# Credit Fraud Transactions Detection with Random Forest

Hans Peter Ndeffo

**Professor - Jonathan Natov**

June 20, 2020

## Abstract

In this project we aim at detecting fraudulent transactions using random forest with the following steps:

- Import a credit card customers dataset. The dataset can be downloaded here:https://www.kaggle.com/mlg-ulb/creditcardfraud/download. Due to confidentiality issues the data has been transformed using a technic called PCA which stands for Principal Component Analysis.
- Display an overview of the bank dataset which has about 285000 rows.
- Setup Spark context for computations.
- Setup the random seed for sampling. The dataset will be split into 80% for training and 20% for testing.
- Use default R random forest algorithm and then the one coming from Spark ML libraries.
- Compare the computational time of the 2 procedure using a plot.
- Extract the important variables using Random Forest from Spark.
- Determine model effectivness and predicting power by computing training error and testing error.

## Introduction

Banks and other credit card issuers companies have had a hard time detecting, preventing and responding to frauds as the number of transactions increased. As a result, it has become extrement important for banks to detect those anomalies effectively and faster. Each year, according to the Nilson report, it cost about $35 billions.Identifying those frauds by human hands has become impossible. For example in the dataset we will use, there is an estimated 285000 transactions over a 2 day period due to the high number of transactions. It is in that optic that data science has become critical to help understand data trend and predict anomalies at a larger scale. While looking for the right algorithm to use, there are certain procedures to follow in order to remain consistent and precise in any analysis.The outcome of those procedures will determine which machine learning algorithm suit best the problem.

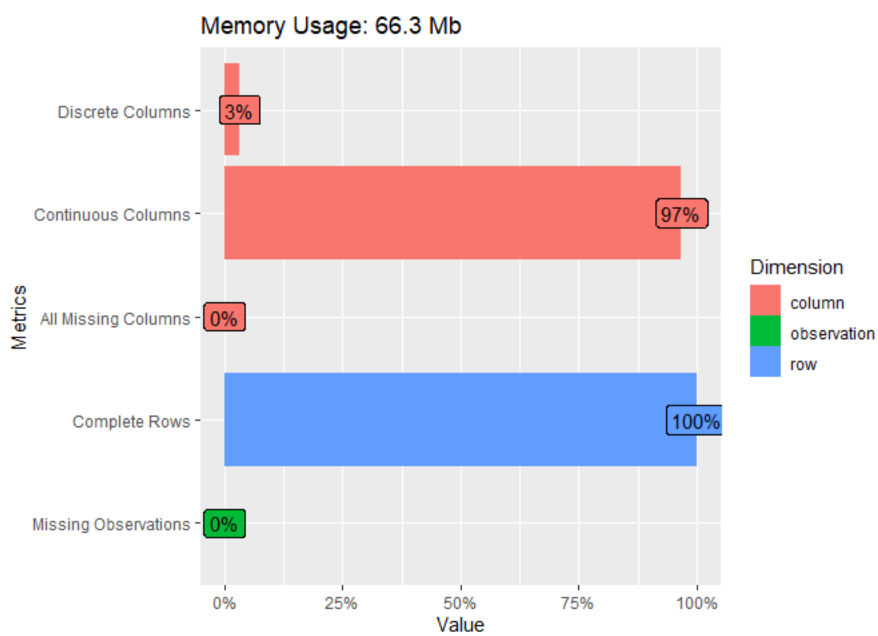
### Pre-requisites

# Step 1: Read customers dataset' file and load the required libraries

# Step 2: Show an overview and a sample of the dataset

Having an overview of the dataset structure is extremely important because this could show several characteristics. For example whether the dataset is balanced or imbalanced. A dataset is balanced if it follows these criterias. If there is n distinct preferably discrete output, Then the data should be evenly distributed among each output. In that case the probability p of any output i for example to occur is approximatively p $=\frac{1}{n}$. However, in this case the bank dataset we are currently using is most likely **imbalanced** because we assume in general that most credit card transactions are normal. This assumption will be verified through the steps below.

