Harvard Data Scientist Capstone project Credit Fraud

Elin Brännström

26 May 2020

Introduction

Mastering the skill of predicting fradulent transactions has high potential in form of savings for every type of business. Although fraud detection is a common machine learning problem, it is among the more difficult ones since often the rate of fraudulent transactions is negligible in size compared to non-fraudulent transactions, turning the problem into finding a needle in a haystack. Modelling also needs to be done with extreme sensitivity to avoid classifying non-fraudulent transactions as fraudulent, leading to decreased customer satisfaction.

This report will present a machine learning approach to predict credit fraud. The data set used is a Synthetic Financial Fraud Detection found on Kaggle. The data set contains a binary variable for fraud / non fraud, which will be the target variable for predictions. Furthermore, the data set contains detailed information about the transactions, such as transaction type, time of transaction, receiver & sender, amount of transaction, initial balance & balance after transaction for receiver and sender. The report will firstly attend to examining and cleaning of data, then proceed to splitting data for training, testing & validation and finally to modelling and results. As a last step, conclusions and lessons learned will be discussed.

The data set can be found here: https://www.kaggle.com/ntnu-testimon/paysim1

Note that TinyTex is a necessary library for knitting to PDF.

Method & analysis

Data import

As a first step, the data set is imported into R from the link provided above, using the following line of code.

```
#Import necessary libraries and data set
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")

library(tidyverse)
library(caret)

#Reading data set from Dropbox repository. This could take a few minutes
credit_fraud <- read.csv("https://www.dropbox.com/s/mz93hln4efohpn4/credit_fraud.csv?dl=1")
head(credit_fraud)
```

```
##   step     type   amount    nameOrig oldbalanceOrg newbalanceOrig    nameDest
## 1    1  PAYMENT  9839.64 C1231006815        170136      160296.36 M1979787155
## 2    1  PAYMENT  1864.28 C1666544295         21249       19384.72 M2044282225
## 3    1 TRANSFER   181.00 C1305486145           181           0.00  C553264065
```

```
## 4    1 CASH_OUT   181.00  C840083671           181          0.00   C38997010
## 5    1  PAYMENT 11668.14 C2048537720         41554      29885.86 M1230701703
## 6    1  PAYMENT  7817.71   C90045638         53860      46042.29  M573487274
##   oldbalanceDest newbalanceDest isFraud isFlaggedFraud
## 1              0              0       0              0
## 2              0              0       0              0
## 3              0              0       1              0
## 4          21182              0       1              0
## 5              0              0       0              0
## 6              0              0       0              0
```

Initial data preparations

Checking the heading of the data, we start with examining how many of the rows in the "isFlagged"-column that is flagged as fraud percentage wise. Based on this, and the explanation from the source page, we will remove this variable from the data set since it does not provide enough information to be useful for modelling.

```
#Removing the variable "isFlagged" since it only
#contains 16 rows with values and is a small fraction
#of the total dataset
nr_isFlagged <- nrow(credit_fraud[credit_fraud$isFlaggedFraud==1,])
fraction_isFlagged <-nrow(credit_fraud[credit_fraud$isFlaggedFraud==1,]) / nrow(credit_fraud)
print(nr_isFlagged)
```

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

```
print(fraction_isFlagged)
```

```
## [1] 2.514687e-06
```

```
credit_fraud_new <- subset(credit_fraud, select=-isFlaggedFraud)
```

As a next step number of NaN's in the data set is examined, only to find there are no missing values.

