DATA586 Project - Credit Card Fraud Detection

Neural network anomaly detector

Group #8: Alyssa Foote, Viji Rajagopalan

Abstract:

According to European Central Bank's (ECB) report released in Oct 2020, the total amount of fraud related losses amounts to 1.8 billion Euros just in one of the areas namely Single European Payment Area (SEPA). The fraud count also shows a growing trend over the years which explains the severity of the problem. There are several solutions constantly assessed to solve this problem and some of the prominent solutions include using Bayesian belief networks, artificial neural networks, random forests, auto encoders and recurrent neural networks. The scope of this project is to assess simple deep neural net for solving this problem and we hypothesize that a simple deep network would be able to provide an accuracy over 80% in detecting true fraud. Multi-layer perceptron based model and neural network model with 3 layers are compared against each other and the neural network with 3 layers is selected. Hyper parameters for this model are tuned and the resulting model successfully detects true fraud at 86% and has an F1-score 74%. CNNs were also assessed for this dataset and as the goal is to build a simple model with neural network, it is not carried forward during the course of this project.

Introduction and Background:

According to European Central Bank's (ECB) report released in Oct 2020, the total amount of fraud related losses amounts to 1.8 billion Euros just in one of the areas namely Single European Payment Area (SEPA). The green line in the below *fig 1* shows an increasing trend in credit card fraud over the years given a lot of enhancements in the payments industry. This shows the need for and importance of detecting credit card fraud accurately.

Fig 1 - Credit card fraud trend

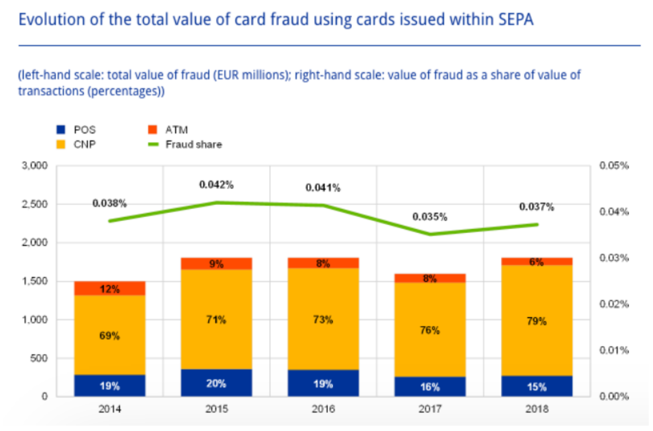


Fig. 1. Evolution of total value of card fraud using cards issued within SEPA.
Card-not-present frauds account for the majority of reported frauds.

Image source - https://fraud-detection-handbook.github.io/fraud-detection-handbook/Chapter_2_Background/CreditCardFraud.html

To allow faster and convenient services for customers, credit card agencies collect an enormous amount of data. To prevent fraud, the credit card companies need to be able to flag in real time to customers and take action like blocking the card. This must be done with caution and high precision as otherwise it would be an annoying and unpleasant experience for consumers. So, for detecting fraud transactions, huge volumes of data need to be processed in real time and with high precision. Thus this becomes a classic case for using machine learning algorithms to find a suitable solution. The scope of this project involves building a suitable model for anomaly detection for credit card transactions.

The goal of credit card fraud detection process is to identify fraudulent transactions made using stolen or compromised credit card information. ECD dataset that contains two days of transaction data of European cardholders in September 2013 is used for building the desired model. The dataset is hosted on Kaggle.com and can be accessed here. This dataset has 284,807 total transactions and 492 fraud transactions in them. As we can see, the total number of fraud transactions is about 0.0172% of total transactions and this kind of unbalanced dataset needs special attention from dataset preparation to model validation for our model to be successful which we will explain further in specific sections of the report.

