CodeOp's DA cohort1 Group Project

Title:
Predictive analysis for fraud detection using synthetic mobile money transaction data

Hyojung Lee
Maria Gayà Fiol

Summary

PaySim is a financial mobile money simulator devised for fraud detection. The data generated by PaySim is based on a sample of real transactions extracted from one month of financial logs from a mobile money service implemented in an African country. Hence, we can benefit from this simulated data, which is similar to the original records, to conduct fraud analytics overcoming the lack of public access to private records.

Here, we used one of PaySim's public datasets available on the Kaggle website. Using this synthetic data, we aim to understand better all the features related to fraudulent profit by data analysis and make good predictions on the unseen data based on our findings. To enhance prediction performance, we implemented different machine learning algorithms and evaluated each model.

• Jupyter Notebooks: https://github.com/CodeOp-tech/projectfraud-paysim
• Data used for analysis: https://www.kaggle.com/ntnu-testimon/paysim1
• Reference: E. A. Lopez-Rojas , A. Elmir, and S. Axelsson. "PaySim: A financial mobile money simulator for fraud detection". In: The 28th European Modeling and Simulation Symposium-EMSS, Larnaca, Cyprus. 2016

Outline

1. Exploratory Data Analysis
   - When does fraud happen?

2. Feature Engineering
   - Encode non-numeric to numeric values

3. Machine Learning To Detect Fraudulent Transactions
   - Build a baseline
   - Compare the performance of different ML models:
     Logistic regression vs. Neural Network vs. Random Forest vs. XGBoost

1. Exploratory Data Analysis

1.1 Data collection

Simulated transaction data was downloaded from the Kaggle website and saved as .CSV file under the 'data' directory. For analysis, the file was imported into Jupyter Notebooks as a DataFrame object and processed in Python programming language.

1.2 Data summary

The file was retrieved as a DataFrame structure. It has 6,362,620 rows x 11 columns. Table 1 contains extended information about each column.

Table 1. Fields information

Column name	Data type	Contents
step	integer	a unit of time in the real world. 1 step is 1 hour of time. Total steps 744 (30 days simulation)
type	string/categorical	type of transaction
amount	float	amount of the transaction in local currency
nameOrig	string	customer who initiated the transaction
oldbalanceOrg	float	initial balance before the transaction
newbalanceOrig	float	new balance after the transaction
nameDest	string	customer who is the recipient of the transaction
oldbalanceDest	float	initial balance of recipient before the transaction
newbalanceDest	float	new balance of recipient after the transaction
isFraud	boolean	if transaction is fraudulent (encoded as 1) or valid (encoded as 0)
isFlaggedFraud	boolean	if transaction is flagged as fraudulent (encoded as 1) or not flagged at all (encoded as 0).

The data includes an initial prediction on frauds in the 'isFlaggedFraud' column. The flagging system used here was based on very simple observation: flag(=1) if a known fraudulent transaction involved a transfer of over 200,000 in the local currency. In other words, it is missing all other frauds with less than the amount of 200,000. Hence, in this report, we present how to improve this poor prediction on financial fraud first by exploring

the data, secondly by selecting most relevant variables and lastly by applying different kinds of machine learning algorithms.

1.3 Data exploration (visualization and analysis)

The distribution of non-binary numeric variables is shown in Figure 1. These variables show highly skewed distributions and none of these are normally distributed.

