Online Payment Fraud Detection Project

Matteo Angelini

November 8, 2022

Contents

1 Objectives of this project

In this project we try to detect online payment frauds with Machine Learning models. We will compare and evaluate different models to detect if a transaction is fraudolent or not by training these models on a dataset containing more than 6 millions data points. This is a Classification Task in a Supervised Learning environment, the model will need to recognize whether the transaction is fraudolent or not by detecting abnormal activities in the input features. We will use traditional performance metrics for classification to measure how well various models classify the transactions, for example:

- **accuracy** to measure how often the classifier correctly predicts the label, this metric alone is not enough to determine the performance of a model and can be misleading as we will show later
- **precision** to show how many of the correctly predicted cases actually turned out to be positive, useful in cases where detecting False Positives is more important than False Negatives
- **recall** to show how many positive cases we were able to predict correctly, useful where detecting False Negatives is more important than False Positives
- **f1 score** to combine precision and recall metrics, useful when the number of True Negatives is high

2 Splitting training/test set

We need to split our dataset even before doing any other analysis because otherwise there's a high chance of overfitting. Another reason is that there's a high risk of leakage, this means that the training set is not completely new for the model. We first split the dataset in two sets called *train set* and *test set* with a percentage of 90 and 10 respectively because Machine Learning and especially Deep Learning models needs a lot of data to execute the training. We put aside the *test set* and we don't touch it anymore until the end.

3 Data exploration

In this section we will explore the dataset analyzing each feature and their relationships. We can start by plotting the first rows and describing each column.

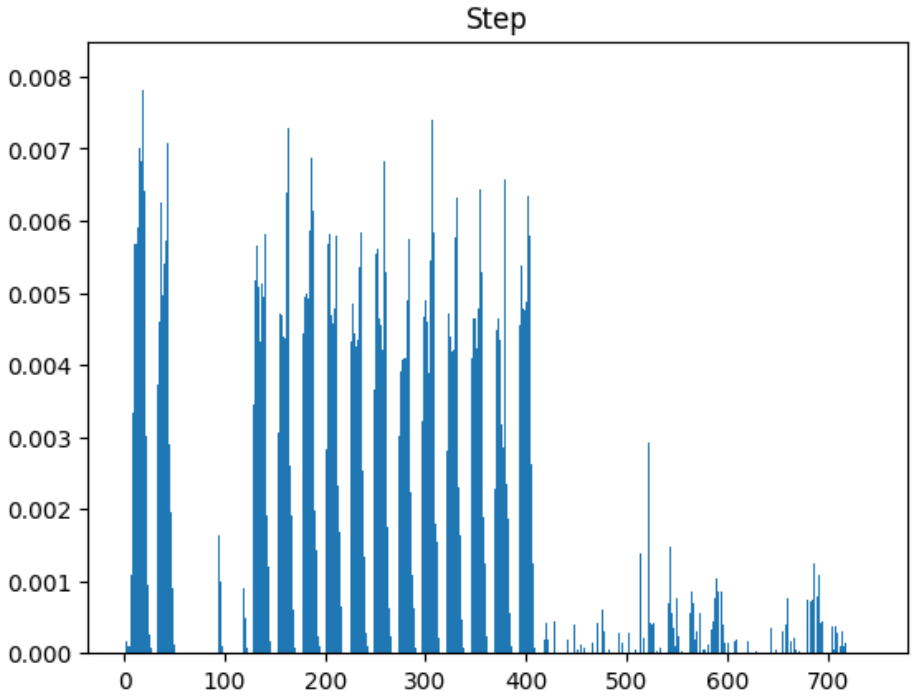
- step: it represents a unit of time where 1 step equals 1 hour
- type: type of online transaction
- amount: the amount of the transaction
- nameOrig: customer starting the transaction
- oldbalanceOrg: balance of customer before the transaction
- newbalanceOrig: balance of customer after the transaction
- nameDest: receiver customer of the transaction
- oldbalanceDest: initial balance of receiver customer before the transaction
- newbalanceDest: the new balance of receiver customer after the transaction
- isFraud: fraud transaction, this is the target label
- isFlaggedFraud: this feature is not described in the dataset description

3.1 Visualizing features

In this section we will plot and explore the most interesting features.

3.1.1 Step

It represents a unit of time where 1 step equals 1 hour, in the following graph we can see the time distribution.



3.1.2 Type

It represents a type of online transaction, there are 5 classes:

- CASH OUT
- PAYMENT
- CASH IN
- TRANSFER
- DEBIT

We can plot its distribution and check which one is the most frequent in our dataset. Type

It looks like most of the transactions are either CASH OUT or PAYMENT, so the majority of frauds will be of one of this two categories.

3.1.3 isFraud

This is our target feature, or label, and we can notice from the graph below that it is imbalanced, which is not ideal for a classification task. We will discuss about the consequences and how to improve it in the next chapters. isFraud

3.1.4 Amount

Let's plot the 10 largest transaction imports. amount

As we can see from the plot above for the first 10 transactions, and also in the attached code for the first 100, neither of these is a fraud. We can conclude that frauds are happening in transactions that have low amount of money

3.1.5 oldBalanceOrig and newBalanceOrig

We can plot the 10 largest balances of the dataset and compare them with the newBalanceOrig

3.2 Finding correlations

The sampling distribution shows a strong positive correlation between the balance before and after the transaction. Now let's have a look at the correlation between features of dataframe compared to the isFraud column.

