, compute the new errors from these predictions and build a next model. We then add it to the 'strong learner model'. So it is expected that the first predictions would be quite inaccurate, but as we train the model further it substantially improves the final models accuracy.

To understand the basic concept behind the Xgboost model, suppose we have K trees, the model can be defined as:

$$\sum_{k=1}^{K} f_k$$

where each f_k is the prediction from a decision tree. The model is composed of a collection (ensemble) of decision trees.

The training objective is accomplished by making a prediction based on all decision trees:

$$\hat{y} = \sum_{k=1}^{K} f_k(x_i)$$

where x_i is the feature vector for the i-th data point.

Similarly, the prediction at the t-th step can be defined as

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i)$$

The trees are composed of *internal nodes* which splits the data points by one of the features (the condition on the edge specifies what data can flow through) and *leaves* where data points which reach a leaf will receive a weight (the prediction).

Data preparation and value imputation

We observed that cases where isFraud is set to 1, for the most part also have *oldbalanceOrig* == 0 and *newbalanceOrig* == 0. We include a NA value on these cases to increase the separation between legitimate and fraudulent transactions. This was not useful for the SVM since it does not operate with observations with NA values (can't compute distances between a vector/data_point and a 'NA' object), but the XGBoost model does since one of the nodes of the tree could be based on a value or NA to choose a left or right branch on that tree. The larger the number of such distinctions on our data the higher the chances to make better predictions.

Confirming that all values have numeric format:

```r
# Get all the data types from the columns in the data using sapply() function
sapply(paysim, class)
```

```
##               step               type             amount     oldbalanceOrig
##          "numeric"          "numeric"          "numeric"          "numeric"
##     newbalanceOrig     oldbalanceDest     newbalanceDest balance_error_orig
##          "numeric"          "numeric"          "numeric"          "numeric"
## balance_error_dest            isFraud
##          "numeric"          "numeric"
```

Xgboost Parameters

The first parameter we need to set is the objective of the learning task. Since we have a single binary class (fraud or not-fraud) we will select the *binary:logistic*, which employs logistic regression and its output is the probability of the outcome. Output value larger than 0.5 is considered 1, and 0 otherwise. Secondly we define the evaluation method, *error* uses a binary classification error rate. It is calculated as #(wrong cases)/#(all cases).

For the predictions, the evaluation will regard the instances with prediction value larger than 0.5 as positive instances, and the others as negative instances. Next we set *max_depth* for the learning tree, increasing this value will make the model more complex and more likely to overfit, making the model perform extremely well on known data but pooly on new data. Training a deep tree will aggressively consume memory. Range $[0,\infty]$. The parameter *ETA*, also known as "learning_rate", is the step size shrinkage used in update to prevent overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative. The *colsample_bytree* is the subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed. Parameter *subsample* defines the ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees, this will prevent overfitting. Subsampling will occur once in every boosting iteration and its range: is [0,1].

The XGBoost has a cross-validation function that help us to fine tune the nrounds (numer of iterations) to achieve better results.

xgb.cv: Cross Validation function of xgboost

```
*data* takes an xgb.DMatrix, matrix, or dgCMatrix as the input.

*nfold* the original dataset is randomly partitioned into n-fold equal size subsamples.

*nrounds* the max number of iterations

*Maximize* If feval and early_stopping_rounds are set, then this parameter must be set as well. When it
```

```r
### XGB MODEL

# Converting train and test into xgb.DMatrix format
Dtrain <- xgb.DMatrix(
        data = as.matrix(train_set[, !names(train_set) %in% c('isFraud')]),
        label = train_set$isFraud)
Dtest <- xgb.DMatrix(
         data = as.matrix(test_set[, !names(test_set) %in% c('isFraud')]),
        label = test_set$isFraud)

# Model Building: XGBoost
param_list = list(
  objective = "binary:logistic",
  eval_metric = 'error',
  eta = 1,
  gamma = 1,
  max_depth = 3,
  subsample = 0.8,
  colsample_bytree = 0.5)

# # 5-fold cross-validation to
# # find optimal value of nrounds
# set.seed(1)  # Setting seed

# # Cross-validation to determine the best
# # parameter for nrounds
# xgbcv = xgb.cv(params = param_list,
#                 data = Dtrain,
#                 nrounds = 300,
#                 nfold = 5,
#                 print_every_n = 10,
#                 early_stopping_rounds = 10,
#                 metrics = list('error'),
#                 maximize = F)

# xgbcv

# In order to replicate the results here, you need to set the seed to 1.
set.seed(1, sample.kind = 'Rounding')
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
# Training XGBoost model at nrounds = 100
xgb_model = xgb.train(data = Dtrain,
                      params = param_list,
                      nrounds = 21)   # 21 best value from Cross-Valid
```