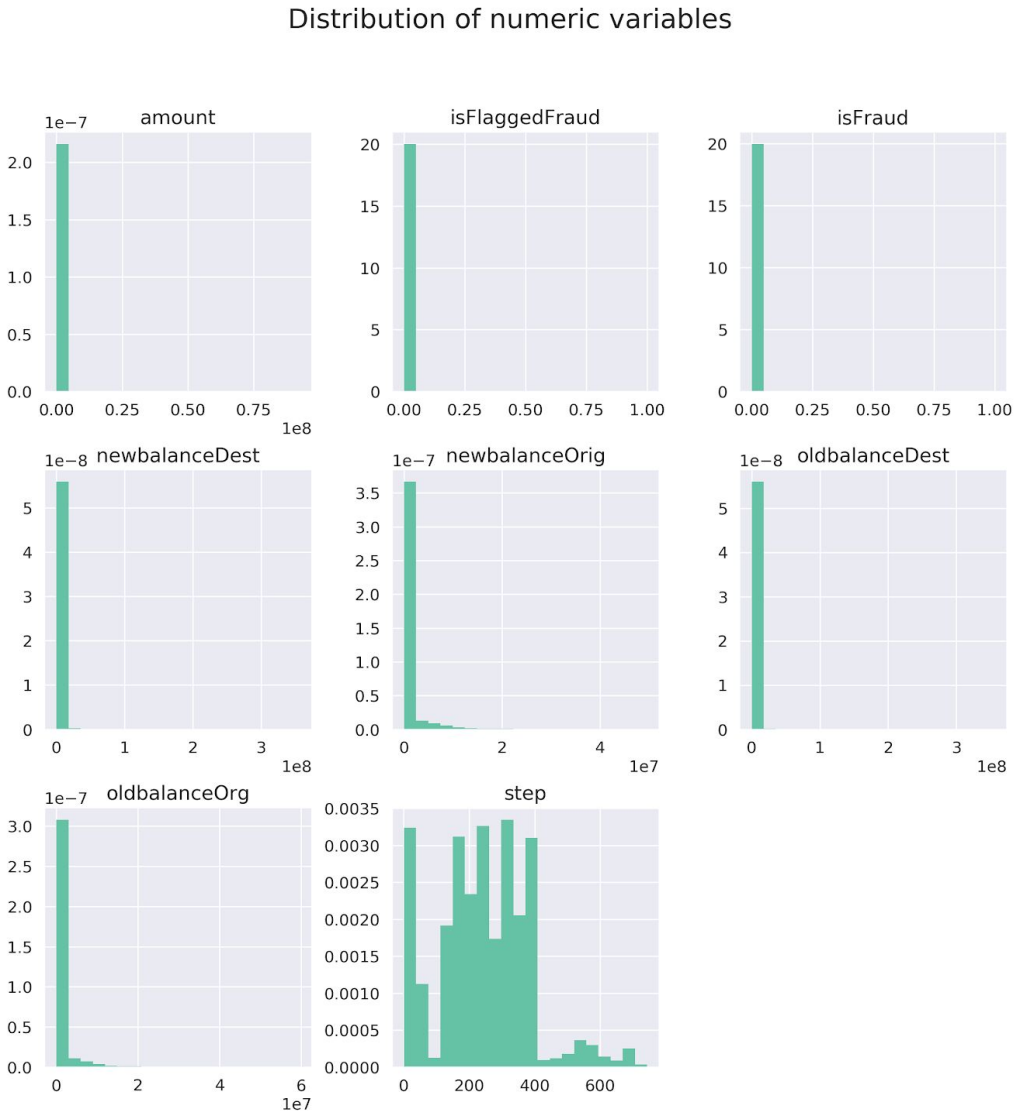


Figure 1. Distribution of numeric variables. Histograms generated by its counts normalized to form a probability density. The area under each histogram sums to 1.

Binary variables ('isFraud' and 'isFlaggedFraud') of this data contain boolean-type values. Value '1' means the affirmation of each action ('fraud', 'flagged as fraud') while '0' represents the opposite. As shown in Figure 2, both 'isFraud' and 'isFlaggedFraud' columns have extremely unbalanced distributions.

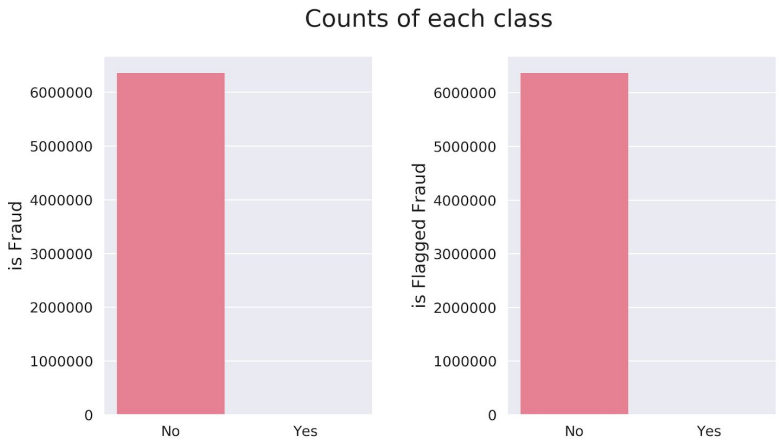


Fig 2. Target distribution. Left: 99 % (6,354,407 cases) of the records in this dataset were valid transactions whereas only 0.1 % (8,213 cases) were fraudulent. Right: Only 0.0003 % (16 cases) of the records were flagged as fraud. The rest 8197 cases were falsely reported as negative.

A correlation is a metric that represents how much two random variables vary together. To quantify correlative relationships between all the variables, we implemented .corr() function, which measures the direction and strength of linear relationship and represents it as a score between -1 and 1 (Figure 3). There are perfectly correlated (Pearson's r = 1) column pairs such as 'newbalanceOrig'- 'oldbalanceOrig' and 'newbalanceDest'-'oldbalanceDest'. It is natural that these balance-related columns have such a particular relationship since these pairs indicate the amount of mobile money saved in an account before and after a transaction. Likewise, 'amount' has rather strong correlation (r = 0.3 - 0.5) with the above mentioned four columns since it is the difference between the old and new balance.

This project deals with a classification problem where the goal is to predict highly unbalanced binary class labels (0 and 1) of 'isFraud'. To build our own prediction model, we need to select our features-of-interest which provide a good forecast of our target, 'isFraud'. However, it was not possible to specify our scope to a few features using this metric, since there were no columns having good correlation scores with the target (all measured values were below |0.1|) (Figure 3 and Table 2).