As we can see from this heatmap plot and the observations we made in the previous chapter, it seems that isFraud is not strongly correlated with other columns.

4 Preprocessing and Feature Selection

We can show that there aren't any missing or incomplete feature in the dataset, so we don't need to perform data cleaning tasks.

We observed at the beginning that for the *isFlaggedFraud* column there's not much information, it is not clear what is it's purpose. We can see that it is utilized in the dataset only few times, so we drop it from the dataframe.

The correlation map also shows that there's no correlation between frauds and the customers IDs, so we can remove them. To drop columns we need to create a custom Transformer with scikit-learn, and this can be included in a Pipeline. Building our pipline to do preprocessing on data is useful not only for the training set but also for the test set, this way we have a predefined path that data needs to take before being processed by a model. Apart from the ColumnDropper our pipeline will have the following objects:

- a **OneHotEncoder** to encode strings and still be able to preserve their meaning without the risk of being misinterpreted by models for having some sort of hierarchy or order in them. In our case we will use it on the *type* column
- a **StandardScaler** to scale our numerical data, we need to do it because our columns have different ranges each

We can visualize the first dataset rows:

5 Models Comparison

Let's train some models and compare them, for this project we will use:

- Logistic Regression
- MLP Neural Network
- Random Forest

We will also use cross validation, this will give us better results by making it harder for the model to overfit.

5.1 First struggles

As soon as we start training our models for the first time we understand that the models initialization will need some tweaks to decrease the complexity and build time. In our host machine the program was taking up to 3GB of RAM and 10 minutes to build even the simplest Logistic Regressor. So we limit the iterations needed by the model to converge to 300, we start training again and we get the following results:

The accuracy is very high, if we look at the Confusion Matrix we notice that the model predict almost all the "notFraud" transaction correctly, but also that it does not classify the fraudulent ones at all! Basically the model learnt to always predict the notFraud class without doing any analysis, because this is the most convenient way to minimize the Loss Function. This can also be noted by looking at the Recall Score, which is very low, meaning that the classifier failed to predict the minority class label. We can show this even better by doing the training only on one column and computing the confusion matrix once again:

As we can see from the results the accuracy is still very high, and the confusion matrix is almost identical. This is happening because we have a highly imbalanced dataset, meaning that the occurrencies where "isFraud" is 0 are almost 27 times more than where "isFraud" is 1. We try to deal with this imbalance by applying resampling.

5.1.1 Balancing the dataset

We want to visualize this imbalance, but to do so we need to use a Dimsensionality Reduction technique to reduce dimensions of our dataset from 11 to 2. In this project we will use PCA for reducing dimensions combined with undersampling to remove the imbalance. The followings are plots before and after balancing the dataset.

Even if this operations balance the dataset, they also have some weaknesses. For example oversampling will increase the risk of overfitting, and undersampling will increase the risk of information loss. The initial idea was to use a specific library called "imblearn" and execute both undersampling and oversampling together, but it revealed to be more difficult than I initially thought. After 20 minutes of running without any result and after tweaking various versions of both sklearn and imblearn without any improvement I decided to just do undersampling with basic pandas functions.

5.2 Cross validation with KFold

After balancing the dataset we can start training our models using KFold and we get the following results

This results are not bad but also not exceptional, we can try to implement a Neural Network.

5.3 Neural Networks

In previous chapters we only used ML models, now we introduce our final DNN model and compare its performance with the others. In this project we use Tensorflow and Keras, so let's create a base model that later will be tweaked. The structure is the following:

- input layer with 11 neurons because our dataframe has 11 columns
- a hidden layer of 100 neurons. This will be tweaked afterwards, but a paper found out that it's better to create several hidden layers with the same number of neurons than having a single layer with a lot of neurons
- the output layer containing a single neuron because this is a binary classification task

Both the input and hidden layer have a ReLU activation function, instead the output layer has a sigmoid activation function. For a binary classification task it is more suitable a binary cross entropy Loss Function, and the Adam algorithm for gradient descent. After executing training for selected models we get the following results:

MLP results are obviously extremely dependent on the number of epochs, this hyperparameter can be tweaked to get better performances. So we choose the Neural Network as our final model and begin the hyperparameter tuning.

5.4 Hyperparameter optimization with KerasTuner

Now that we chose the Multi Layer Perceptron as our final model we can tweak some hyperparameters like the number of hidden layers, number of neurons for each layer, number of epochs for achieving better results. Doing so manually is inefficient and painful, so we can use the recommended libray for Keras called KerasTuner.

6 Testing our final model

The first thing we do is processing our custom Pipeline calling only with the *transform* method because we don't want to fit it again. Now that we have our final tuned model with the correct weights we can test its performance on the training set.

Achieving a 98% accuracy with 0.4% loss is a great result, maybe with a more advanced NN model we could improve even more the performance.