Credit Card Fraud detection using Machine Learning and Deep Learning models

Subject: CM3070
Student: 190128812

Template: Machine Learning and Neural Networks – Public dataset

Table of contents

Introduction.. 2
Literacy review... 5
Project design... 12
Domain and users... 12
Project structure... 12
Work plan... 14
Evaluation.. 15
Technologies and methods.. 15
Project Prototype.. 17
Appendix.. 20
References.. 24

Introduction

Frauds perpetrated in digital payments, and in particular using credit cards, are a constant threat to users and cause damages for billions of dollars every year.

According to the 2022 report by Nilsson[14], an authoritative provider of data and news on the payment card industry, losses due to payment card fraud amount to approximately USD 32 billion worldwide. The forecast for the next 10 years is even more worrying, as a total loss of USD 397 billion is estimated. If we consider the phasing out of cash and the continued growth of digital payments, we can agree that fraud attempts are set to rise.

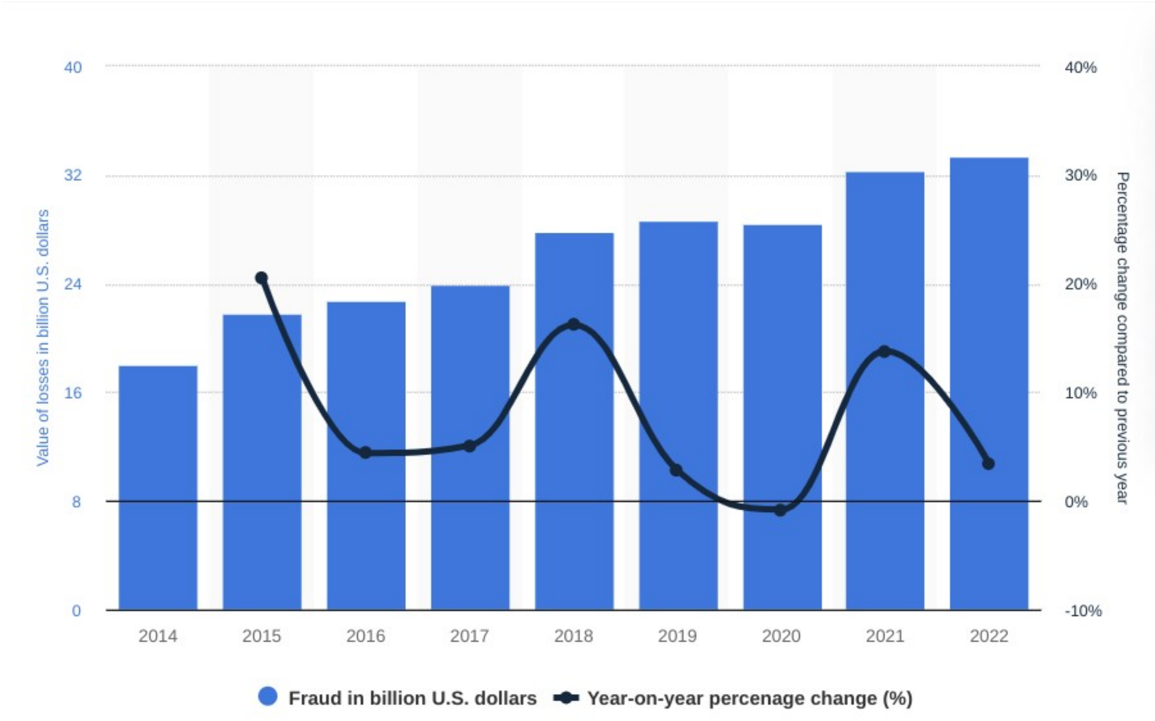


Figure 1: Credit card losses worldwide. https://www.statista.com/statistics/1394119/global-card-fraud-losses/

The ability to make payments online has opened up new possible scenarios for fraudsters, who no longer have to make the payment in person with a stolen card, running the risk of being stopped upon verification of their identity, but can perpetrate their criminal activities anywhere in the world an Internet connection is available.

This type of payment is called 'Card Not Present' (CNP) and is by far the most common method used in fraud attempts.[6]

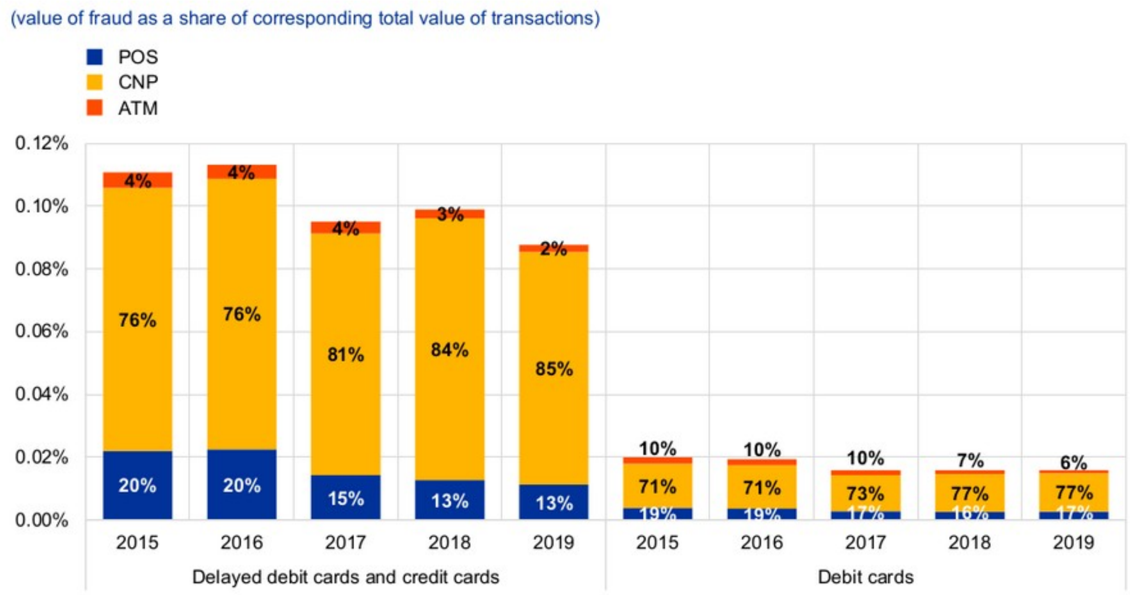


Figure 2: Type of Credit Card Fraud [6]

It is therefore important to have a secure system that allows users to use their credit cards in complete safety; this implies the need for an efficient system to detect fraud in real time.

The idea of this project was born with the aim of verifying the effective use of a fraud detection model, in a real scenario, where not only an excellent performance in detecting malicious transactions but also an extreme speed of transaction analysis is required.

My interest in this area stems from my current work situation, as I am employed in a multinational digital payments company; therefore, this project could lead to new career opportunities within my company. However, I am also a frequent user of payment cards, both in person and online, and I am aware of the risks associated with these types of transactions.
The study of fraud prevention methods today, therefore, also directly concerns me from a business perspective, and I set out to understand what methods financial institutions use to prevent fraud and how these methods can be improved.

Most companies that handle digital payments use control systems that analyse each recorded transaction in order to detect fraud attempts. The vast majority of software [15] [16] [17] [18] [19] currently on the market makes use of the latest artificial intelligence techniques. Being commercial and not open source software, they do not, however, provide detailed information on their models used, and merely mention techniques or algorithms.

To understand in detail what the state of the art is in this area, however, it is possible to read the available research, where Machine Learning (ML) and Deep Learning (DL) techniques and algorithms have been used to obtain the results available today.

Starting with the research I found most interesting, I will create, by combining various ML and DL techniques and algorithms, an unsupervised Deep Neural Network (DNN) that is capable of detecting

credit card fraud (CCF) with results possibly at least equivalent to those obtained in the research read, and that is fast enough in analyzing new transactions that it can be used in a production system.
In order for the fraud detection system to be used in a real-world scenario, it will have to:

- be effective in detecting fraud
- produce few false positives
- be fast in classifying payment transactions.

The importance of generating few false positives hinges on the fact that these types of transactions require subsequent analysis, probably by an operator following a user report, who will need to check the actual legitimacy of the transaction. This entails a cost to the payment processing company, which although less than the figures detected in fraud, can still have an important impact, especially if the transactions classified as False Positives, were of a significant amount.

Since transactions have to be executed very quickly, a system that is too slow in analyzing new transactions would unacceptably slow down operations and would in fact be unusable in a production system. If we consider that the number of transactions processed every second is in the order of thousands [20] [21], we can guess how crucial it is to have components working in real or near real time.