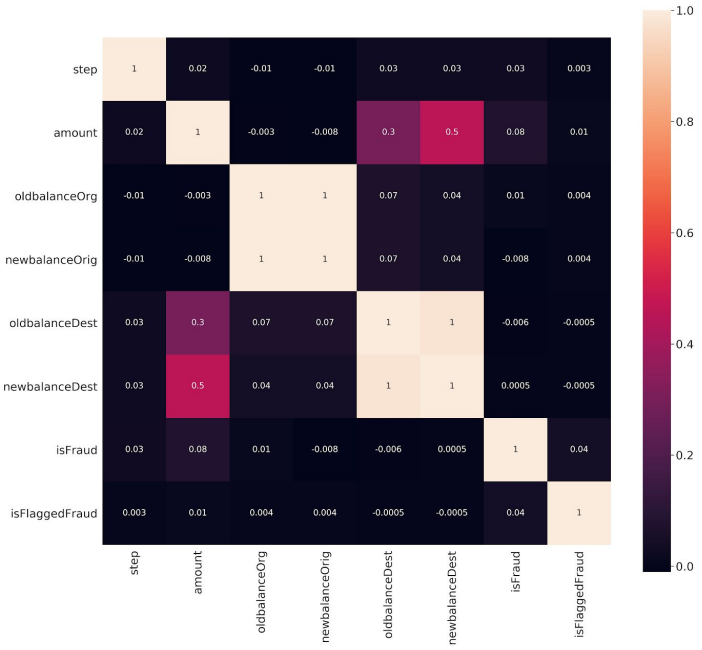


Fig 3. A heatmap showing quantified linear correlation with Pearson correlation coefficient (Pearson's r scores). Amount of each transaction and balances of the accounts where the transaction happened show strong correlation.

Column name	r
amount	0.077
isFlaggedFraud	0.044
step	0.032
oldbalanceOrg	0.010
newbalanceDest	0.001
oldbalanceDest	-0.006
newbalanceOrig	-0.008

Table 2. Correlation coefficient measured between 'isFraud' and featured columns

By understanding the pattern of fraudulent transactions in the past, we can predict how the future events will look like. Therefore, before creating a predictive model, we decided to revise our current records and divided it into four different groups: TP (True Positive), FP (False Positive), TN (True Negative), FN (False Negative)

Following codes were utilized as masks for filtering the dataframe:

```
is_fraud = paysim['isFraud']==1
is_flagged = paysim['isFlaggedFraud']==1
```

Results (# of instances)

- TP (True positive): 16
- FP (False positive): 0
- TN (True negative): 6354407
- FN (False negative): 8197

In the current dataset, there are a total 6,362,620 observations and 8213 of them were fraudulent (TP + FN). However, only 16 fraud cases were flagged as 'fraud' (TP) and the other 8197 cases were flagged as 'not a fraud' (FN). Therefore, the current flagging system has failed on detecting >99% of frauds from this particular data (=low recall). On the other hand, it successfully detects all the negative cases without any FPs (=high precision).

Recall and Precision

- Recall
  - TP / (TP+FN)
  - How many of the relevant issues are detected as relevant?

- Precision
  - TP / (TP+FP)
  - How many events detected as relevant were actually relevant?

One of the most probable reasons for this low recall is the high imbalance observed from the target class distribution. Thus, we can expect to enhance our forcast by reducing FNs and improving the recall score. To do so, we looked into characteristics of features in two datasets, TP and FN, and compared.

1.3.a Different types of transactions

There are five types of different transactions in this data: CASH-IN, CASH-OUT, DEBIT, PAYMENT, TRANSFER (Table 3 and Figure 4)

Table 3. Transaction type

Transaction type	information
CASH-IN	Paying in cash to a merchant; Increases the balance of the account.
CASH-OUT	Withdrawal of cash; Decreases the balance of the account.

DEBIT	Sending the money from the mobile money service to a bank account; Decreases the balance of the account.
PAYMENT	Paying for goods or services to merchants; Decreases the balance of the account & increases the balance of the receiver.
TRANSFER	Sending money to another user through the mobile money platform; Decreases the balance of the account.

Unlike the frequency of transactions that appeared in the entire data (Figure 4), TP cases were solely reported when the transaction type was 'TRANSFER'. In FN events, typical transaction types observed were 'TRANSFER' and 'CASH_OUT'. From this result, we concluded that watching out for specific types of money mobilization involved in a reported fraud can be useful for our future prediction.