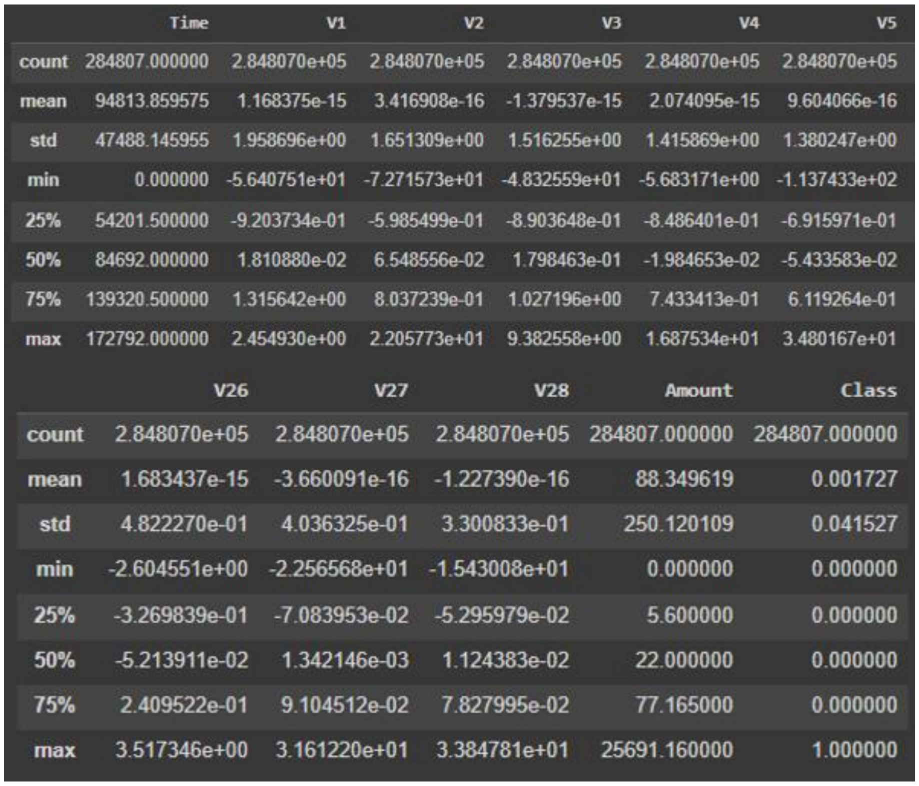
Armed with above knowledge about credit card fraud, we start to analyze the given dataset and see that it has 30 features overall. Based on information from Kaggle about this dataset, there are 28 PCA transformed features (PCA components) from real transactions and the feature related details are undisclosed to protect consumer data. The fields are named from V1, V2…V28. Two other fields present are amount and time that indicate the transaction amount and seconds elapsed between each transaction and the first transaction in the dataset. Last but not least is the Class variable which is present in the 31st column and it is a binary classification variable. A value of 1 in this field indicates it’s a fraud transaction and 0 indicates it’s a non-fraud transaction. The number of fields and the size of the dataset are comparatively huge, and we conclude designing a neural network-based machine learning model/solution will be appropriate and recommended for this type of problem.

Methodology:

Exploratory Analysis: Similar to other machine learning and data analysis methods, it is important to understand the data before applying developing deep learning models for making predictions or classifications. For this study, after the credit card fraud data was downloaded from Kaggle and loaded into a Pandas dataframe, a brief exploration of the data was conducted. We began the exploratory analysis by reviewing the descriptive statistics. As seen in *Fig 2* below, we can confirm that the variables from the PCA are scaled, while the 'Time' and 'Amount' are not scaled. Neural Network models require all features to be scaled or normalized so this is something that will be addressed before training the model.

Various functions from the Pandas library was then used to analyze the data further. A check for nulls confirmed that the data did not have any. While a review of duplicates revealed that the data had a number of duplicate records with the majority, 1062, being non-fraud cases and only 19 records recorded as fraudulent. The duplicates were removed as part of the preprocessing steps so that the neural net models only saw unique data.