Literacy review

The state of the art in credit card fraud (CCF) detection models involves the use of unsupervised deep neural networks (DNNs). The unsupervised learning method eliminates the need for prior classification of transactions, giving the system the ability to be able to constantly update itself by training on new data. A supervised system, on the other hand, will require classification of new data before it can be integrated into the model; this obviously involves a certain amount of time, proportional to the amount of data to be classified.

Machine Learning and Deep Learning models, perform best when the dataset is balanced, that is, contains a similar number of elements, for each class contained. The number of frauds though is significantly small compared to the total volume of transactions; the database used in my project [22] contains about 0.17% of transactions classified as frauds. This imbalance prevents the ML algorithms from performing well, and it is therefore necessary to use techniques to balance the dataset.

Two techniques can be used to overcome this problem: oversampling [10] and under-sampling [3] of data. Oversampling consists of creating new 'synthetic' records of the minority class, as similar as possible to the original ones, so as to have an equivalent number of data for the classes in the dataset. Instead, under-sampling is to reduce the number of records in the majority class until the number of records in the minority class is reached.

Fraud detection models have been used since the introduction of digital payments. Before artificial intelligence was used, experts in the financial sector manually entered a set of rules into their systems to detect fraud attempts; however, this system was difficult to maintain and required constant updating to adapt to the new techniques used by fraudsters.

The first ML models used had to deal with the problem of unbalanced datasets, which prevented their algorithms from performing well. Haibo He et al [10] compared the oversampling methods developed so far (SMOTE, ADASYN, SVM, etc.) for balancing data, comparing their effectiveness; an interesting part of their work lies in the analysis of the assessment metrics used, which must be evaluated depending on the characteristics of the dataset used; in the case of highly unbalanced datasets, in fact, some methods lose their effectiveness. They therefore call for the creation of a uniform benchmark for standardized performance assessments using curve-based methods (ROC, PR Curves, Cost Curves) for evaluation. This research is useful for my project because I was able to learn about evaluation techniques based on curves, which I did not know before; I will definitely introduce them as an evaluation method for the models I will create later.

Drummond el al. [3], [4] in their research analyze data under-sampling methods, evaluating their effectiveness with the Decision Tree Learner algorithm. Using the Cost Curves algorithm to evaluate the performance of their models, they verify a better classification result than oversampling methods. Only one algorithm and one dataset is tested in the research, and thus the effectiveness of their method under different conditions is to be evaluated. I find this work interesting, though, and will evaluate the use of an under-sampling technique in my project. The CCF dataset contains only about 400 fraud records out of a total of more than 280,000 records and is therefore extremely unbalanced; it will be interesting to see if reducing the number of records to such a low value will actually lead to better results.

Sisodia et al. [13] compared different oversampling and under-sampling methods and evaluated their effectiveness on three highly unbalanced datasets. The rebalanced data were used for training different ML algorithms, and different methods were also used for performance measurement, giving the possibility to have an accurate analysis of the created models. The interesting part of this research is the comparison made between the different models used, combining ML algorithms with data balancing techniques. In my project I can then decide to focus on the best performing techniques, leaving out the methodologies that gave the worst results. I can certainly use the information from this research, combining it with different ML and DL algorithms, which have not been addressed in this work; I can also include measurements regarding the timing required for training the data and analyzing new transactions.

Mrozek et al. [12] Instead, Mrozek et al. in their work focused on the evaluation of several ML algorithms (Logistic Regression, Random Forest, K-Nearest Neighbors and Stochastic Gradient Descent) using, however, only two oversampling methods: Random Under-sampling and Synthetic Majority Oversampling Techniques (SMOTE). This research, combined with those previously analyzed, broadens my field of knowledge, giving me a way to have an analysis of numerous ML algorithms combined with as many data balancing techniques. Starting from this study, I can create DL models, combining the different mentioned oversampling techniques.

Fiore et al. [7] analyzing the oversampling methods used up to that point, they verified that they generated a high number of false positives (FPs), that is, legitimate transactions classified as fraud. These transactions, while posing no real danger to the user, create problems for the user and the financial institution, which must check them to verify the actual legitimacy of the transactions.
They therefore created and tested an oversampling method that, using a Generative Adversarial Network (GAN), can generate synthetic data indistinguishable from the originals, reducing the number of FP. A GAN is composed of two feed-forward Neural Networks (NN), having multiple layers (Deep Neural Network) interconnected with each other.

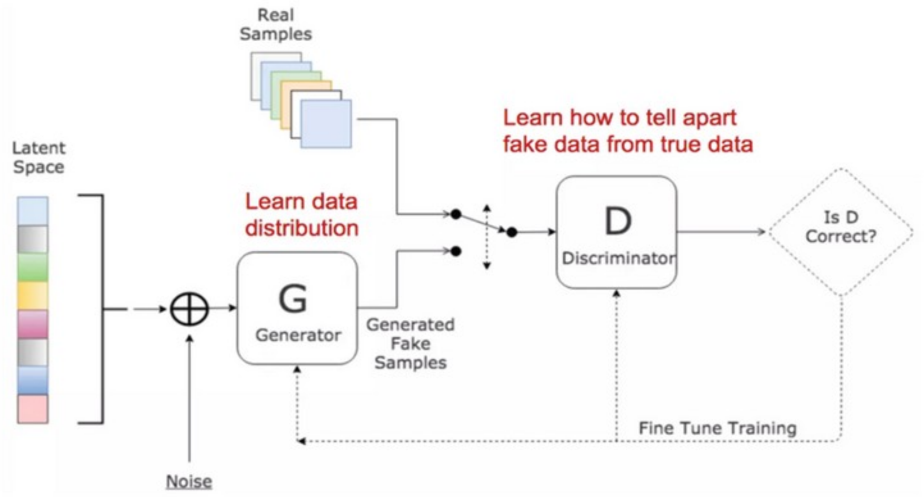
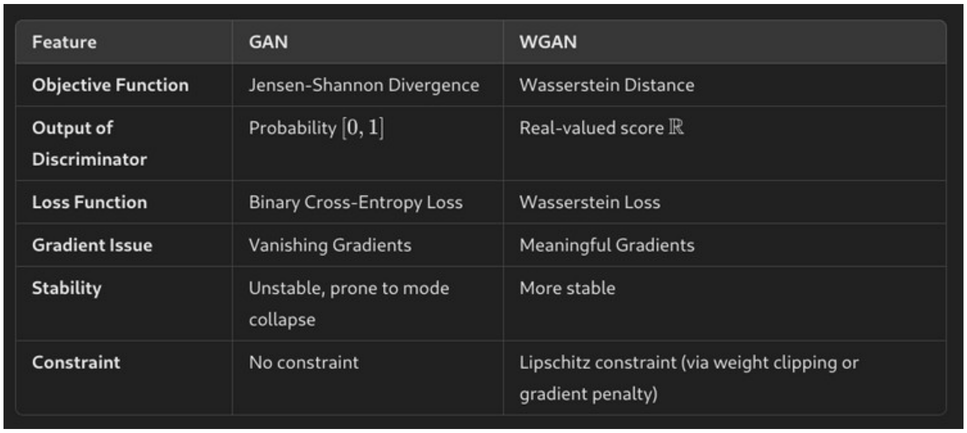


Figure 3: Generative Adversarial Network [8]

The Generator Neural Network (G) is in charge of creating the synthetic data, which is evaluated by the Discriminator (D), who will decide whether the quality of the data created by G is sufficient or not.

G will adjust its parameters based on D's evaluation in order to improve its output to a value considered suitable.
The results of their work show how the use of GAN neural networks has very similar performance to that obtained by SMOTE method, but significantly reduces false positives.
This research introduces a new method using DNN techniques for data balancing; I find this method very interesting and will use it in my project, although this will add difficulties, as it will require further study of this type of neural network. I will add an analysis of time performance related to data training, which is not reported in this paper. Also, I will combine this technique with a DNN for classification, which was not done in this study.