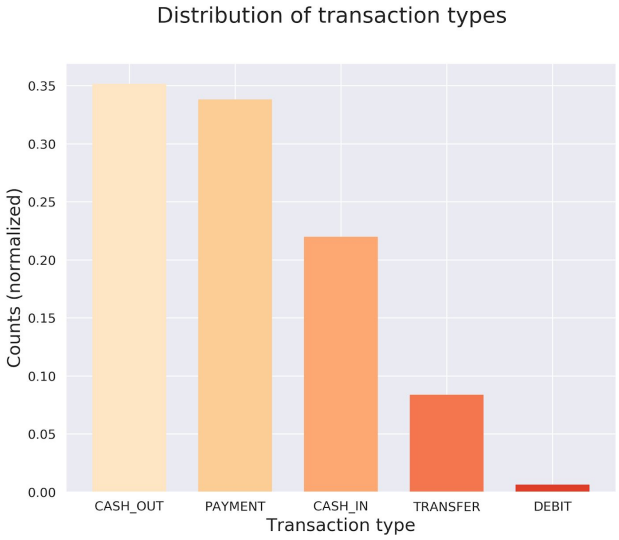


Figure 4. Different types of transactions were realized. In order of frequency: CASH_OUT (35 %) > PAYMENT (34 %) > CASH_IN (22 %) > TRANSFER (8 %) > DEBIT (1 %).

1.3.b Transaction amount

In Figure 5, amounts of each transaction are counted and normalized for each case (TP, FN and TN). In TNs, there were barely any events that surpassed 200,000 of amounts in transaction, whereas fraudulent activities involved more varied amounts of money mobilization. We did not perform the one-way ANOVA, a parametric test for statistical differences among 3 or more groups, on this set of data due the huge difference in sample size and internal variance of the groups.

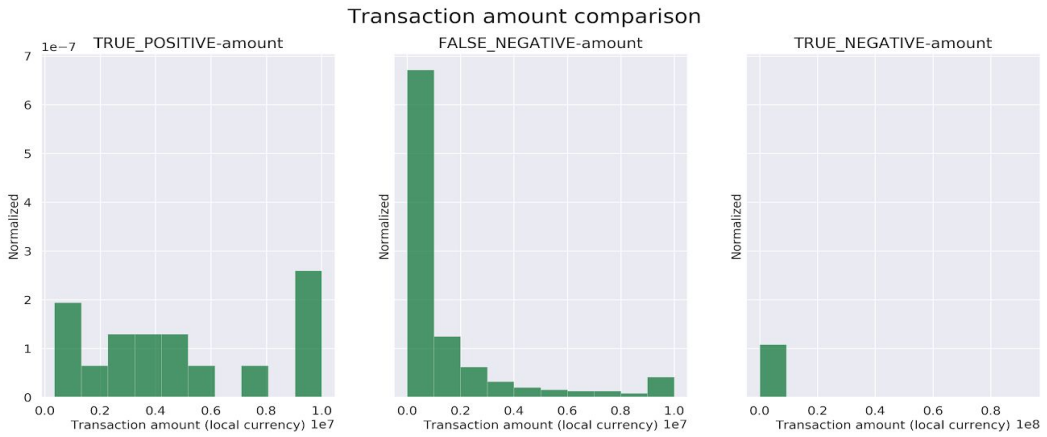


Figure 5. Probability density histogram of transaction amount in TP, FN and TN.

1.3.c Time (step) of transactions

More fraudulent activities were observed when step is over 400. Unlike TNs (right, Figure 6), FNs (middle, Figure 6) were observed regularly over the entire steps without large variance as well as TPs (left, Figure 6). In TN cases, the transaction activity decreases after the first half of the simulation steps. According to the author of this simulator, it is because of the introduction of income during the first days of the month.

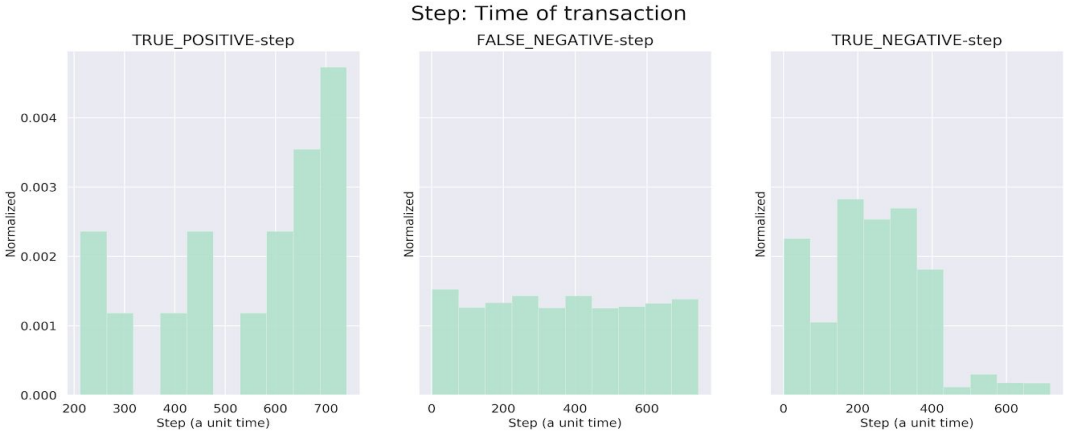


Figure 6. Probability density histogram of transaction step in TP, FN and TN.