## The plot below count the missing values and display how many discrete and continuous columns.




Sample

| Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | V10 | V11 | V12 | V13 | V14 | V15 | V16 | V17 | V18 | V19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -1.3598071 | -0.07278117 | 2.5363467 | 1.3781552 | -0.33832077 | 0.46238778 | 0.23959855 | 0.09869790 | 0.3637870 | 0.09079417 | -0.5515995 | -0.61780086 | -0.9913898 | -0.3111694 | 1.4681770 | -0.4704005 | 0.2079712 | 0.02579058 | 0.4039930 |
| 0 | 1.1918571 | 0.26615071 | 0.1664801 | 0.4481541 | 0.06001765 | -0.08236081 | -0.07880298 | 0.08510165 | -0.2554251 | -0.16697441 | 1.6127267 | 1.06523531 | 0.4890950 | -0.1437723 | 0.6355581 | 0.4639170 | -0.1148047 | -0.18336127 | -0.1457830 |
| 1 | -1.3583541 | -1.34016307 | 1.7732093 | 0.3797796 | -0.50319813 | 1.80049938 | 0.79146096 | 0.24767579 | -1.5146543 | 0.20764287 | 0.6245015 | 0.06608369 | 0.7172927 | -0.1659459 | 2.3458649 | -2.8900832 | 1.1099694 | -0.12135931 | -2.2618571 |
| 1 | -0.9662717 | -0.18522601 | 1.7929933 | -0.8632913 | -0.01030888 | 1.24720317 | 0.23760894 | 0.37743587 | -1.3870241 | -0.05495192 | -0.2264873 | 0.17822823 | 0.5077569 | -0.2879237 | -0.6314181 | -1.0596472 | -0.6840928 | 1.96577500 | -1.2326220 |
| 2 | -1.1582331 | 0.87773675 | 1.5487178 | 0.4030339 | -0.40719338 | 0.09592146 | 0.59294075 | -0.27053268 | 0.8177393 | 0.75307443 | -0.8228429 | 0.53819555 | 1.3458516 | -1.1196698 | 0.1751211 | -0.4514492 | -0.2370332 | -0.03819479 | 0.8034869 |

# Step 3: Seting up Apache Spark

Setting up Spark follows the substeps below:

- Create a spark config variable that will contain all the details about how ressources we want Spark to use henceforth
- Since I am using Spark locally I am able to allocate the number of cores and how much RAM will be used

```
conf <- spark_config()
conf$`sparklyr.cores.local` <- 4
conf$`sparklyr.shell.driver-memory` <- "4G"
```

- Then create and connect a Spark cluster that will have the config above with this command

```
##    user  system elapsed
##    0.15    0.11    8.01
```

Copy the dataset into the Spark Cluster for faster computations and create a reference to it locally

## Count the number of values with 0 and 1 as output using dplyr

Count data with output 0.

```
zero_count <- count(customer_data%>%
  select(Class)%>%
  filter(Class == 0)%>%
  collect())
```

Count data with output 1.

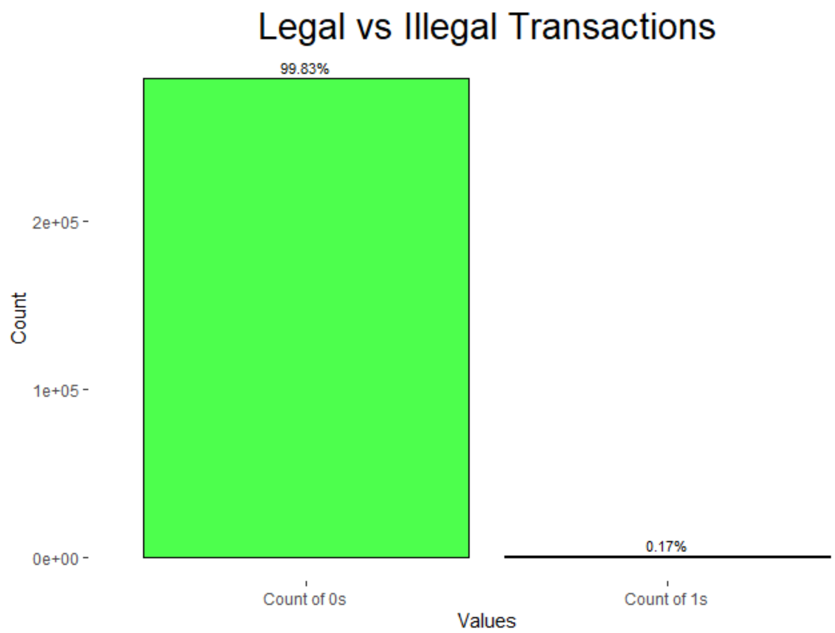
```
one_count <- count(customer_data%>%
  select(Class)%>%
  filter(Class == 1)%>%
  collect())
```