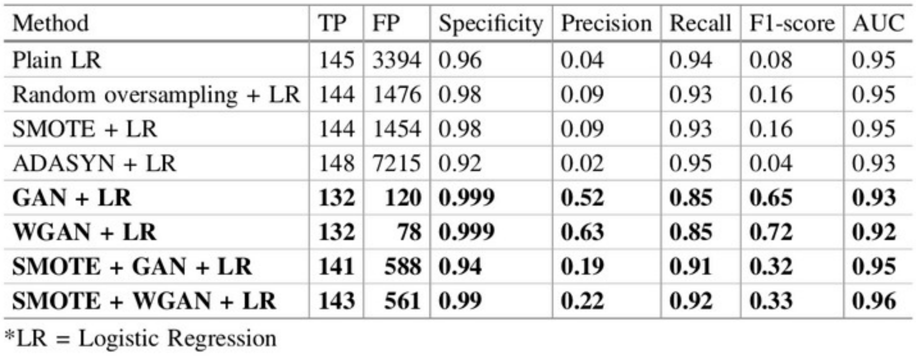
Gangwar et al. [8] proposed the use of a Wasserstein GAN (WGAN) [1] network for synthetic data generation. GANs are often used for realistic image generations, and while they offer great performance, they sometimes suffer from two main problems: mode collapse and convergence failure. In the case of mode collapse, the model fails to generate original data but tends to repeat the same data over and over again. For example, starting from the MNIST dataset that consists of handwritten numbers from 0 to 9, the GAN network might keep generating the same number, or only a subset of numbers. In the case of CCF records, it could create duplicate data or data with little difference between them.
In the case of convergence error, the generator and discriminator losses, instead of improving, diverge, fluctuate or fail to provide meaningful updates. If the discriminator is much stronger than the generator, it "wins" easily, providing no meaningful feedback to the generator.
Alternatively, if the generator becomes too strong, it might produce trivial outputs that fool the discriminator completely, causing the discriminator to fail. WGANs solve these problems by modifying certain parts of the neural network, as can be seen in the image below



Feature	GAN	WGAN
Objective Function	Jensen-Shannon Divergence	Wasserstein Distance
Output of Discriminator	Probability $[0, 1]$	Real-valued score $\mathbb{R}$
Loss Function	Binary Cross-Entropy Loss	Wasserstein Loss
Gradient Issue	Vanishing Gradients	Meaningful Gradients
Stability	Unstable, prone to mode collapse	More stable
Constraint	No constraint	Lipschitz constraint (via weight clipping or gradient penalty)

Figure 4: Differences between GAN and WGAN

The results of this research show excellent performance in fraud detection and a significant decrease in false positives. This means that high-quality synthetic data similar to the originals were generated through the WGAN network. The table below shows the results obtained with different oversampling techniques combined with the Logistic Regression algorithm.



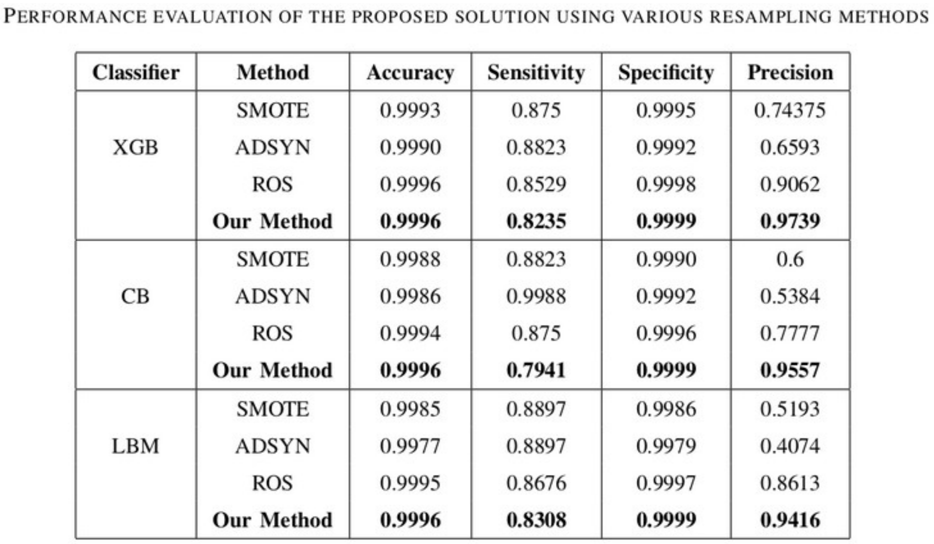
Method	TP	FP	Specificity	Precision	Recall	F1-score	AUC
Plain LR	145	3394	0.96	0.04	0.94	0.08	0.95
Random oversampling + LR	144	1476	0.98	0.09	0.93	0.16	0.95
SMOTE + LR	144	1454	0.98	0.09	0.93	0.16	0.95
ADASYN + LR	148	7215	0.92	0.02	0.95	0.04	0.93
GAN + LR	**132**	**120**	**0.999**	**0.52**	**0.85**	**0.65**	**0.93**
WGAN + LR	**132**	**78**	**0.999**	**0.63**	**0.85**	**0.72**	**0.92**
SMOTE + GAN + LR	**141**	**588**	**0.94**	**0.19**	**0.91**	**0.32**	**0.95**
SMOTE + WGAN + LR	**143**	**561**	**0.99**	**0.22**	**0.92**	**0.33**	**0.96**

*LR = Logistic Regression

Figure 5: Research results [8]

The objective of this research was to verify the effectiveness of the WGAN method compared with other data balancing methods; therefore, only two ML algorithms were used for data classification, and none for DL. There is also a lack of time performance analysis for model training and new data analysis. In my project I will test the effectiveness of a WGAN network combined with DL models for classification.

El Kafhali et al. [5] start from previous research on using GAN as oversampling, testing new ML algorithms, such as LightGBM (LBM), XGBoost (XGB), Cat-Boost (CB).



PERFORMANCE EVALUATION OF THE PROPOSED SOLUTION USING VARIOUS RESAMPLING METHODS

Classifier	Method	Accuracy	Sensitivity	Specificity	Precision
XGB	SMOTE	0.9993	0.875	0.9995	0.74375
	ADSYN	0.9990	0.8823	0.9992	0.6593
	ROS	0.9996	0.8529	0.9998	0.9062
	Our Method	**0.9996**	**0.8235**	**0.9999**	**0.9739**
CB	SMOTE	0.9988	0.8823	0.9990	0.6
	ADSYN	0.9986	0.9988	0.9992	0.5384
	ROS	0.9994	0.875	0.9996	0.7777
	Our Method	**0.9996**	**0.7941**	**0.9999**	**0.9557**
LBM	SMOTE	0.9985	0.8897	0.9986	0.5193
	ADSYN	0.9977	0.8897	0.9979	0.4074
	ROS	0.9995	0.8676	0.9997	0.8613
	Our Method	**0.9996**	**0.8308**	**0.9999**	**0.9416**

Figure 6: Research results [5]

The final results obtained confirm the performance of GAN in generating synthetic data and, while using different evaluation metrics, do not provide detailed information on the number of false positives generated. However, it is interesting to evaluate the performance of GAN for oversampling data, with ML algorithms not tested in previously read research.

Chen et al. [2] use a model consisting of a Sparse Auto Encoder (SAE) and a GAN network for fraud detection. Compared with classical classification methods, they use a one-class classification method, where only one class is considered for training the model, so that it is able to distinguish it accurately from other classes. This method solves the dataset imbalance problem, as the minority class is

excluded from the training data. Data from legitimate transactions are given as input to the SAE, which extracts the key features.
Auto Encoders (AEs) generally have only one hidden layer that contains fewer neurons than input neurons and are used for dimensionality reduction and feature extraction. SAEs, on the other hand, have a hidden layer that contains more neurons than input neurons, and this allows them to have better anti-noise capability.

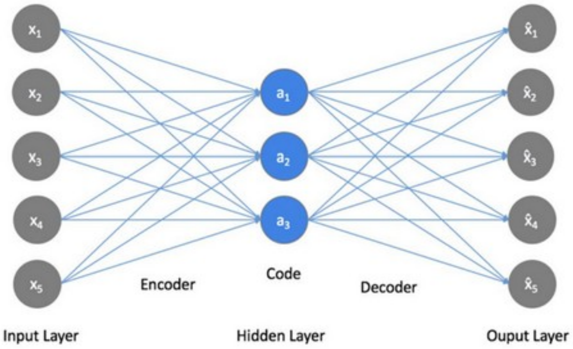


Figure 7: Normal Auto Encoder

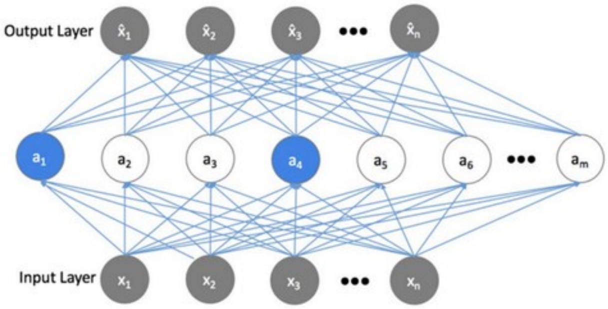
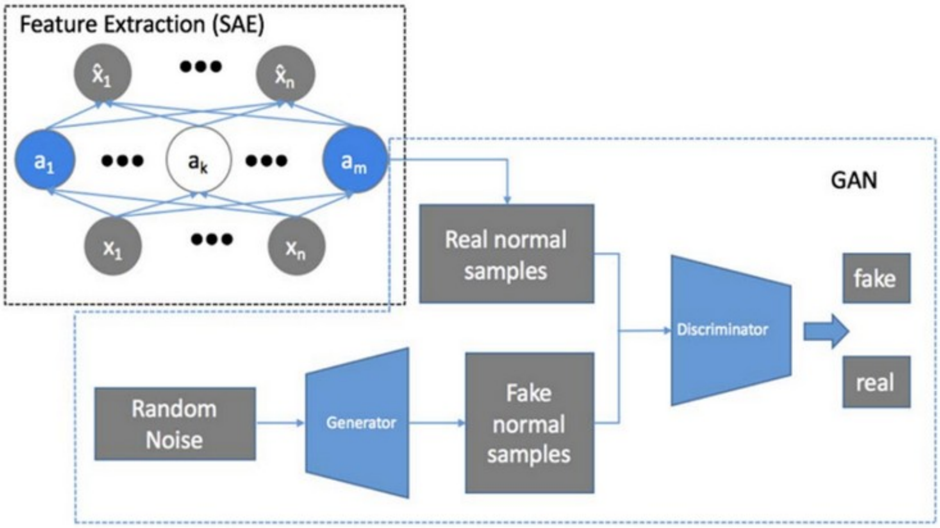