1.3.d Name of the service user and the recipient

There are two columns containing string values and each of them indicates the name of the client (user of the mobile service) and the recipient of a transaction (nameOrig and nameDest, respectively). Considering that an input that starts with 'C' is the Customer and one starts with 'M' is the Business, we categorized string values by its customer-recipient type:

- CC: Customer - Customer
- CB: Customer - Business
- BC: Business - Customer
- BB: Business - Business

Then we created a new column called ‘validTransactionType’ and assigned these new categorical values (‘CC’, ‘CB’, ‘BC’, ‘BB) to the corresponding row. The most dominant and only customer-recipient type in all fraud transactions was ‘CC’ (total 8213 cases), while both ‘CC’ and ‘CB’ were prevalent in all valid transactions (CC: 4202912, CB: 2151495 cases).

2. Feature Engineering

Data preprocessing is one of the most crucial steps to prepare raw data in the form and shape ready to be applied to any machine learning algorithm. Since the current data was generated intentionally, it is relatively clean without any missing value in it. Hence, it was not necessary to remove or impute missing values. On the other hand, transforming categorical data with string values was required to encode all non-numeric values into numeric ones. This transformation is necessary depending on the type of the model, such as logistic regression and neural networks, which digests numeric inputs only.

2.1 Feature transformation

By using .map() method, two categorical columns with string values were converted into a numeric form: ‘type’ and ‘validTransactionType’.

- ‘PAYMENT':0, 'TRANSFER':1, 'CASH_OUT':2, 'DEBIT':3, 'CASH_IN':4
- 'CC':1, 'CB':2, 'BB':3, 'BC':4

Then, we created a new column flagging two transaction types (transfer and cashout) which could possibly be frauds. To this end, we mapped the rows only if their transaction types were either transfer or cashout with following codes:

```
type_filter = (paysim['type'] == 'TRANSFER') | (paysim['type'] == 'CASH_OUT')
```

And then the result was reflected in a new column called ‘type_to_watch’:

```
paysim['type_to_watch'] = type_filter*1
```

Final dataframe has 6362620 rows x 13 columns.

2.2 Final feature selection

Since not all features are providing information relevant to detect fraud, it is necessary to select only the best indicators for our target and build a model based on these features only. For this report, we selected following 10 features:

- step
- type
- amount
- oldbalanceOrg
- newbalanceOrig
- oldbalanceDest
- newbalanceDest
- isFlaggedFraud
- validTransactionType
- type_to_watch

Columns that were excluded for generating models are: ‘isFraud’, ‘nameOrig’ and ‘nameDest’. 'nameOrig' and 'nameDest' are nominal values containing the identity of each individual involved in a transaction and these are better represented in a newly created column, ‘validTransactionType’. Therefore we decided to remove the name columns to reduce noise. In addition, ‘isFraud’ was excluded from the feature list because it is our target.

3. Modeling

3.1 Data split into train and test set

Classification for predicting binary target labels is a subcategory of supervised learning. In supervised learning, a model learns from training data with labels and predicts the labels of unseen data. Since there is only one data available, we need to split it into two sets for fitting and predicting. To this end, we divided 70% (4453834 rows) of the data as a train, and another 30% (1908786 rows) as a test by using the train_test_split method of scikit-learn.

3.2 Baseline

With the train and test set created in the previous step, we created the most simple and basic model as our baseline. We can benefit from the baseline to get a better insight on how to improve our next model. To generate a baseline, we first standardized the data to

regularize the input and then utilized the logistic regression algorithm from scikit-learn library.

A. Standardization

Standardizing the data is to rescale all variables into a relatively similar range. This will result in a mean of 0 and similar distribution of standard deviation of 1 for all the variables. This step guarantees a unit difference of a variable to be in a similar range with another even though the original scale of both features were measured in different units. Hence the same size of the weight of a coefficient can exert similar effects on two features which makes them comparable and easier to interpret.

Among various standardization techniques available in scikit-learn library (e.g. MinMaxScaler, RobustScaler, Normalizer), we chose StandardScaler so that all features can have the same mean and unit variance.

B. Building a baseline model

A common and simple approach to predict the class is a logistic function (sigmoidal function). Logistic regression is a classification model to predict probabilities of a sample observation to be labeled as class value 1. If the probability is larger or equals to 0.5, the outcome will be '1' and otherwise '0'. Here, we instantiated the LogisticRegression object imported from scikit-learn library and fit it with our train data. After the fitting, we generated predictions using features from the test data.