Fig 2 – Dataset Summary

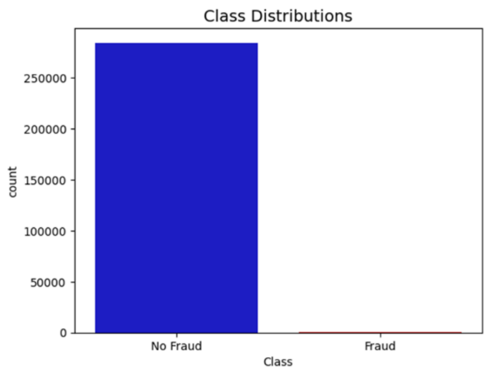


	Time	V1	V2	V3	V4	V5
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01

	V26	V27	V28	Amount	Class
count	2.848070e+05	2.848070e+05	2.848070e+05	284807.000000	284807.000000
mean	1.683437e-15	-3.660091e-16	-1.227390e-16	88.349619	0.001727
std	4.822270e-01	4.036325e-01	3.300833e-01	250.120109	0.041527
min	-2.604551e+00	-2.256568e+01	-1.543008e+01	0.000000	0.000000
25%	-3.269839e-01	-7.083953e-02	-5.295979e-02	5.600000	0.000000
50%	-5.213911e-02	1.342146e-03	1.124383e-02	22.000000	0.000000
75%	2.409522e-01	9.104512e-02	7.827995e-02	77.165000	0.000000
max	3.517346e+00	3.161220e+01	3.384781e+01	25691.160000	1.000000

Although the data description provided, as described in the background section above, indicted that the data was highly imbalanced, we ended our data exploration by creating a simple bar chart of the class distributions as seen in *Fig 3*. Only a sliver of red can be seen at the bottom which visually represents the severity of imbalance in this dataset.

Fig 3 – Data Class Imbalance



Data Preprocessing: Before any data is fed into any type of model, including deep learning, it is vital that various preprocessing techniques are used. The applicable methods will highly depend on the dataset being studied. For this project, scaling, feature reduction, and data balancing strategies were implemented. To scale the 'Time' and 'Amount' columns we opted to use the RobustScaler method from the sklearn package as it scales the values based on the quartile range instead of the mean and standard deviation as is done with the regular scaling method. For feature selection, we looked at using feature reduction using the GenericUnivariateSelect with the FPR (false positive ratio) mode, with a resulting 22 features, but did not proceed with it as full 30 feature models were providing better preliminary test results. Then for the data balancing we opted to try both undersampling and oversamplig methods both of which were only applied to the training set. This is because the testing should be executed on an original unbalanced dataset so as to more accurately depict real word number of fraud versus non fraud transactions. For the undersampling, a random subset of the non fraud transactions equal to the number of fraud transactions were kept. As seen in *Fig 4* below, this meant that there were a total of 418 each of fraud and non fraud transactions in the training set. While for the oversamplying, a technique called SMOTE - Synthetic Minority Over-sampling Technique - was used. This synthesized additional fraud transactions through the use of the k-nearest neighbours algorithm. As seen in *Fig 5* below, there is still a balance between fraud and non fraud transactions except this time there was an increase in the number of fraud transactions.

Fig 4 – Data Undersampling

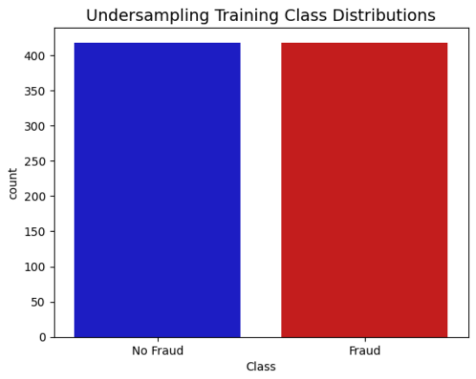
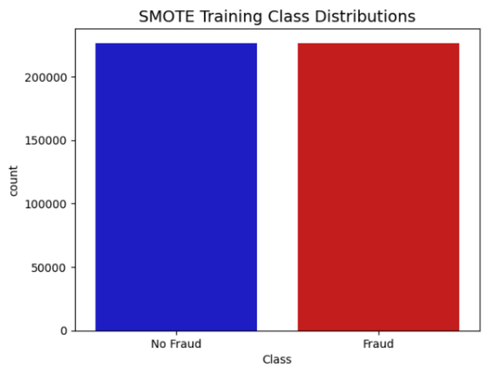


Fig 5 – Data Oversampling



Training and Evaluation Strategy - In addition to ensuring the dataset is balanced, we implemented various learning strategies to help train our models. We implemented batching so that the weights would be updated multiple times within each epoch. For our final model, we also tuned a number of additional hyperparamters including the number of epochs, the size of the batches, and the number of dropout layers. Other tuning parameters such as the learning rate were tested but deemed to not have much effect on the end results.