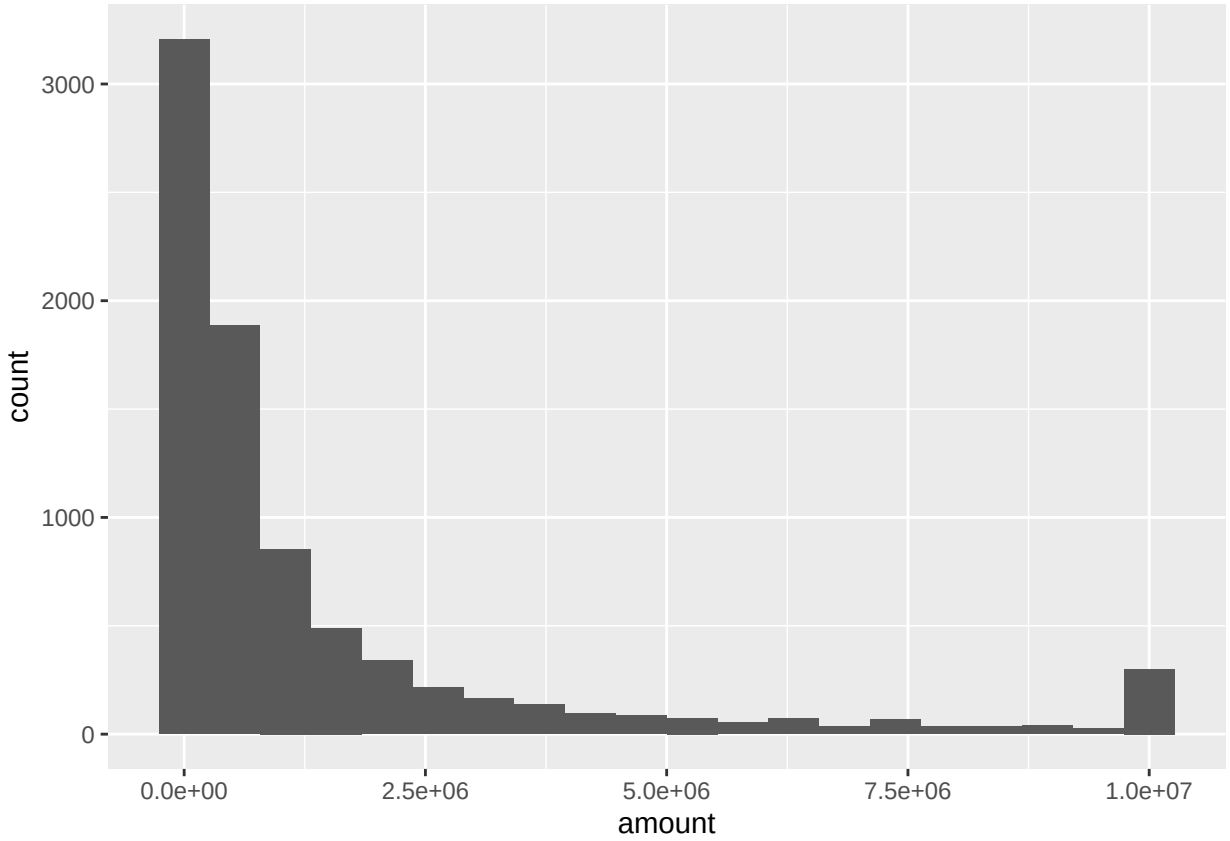
```
#Checking for missing values
sum(is.na(credit_fraud_new))
```