Results

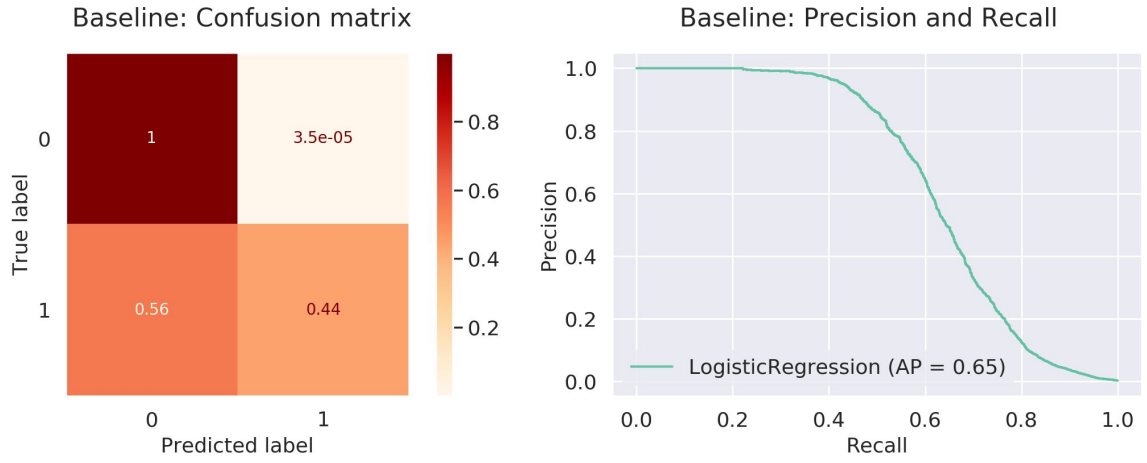


Figure 7. Confusion matrix (left) and Precision-Recall curve (right) of baseline model.

Due to high imbalance in target label distribution, the result shows high FN and relatively low recall (score: 0.438). Prediction for 'not a fraud' converged to 100% accuracy and hence there is negligible amount of FPs which leads to high precision of the model.

In the next step, we will apply different machine learning algorithms to outperform this baseline. More specifically, we will evaluate recall scores of each model as an important metric to improve the forecast.

3.2 More Machine Learning

In order to improve our current baseline of this particular case, we implemented four different models of supervised learning to improve the target classification:

1. Logistic regression
2. Neural network,
3. Random forest
4. XGBoost

A. Logistic regression

As the proportion of the target ratio (not_fraud(0) vs. fraud(1)) converges to 99.9 vs. 0.1 (%), we can penalize the wrong prediction of the minorities much higher than that of the majority by defining the class weights as the inverse of label distribution. To this end, we added 'class_weight' hyperparameter and built another logistic regression model.

```
# Define class weights (inverse ratio of the label)
weight = {0:0.1, 1:99.9}

# Fit again with logistic regression
logistic2 = LogisticRegression(class_weight=weight)
logistic2.fit(X_train, y_train)
```

Results

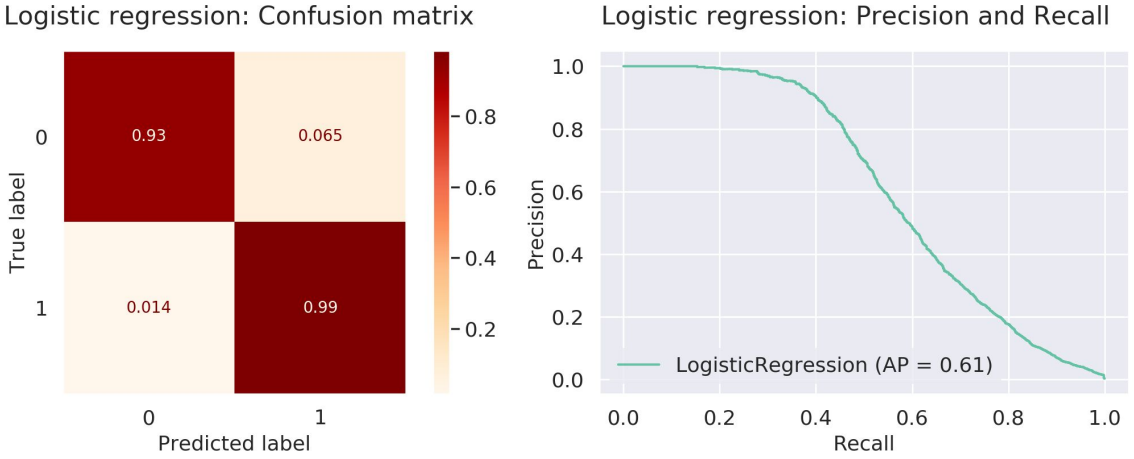


Figure 8. Confusion matrix (left) and Precision-Recall curve (right) of improved logistic regression model.

By applying different weights to each class, the prediction for label '1' increased significantly (recall score: 0.986). Naturally, it increased FPs slightly as a precision-recall tradeoff, but the model can detect 93% of the negative events correctly. Therefore this model can predict the target in a more balanced way than the baseline.

B. Neural network

Second algorithm of the supervised learning model we selected is the Neural network. To generate and evaluate a neural network, we imported the Sequential model of Keras library. This model can be simply constructed by passing a list of hidden layers which contain a number of nodes (neurons). A node in a layer computes the inputs by applying different weights and sends the output out to the next layer.