Figure 8: Sparse Auto Encoder

In the training phase, the 'output of the SAE, which contains the key-features of legitimate transactions, is given as input to the GAN network, where, the Generator takes random noise as input to generate synthetic data close to the real data, and the Discriminator, based on the features learned from the legitimate transactions, will have to decide whether the generated data are real or fake.
For testing, the new data is given as input to the SAE, which extracts their features and then the GAN Discriminator will classify it as legitimate or fraud.



This research is interesting for several aspects: I did not know the one-class classification method, which can be useful in the case of unbalanced datasets; furthermore I deepened my knowledge of Auto Encoders, which I will use in my project and I was able to study the use of GAN networks, not for oversampling the data, but for their classification.

Although GAN networks will certainly be part of my project, I do not think to use them for classification, but only for balancing the dataset. It would have been interesting also to have an analysis of the time performances for the training of this model.

Jiang et al. [11] propose the use of a Denoising Auto Encoder, to eliminate the 'noise' introduced by the database oversampling process. A DAE is in fact able to eliminate the noise generated by the oversampling process, so that the newly generated transactions are as close as possible to the original transactions, reducing the generation of transactions classified as False Positives.

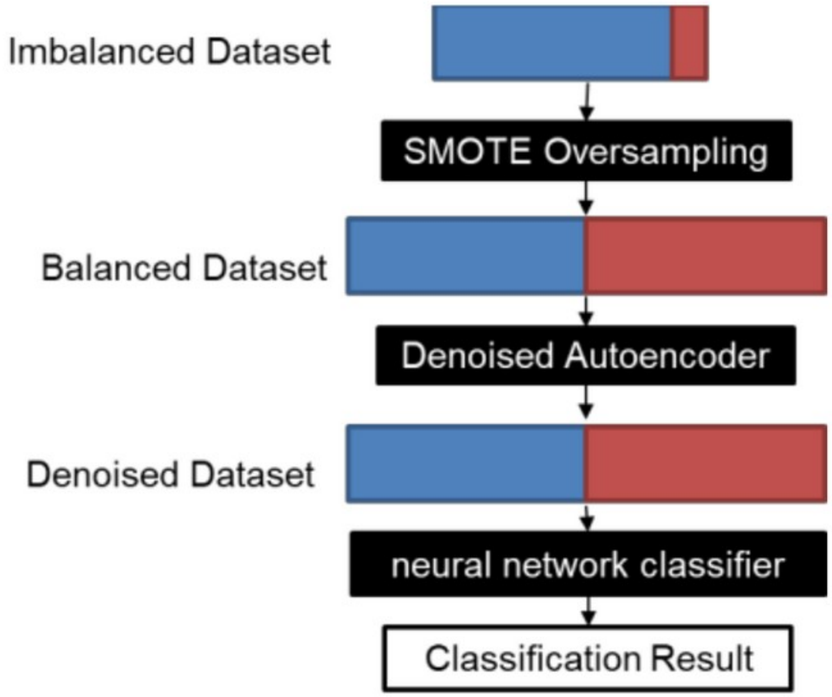


Figure 9: Proposed model [11]

SMOTE was used to balance the data and the resulting dataset was used as input to the DAE for data cleaning. The Auto Encoder is therefore not used for classification, but as an intermediate layer for the treatment of training data. The output of the DAE is then used as input to a 6-layer DNN that uses SoftMax with cross-entropy as loss function for the final classification.
The results show good results in accuracy and recall but are not compared with results obtained using other techniques and algorithms. It is therefore not clear whether this model improves fraud detection compared to others.
It is interesting to note a new way of using an Auto Encoder, not as a classifier but as a method for cleaning the dataset after oversampling. This research also uses a DNN for classification, which can be a starting point for my project.

Gomez et al. [9] introduce a data aggregation process, with the aim of identifying behavior patterns in users' payment transactions. The model tries to detect an anomaly in users' habits, such as the transaction amount or the transaction time, to identify possible fraud attempts.

The database used consists of more than 900 million records collected over an 18-month period.
To overcome the data balancing problem, two cascading filters are used; the goal is to perform an under-sampling of the data, eliminating as many legitimate transactions as possible, while keeping the fraudulent transactions. The ratio between fraud and legitimate transactions is thus reduced from 5000:1 to 100:1.
The dataset is then analyzed by a MultiLayer Perceptron (MLP) for classification.

Two new methods are introduced to measure performance: Value Detection Rate (VDR) and the True False Positive Ratio (TFPR). VDR is the ratio between the amount of transactions detected as fraud and the total amount of fraud; TFPR represents the cost of transactions classified as false positives.

The results obtained are good even if they vary from month to month, probably because payment transactions can vary a lot in some periods of the year.
Unfortunately, it is not possible to make a real comparison between the performances of this model and those of other research because the database used is not public. Furthermore, the dataset contains some data related to the user (card number) and the time of payment, which are used to perform a data aggregation and detect patterns.
This does not pose ethical problems, since the dataset remains private, but it differs from the database that I use in my project where there is no sensitive information, and the time information is not related to the date of the transaction.
Although this research is interesting for the method used for data balancing, I will not be able to use the same aggregation techniques, since I do not have the same type of data available; I can however use the VDR and TFPR methods to measure performance.
Although this research mentions the problem of having a system with short analysis times, it does not report such an evaluation for the created model.

Project design

Reading research related to CCFDs has been helpful in understanding what the state of the art is in this area. The main problems present in this field are due to both the strong imbalance in the dataset and the continuous evolution in the techniques used by fraudsters, which can lead to concept drift. Concept drift occurs when a model stops performing because the patterns in the dataset have changed. It would be easy enough to detect fraud attempts if a stolen credit card, making small-amount transactions, were suddenly used for payments of the maximum allowable amount. Instead, it is more likely that the fraudster performs multiple transactions of non-suspicious amounts in a manner similar to legitimate transactions. In addition, payment technology is evolving rapidly, and new possibilities for users are being introduced, corresponding to new attack surfaces for fraudsters.

For this reason, my hypothesis is that an unsupervised system of CCFD is superior to supervised models. The elimination of the transaction classification step allows new data to be continuously introduced into the model, which will thus be able to detect the most up-to-date fraud patterns. Billions of payment transactions are executed each year, and classifying them would take a long time; this, as mentioned earlier, could lead to the concept drift phenomenon.

Therefore, the ultimate goal of my project is to create an unsupervised Deep Neural Network model that can analyze unclassified data and is fast enough to be used in a production environment. The time factor is very important in an industry where thousands of operations are performed every second [20] [21] , with response times in the millisecond range. So it will not be enough to have good results in fraud detection, and to have a low number of false positives, but it will be equally important to evaluate the speed of the system.

Domain and users

The project falls within the domain of digital credit card payments and the software developed will be installed on a server owned by a banking institution or a company that handles digital payments. The users, in the strict sense of the term, will be the companies themselves, who will use the software for real-time fraud detection. In reality, however, the concept of user extends to all persons who will use, albeit transparently and unknowingly, their credit card to make a payment online or at a physical Point Of Sale (POS).

Project structure

In the first phase of the project I will need to analyze the dataset [22] and check for sensitive user data or outliers that could impact the performance of the algorithms used.

Next I will have to handle the problem of strong data imbalance by evaluating different techniques. My hypothesis, as a result of reading previous research, is that GAN networks, or its variants (WGAN, etc.) are the best method for creating synthetic data that will lead to fewer false-positive classifications. To test my hypothesis I will start with using other oversampling methods (SMOTE, ADYSIN etc,) to have a benchmark.