Although building a model that can accurately predict fraud transitions is the main goal of this project, it is also important to account for the number of correctly predicted non fraud classes. No one wants their credit card locked due to a legitimate purchase being flagged as fraudulent. As a result, looking at just the model accuracy will not suffice. Instead we need to review the both the precision and recall as well as the F1 score. Precision is a ratio of the correctly classified cases over these cases plus the wrongly classified negative cases. While recall is a ratio of the correctly classified cases over these cases plus the wrongly classified positive cases. The F1 score is a harmonic mean between precision and recall. Thus, together with a confusion matrix we can effectively evaluate the performance of our model.

Experiments and Results:

For the first experiment, simple methods are employed. The dataset is randomly split into 80%-20% for training and testing sets respectively. In order to come up with a base model, we first start our research with understanding existing models for this dataset. An IEEE paper titled "Anomaly Detection in Credit Card Transaction using Deep Learning Techniques"(part of 2022 7th International Conference on Communication and Electronics Systems (ICCES)) presents and evaluates a few of the models using Random Forest, Auto Encoders and Auto Encoders with LSTM algorithms. It has shown that the Random Forest based algorithm has shown superior performance over the other two models based on AUROCH score. Our goal for this project is to come up with a suitable neural network-based model. Drawing inspiration from this study and also from DATA586 labs, we first proceed with identifying a very simple model. Based on Random Forest algorithm's performance, we hypothesize that simple deep learning models would be able to perform well on this dataset and we proceed to explore algorithms with simple parameters.

Simple Base Model: The first set of models includes 1. A Multilayer perceptron with both linear and non-linear activations and one hidden layer. 2. Inspired by LeNet 5 architecture, we use a similar classifier with fully connected layers and tanh activation functions. The output layer has softmax classification for predicting the output. The architecture and performance of the models are tabulated in *Table 1* below.

Table 1 – Base Models Summary

Data setup	Model name	Architecture	Initial output
Unbalanced data Random split of 80-20 for training and test	Multilayer percentron	1. Number of features – 28, only PCA components V1 to V28 used 2. One hidden layer with 50 nodes 3. Soft max classification for output	54% of true fraud were identified correctly by the model
Unbalanced data Random split of 80-20 for training and test	Neural net classifier	1. Number of features – 28, only PCA components V1 to V28 used 2. Two linear transformation layers with (28,50) and (50,50) for (input,output) 3. Tanh activation functions 4. Soft max classification for output with (50,2). 50 input node and 2 output classification	80% of true fraud were identified correctly by the model

Balancing data: Based on the initial results, neural net model is selected for new assessments and hypertuning. Though we see the classification results for true fraud detection, it must also be mentioned that the algorithms have a perfect score of 100% for classifying the non-fraud transactions. This is a classic problem encountered with unbalanced dataset like the ECD dataset. It is necessary to handle this unbalance to avoid over-classifying the majority group. There are several techniques available to handle this at the data-level, algorithm-level and a hybrid-level which is a combination of both. We mainly employ data-level techniques for this dataset and model through under-sampling and over-sampling.

Under and Over-Sampling: We start with under-sampling training data. Using this technique, we created a sub-sample with 492 of fraud and non-fraud transactions and used that to train the neural net classifier model above. The performance of this model is not satisfactory. With less training data (compared to available 284,807 transactions in total), the model is prone to underfitting and loses opportunities to learn more from data.

We then oversampled the training dataset using SMOTE function and trained the neural net classifier model with this data. The model with random initial set of hyperparameters provides an F1-score of approximately 66%. We select this data sampling technique with the neural net classifier architecture and proceed to tuning hyper parameters to improve the overall performance of the model.

Hyperparameter tuning: As this model has multiple layers and is a deep neural network, we have several hyperparameters available for tuning. The important hyperparameters that are considered for tuning our model are the learning rate, weight decay, dropout rate, batch size, activation function, EPOCH, random seed.

No tuning is performed with random seed as selecting a particular seed and improving accuracy is also considered as an unstable model. So, in the interest of better long run performance on unseen data, this parameter is not tuned.

Feature reduction is also performed by removing the PCA variables which are considered of less importance (assumed that the components are present in the order of explaining the response Class variable. For example, V1 captures more information about the response variable than V2 and so on). Also, duplicate records are removed from the training dataset.

Metrics and Results:

As the dataset is imbalanced, measure of accuracy will always indicate a high score as the data will predict the majority class to perfection. Hence, use of a mixed metric that is sensitive to imbalance and one that can identify the mispredictions is necessary. One such measure is F1 score and so for the model predictions, the tests are measured primarily on this metric. Full classification table is also validated to observe the performance of a model.

There are several models that fit during the tuning process. Below table *Table 2* summarizes some of the successful milestones and shows details on initial, an intermediate and final setup with respect to hyper parameters.

Table 2 – Models Summary

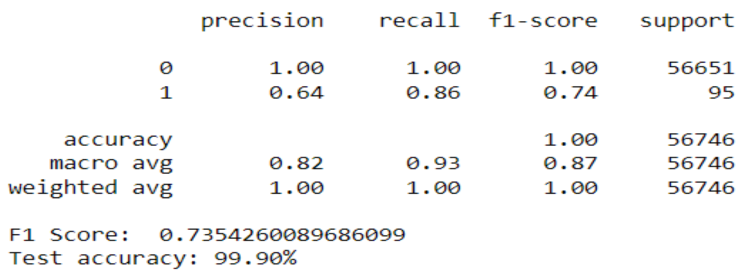
	Starting parameters	**Intermediate**	**Final**
Model Name	Neural Net Classifier (Basic)	Neural Net Classifier (Balanced data with SMOTE with all data including duplicates)	Neural Net Classifier (Balanced data with SMOTE, no duplicates)

Hyper parameters	RANDOM_SEED = 123
	
	LEARNING_RATE = 0.1
	
	BATCH_SIZE = 128
	
	NUM_EPOCHS = 25
	
	NUM_FEATURES = 28
	
	NUM_CLASSES = 2
	
	Adam optimizer
		RANDOM_SEED = 123
		
		LEARNING_RATE = 0.1
		
		BATCH_SIZE = 128
		
		NUM_EPOCHS = 20
		
		NUM_FEATURES = 30
		
		NUM_CLASSES = 2
		
		Adam optimizer
			RANDOM_SEED = 123
			
			LEARNING_RATE = 0.1
			
			BATCH_SIZE = 40000
			
			NUM_EPOCHS = 100
			
			NUM_FEATURES = 30
			
			NUM_CLASSES = 2

			Adam optimizer, dropout=0.02, weight_decay=0.00000000123
F1-Score	F1-Score – 66% True fraud detection (negative)– 80%	F1-Score – 66%-70% True fraud detection (negative)– 91%	F1-Score - 74% True fraud detection – 86%

For a visual understanding of the results, classification table from the final run and the confusion matrix are provided below in *Table 3* and *Fig 6*. As seen from the Classification table, the final f1-score is 74%.

Table 3 – Classification Table Summary



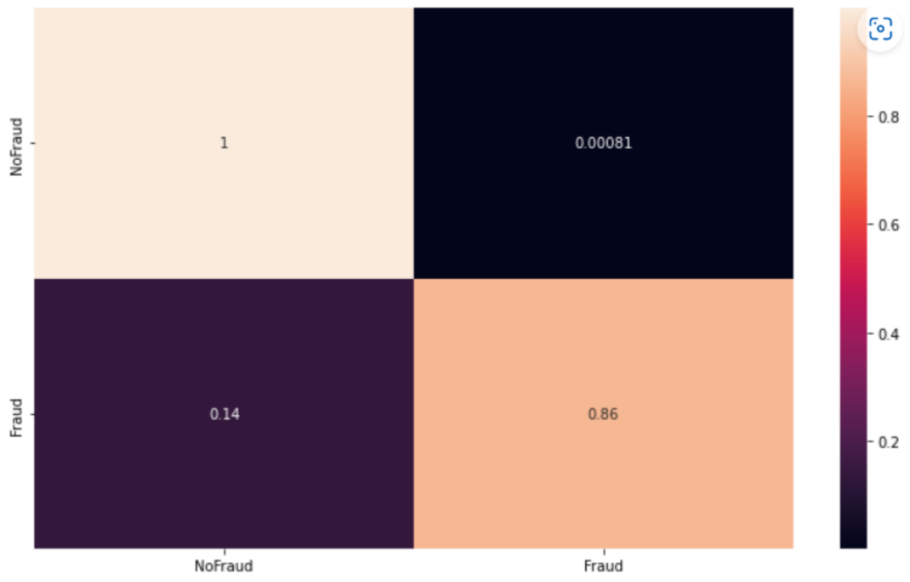
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.64      0.86      0.74        95