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

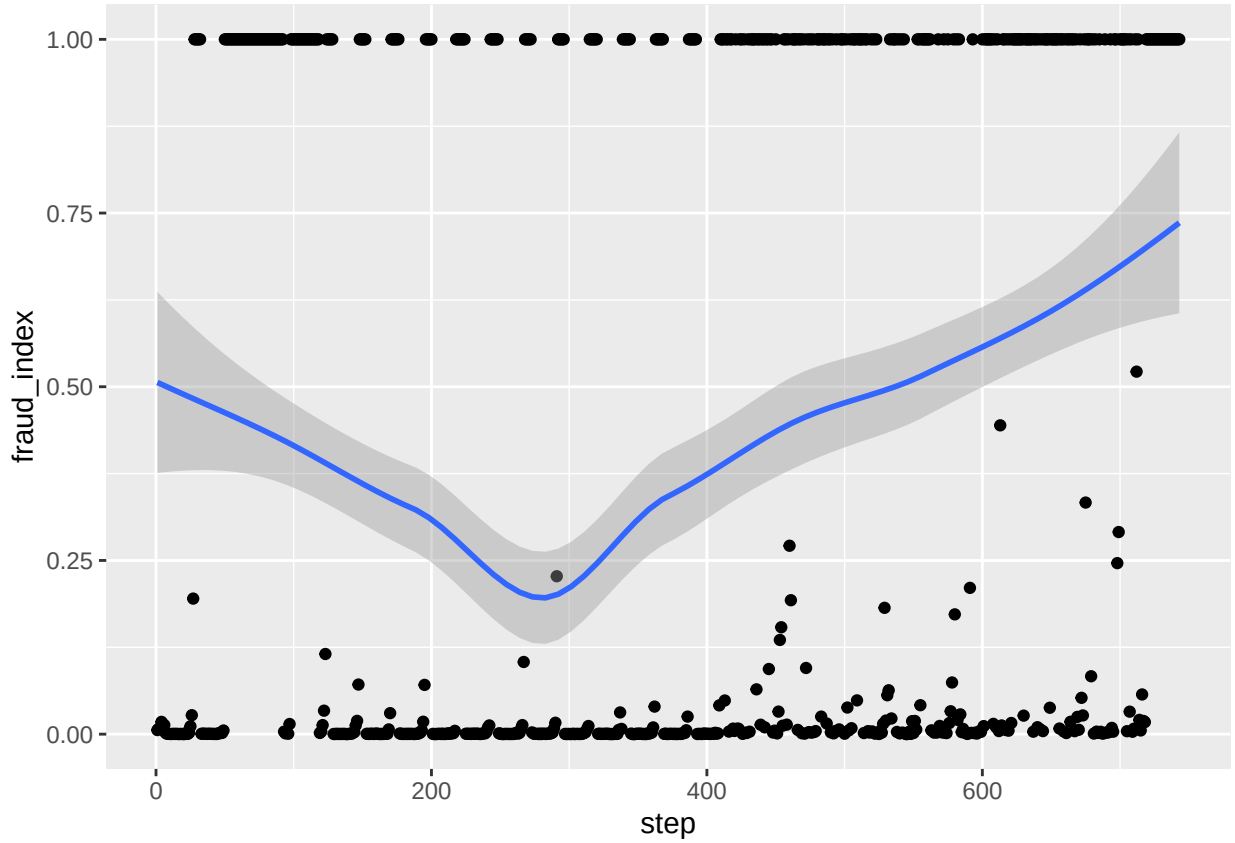
Data visualization

Next step is to visualize some of the data to get a better understanding of some of the variables.

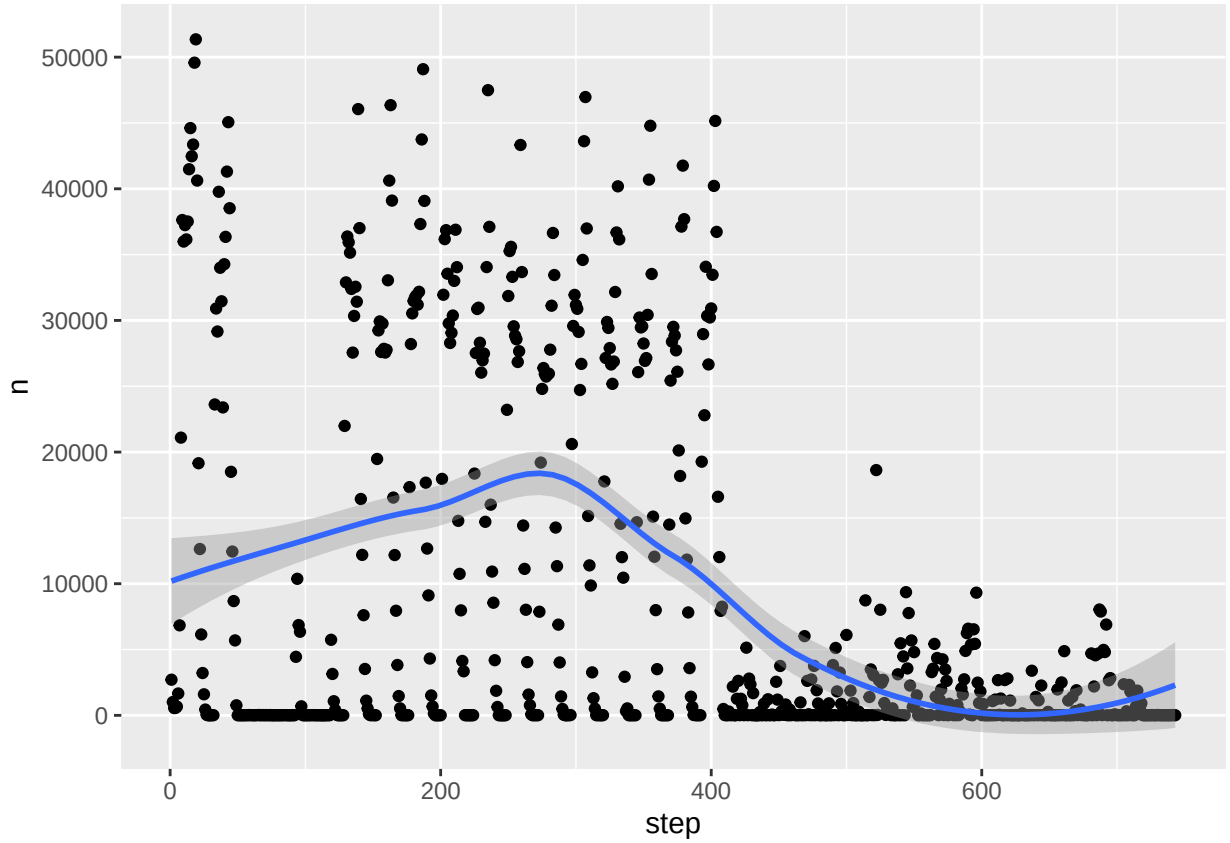
Firstly, look at amounts of fraud and what types of fraud are most common. The graph helps us conclude small amounts of fraud are the most common.