For performance testing I will initially create models using different ML algorithms, such as Logistic Regression, kNN, Naive Bayes and others, combining them with the oversampling methods described above to find the best combination of techniques. The goal of this phase is to test whether oversampling with GAN networks lead to a better results in terms of fraud detection but also fewer false positives. I will also test an under-sampling technique, which in my opinion with this dataset does not perform well, given the low number of records in the minority class; it will be interesting to evaluate whether my hypothesis is correct.

In the next phase I will use Deep Learning techniques to create an unsupervised DNN. I will use AutoEncoders for classification, subsequently testing some of its variants, such as the Denoising Auto Encoder. I will also evaluate the use of the latter as an intermediate layer for the 'cleaning' of data following oversampling.

Among the evaluation metrics that I will use, in addition to the classic precision, accuracy and F1, I will also include curved-based ones. I will also use the confusion matrix to verify the number of false positives created with the different algorithms.

For all the models created I will also analyze the time performances, both for training and for the analysis of test data.

Once I have all the data, I will be able to verify whether my initial hypothesis about DNNs and oversampling methods is correct.

In the figure below you can see a diagram of the project.

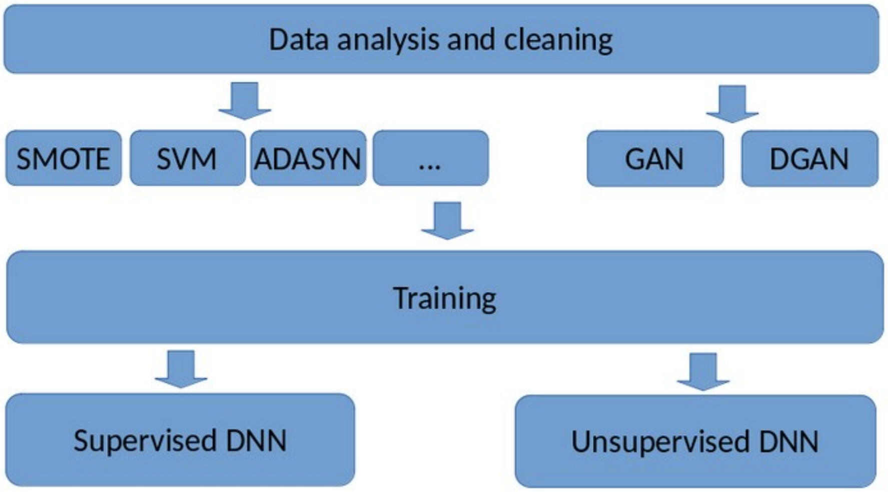


Figure 10: Project structure

During the code development phase I will continue to read new research, so there may be some changes or additions to the initial design. I do not expect to revolutionize the process, but I may use some new techniques learned in the new research analyzed.

Work plan

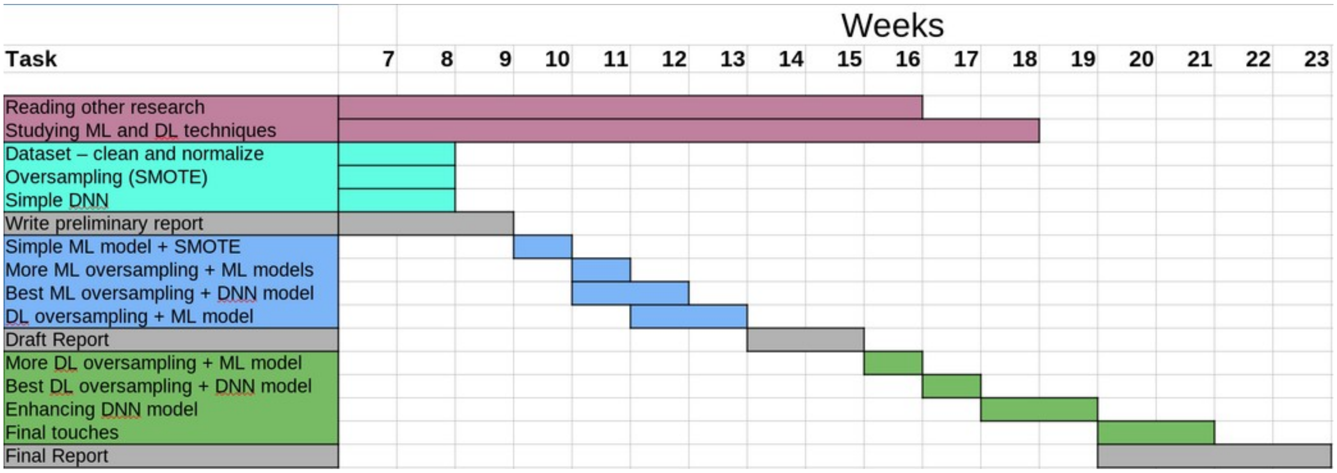
The difficulty of this project lies in the complexity of the algorithms and techniques of ML and DL.
My experience in this field is limited to the courses taken during the academic path, and therefore a good number of hours of study will be necessary to be able to apply techniques of which I have a basic knowledge or no knowledge; this time required must therefore be planned precisely in the work plan.
Furthermore, it will be challenging to reproduce the results obtained in the research read, without having any code; I will have to create many models from scratch, using different algorithms.

In my work plan I have therefore decided to continue with the reading of the research up to week 16, and with the study up to week 17; at that point I will no longer introduce any new techniques or algorithms and will concentrate on improving the model created up to that point.

In parallel I will obviously have to continue with the development of the fraud detection system.
For the preliminary report, since the development of the most difficult part of the project is required, I will create a Deep Learning Network for fraud detection, but using SMOTE for oversampling the data, instead of using a GAN for this purpose, which is the final goal.

In the following weeks I will create a Machine Learning model using different statistical algorithms for evaluation, evaluating the performance of other data oversampling techniques, until the creation of a GAN network that performs this purpose.

Starting from week 17, I will start building the final model, evaluating the best method of oversampling the data, in combination with an unsupervised DNN for fraud detection.
The last weeks will be used for writing the final report and for correcting and improving the project details.



	Weeks																
Task	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Reading other research	■	■	■	■	■	■	■	■	■	■							
Studying ML and DL techniques	■	■	■	■	■	■	■	■	■	■	■	■					
Dataset – clean and normalize	■	■															
Oversampling (SMOTE)	■	■															
Simple DNN	■	■															
Write preliminary report	■	■	■														
Simple ML model + SMOTE				■													
More ML oversampling + ML models					■												
Best ML oversampling + DNN model					■	■											
DL oversampling + ML model						■	■										
Draft Report								■	■								
More DL oversampling + ML model										■							
Best DL oversampling + DNN model											■						
Enhancing DNN model												■	■				
Final touches														■	■		
Final Report														■	■	■	■

Evaluation

The models created will be evaluated on three different aspects:

- Performance in fraud detection
- Number of false positives generated
- Speed in training and classification of test data

As I pointed out earlier, the evaluation of the best model will necessarily have to take into account the speed of analysis of the test data. A model evaluated too slow, that is, not capable of analyzing a new transaction on the order of milliseconds, will not be able to be used in a real-world scenario.

For the evaluation of the detection performance, I will use different methods (accuracy, precision, F1, curve-based), also analyzing the confusion matrix to check the amount of false negatives (FN) and false positives (FP) detected, which, although in different terms, represent a cost for the payment management company.

Initially I will compare the performances of the models created to find the best combination between oversampling technique and ML algorithm for fraud detection.
The comparison will take into great consideration the time performances, eliminating the models considered too slow, even if with better detection performances.

Then I will evaluate the use of GAN and WGAN for data oversampling, and compare them with the results obtained by the other methods. The final model will use the best evaluated oversampling technique, in combination with a Deep Neural Network.

The comparison will test whether Deep Neural Network models prove superior to Machine Learning models for this specific purpose, and which combinations of techniques are most effective.

Technologies and methods

This project will be developed in a Jupyter Notebook, where descriptive parts will alternate with parts of code. The descriptive parts will serve to clarify the purpose of the project, explain its various steps, introduce theoretical concepts and present final conclusions.

The code part will be written in Python, using native and third-party libraries for the realization of artificial intelligence models.

The time performances of the created models will be relative to the hardware I have available in my PC:
CPU: Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
GPU: none
RAM: 32 GB
OS: Debian 12 bookworm

If I have time available, I will compare the performances with those obtained on my laptop, which has a less performing processor but has a GPU; I will probably have to modify the code to use libraries optimized for GPU use.