    accuracy                           1.00     56746
   macro avg       0.82      0.93      0.87     56746
weighted avg       1.00      1.00      1.00     56746

F1 Score:  0.7354260089686099
Test accuracy: 99.90%
```

Fig 6 – Confusion Matrix



The model detects about 86% of the true fraud cases.

Following trials are executed as well during the process. 1. Learning rate – Changed from 0.1 to 0.01, 0.001 and also to 1. 2. Optimizers – Changed from Adam to SGD, NAdam. 3. Activation function – Change from Tanh to Relu6. 4. Batch size – Changed to 40000, 4000, 400. 5. Dropout – Changed to 0.2, none, 0.1 and 0.5. 6. Hidden layers and nodes – Increased and decreased hidden layers and nodes. 7. Weight decay – Changed to 0.00000000123, none and other values. The changes did not help in improving the performance of the model. As seen from the Summary table above, introducing drop out, weight decay and increasing the EPOCH helped to gain accuracy roughly from 66% to 74% on balanced data.

Conclusion:

The base models were built using multi-layer perceptron and simple deep neural network models. Neural network with TanH activation function performed well for this dataset and hyper parameters learning rate, EPOCH, batch size, optimizer, dropout and weight_decay parameters are added and tuned which increased F1-score of the model from 66% to 74%. Many iterations are executed on the model with different values. Layers are also added and removed but changes related to layers and nodes did not help to improve the F1-score further. CNNs were also tried but there were mixed reviews on using CNNs for non-image data. There is also less material and examples available to model non-image datasets with Pytorch and CNN. The team is not experienced with anomaly detection or unbalanced datasets, so this was a new learning to analyze and select the sampling techniques to handle imbalance. Towards end of the project, 74% is the highest F1-score achieved. This gives a good start for fraud detection with credit cards and we understand there are more opportunities to improve the score further by training the model by using train-validation-test datasets and increasing EPOCHs further using GPUs. It is to be noted that the final model takes approximately 30 minutes on a standard laptop and Jupyter notebook.

References:

1. Dataset - https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
2. Background business problem - https://fraud-detection-handbook.github.io/fraud-detection-handbook/Chapter_2_Background/CreditCardFraud.html#credit-card-fraud-scenarios
3. Advantages of machine learning algorithms in credit card anomaly detection - https://www.infosysbpm.com/blogs/bpm-analytics/machine-learning-for-credit-card-fraud-detection.html#:~:text=How%20can%20machine%20learning%20help%20with%20credit%20card%20fraud%20detection,probability%20of%20fraud%5B2%5D.
4. Data sampling for class imbalance - https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0192-5#ref-CR117
5. Confusion matrix - https://christianbernecker.medium.com/how-to-create-a-confusion-matrix-in-pytorch-38d06a7f04b7
6. Classification report - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
7. Batch for pytorch for non-image data - https://machinelearningmastery.com/training-a-pytorch-model-with-dataloader-and-dataset/
8. For the idea of using sklearns feature selection: - https://www.kaggle.com/code/nareshbhat/fraud-detection-feature-selection-over-sampling
9. https://towardsdatascience.com/pytorch-tabular-regression-428e9c9ac93
10. https://www.kaggle.com/code/renanmour4/credit-card-fraud-detection
11. https://www.kaggle.com/code/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets
12. https://www.kaggle.com/code/drscarlat/fraud-detection-under-oversampling-smote-adasyn