Here, we defined 3 hidden layers with decreasing node numbers (16-8-1). For the first 2 layers, a rectified linear activation function(ReLU) was given as an activation function, a nonlinear function which works like a linear function. In the last layer, sigmoid function was applied to transform the output as a probability between 0 and 1.

```
# Layers
nn_layers = [
                 layers.Dense(16, activation='relu', input_shape=(10,)),   # 10 features
                layers.Dense(8, activation='relu'),
                layers.Dense(1, activation='sigmoid') # 1 output
                   ]


# Define a Sequential model
nn_model = Sequential(nn_layers)

# Recompile and train the model
nn_model.compile(optimizer='Adam',
                           loss='binary_crossentropy',
                           metrics=metrics.Recall())

# Fit the model on our data
nn_model.fit(X_train, y_train, epochs=3, validation_data=(X_test, y_test))

# Generate predicted y values
y_prediction_nn = nn_model.predict(X_test)
```

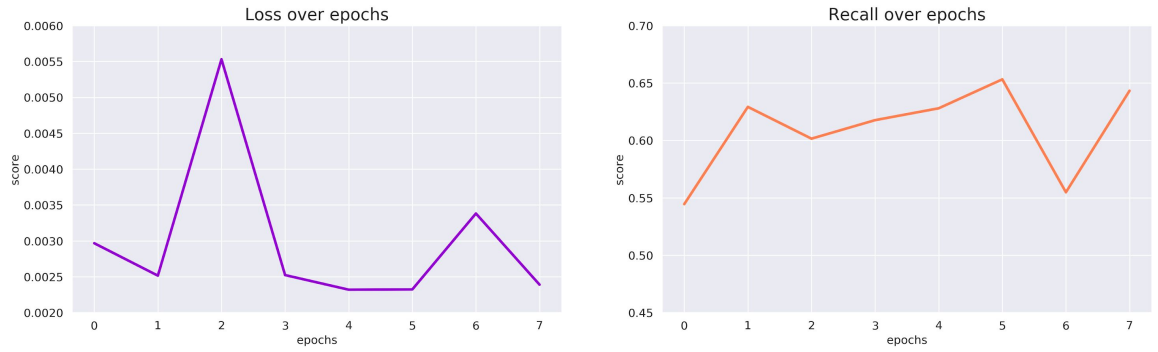
When compiling the model, 'binary crossentropy' was chosen as a function to estimate the error of the model since it is a loss function that is typically used for binary classification. In a neural network model, calculating the loss helps update the weights and reduce the loss on the next evaluation. To evaluate the model, both AUC & recall values were retrieved to compare them with our baseline and other types of models. Adam optimizer is expected to optimize individual learning rates for each parameter.

Results

Epoch numbers between 3-9 have been tried so far , so that our model could learn the entire train data for the defined number of times. Class weight was not included in this particular model since the weight caused overfitting and the loss increased meanwhile the recall score decreased. The best performance observed was when tried with epoch = 8.

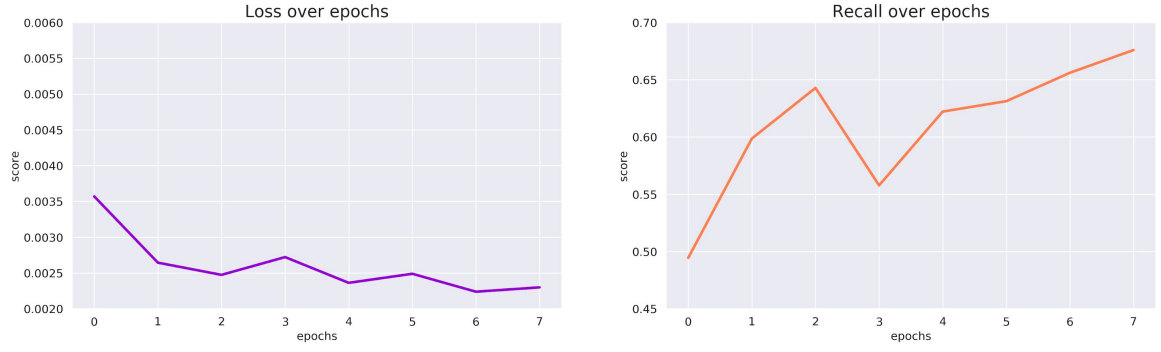
Here, results from two different attempts are attached. In both trials, first graph (purple line) indicates the changes in loss value over the epochs and the second (orange line) shows the recall scores.

Trial 1:



- 4 hidden layers (# neurons: 24 - 16 - 8 - 1)
- 8 epochs
- Final recall score: 0.643

Trial 2:



- 3 hidden layers (# neurons: 16 - 8 - 1)
- 8 epochs
- Final recall score: 0.676

Trial 2 shows better prospects since it demonstrates a continuous tendency to 1. Decrease the loss, and 2. Increase the recall. The best recall score this model gave was 0.676, but there is a room to enhance this result by adjusting the number of layers, number of nodes and the learning epochs.