Some of Python Libraries used:
Keras, TensorFlow, Numpy, Sklearn, Pandas, Matplotlib

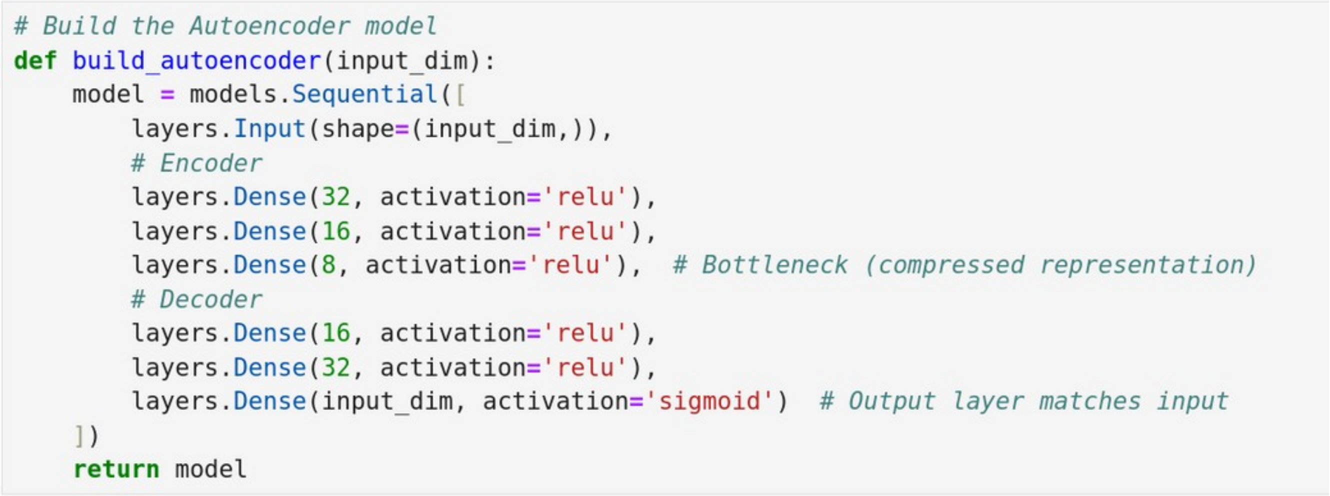
Some of the techniques and algorithms used:
Oversampling techniques (SMOTE, ADASYN, GAN, WGAN, …)
Logistic Regression, SVM, kNN, AdaBoost, etc.

Evaluation metrics:
Precision, Recall, F1, curve-based methods

Project Prototype

In the presented prototype I developed what I think is the most complex part of the project: an unsupervised DNN for fraud classification. The difficulty lies both in the complexity of the topic and in my lack of experience with these techniques, which therefore required some prior study before I was able to create such a model.

For DNN I used an AutoEncoder (AE), an NN that is also used in cases of anomaly detection, in this context represented by fraud transactions.

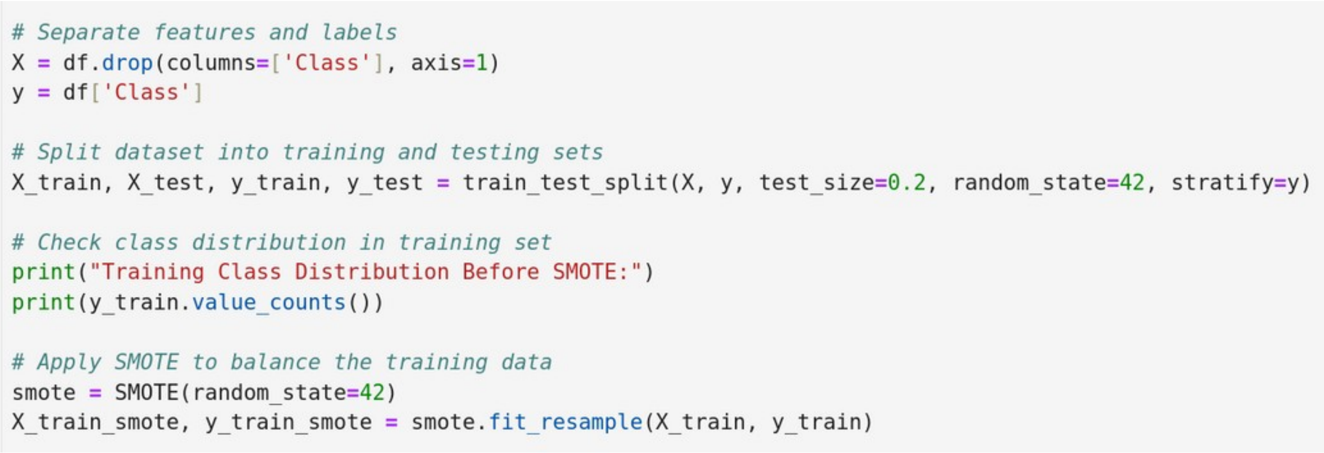


```python
# Build the Autoencoder model
def build_autoencoder(input_dim):
    model = models.Sequential([
        layers.Input(shape=(input_dim,)),
        # Encoder
        layers.Dense(32, activation='relu'),
        layers.Dense(16, activation='relu'),
        layers.Dense(8, activation='relu'),  # Bottleneck (compressed representation)
        # Decoder
        layers.Dense(16, activation='relu'),
        layers.Dense(32, activation='relu'),
        layers.Dense(input_dim, activation='sigmoid')  # Output layer matches input
    ])
    return model
```

I set that number of layers based on the features contained in the dataset. Later I will try to modify them to see if the performance can be improved.

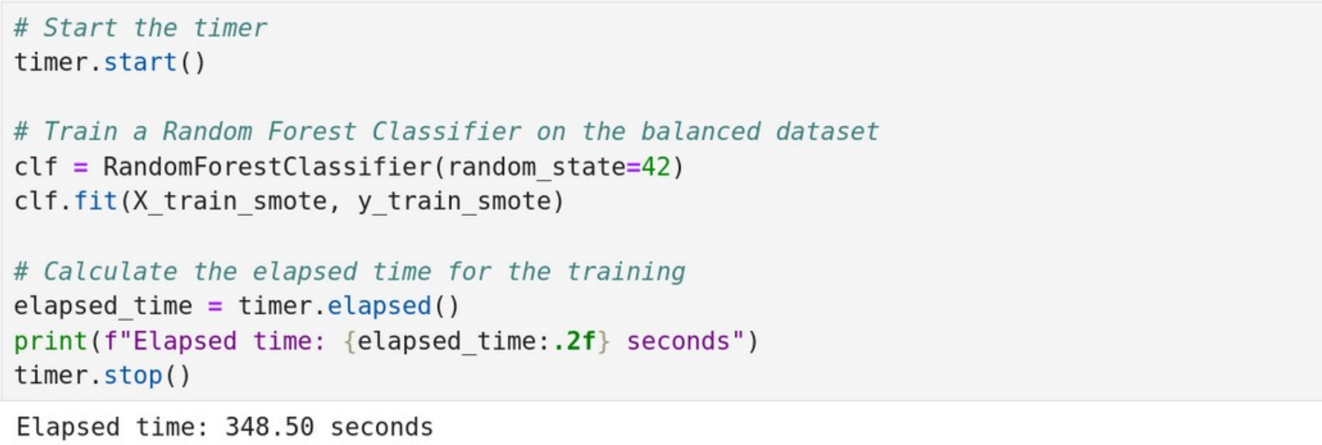
The AE's Encoder, trained with only legitimate transactions, compresses the input into a lower-dimensional representation in order to extract the most important features. The compressed data is then reconstructed trying to have a result as close to the original as possible. Based on the reconstruction errors, the AE modifies its parameters until it achieves the best possible result.
Once the AE's training phase is finished, it is given the entire dataset as input, also containing the fraud transactions that, having not been analyzed before, should have a larger reconstruction error than the legitimate ones.