Next we will examine the "step"-variable describing time, and whether or not there is a time dependency connected to fraud. The graph describes proportion of frauds per time step, and we can clearly see a large time variability with some time steps only containing frauds, and other steps without any frauds.



We can also examine number of fraudulent transactions per time step. Interestingly, the number of fraudulent transactions peaks at the same step as the rate of fraudulent transactions is at a minimum. We can also see a comparatively small amount of fraudulent transactions at the step of which the rate of fraud is higher. This implies there is a time-dependency of fraud, which might be explained by non-fraudulent transactions decreasing more at night, compared to fraudulent transactions, explaining the two curves. We will keep the step variable for modelling since it contains valuable information.



Two of the columns contains ID of the customer initiating the transaction and of the customer receiving the transaction. Checking number of unique IDs in the data, one can conclude that the variable nameOrig has 99.9 % unique values and that nameDest has 42.8 % unique values. In this form, with many unique values, these variables are not useful for making predictions. However, every ID starts with a letter that might be useful for predictions. Looking at number of unique starting letters, we see only one value for nameOrig, hence this variable is deleted because it does not contain any valuable information. For nameDest there are two different letters, and this variable is therefore kept.

```
#Checking number of unique values for "nameOrig" and "nameDest"
nr_unique_nameOrig <- length(unique(credit_fraud_new$nameOrig))/dim(credit_fraud)[1]
nr_unique_nameDest <- length(unique(credit_fraud_new$nameDest))/dim(credit_fraud)[1]

print(nr_unique_nameOrig)
```

```
## [1] 0.9985363
```

```
print(nr_unique_nameDest)
```

```
## [1] 0.4278681
```

```
#Replace the variables with first letter of every row
credit_fraud_new <- credit_fraud_new %>% mutate(nameOrigLetter = str_sub(nameOrig,1,1)) %>%
  mutate(nameDestLetter = str_sub(nameDest,1,1)) %>%
  select(-nameOrig,-nameDest)
```

```r
#Check unique values again and remove "nameDestLetter" since it
#only contains one value
length(unique(credit_fraud_new$nameOrigLetter))
```

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

```r
credit_fraud_new <- credit_fraud_new %>% select(-nameOrigLetter)

length(unique(credit_fraud_new$nameDestLetter))
```

```
## [1] 2
```

Now let us move on to columns with numeric variables. For this analysis I will temporarily remove other columns. Notice a strong correlation between oldbalanceDest & newbalanceDest and newbalanceOrig & oldbalanceOrig. Columns with strong correlations are removed, since these only contain redundant data which can cause storage problems and potentially overfitting.

```r
#Checking correlation between numeric variables
test_corr <- credit_fraud_new %>% select(-nameDestLetter,-type)
cor(test_corr)
```

```
##                      step       amount oldbalanceOrg newbalanceOrig
## step            1.00000000  0.022372995  -0.010058378   -0.010299037
## amount          0.02237299  1.000000000  -0.002762475   -0.007860925
## oldbalanceOrg  -0.01005838 -0.002762475   1.000000000    0.998802763
## newbalanceOrig -0.01029904 -0.007860925   0.998802763    1.000000000
## oldbalanceDest  0.02766536  0.294137450   0.066242501    0.067811518
## newbalanceDest  0.02588818  0.459304267   0.042028619    0.041837497
## isFraud         0.03157757  0.076688429   0.010154422   -0.008148161
##                oldbalanceDest newbalanceDest       isFraud
## step              0.027665360   0.0258881757  0.0315775686
## amount            0.294137450   0.4593042673  0.0766884288
## oldbalanceOrg     0.066242501   0.0420286188  0.0101544219
## newbalanceOrig    0.067811518   0.0418374971 -0.0081481613
## oldbalanceDest    1.000000000   0.9765685054 -0.0058852782
## newbalanceDest    0.976568505   1.0000000000  0.0005353471
## isFraud          -0.005885278   0.0005353471  1.0000000000
```

```r
# Find which columns to remove
findCorrelation(cor(test_corr),cutoff = .7, verbose = TRUE, names = TRUE, exact = TRUE)
```

```
## Compare row 6  and column  5 with corr  0.977
##   Means:  0.258 vs 0.137 so flagging column 6
## Compare row 4  and column  3 with corr  0.999
##   Means:  0.219 vs 0.088 so flagging column 4
## All correlations <= 0.7
```

```
## [1] "newbalanceDest" "newbalanceOrig"
```

```
#Remove columns
credit_fraud_new <- credit_fraud_new %>% select(-newbalanceOrig,-newbalanceDest)
```