The preferred format for the input data is xgb.DMatrix, so we start out by converting our data to this format. Next we determine all the initial parameters and call the xgb.cv() function. With the nrounds defined we are finally able to train the model. After the training we can view the information by typing the name of the model on the console.

```r
# Display Xgb model info
xgb_model
```

```
## ##### xgb.Booster
## raw: 26.9 Kb
## call:
##   xgb.train(params = param_list, data = Dtrain, nrounds = 21)
## params (as set within xgb.train):
##   objective = "binary:logistic", eval_metric = "error", eta = "1", gamma = "1", max_depth = "3", subs
## xgb.attributes:
##   niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
## # of features: 9
## niter: 21
## nfeatures : 9
```

It is possible to visually inspect the importance of each feature by calling the *xgb.plot.importance()* function. The figure *XGBoost feature importance plot* has the output of this function.

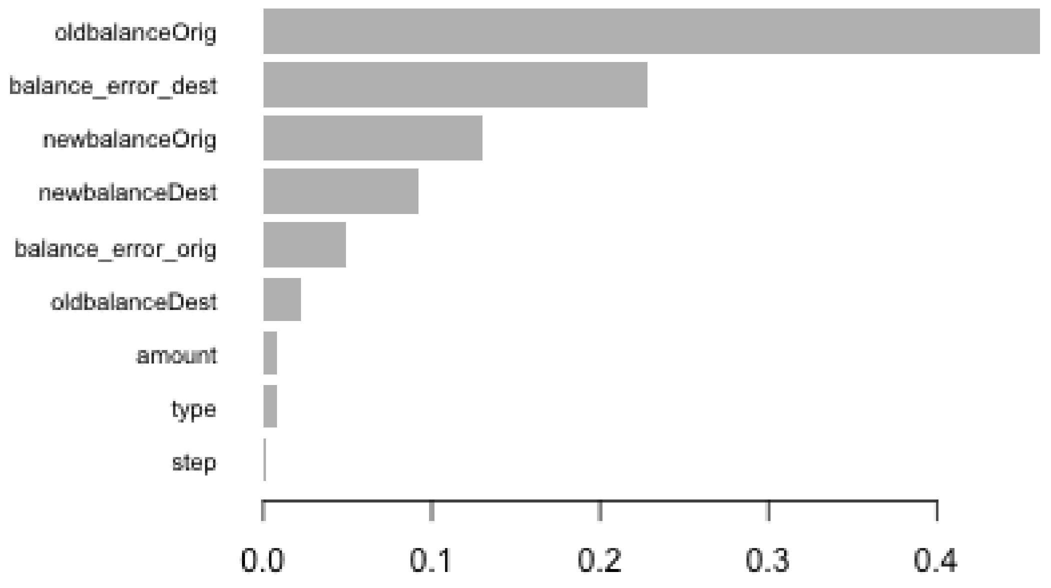


Figure 8: XGBoost feature importance plot

To make predictions we will use the *predict()* function with the model name and the data to use. We will make predictions for the test set and also for the validation set.

```
## [1] "Test error:  0.17647 %"
```

```
## [1] "Validation error 0.17037 %"
```

To measure the performance of the XGBoost model, we can compute the average error: "as.numeric(pred > 0.5)" applies the rule that if the probability is > 0.5, then the observation is classified as 1 and 0 otherwise. Use the "!=" to compare the vectors and the "mean()" to calculate the average error itself.

```r
mean(as.numeric(predictions > 0.5) != test_set$isFraud )
```

Table 6: XGBoost Results

model	kernel	data	accuracy	sensitivity	specificity
XGBoost	binary:logistic	validation_set	0.9982963	0.8009313	0.8009313
NA	NA	NA	NA	NA	NA