C. Random forest

Maria

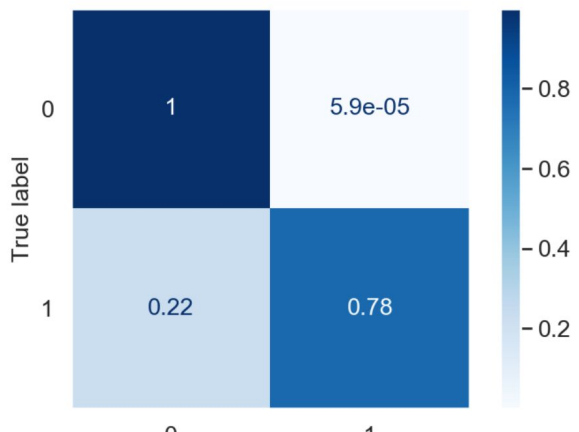
D. XGBoost

XGBoost is short for **Extreme Gradient Boosting** and is an efficient implementation of the stochastic gradient boosting machine learning algorithm. The stochastic gradient boosting algorithm, also called gradient boosting machines or tree boosting, is a powerful machine learning technique that performs well or even best on a wide range of challenging machine learning problems. Before any modification or tuning is made to the XGBoost algorithm for imbalanced classification, it is important to test the default XGBoost model and establish a baseline in performance.

```
# Create a simple classifier
weights=0.99
classifier = XGBClassifier(learning_rate =0.1, n_estimators=1000,
    max_depth=5, min_child_weight=1, gamma=0,subsample=0.8,colsample_bytree=0.8,
    objective= 'binary:logistic', nthread=4,scale_pos_weight=1,seed=27)
classifier.fit(X_train, y_train)
presult(classifier,X_test,y_test,1)
```

A good practice, for these cases is to establish the below parameters. For example, the scale_pos_weight set to 1, will

- max_depth = 5 : This should be between 3-10. I've started with 5 but you can choose a different number as well. 4-6 can be good starting points.
- min_child_weight = 1 : A smaller value is chosen because it is a highly imbalanced class problem and leaf nodes can have smaller size groups.
- gamma = 0 : A smaller value like 0.1-0.2 can also be chosen for starting. This will anyways be tuned later.
- subsample, colsample_bytree = 0.8 : This is a commonly used start value. Typical values range between 0.5-0.9.
- scale_pos_weight = 1: Because of high class imbalance.



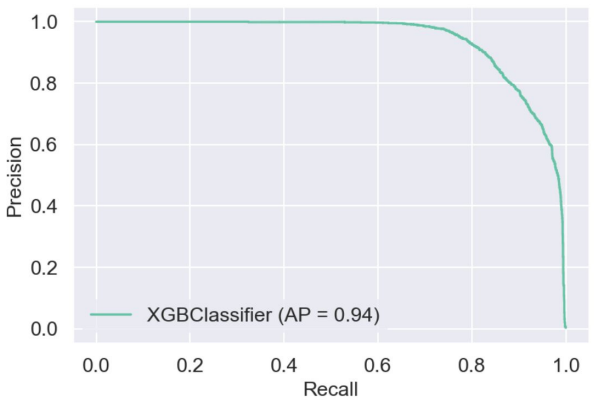


Figure 10. Confusion matrix (left) and Precision-Recall curve (right) of improved logistic regression model.

The recall score given is 0.737. Naturally, it increased FPs slightly as a precision-recall tradeoff, but the model can detect 100% of the negative events correctly. Therefore this model can predict the target in a more balanced way than the baseline.

4. Conclusion

In summary, we built highly functioning predictive classifier models to improve the financial fraud detection system. As a result, we were able to detect fraudulent operations with higher recall scores(up to 0.99) when using all the three different algorithms than the original 'isFlaggedFraud' or baseline model.
Here are our conclusions:

1. The logistic regression algorithm implemented with the 'class_weight' parameter displayed the highest recall score(0.99), but we got slightly lower precision scores, by introducing more FPs, as a tradeoff.
2. With XGBoost, recall score(0.78) is lower than logistic regression, but it maintains high precision, therefore the loss caused by precision-recall tradeoff is smaller.
3. NN and RF need to be optimized by further hyperparameter tuning.

As our observation, logistic regression successfully detected 99% of the fraud transactions maintaining relatively low FP rate(6.5%). We can suspect two reasons of why the simplest model of all defeated other three:

1. The number of input features were 10, 5 of which have strong linear correlation (Fig 3). Therefore, data itself was not too complex to deploy high dimensional calculations. (We noticed overfitting of the model when introducing Keras model - page 14).
2. Some models(NN, RF and XGBoost) do need further optimization of its hyperparameters

By optimizing the present models and by introducing more complicated dataset with extra features, we expect to address a practical solution for mobile money transaction business to detect fraud with high accuracy.