Now, let us move on to examining the fraudulent cases. Notice how fraud only occurs for the types "CASH_OUT" & "TRANSFER". Fraud also only occurs for "C" as "nameDestLetter" and at a maximum of 10 million. Filtering for these conditions will help us narrow down the data set to only contain rows for which fraud can occur.

```
# Summary of only fraudulent cases
only_fraud <- credit_fraud_new %>% filter(isFraud==1)
summary(only_fraud)
```

```
##       step             type          amount         oldbalanceOrg
##  Min.   :  1.0   CASH_IN :   0   Min.   :       0   Min.   :       0
##  1st Qu.:181.0   CASH_OUT:4116   1st Qu.:  127091   1st Qu.:  125822
##  Median :367.0   DEBIT   :   0   Median :  441423   Median :  438983
##  Mean   :368.4   PAYMENT :   0   Mean   : 1467967   Mean   : 1649668
##  3rd Qu.:558.0   TRANSFER:4097   3rd Qu.: 1517771   3rd Qu.: 1517771
##  Max.   :743.0                   Max.   :10000000   Max.   :59585040
##  oldbalanceDest         isFraud  nameDestLetter
##  Min.   :        0   Min.   :1   Length:8213
##  1st Qu.:        0   1st Qu.:1   Class :character
##  Median :        0   Median :1   Mode  :character
##  Mean   :   544250   Mean   :1
##  3rd Qu.:   147829   3rd Qu.:1
##  Max.   :236230517   Max.   :1
```

```
# Filter data set based on summary of fraudulent cases
credit_fraud_new <- credit_fraud_new %>% filter(nameDestLetter == "C") %>%
  filter(amount <= 10000000) %>% filter(type %in% c("CASH_OUT", "TRANSFER")) %>%
  select(-nameDestLetter)

# Size of reduced data set compared to original data set
dim(credit_fraud_new)[1]/dim(credit_fraud)[1]
```

```
## [1] 0.4350356
```

Data partitioning

At this point, I originally proceeded to data partitioninig and modelling. However, among the first model I tried to run was a knn-model, which took over 24 hours to run on my local machine. As a result of this, I have decided to reduce the data set even further to gain flexibility and time when modelling. Ideally, more data should have been kept for training the model, however due to computational restrictions I have decided to proceed with under-sampling my data[1]. This has the benefits of both heavily reducing the data set, as well as balancing the data into equal parts of fraud and non-fraud, which will help train models for fraud. I will randomly pick a subset of non-fraudulent transactions having the same size as the fraudulent transactions which will make up the data set. Observe that ideally, several random subsets should be picked and the model should be trained on every subset, to include as much information as possible.

```r
# Forming an under sampled data set containing 50/50 percent fraudulent cases
# and non-fraudulent cases, keeping all fradulent cases and picking a
#subset of non-fraudulent cases of the same size
number_of_frauds <- sum(credit_fraud_new$isFraud==1)
only_fraud <- credit_fraud_new[which(credit_fraud_new$isFraud==1),]
only_no_fraud <- credit_fraud_new[which(credit_fraud_new$isFraud==0),]

set.seed(1,sample.kind="Rounding")
under_sampling_no_fraud <- sample_n(only_no_fraud, number_of_frauds)
under_sampled_set <- bind_rows(under_sampling_no_fraud, only_fraud)
```