Before I could create the AE though, I had to solve the problem of high imbalance that distinguishes the dataset used. The goal of the project is to test different oversampling methods to see if using a GAN (or its variant) is actually superior in creating synthetic data. Superior means that the data created are as close as possible to the original data and do not generate false positives.
For the prototype, though, I used SMOTE, a method that is easier to implement and widely used; this choice was made as a matter of time, since I would have to study the operation and implementation of GANs in detail. However, the code I wrote will be used in later stages of project development, in which I will test the performance of other balancing methods, comparing them to each other to confirm my theory about GANs.



```
# Separate features and labels
X = df.drop(columns=['Class'], axis=1)
y = df['Class']

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Check class distribution in training set
print("Training Class Distribution Before SMOTE:")
print(y_train.value_counts())

# Apply SMOTE to balance the training data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

In the prototype I also used a RandomForest classifier that will be compared with other classifiers that use both ML and DL algorithms.

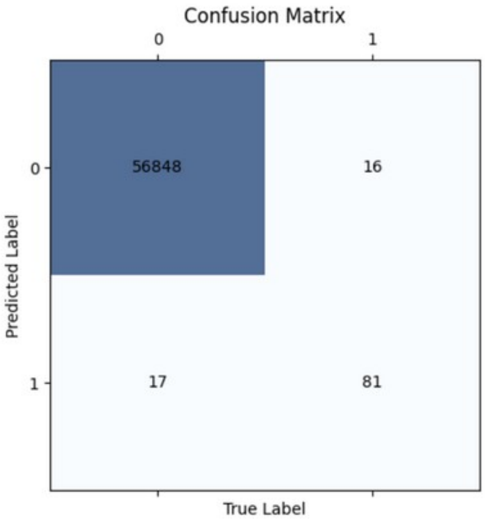


```
# Start the timer
timer.start()

# Train a Random Forest Classifier on the balanced dataset
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_smote, y_train_smote)

# Calculate the elapsed time for the training
elapsed_time = timer.elapsed()
print(f"Elapsed time: {elapsed_time:.2f} seconds")
timer.stop()
```

```
Elapsed time: 348.50 seconds
```

As you can see, the time needed to train the dataset was about 348 seconds.

For performance evaluation, I used accuracy, precision and F1 values for the time being, as well as the confusion matrix to check the number of false positives classified. Later I will also introduce curve-based methods.



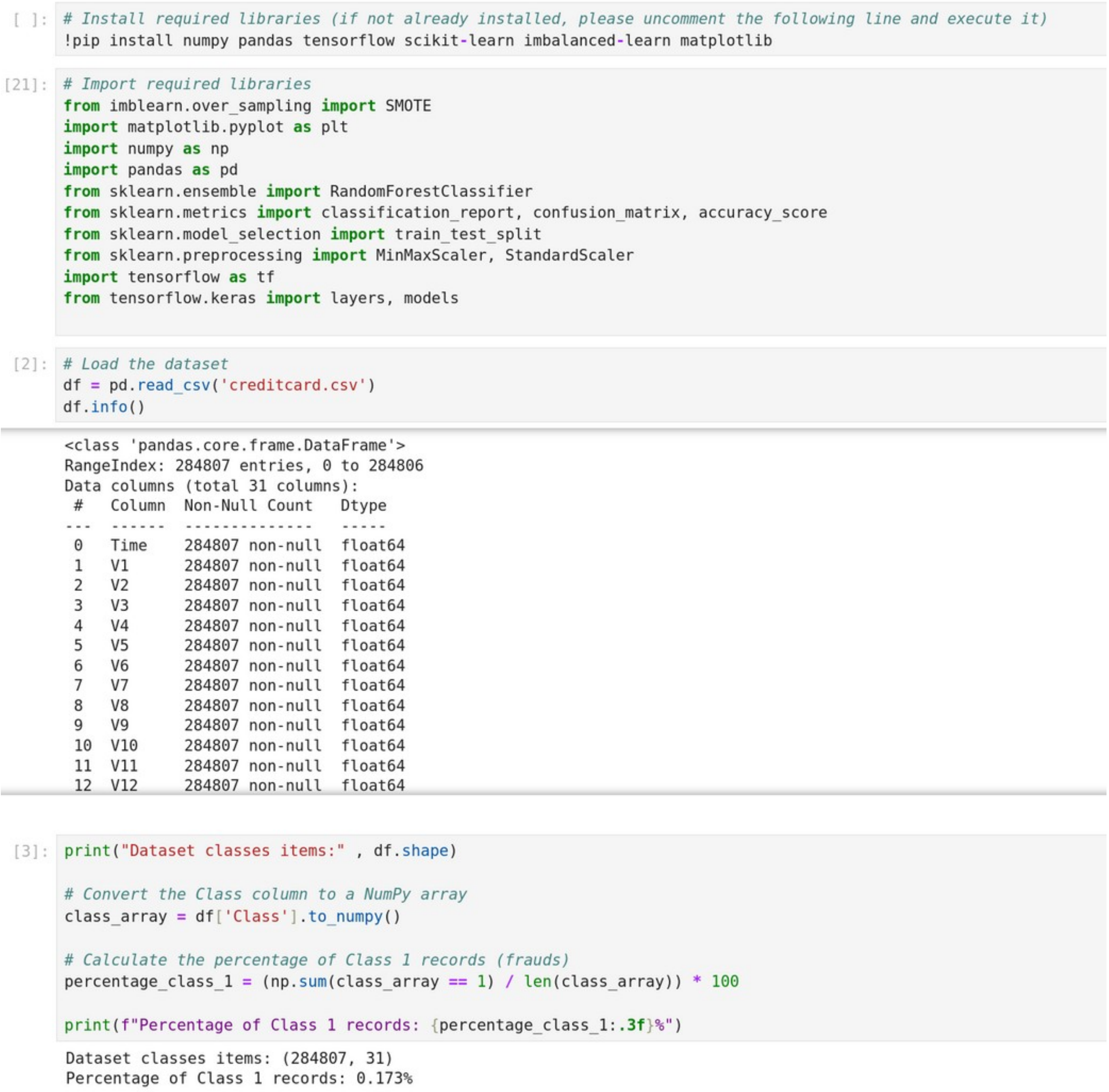
For time performance evaluation, I created a simple class that calculates the seconds taken by a piece of code. I used this class to evaluate the time taken to train both the RandomForest classifier and the Auto Encoder.

In the next stages of the project, I will continue with the analysis of different oversampling methods (and one under-sampling method), testing them in combination with some ML algorithms. Then I will use different variants of GAN, combining them with the same algorithms evaluated earlier, and, finally, I will focus on developing a DNN.
The ultimate goal is the creation of an unsupervised DNN that has similar or better performance than the other models created.
As I continue reading more research, it's possible that the work plan will change, and new techniques or algorithms will be introduced.

The code was written and tested in a Jupyter Notebook, using third-party libraries for the ML and DL algorithms. I have not used any GPU-optimized libraries for the time being, as I do not have one available on my PC; however, I do not exclude adapting the code for this purpose and running the tests on a laptop of mine with a less performing CPU, but with a dedicated GPU.

Appendix

Python code for the prototype



```
[ ]: # Install required libraries (if not already installed, please uncomment the following line and execute it)
     !pip install numpy pandas tensorflow scikit-learn imbalanced-learn matplotlib
```

```
[21]: # Import required libraries
      from imblearn.over_sampling import SMOTE
      import matplotlib.pyplot as plt
      import numpy as np
      import pandas as pd
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
      from sklearn.model_selection import train_test_split
      from sklearn.preprocessing import MinMaxScaler, StandardScaler
      import tensorflow as tf
      from tensorflow.keras import layers, models
```

```
[2]: # Load the dataset
     df = pd.read_csv('creditcard.csv')
     df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
```

```
[3]: print("Dataset classes items:" , df.shape)

     # Convert the Class column to a NumPy array
     class_array = df['Class'].to_numpy()

     # Calculate the percentage of Class 1 records (frauds)
     percentage_class_1 = (np.sum(class_array == 1) / len(class_array)) * 100

     print(f"Percentage of Class 1 records: {percentage_class_1:.3f}%")
```

```
Dataset classes items: (284807, 31)
Percentage of Class 1 records: 0.173%
```

Class for calculating the time taken to execute the code

[25]:

```python
import time

class Timer:
    def __init__(self):
        """Initialize the timer with no start time"""
        self.start_time = None

    def start(self):
        """Start the timer"""
        self.start_time = time.time()

    def elapsed(self):
        """Calculate the time elapsed since the timer was started"""
        if self.start_time is None:
            raise ValueError("Timer has not been started. Call `start()` before `elapsed()`")
        elapsed_time = time.time() - self.start_time
        return elapsed_time

    def stop(self):
        """Stop the timer and reset the start_time"""
        self.start_time = None

# Create the Timer object
timer = Timer()
```

Balancing the dataset

The dataset is very unbalanced, so I will use the SMOTE technique to make the two classes equivalent in the number of records.
The balancing of the dataset is done after its division into training and test data, so that the latter contain no 'synthetic' data.

[26]:

```python
# Display class distribution
print("Class Distribution Before SMOTE:")
print(df['Class'].value_counts())
df['Class'].value_counts().plot(kind='bar', title='Class Distribution (Before SMOTE)', xlabel='Class', ylabel='Count')
plt.show()

# Separate features and labels
X = df.drop(columns=['Class'], axis=1)
y = df['Class']

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Check class distribution in training set
print("Training Class Distribution Before SMOTE:")
print(y_train.value_counts())

# Apply SMOTE to balance the training data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Check class distribution after SMOTE
print("Training Class Distribution After SMOTE:")
print(pd.Series(y_train_smote).value_counts())

# Visualize the balanced class distribution
pd.Series(y_train_smote).value_counts().plot(kind='bar', title='Class Distribution (After SMOTE)', xlabel='Class', ylabel='Count')
plt.show()
```

[27]:

```python
# Start the timer
timer.start()