## Show the distribution of data for each output

Class column will be the outcome has 2 distinct and discrete values which are 0 and 1. **0** stand for **normal** and **1** for **fraudulent transaction**. The **timesteps** in this dataset is likely to be represented **in seconds(s)**.

The code below will show the distribution of data among each outcome(0 and 1)

```
## No summary function supplied, defaulting to `mean_se()`
```




## Step 4: Split the dataset into 80% training and 20% testing

```
#80% train sample
train_sample = sample(1:n, floor(n*0.80))
train_data = bank_customers[train_sample, ]
test_data = bank_customers[-train_sample, ]
```

Copy the training dataset into the Spark Cluster for faster computations and create a reference to it locally

```
spark_train_data <- copy_to(sc, train_data)
```

Copy the test dataset into the Spark Cluster for faster computations and create a reference to it locally

```
spark_test_data <- copy_to(sc, test_data)
```

## Step 5: Train the model using Spark and default R ML libraries

## Step 6: Determine important variables or columns

The important features are listed below although it is impossible to determine exactly what they are or mean.

```
ml_feature_importances(rf_spark)
```

```
##    feature   importance
## 1      V17 0.1542470088
## 2      V12 0.1304192321
## 3      V16 0.1191520262
```

```
## 4       V7 0.1056314695
## 5      V10 0.0910974353
## 6       V9 0.0581049076
## 7       V4 0.0553595165
## 8      V14 0.0547563195
## 9      V11 0.0538219815
## 10      V3 0.0429298746
## 11     V18 0.0223350475
## 12     V21 0.0181520431
## 13      V2 0.0130084585
## 14     V27 0.0088154497
## 15     V26 0.0086772489
## 16      V8 0.0082439259
## 17    Time 0.0069234740
## 18      V6 0.0067006526
## 19     V20 0.0060866532
## 20      V5 0.0059132389
## 21  Amount 0.0045336904
## 22     V13 0.0041094502
## 23     V15 0.0034883972
## 24     V28 0.0034804155
## 25     V22 0.0031073659
## 26     V24 0.0028237096
## 27      V1 0.0027956189
## 28     V25 0.0022475290
## 29     V19 0.0020848742
## 30     V23 0.0009529851
```

6 Majors factors can actually help perform anomaly detection relatively accurately. The percentage may slightly vary depending on the sampling used for training. The important features are:

- **v17** counts for about **15.34%**
- **v12** use by the customer counts for **15.03%**
- **v16** influence the model for about **11.206%**
- **v10** influence the model for about **10.25%**
- **v9** influence the model for about **8.29%**
- **v7** influence the model for about **7.83%**

# Steps 7: Determining the model effectiveness by computing the testing error

Predict and collect data from Spark clusters and create a reference to the predicted data locally

```
test_predicted <- ml_predict(rf_spark,spark_test_data)%>%
  collect()
```

## Computing the testing error

```
rdmFor_testing_error <- sum(test_predicted$prediction!= test_data$Class)/nrow(test_data)
rdmFor_testing_error
```

```
## [1] 0.0006495558
```

The **testing error** is about **0.05%**. This means the **accuracy** is **99.95%**.

Close Spark connection

```
spark_disconnect(sc)
```

## Conclusion

In this project, we were successfully able to identify fraudulent transactions with an **accuracy of 99.95%**.Again it is hard to discuss why there is **0.05%** based on features because they are anynomous. However, a further analysis of the structure of the dataset using different Machine learning algorithms might help reduce that error.