Now let us partition the under sampled data for training set, test set and validation set. I choose to use 10 % of the data for validation, which will be held out until the very end, to validate the final model on. The reason for this is to get an un-biased evaluation of the model, and picking only 10 % of the data still allows for most of the data to be used for training. I choose to partition the remaining data for 80 % training and 20 % testing, keeping the majority for training to achieve as good a model as possible. I also change class of integers for modelling purposes.

```r
# Replacing data set with under sampled one. Overwriting the name to be able to reuse code
credit_fraud_new <- under_sampled_set

#Changing integers to factors
credit_fraud_new$type <- as.factor(credit_fraud_new$type)
credit_fraud_new$isFraud <- as.factor(credit_fraud_new$isFraud)

#Set seed for reproducibility
set.seed(1,sample.kind="Rounding")

# Create data partition, using 10 % for validation,
#and 20 % of the remaining data for testing.
#Remaining part is used for training.
test_index <- createDataPartition(y = credit_fraud_new$isFraud, times=1, p=0.1, list=FALSE)
validation_set <- credit_fraud_new[test_index,]
test_set <- credit_fraud_new[-test_index,]

set.seed(1,sample.kind="Rounding")
test_index <- createDataPartition(y = test_set$isFraud, times=1, p=0.2, list=FALSE)
training_set <- credit_fraud_new[-test_index,]
test_set <- credit_fraud_new[test_index,]
```

Modelling

Due to the heavy imbalance of the original data being skewed to many more non-fraudulent transactions compared to fraudulent transactions, we will first examine the accuracy of just predicting all transactions as non-fraudulent. By doing this, we would predict 99.9 % of all transactions in the data correctly. Even though this is a very good result, this baseline model is defeating the purpose of modelling for fraud, and also gives a misleading accuracy due to not considering size of the classes. When modelling on heavily imbalanced data sets a better measurement would be the "balanced accuracy" weighing accuracy based on size of the class.

```r
#Accuracy when predicting all transactions in original data as legal
fraud_as_numeric <- as.numeric(as.character(credit_fraud$isFraud))
accuracy_all_legal <- 1 - sum(fraud_as_numeric)/dim(credit_fraud)[1]
print(accuracy_all_legal)
```

```
## [1] 0.9987092
```

Proceeding with modelling on the under-sampled data, I will start with defining a control parameter to use for all models to control the computational nuances.

```
#Setting control parameter for modelling
control <- trainControl(method = "cv", number = 10, p = .9)
```

Starting simple with a regression (glm) model gives the following result. I use the measure "Accuracy" for comparing models, since the undersampled data set is balanced in classes this measure is sufficient to use.

```
#Training glm model
train_glm <- train(isFraud ~ ., method = "glm",
                   data = training_set,
                   trControl = control)

#Predict onto test set and display confusion matrix and accuracy
glm_pred <- predict(train_glm, test_set)
cm_glm <- confusionMatrix(glm_pred, test_set$isFraud)

print(cm_glm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1515  208
##          1  147 1088
##
##                Accuracy : 0.88
##                  95% CI : (0.8677, 0.8915)
##     No Information Rate : 0.5619
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.755
##
##  Mcnemar's Test P-Value : 0.00145
##
##             Sensitivity : 0.9116
##             Specificity : 0.8395
##          Pos Pred Value : 0.8793
##          Neg Pred Value : 0.8810
##              Prevalence : 0.5619
##          Detection Rate : 0.5122
##    Detection Prevalence : 0.5825
##       Balanced Accuracy : 0.8755
##
##        'Positive' Class : 0
##
```

```
glm_accuracy <- cm_glm$overall['Accuracy']

accuracy_results <- data_frame(method= "GLM", ACC=glm_accuracy)
accuracy_results %>% knitr::kable()
```

method	ACC
GLM	0.8799865

Next I will try a more complex knn model evaluating if there is a cluster effect in the data. Let us examine if this gives a better result.

```
#Train knn model
train_knn <- train(isFraud ~ ., method = "knn",
                   data = training_set,
                   trControl = control)

#Predict onto test set and display confusion matrix and accuracy
knn_pred <- predict(train_knn, test_set)
cm_knn <- confusionMatrix(knn_pred, test_set$isFraud)

print(cm_knn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1567   38
##          1   95 1258
##
##                Accuracy : 0.955
##                  95% CI : (0.9469, 0.9622)
##     No Information Rate : 0.5619
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9091
##
##  Mcnemar's Test P-Value : 1.199e-06
##
##             Sensitivity : 0.9428
##             Specificity : 0.9707
##          Pos Pred Value : 0.9763
##          Neg Pred Value : 0.9298
##              Prevalence : 0.5619
##          Detection Rate : 0.5297
##    Detection Prevalence : 0.5426
##       Balanced Accuracy : 0.9568
##
##        'Positive' Class : 0
##
```

```r
knn_accuracy <- cm_knn$overall['Accuracy']

accuracy_results <- bind_rows(accuracy_results, tibble(method="KNN", ACC=knn_accuracy))
accuracy_results %>% knitr::kable()
```

method	ACC
GLM	0.8799865
KNN	0.9550372

Next step is to evaluate a Random Forest model, which is a collection of decision trees working in a rule-based matter. I will start with the standard Random Forest model.

```r
#Train Random Forest model
train_rf <- train(isFraud ~ ., method = "rf",
                  data = training_set,
                  trControl = control)

#Predict onto test set and display confusion matrix
rf_pred <- predict(train_rf, test_set)
cm_rf <- confusionMatrix(rf_pred, test_set$isFraud)

print(cm_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1613    8
##          1   49 1288
##
##                Accuracy : 0.9807
##                  95% CI : (0.9751, 0.9854)
##     No Information Rate : 0.5619
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.961
##
##  Mcnemar's Test P-Value : 1.17e-07
##
##             Sensitivity : 0.9705
##             Specificity : 0.9938
##          Pos Pred Value : 0.9951
##          Neg Pred Value : 0.9634
##              Prevalence : 0.5619
##          Detection Rate : 0.5453
##    Detection Prevalence : 0.5480
##       Balanced Accuracy : 0.9822
##
##        'Positive' Class : 0
##
```

```
rf_accuracy <- cm_rf$overall['Accuracy']

accuracy_results <- bind_rows(accuracy_results, tibble(method="RF", ACC=rf_accuracy))
accuracy_results %>% knitr::kable()
```

method	ACC
GLM	0.8799865
KNN	0.9550372
RF	0.9807302

Next we will test a Support Vector Machine with Radial kernel, to see if the two classes can be separated on a hyperplane by a non-linear boundary.

```
#Train Support Vector Machine model
train_svm <- train(isFraud ~ ., method = "svmRadial",
                   data = training_set,
                   trControl = control)

#Predict onto test set and display confusion matrix
svm_pred <- predict(train_svm, test_set)
cm_svm <- confusionMatrix(svm_pred, test_set$isFraud)

print(cm_svm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1505   86
##          1  157 1210
##
##                Accuracy : 0.9178
##                  95% CI : (0.9074, 0.9275)
##     No Information Rate : 0.5619
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8341
##
##  Mcnemar's Test P-Value : 7.106e-06
##
##             Sensitivity : 0.9055
##             Specificity : 0.9336
##          Pos Pred Value : 0.9459
##          Neg Pred Value : 0.8851
##              Prevalence : 0.5619
##          Detection Rate : 0.5088
##    Detection Prevalence : 0.5379
##       Balanced Accuracy : 0.9196
##
##        'Positive' Class : 0
##
```

```r
svm_accuracy <- cm_svm$overall['Accuracy']

accuracy_results <- bind_rows(accuracy_results, tibble(method="SVM", ACC=svm_accuracy))
accuracy_results %>% knitr::kable()
```

method	ACC
GLM	0.8799865
KNN	0.9550372
RF	0.9807302
SVM	0.9178499

Examining the results from the different tested models above, a random forest performs the best and will be the deployed model. But before going to the next step of unlocking the validation set, let us see if we can tune the parameters to reach an even better performance of the random forest. I will involve a cross-validation for the "mtry" parameter which controls the number of variables available at each tree node. By default this is set to the square root of predictors available, which is 3 in my case. I will therefore cross-validate for values ranging between 1 and 5, to examine if a better model can be developed.

```r
#Define tuning grid for the mtry parameter
tunegrid <- expand.grid(.mtry=c(1:5))
#Train Random Forest model
train_rf_crossv <- train(isFraud ~ ., method = "rf",
                         tuneGrid = tunegrid,
                         data = training_set,
                         trControl = control)

#Predict onto test set and display confusion matrix
rf_crossv_pred <- predict(train_rf_crossv, test_set)
cm_rf_crossv <- confusionMatrix(rf_crossv_pred, test_set$isFraud)

print(cm_rf_crossv)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1607    9
##          1   55 1287
##
##                Accuracy : 0.9784
##                  95% CI : (0.9725, 0.9833)
##     No Information Rate : 0.5619
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9562
##
##  Mcnemar's Test P-Value : 1.855e-08
##
##             Sensitivity : 0.9669
##             Specificity : 0.9931
##          Pos Pred Value : 0.9944
```

```
##          Neg Pred Value : 0.9590
##              Prevalence : 0.5619
##          Detection Rate : 0.5433
##    Detection Prevalence : 0.5463
##       Balanced Accuracy : 0.9800
##
##        'Positive' Class : 0
##
```

```
rf_crossv_accuracy <- cm_rf_crossv$overall['Accuracy']

accuracy_results <- bind_rows(accuracy_results, tibble(method="RF crossv", ACC=rf_crossv_accuracy))
accuracy_results %>% knitr::kable()
```

method	ACC
GLM	0.8799865
KNN	0.9550372
RF	0.9807302
SVM	0.9178499
RF crossv	0.9783638

Results

Based on the models examined above, a Random Forest model with default mtry value looks the most promising. Now let us unlock the validation set for final verification, using the random forest model.

```
#Predicting onto the hold out validation set and displaying confusion matrix and accuracy
validation_pred <- predict(train_rf, validation_set)
cm_validation <- confusionMatrix(validation_pred, validation_set$isFraud)

print(cm_validation)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 817   0
##          1   5 822
##
##                Accuracy : 0.997
##                  95% CI : (0.9929, 0.999)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9939
##
##  Mcnemar's Test P-Value : 0.07364
##
##             Sensitivity : 0.9939
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
```

```
##          Neg Pred Value : 0.9940
##              Prevalence : 0.5000
##          Detection Rate : 0.4970
##    Detection Prevalence : 0.4970
##       Balanced Accuracy : 0.9970
##
##        'Positive' Class : 0
##
```

```
validation_accuracy <- cm_validation$overall['Accuracy']
accuracy_results <- bind_rows(accuracy_results, tibble(method="VALID.", ACC=validation_accuracy))
accuracy_results %>% knitr::kable()
```

method	ACC
GLM	0.8799865
KNN	0.9550372
RF	0.9807302
SVM	0.9178499
RF crossv	0.9783638
VALID.	0.9969586

Trying the random forest model on a larger subset of the non-fraudulent cases, that were excluded in sampling, so where fraud could still occur.

```
#Sampling a subset of legal transactions
set.seed(1,sample.kind="Rounding")
subset_no_fraud <- sample_n(only_no_fraud, dim(under_sampled_set)[1])
subset_no_fraud$type <- as.factor(subset_no_fraud$type)
subset_no_fraud$isFraud <- as.factor(subset_no_fraud$isFraud)

#Predict using the random forest model and calculate accuracy
no_fraud_pred <- predict(train_rf, subset_no_fraud)
no_fraud_accuracy <- 1 - sum(no_fraud_pred==1)/dim(subset_no_fraud)[1]
accuracy_results <- bind_rows(accuracy_results, tibble(method="NO FRAUD", ACC=no_fraud_accuracy))
accuracy_results %>% knitr::kable()
```

method	ACC
GLM	0.8799865
KNN	0.9550372
RF	0.9807302
SVM	0.9178499
RF crossv	0.9783638
VALID.	0.9969586
NO FRAUD	0.9822842

Conclusions

The random forest model showed the best accuracy, and also scored the lowest “false-positives”.

Modelling for fraud turned out to be complicated due to the over representation of non-fraudulent transactions. The data set had to be reduced in size by under-sampling methods, mostly due to computational power but also to even out proportions between classes. The final model reached an accuracy of over 99 % on the validation data, which is great. Keeping in mind that 57 % percent of the data was excluded because the summary showed no fraud exist under these circumstances, the total accuracy for the full scoped business case would be increased. An implemented solution could work in a matter to first examine if the transaction is of a character where fraud occurs, and depending on if this is the case or not predict "no fraud" without even involving the model if the conditions are not met. It is however important to note that the data set is showing the truth as of now, and that the characteristic of fraudulent cases may very well change, and so also the modelling approach.

If modelling would have been done for the full data set, I would have focused on the balanced accuracy as a measure, taking size of classes into account. When examining confusion matrices, I have chosen to focus on "false-positives" as the number to minimize, since this is a measure of how many non-fraudulent cases are classified as fraudulent. Having a large such number would lead to decreased customer satisfaction.

A next step for improving the model could also be to separate the types of fraud and create two models. We saw fraud only occurs for "CASH_OUT" and "TRANSFER", and a natural next step could be to examine the behaviour of these two types of fraud individually, and if a better model could be built based upon that knowledge.

I would also like to continue working with parameters and larger data sets, now when a preferred model has been chosen. A next step would include tuning based on different depths of trees and include more data into training.

References

[1]. Jaitley,U. Comparing Different Classification Machine Learning Models for an imbalanced dataset. (2019). Towards Data Science. Available from: https://towardsdatascience.com/comparing-different-classification-machine-learning-models-for-an-imbalanced-dataset-fdae1af3677f