Table 7: SVM and XGBoost Results

model	kernel	data	accuracy	sensitivity	specificity
SVM	linear	test_set	0.9134550	0.8951995	0.8951995
SVM	polynomial	test_set	0.9675456	0.9729547	0.9729547
SVM	linear	validation_set	0.9069343	0.8819951	0.8819951
SVM	polynomial	validation_set	0.9628954	0.9708029	0.9708029
XGBoost	binary:logistic	test_set	0.9982353	0.7798697	0.7798697
XGBoost	binary:logistic	validation_set	0.9982963	0.8009313	0.8009313

The algorithm does not use the test_set data during the construction of the model. When making classifications we can do a regression to the *isFraud* label and use a threshold (0.5). We finally calculate the Confusion Matrix to evaluate the performance of the model.

XGBoost results

The final performance results for the XGBoost model can be seen on the table *XGBoost Results*.

```
## [1] "Table XGBoost Results"
```

Overall Results

No that we have downloaded, decompressed, loaded to the memory, explored, created visualizations, and applied two machine learning models, it is time to see all the results in one place. The table *SVM and XGBoost Results* brings it all together.

```
# display table with all results
all_results
```

Even though both models performed extremely well, the XGBoost ended up at the podium with flying colors. It is worth remembering that the SVM model was trained on a much smaller dataset since we down-sampled the majority class observations to match the number of the minority class and scored with accuracy on the validation data above 90%. The XGBoost algorithm actually was able to be trained using all the available training data without need to down-sample and performed extremely well.

Conclusion

The motivation behind choosing the PaySim dataset as starting point to this project was **to find a real world problem and try to solve it**. Up to this date Financial Crimes such as Fraud impacts Merchants and Consumers alike, since the first will eventually raise the cost of their products/services to cover financial losses. I am quite pleased with the results achieved here. As final remarks, it is useful to know that training models on datasets with millions of observations can take a long time to process and it helps a lot to reduce the size of the data to figure initial parameters and testing of how the different models work can be a time saver. Specially when dealing with large datasets and working on a single computer with i5 processor and 8GB RAM! This project has been a great joy and serious motivation to keep learning R programming and exploring new packages and resources. I look forward to exploring new machine learning challenges with other datasets for further develop my skill set.

Acknowledgment

I would like to thank Prof. Rafael A Irizarry for the great class content throughout all the modules: R Basics, Visualization, Probability, Inference and Modeling, Productivity Tools, Data Wrangling, Linear Regression, Machine Learning and, the last one, Data Science: Capstone. I also would like to thank my student peers during this amazing Data Science Professional Certificate journey.

References

- E. A. Lopez-Rojas, A. Elmir, and S. Axelsson.

  https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2584265/2017+IJSPM+EDL+Final+version.pdf?isAllowed=y&sequence=2 "PaySim: A financial mobile money simulator for fraud detection". In: The 28th European Modeling and Simulation Symposium-EMSS, Larnaca, Cyprus. 2016 Kaggle has featured PaySim1 as dataset of the week of april 2018. See the full article: http://blog.kaggle.com/2017/05/01/datasets-of-the-week-april-2017/

- Nathaniel Lee

  title: Credit card fraud will increase due to the Covid pandemic, experts warn accessed on March, 26th, 2021 @ 9PM https://www.cnbc.com/2021/01/27/credit-card-fraud-is-on-the-rise-due-to-covid-pandemic.html

- Max Kuhn et. al.

  Caret package https://cran.r-project.org/web/packages/caret/caret.pdf

- David Meyer et. al.

  e1071 package https://cran.r-project.org/web/packages/e1071/e1071.pdf

- Tianqi Chen et. al.

  https://cran.r-project.org/web/packages/xgboost/index.html

- Statistical tools for high-throughput data analysis

  title: ggplot2 axis scales and transformations http://www.sthda.com/english/wiki/ggplot2-axis-scales-and-transformations

- Kaggle Winning Solution Xgboost Algorithm

  https://www.slideshare.net/ShangxuanZhang/kaggle-winning-solution-xgboost-algorithm-let-us-learn-from-its-author?from_action=save