# Train a Random Forest Classifier on the balanced dataset
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_smote, y_train_smote)

# Calculate the elapsed time for the training
elapsed_time = timer.elapsed()
print(f"Elapsed time: {elapsed_time:.2f} seconds")
timer.stop()
```

```
Elapsed time: 348.50 seconds
```

[28]:

```python
# Start the timer
timer.start()

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Calculate the elapsed time for the training
elapsed_time = timer.elapsed()
print(f"Elapsed time: {elapsed_time:.2f} seconds")
timer.stop()

# Evaluate the model
print("Classification Report:\n", classification_report(y_pred, y_test))
print("Accuracy:", accuracy_score(y_pred, y_test))

# Visualize confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.matshow(conf_matrix, cmap='Blues', alpha=0.7)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        plt.text(x=j, y=i, s=conf_matrix[i, j], va='center', ha='center')

plt.title('Confusion Matrix')
plt.xlabel('True Label')
plt.ylabel('Predicted Label')
plt.show()
```

```
Elapsed time: 0.46 seconds
```

[43]:

```python
# Build the Autoencoder model
def build_autoencoder(input_dim):
    model = models.Sequential([
        layers.Input(shape=(input_dim,)),
        # Encoder
        layers.Dense(32, activation='relu'),
        layers.Dense(16, activation='relu'),
        layers.Dense(8, activation='relu'),  # Bottleneck (compressed representation)
        # Decoder
        layers.Dense(16, activation='relu'),
        layers.Dense(32, activation='relu'),
        layers.Dense(input_dim, activation='sigmoid')  # Output layer matches input
    ])
    return model
```

[44]:

```python
# Separate features and labels
# The Class feature is not required in an unsupervised learning model
X = df.drop(columns=['Class'], axis=1)  # Features
y = df['Class']  # Labels

# Normalize the features
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Separate normal (non-fraudulent) transactions for training the Autoencoder
X_normal = X_normalized[y == 0]

# Split the normal transactions into training and validation sets
X_train, X_val = train_test_split(X_normal, test_size=0.2, random_state=42)

# Initialize the Autoencoder
input_dim = X_train.shape[1]
autoencoder = build_autoencoder(input_dim)

# Compile the Autoencoder
autoencoder.compile(optimizer='adam', loss='mse')

# Start the timer
timer.start()

# Train the Autoencoder
history = autoencoder.fit(
    X_train, X_train,  # Input is the same as the target
    validation_data=(X_val, X_val),
    epochs=50,
    batch_size=64,
    verbose=1
)

# Calculate the elapsed time for the training
elapsed_time = timer.elapsed()
print(f"Elapsed time: {elapsed_time:.2f} seconds")
timer.stop()
```

[46]:

```python
# Use the Autoencoder to calculate reconstruction errors
X_reconstructed = autoencoder.predict(X_normalized)
reconstruction_errors = np.mean(np.square(X_normalized - X_reconstructed), axis=1)

# Set a threshold for anomalies based on normal transactions' errors
threshold = np.percentile(reconstruction_errors[y == 0], 99)  # 98th percentile

# Classify anomalies (fraud) based on reconstruction error
y_pred = (reconstruction_errors > threshold).astype(int)

# Evaluate the results
print("Classification Report:\n", classification_report(y, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y, y_pred))
print("Accuracy:", accuracy_score(y, y_pred))

# Visualize reconstruction error distributions
plt.figure(figsize=(8, 5))
plt.hist(reconstruction_errors[y == 0], bins=50, alpha=0.6, label='Normal')
plt.hist(reconstruction_errors[y == 1], bins=50, alpha=0.6, label='Fraud')
plt.axvline(threshold, color='red', linestyle='--', label='Threshold')
plt.xlabel('Reconstruction Error')
plt.ylabel('Frequency')
plt.legend()
plt.title('Reconstruction Error Distribution')
plt.show()
```

[47]:

```python
# Visualize confusion matrix
conf_matrix = confusion_matrix(y, y_pred)
plt.matshow(conf_matrix, cmap='Blues', alpha=0.7)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        plt.text(x=j, y=i, s=conf_matrix[i, j], va='center', ha='center')

plt.title('Confusion Matrix')
plt.xlabel('True Label')
plt.ylabel('Predicted Label')
plt.show()
```

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein GAN. https://doi.org/10.48550/arXiv.1701.07875

[2] Jian Chen, Yao Shen, and Riaz Ali. 2018. Credit Card Fraud Detection Using Sparse Autoencoder and Generative Adversarial Network. In *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, November 2018. IEEE, Vancouver, BC, 1054–1059. https://doi.org/10.1109/IEMCON.2018.8614815

[3] Chris Drummond, Chris Drummond, and Robert C Holte. C4.5, Class Imbalance, and Cost Sensitivity: Why Under-Sampling beats Over-Sampling.

[4] Chris Drummond and Robert C. Holte. 2005. Severe Class Imbalance: Why Better Algorithms Aren't the Answer. In *Machine Learning: ECML 2005*, João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge and Luís Torgo (eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 539–546. https://doi.org/10.1007/11564096_52

[5] Said El Kafhali and Mohammed Tayebi. 2022. Generative Adversarial Neural Networks based Oversampling Technique for Imbalanced Credit Card Dataset. In *2022 6th SLAAI International Conference on Artificial Intelligence (SLAAI-ICAI)*, December 01, 2022. IEEE, Colombo, Sri Lanka, 1–5. https://doi.org/10.1109/SLAAI-ICAI56923.2022.10002630

[6] European Central Bank. 2021. Seventh report on card fraud. (2021). Retrieved from https://www.ecb.europa.eu/pub/pdf/cardfraud/ecb.cardfraudreport202110~cac4c418e8.en.pdf

[7] Ugo Fiore, Alfredo De Santis, Francesca Perla, Paolo Zanetti, and Francesco Palmieri. 2019. Using generative adversarial networks for improving classification effectiveness in credit card fraud detection. *Inf. Sci.* 479, (April 2019), 448–455. https://doi.org/10.1016/j.ins.2017.12.030

[8] Akhilesh Kumar Gangwar and Vadlamani Ravi. 2019. WiP: Generative Adversarial Network for Oversampling Data in Credit Card Fraud Detection. In *Information Systems Security*, Deepak Garg, N. V. Narendra Kumar and Rudrapatna K. Shyamasundar (eds.). Springer International Publishing, Cham, 123–134. https://doi.org/10.1007/978-3-030-36945-3_7

[9] Jon Ander Gómez, Juan Arévalo, Roberto Paredes, and Jordi Nin. 2018. End-to-end neural network architecture for fraud scoring in card payments. *Pattern Recognit. Lett.* 105, (April 2018), 175–181. https://doi.org/10.1016/j.patrec.2017.08.024

[10] Haibo He and E.A. Garcia. 2009. Learning from Imbalanced Data. *IEEE Trans. Knowl. Data Eng.* 21, 9 (September 2009), 1263–1284. https://doi.org/10.1109/TKDE.2008.239

[11] Ping Jiang, Zhang Jinliang, and Zou Junyi. 2019. Credit Card Fraud Detection Using Autoencoder Neural Network. *Neurocomputing* 139, (2019), 84–96.

[12] Petr Mrozek, John Panneerselvam, and Ovidiu Bagdasar. 2020. Efficient Resampling for Fraud Detection During Anonymised Credit Card Transactions with Unbalanced Datasets. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, December 2020. IEEE, Leicester, UK, 426–433. https://doi.org/10.1109/UCC48980.2020.00067

[13] Dilip Singh Sisodia, Nerella Keerthana Reddy, and Shivangi Bhandari. 2017. Performance evaluation of class balancing techniques for credit card fraud detection. In *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, September 2017. IEEE, Chennai, 2747–2752. https://doi.org/10.1109/ICPCSI.2017.8392219

[14] payment-card-fraud-losses-reach-32-34.

[15] DataVisor. Retrieved from https://www.datavisor.com/products/fraud-platform/

[16] FICO Falcon Fraud Management. Retrieved from https://www.fico.com/en/products/fico-falcon-fraud-manager

[17] Fraud.net. Retrieved from https://fraud.net/

[18] RiskShield. Retrieved from https://www.inform-software.com/en/solutions/risk-fraud/fraud-prevention

[19] SAS Fraud Management. Retrieved from https://www.sas.com/en_us/solutions/fraud-security-intelligence.html

[20] Visa technology. Retrieved from https://corporate.visa.com/content/dam/VCOM/download/corporate/media/visanet-technology/visa-net-booklet.pdf

[21] *The Federal Reserve Payments Study: Cards and Alternative Payments, 2021 and 2022*. Retrieved from https://www.federalreserve.gov/paymentsystems/fr-payments-study.htm

[22] Credit Card Fraud Detection Dataset. Retrieved from https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud