UNIVERSITÀ DEGLI STUDI DI CATANIA

Dipartimento di Matematica e Informatica
Corso di Laurea Triennale in Informatica

Claudio Giusti

Detecting Credit Card Frauds via MLP

Machine Learning project

Professor:
Giovanni Maria Farinella

Anno Accademico 2023 - 2024

Contents

1 Introduction

This project, developed as part of the Machine Learning course, focuses on applying feature generation and transformation techniques to detect fraudulent activities using machine learning models. A key aspect of this work is a custom-generated dataset, designed specifically for fraud detection scenarios through a data generation process. Fraud detection is a critical application of machine learning, where accurately classifying fraudulent and legitimate transactions can help significantly reduce financial losses. In this project, we employ a Multilayer Perceptron (MLP), a type of feedforward neural network, as the primary model for fraud detection. The MLP is trained and tested on the custom dataset, with features that have been engineered and transformed to enhance the model's performance. Effective feature generation and transformation are essential for boosting the predictive power of machine learning models. By applying various techniques such as normalization, standardization, and polynomial feature generation, we aim to create data representations that enable the MLP to detect fraudulent transactions more accurately. The primary objective of this project is to demonstrate how effective feature engineering, combined with a robust classifier like the MLP, can significantly improve accuracy, precision, recall, and the overall performance of a fraud detection system. In Section 2, we explain the process of generating transactional data, detailing the functionality of each component and how they cooperate to build a reliable dataset. Chapter 4 covers the transformation of features, adapting them to enhance the model's predictive capabilities. Section 5 outlines the approaches used for training the networks and optimizing the results, while Chapter 6 analyzes the results obtained from the various networks, identifying the most effective solutions for addressing the fraud detection challenge.

2 Prerequisites

This section outlines the key techniques and metrics used in this project to train a Multilayer Perceptron (MLP) for fraud detection. We cover the feature engineering methods, training techniques, and evaluation metrics, all of which contribute to improving the model's performance and ensuring robust generalization.

2.1 Feature Engineering and Normalization Techniques

Several preprocessing techniques are essential for preparing the dataset for the MLP:

- **Standardization**: Transforms features to have a mean of zero and a standard deviation of one, which is critical for algorithms like MLP. The formula is:

$$x_{\text{standardized}} = \frac{x - \mu}{\sigma}$$

- **Logarithmic Transformation**: Compresses skewed distributions, particularly useful for transaction amounts in fraud detection:

$$x_{\text{log}} = \log(x + 1)$$

- **Polynomial Features**: Generates new features by combining existing ones (e.g., x^2, $x_1 \times x_2$) to capture non-linear relationships.

2.2 Training Techniques

To optimize the MLP, several training techniques were applied:

- **Binary Cross-Entropy (BCE)**: The loss function used for binary classification (fraud vs. non-fraud). It penalizes wrong predictions based on their distance from the true label:

$$\text{BCE}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

- **ADAM Optimizer**: Combines the advantages of SGD and RMSProp, adjusting the learning rate dynamically for faster and more stable convergence.

- **SGD (Stochastic Gradient Descent)**: A straightforward optimization method that updates model parameters using the gradient of the loss function. While it can be noisy, this noise can help the model escape local minima and achieve a global minimum:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

  where θ represents the model parameters, η the learning rate, and $J(\theta)$ the cost function.

- **Dropout**: A regularization technique that randomly drops neurons during training to prevent overfitting. Typically, a dropout rate between 0.2 and 0.5 is applied, reducing the model's reliance on specific neurons.

- **Early Stopping**: Monitors the validation loss and halts training when no improvement is observed after a set number of epochs (patience), helping to avoid overfitting.

2.3 Evaluation Metrics

Multiple metrics are used to assess model performance, especially in the context of class imbalance in fraud detection:

- **Accuracy**: Measures the overall correctness of the model's predictions:
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$
- **Precision**: Indicates the proportion of predicted frauds that are actual frauds:
$$\text{Precision} = \frac{TP}{TP + FP}$$
- **Recall (Sensitivity)**: Measures the model's ability to correctly identify all fraud cases:
$$\text{Recall} = \frac{TP}{TP + FN}$$
- **F1-Score**: The harmonic mean of precision and recall, balancing both metrics:
$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$
- **AUC-ROC**: Measures the area under the ROC curve, assessing the model's ability to distinguish between fraud and non-fraud across all classification thresholds.
- **Average Precision (AP)**: The area under the precision-recall curve, providing a single score that captures the precision-recall tradeoff.

2.4 Models

In this project, several machine learning models were used to tackle the classification task, particularly for fraud detection. These models include **Sigmoid (Logistic Regression)** for binary classification and the core model, the **Multilayer Perceptron (MLP)**. Below, we explain each model, its function, and the associated loss function used during training.

Sigmoid (Logistic Regression)

The sigmoid function, commonly used in logistic regression, is ideal for binary classification tasks. It outputs a probability between 0 and 1, indicating the likelihood of the input belonging to a particular class (fraud or non-fraud).

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Where $z = w^T x + b$, with w representing the weights, x the input, and b the bias.

Loss Function: Binary Cross-Entropy (BCE)

The loss function for logistic regression is Binary Cross-Entropy (BCE), which penalizes incorrect predictions based on how confident the model is:

$$L_{\text{BCE}}(y, \hat{y}) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

Where y is the true label (0 or 1), and $\hat{y}$ is the predicted probability.

Multilayer Perceptron (MLP)

The Multilayer Perceptron (MLP) is a type of neural network composed of an input layer, hidden layers, and an output layer. It is used for more complex classification tasks, such as fraud detection, due to its ability to model non-linear relationships. Each neuron in the network applies a linear transformation to the input, followed by a non-linear activation function, typically **ReLU (Rectified Linear Unit)**:

$$f(z) = \max(0, z)$$

The output layer typically uses sigmoid for binary classification or softmax for multi-class tasks.

Loss Function: Binary Cross-Entropy (BCE)
For binary classification tasks, such as fraud detection, the Binary Cross-Entropy loss is used, as defined earlier. This loss function is suitable when the model outputs probabilities for two classes. The MLP is trained using **backpropagation**, where the error is propagated through the network, and the weights are adjusted to minimize the loss function.

2.5 Python and Libraries

In this project, Python was used as the primary programming language, along with several libraries for data processing, model development, and evaluation. Below is a list of the key libraries used:

- **Python**: A versatile and widely-used programming language known for its readability and extensive ecosystem of libraries, making it ideal for machine learning and data science projects.
- **NumPy**: A fundamental library for numerical computations in Python. It provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- **Pandas**: A library used for data manipulation and analysis, especially for handling structured data like CSV files. It provides data structures like DataFrames for efficiently managing large datasets.

- **Matplotlib**: A plotting library used to generate graphs and visualizations. It is helpful in analyzing model performance, visualizing datasets, and tracking training progress.
- **Scikit-learn**: A popular machine learning library providing tools for model selection, evaluation, and various preprocessing techniques, including scaling, encoding, and splitting datasets.
- **PyTorch**: A deep learning framework used for building and training the Multilayer Perceptron (MLP). It offers dynamic computation graphs, enabling flexibility in model building and easy debugging.

3 Generating the Dataset

Contrary to any other tasks, creating a Fraud Detection System implies that the model should be trained with real transactions. From a practical point of view it is unrealistic to find such a dataset unless it is directly provided by a bank or a company who manages the latter; given this problem, the project features a custom generator class that will emulate the creation of such data, focusing on building appropriate features to help the models perform well at correlating them. The dataset and features were built and inspired by research papers that featured real data, so the emulated data will be as realistic as possible. The generator is structured as a class, where each method is purposely designed to build each feature of the dataset. Next, the process to create such data will be listed.

3.1 Defining the Clients

The method `create_clients` is responsible for generating a set of client profiles, each with random characteristics that mimic real-world behaviors. In fraud detection scenarios, it is crucial to simulate realistic client behaviors in order to have a diverse dataset, and this method does exactly that by introducing variability across clients. The first thing the method does is set a random seed. This ensures that the generated profiles are consistent across multiple runs, which is important for reproducibility. Every time the method is run with the same seed, the same set of random values will be generated. This makes testing and comparison easier, as the data stays consistent unless the seed changes. Once the random seed is set, the method moves on to generate the actual attributes for each client. Each client is assigned a random geographic location using `x_client_coordinate` and `y_client_coordinate`, which are both chosen from a range of 0 to 100. These can be thought of as placeholders for geographic coordinates, giving each client a unique virtual location. While these coordinates don't represent actual map locations, they simulate how clients are distributed geographically, which can be useful for certain fraud patterns. In addition to location, the method creates financial behavior for each client. Each client gets an `average_amount`, which represents the average transaction value for that client. This value is randomly selected from a range between 5 and 100, mimicking different spending habits—some clients make small, frequent purchases, while others might spend larger amounts less frequently. To introduce further variability, the method calculates the `amount_std_dev`, which is simply half of the client's average transaction amount. This standard deviation simulates how much a client's spending deviates from their average; clients with a higher standard deviation might have more inconsistent spending habits, while those with a lower standard deviation are more predictable. Additionally, the method generates an `avg_transactions_per_day` attribute, which defines how often each client transacts. This value ranges from 0 to 4 transactions per day. This variability is key in creating more realistic client profiles because fraud detection systems often look for unusual patterns in transaction frequency—clients who suddenly make far more transactions than usual may be flagged for suspicious activity. Once all the attributes are generated, the method assembles them into a dictionary. Each key in the dictionary corresponds to one of the client attributes, and the values are arrays of data for all the clients. This dictionary is then converted into a Pandas DataFrame, which organizes the data in a clean, tabular format. It is then returned by the method.

3.2 Creating the Terminal Profiles

The `create_terminals` method generates profiles for transaction terminals, assigning random geographic coordinates to simulate their locations. These terminals represent points of interaction in the financial system, such as ATMs or POS terminals, where clients perform transactions. As with the client generation, the method begins by setting a random seed to ensure the reproducibility of the data. This allows the same terminal profiles to be generated consistently across different runs, which is important for ensuring the reliability of model testing and evaluation. The key part of the method is the generation of two-dimensional coordinates for each terminal. Using the `np.random.uniform` function, the method creates X and Y coordinates for all the terminals, with values ranging from 0 to 100. These coordinates simulate how terminals are distributed geographically, though they do not correspond to real-world locations. The coordinates provide spatial data that can later be used to simulate transactions and analyze the proximity between terminals and clients, which is often crucial for detecting fraud based on geographical inconsistencies. Once the coordinates are generated, the method organizes them into a Pandas DataFrame, where each row corresponds to a unique terminal, with columns representing the terminal's ID and its X and Y coordinates. The DataFrame is then returned for further use in the simulation process, where it will help create transactions between clients and these terminals.

3.3 Associating Clients with Nearby Terminals

In order to simulate real-world transactions, it is important to know which terminals (ATMs, point-of-sale devices, etc.) are geographically accessible to each client. The methods `associate_terminals_with_clients` and `add_available_terminals` work together to achieve this by identifying the terminals that fall within a specified radius of a client's location and associating them with the client. The first method, `associate_terminals_with_clients`, focuses on finding which terminals are near a particular client. It takes a client's profile and the coordinates of all the terminals, then calculates the distance between the client and each terminal using the Euclidean distance formula. By comparing these distances to the given radius, it filters out the terminals that are too far away, leaving only those that are within the specified radius. The result is a list of terminal IDs that are close enough for the client to interact with. Building on this, the second method, `add_available_terminals`, expands the process to all clients. It adds a new column to the client profile DataFrame, where each client is associated with a list of terminal IDs that are within the specified radius. This method iterates through each client, applying the `associate_terminals_with_clients` function to determine which terminals are nearby. The result is a new column called `near_terminals` in the client DataFrame, which holds a list of terminal IDs for each client. These two methods work together to establish a realistic connection between clients and terminals, simulating how transactions might take place based on geographical proximity. By associating clients with nearby terminals, the dataset gains a spatial dimension, which is critical for detecting location-based fraud or understanding transaction patterns.

3.4 Generating Transaction Data

The process of generating synthetic transaction data for the clients is handled by three interconnected methods: `generate_daily_transactions`, `generate_transactions`, and `generate_transactions_df`. These methods work together to simulate daily transactions, compile them over multiple days, and aggregate them into a DataFrame for all clients. They create realistic transaction patterns, incorporating random variations in transaction amounts, times, and locations. The first method, `generate_daily_transactions`, is responsible for simulating a single day of transactions for an individual client. It starts by determining the number of transactions the client makes in a day, based on a Poisson distribution that uses the client's average daily transaction count. For each transaction, the method generates a random transaction time, ensuring the time falls within the bounds of a 24-hour period. The transaction amount is also randomized, generated from a normal distribution centered around the client's average transaction amount, with some variation introduced by the standard deviation. If the generated amount is negative, which can happen due to the nature of the distribution, a fallback ensures that the transaction amount is recalculated as a positive value within a reasonable range. Finally, the transaction is assigned to a nearby terminal, randomly chosen from the list of terminals associated with the client. The result is a list of transactions for that particular day, each with attributes like the client ID, terminal ID, time, and amount. Building on the daily transaction generation, the `generate_transactions` method handles the task of generating transactions over multiple days. This method loops through the desired number of days and calls `generate_daily_transactions` for each day. All the transactions generated for a client across these days are collected into a DataFrame. Additionally, the method adds a column for the actual transaction datetime, calculated based on the transaction's timestamp and the specified start date. The final output is a structured DataFrame containing all transactions for the client over the given period, with relevant details such as the transaction time, amount, and associated terminal. The final method, `generate_transactions_df`, scales this process up to handle all clients. It applies the transaction generation process to the entire client dataset, grouping the clients by their IDs and calling `generate_transactions` for each one. This method ensures that every client's transactions are generated for the specified number of days and compiles the results into a single comprehensive DataFrame. This DataFrame contains the transaction records for all clients, capturing the full spectrum of their activity over the simulated period.

3.5 Adding Fraud Scenarios

The method `add_fraud_scenarios` is designed to introduce various fraud patterns into the transactions dataset. By simulating different types of fraud, the method helps to create a realistic dataset where fraudulent activities can be identified and studied. It does this by applying three distinct fraud scenarios, each representing a unique type of fraudulent behavior. At the start, the method initializes two columns in the transactions DataFrame: `IS_FRAUD`, which indicates whether a transaction is fraudulent, and `FRAUD_SCENARIO`, which marks the scenario under which the fraud was triggered. Initially, all transactions are labeled as non-fraudulent.

- **Scenario 1: High-Value Transactions**
  The first fraud scenario targets transactions with unusually large amounts. Any transaction exceeding 300 units is automatically flagged as fraudulent, and the `IS_FRAUD` column is updated to reflect this, along with the `FRAUD_SCENARIO` column being set to 1. This scenario mimics a typical fraud pattern where high-

value transactions are considered suspicious due to their potential risk. After applying this condition, the method prints out the number of transactions that were flagged as fraudulent under this scenario.

- **Scenario 2: Compromised Terminals**
  The second scenario introduces fraud at specific terminals that have been compromised. For each day in the dataset, two random terminals are selected as compromised. Any transaction that takes place at one of these terminals over the next 28 days is flagged as fraudulent, with `FRAUD_SCENARIO` set to 2. This scenario simulates situations where fraudsters gain control over certain terminals, and all transactions at those terminals are at risk of being fraudulent. Once this scenario is applied, the method calculates and prints the number of frauds that occurred under this condition.

- **Scenario 3: Compromised Customers**
  The third scenario targets specific clients whose accounts have been compromised. For each day, three random clients are marked as compromised. Over the following 14 days, any transactions made by these clients are reviewed. One-third of these transactions are selected and their amounts are artificially inflated by a factor of 5, mimicking scenarios where fraudsters exploit a client's account to make unusually large transactions. These inflated transactions are then flagged as fraudulent, with `FRAUD_SCENARIO` set to 3.

By the end of the process, the transactions DataFrame contains a realistic mix of fraudulent and non-fraudulent transactions, each fraud labeled by the scenario that generated it.

3.6 Generating the final Dataset

The dataset method is responsible for generating the entire dataset of synthetic transactions. It integrates all the previous steps—from creating client and terminal profiles, to associating terminals with clients, generating transactions, and introducing fraud scenarios—into one cohesive process. The result is a fully prepared dataset that can be used for training machine learning models or conducting data analysis. It allows to set the dimension of the requested dataset allowing it to be scaled for larger and complex tasks.

4 Transforming the Features

The previously generated simulated dataset is not sufficiently suitable for training a machine learning model intended for accurate classification and prediction tasks. Machine learning algorithms require both numerical and categorical (empirical) features. In the generated dataset, the only features that fit these criteria are the fraud label and the transaction amount. The objective is to create new features that can be effectively utilized for predictive modeling. To achieve this, several sources have been used as a reference, and the feature transformations to be implemented are as follows:

- **One-hot encoding**
  This transformation will adjust the variables related to date and time, creating binary features that emphasize significant time periods. Specifically, the transformation will produce a feature indicating whether a transaction occurred during the day or night, and another feature denoting if the transaction took place on a weekend or a weekday. These features are crucial in real-world scenarios, as fraudulent patterns often differ between day and night and between weekdays and weekends.

- **RFM(Recency, Frequency, Monetary value)**
  This transformation will modify the ID variable to create features that highlight customer spending behaviors. This process will follow the RFM (Recency, Frequency, Monetary) framework proposed in this research [1].

- **Risk Encoding**
  This transformation will create features that highlight the risk of a terminal. The risk will be the average amount of frauds that have occurred in that terminal for 3 periods of time.

4.1 One-hot encoding

Two new features are introduced to categorize transactions based on day type and time. The first feature distinguishes transactions that occur on a weekday (0) versus a weekend (1), which is achieved by converting the transaction's timestamp into a day of the week using a simple check against the day's numeric representation. This is followed by applying one-hot encoding, allowing the system to process the resulting categorical value. A similar method is applied for the second feature, which differentiates between transactions during the day (0) and those occurring at night (1). Here, the time of the transaction is examined, with nighttime being defined as any time between 0 am and 6 am. Again, the results are converted into binary, making it straightforward to incorporate into the model. The resulting binary values enhance the dataset by adding more granular transaction timing details, which can be critical for the machine learning model's performance.

4.2 RFM(Recency, Frequency, Monetary value)

The next step involves transforming client IDs based on two key features: the number of transactions (Frequency) and the average monetary value of these transactions (Monetary value). These features will be calculated over three distinct time frames: one day, seven days, and thirty days. This transformation introduces six new features in total, enabling a more detailed understanding of customer behavior. The idea is to capture both how often a customer transacts within specific periods and the average amount spent, which can help improve the model's predictive power. These time frames can also be fine-tuned later as part of model optimization to better align with the overall performance.

4.3 Risk Encoding

In this section, the focus shifts to transforming terminal IDs to compute a **risk score**, which indicates the exposure of a terminal to fraudulent transactions. The risk score is calculated as the average number of fraudulent transactions associated with a terminal over different time windows: 1, 7, and 30 days. A **delay concept** affects this process in order to emulate the time that it takes for a fraudulent transaction to be identified as such. The risk score calculation is done in two stages. First, the number of both total and fraudulent transactions is computed during the delay period. Then, in the second stage, similar values are computed for the time windows plus the delay. By comparing the results of the two stages, the number of fraudulent transactions per window size is determined, providing the risk score.

5 Proposed Approaches

After fine-tuning the generated dataset, the methods employed in this experiment will be discussed to establish a linear workflow, ensuring optimal results.

5.1 Dataset Organization

A crucial phase in training any model is the proper splitting of the dataset to avoid biases and redundancies between the training and evaluation sets. In this experiment, the dataset consists of transactions that occurred over a span of 3 days and 3 months, containing a total of 726,864 transactions, of which 5,106 are fraudulent. To split this dataset effectively, the transaction timestamps will be utilized. This ensures that each set contains distinct transactions, with intervals of one week separating them. The dataset will be divided into three sets: training, validation (used for evaluation during training), and testing.

5.2 Input Data Normalization

Once the data has been correctly separated, normalization will be applied to improve model performance and enhance its ability to converge without overfitting. The data will be transformed so that values are neither too large nor too small. Each feature will be standardized to have a mean of 0 and a standard deviation of 1, ensuring that the data is centered around 0. This allows the model to treat each feature fairly and prevents dominance by features with larger ranges.

5.3 Convergence Techniques

One of the main objectives is to achieve convergence as efficiently as possible. Two common techniques used for this purpose are Early

Stopping and the Learning Rate Scheduler:

- **Early Stopping:** This technique monitors the model's performance on the validation set during training and halts the process when improvements cease. It prevents overfitting by stopping the training when further learning no longer benefits the model, saving time and improving generalization. Additionally, the model's best performance during training will be saved.
- **Learning Rate Scheduler:** This method adjusts the learning rate during training according to a predefined schedule. It helps the model converge more effectively by reducing the learning rate as training progresses, allowing for finer adjustments as the model nears an optimal solution.

5.4 Regularization

Overfitting is a significant risk during model training, leading to poor performance on evaluation data. Since this experiment employs an MLP, Dropout is used as the primary regularization method. Dropout prevents overfitting by randomly ”dropping” (setting to zero) a certain percentage of neurons during training. This encourages the network to avoid over-reliance on any single neuron, promoting robustness and enhancing generalization. During evaluation, the dropped neurons are reactivated.

5.5 Multi-Layer Perceptron Structures

For this experiment, three different MLP structures will be tested, each with a unique design:

- **Downsampling Structure:** This structure follows a downsampling approach, where each layer has a progressively smaller dimension until the output layer is reached. Figure 1 illustrates this structure.

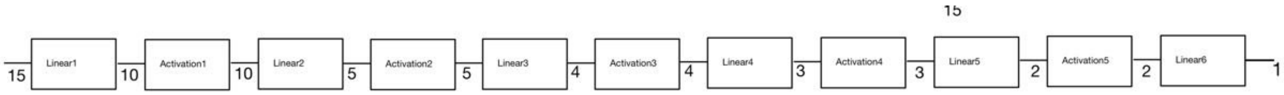

Figure 1: Down-sampling structure

- **Fully-Connected Layers:** This structure consists of only two fully connected layers of equal dimension. The purpose of this MLP is to evaluate the performance of a simple model with a minimalistic design. Figure 2 shows this structure.

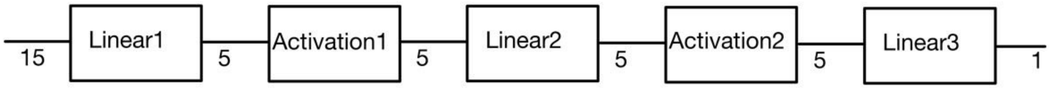

Figure 2: Fully-Connected MLP

- **Funnel-Like Structure:** This design resembles a funnel, starting with layers that expand significantly to allow the network to capture more features, aiding in better correlations and classifications. The layers then contract until the output layer is reached. Figure 3 illustrates this structure.

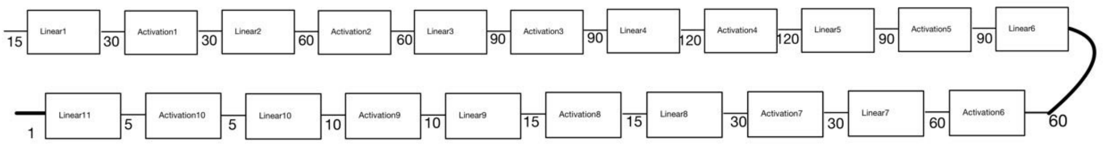

Figure 3: Funnel-like Structure

During training, each structure will undergo dropout testing to assess how performance varies with the application of dropout.

6 Workflow and Analysis

In this section, we delve into the training of the networks using the methodologies listed in the previous section and analyze the results using the chosen metrics applied to the evaluation performance and visualizations.

6.1 Downsampling Structure MLP

This structure will be initially tested without dropout. To determine the best optimizer, the model will be trained using two optimizers: SGD (Stochastic Gradient Descent) and Adam.

Optimizer	Accuracy	Precision	Recall	F1	AUC ROC	Avg Prec
SGD	0.9968	**0.8991**	0.5269	0.6644	0.8340	0.5894
Adam	**0.9969**	0.8315	**0.5968**	**0.6948**	**0.8511**	**0.6436**

Table 1: SGD evaluation metrics compared to Adam's.

Upon analyzing Table 1, we observe that while both models achieve 99% accuracy, this metric holds little significance due to the dataset's class imbalance. With SGD, precision is 97%, but recall is below 50%, meaning the model correctly identifies few fraudulent transactions. Adam, on the other hand, achieves a lower precision but significantly higher recall, offering a better balance overall. In both cases AUC ROC is above 80% but Adam offers a better Average Precision.

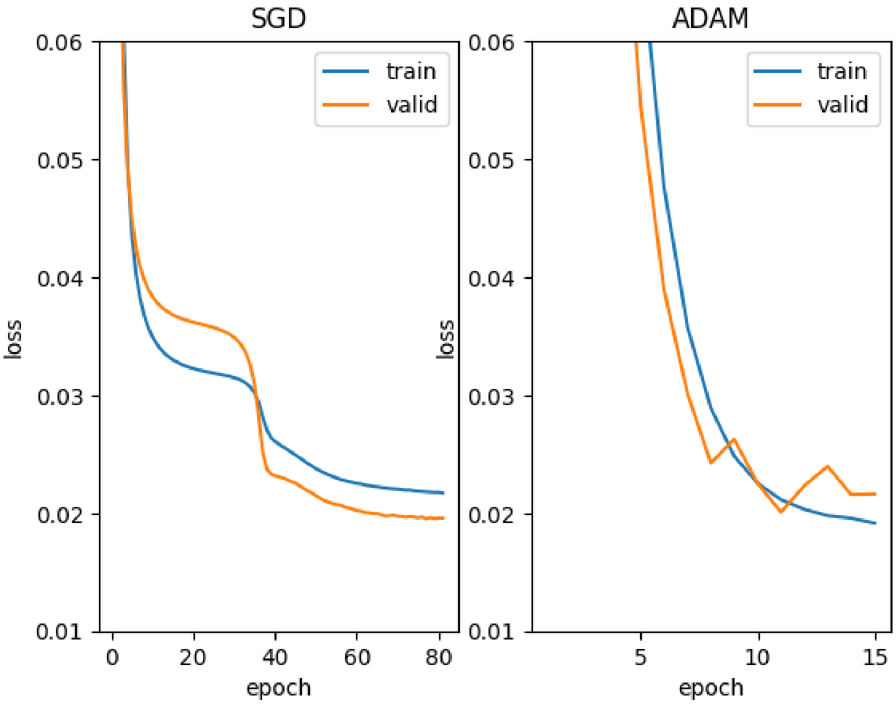

Figure 4: SGD's loss trend against Adam's loss trend.

Figure 4 shows that while SGD appears more stable, Adam converges faster, although at the cost of introducing slight overfitting. This issue can be mitigated by applying dropout.

Optimizer	Accuracy	Precision	Recall	F1	AUC ROC	Avg Prec
Adam	0.9967	0.9766	0.4489	0.6151	0.8597	0.6166

Table 2: Evaluation metrics for the fully-connected MLP.

Table 2 highlights that the precision improved drastically, making the model very precise at identifying fraudulent cases reducing significantly the Recall. AUC ROC still offers a high performance meaning while the Average Precision had a slight improvement.

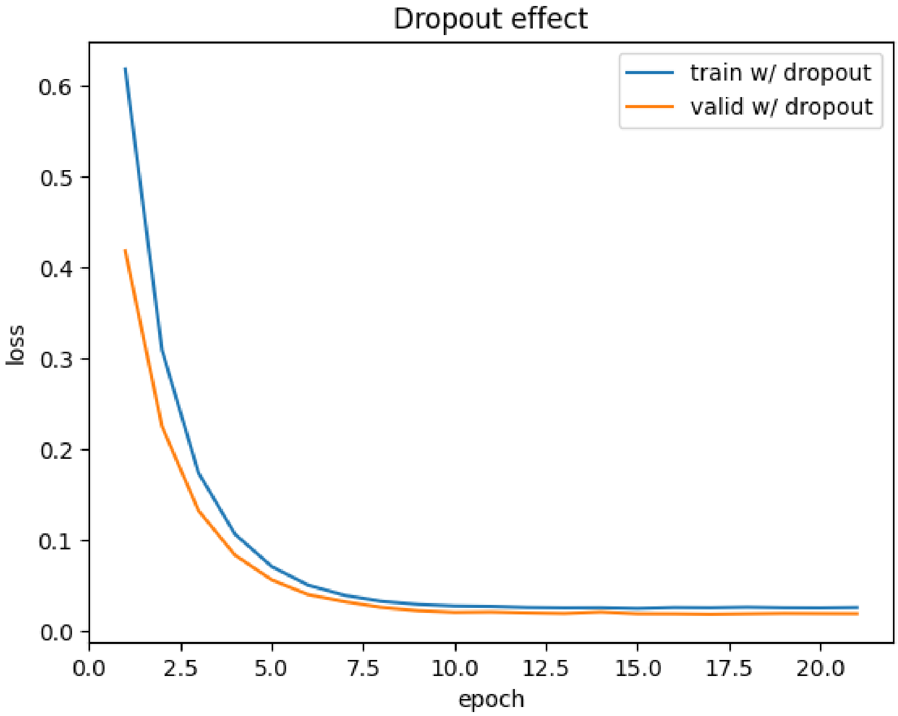

Figure 5: Effect of dropout on the downsampling structure.

As shown in Figure 5, dropout improves convergence and reduces the overfitting introduced by Adam, increasing precision to 97% at the cost of recall.

6.2 Fully-Connected Layers MLP

The training and evaluation will follow the same workflow, but this time Adam will be used directly, given its faster convergence. The first experiment will be conducted without dropout.

Optimizer	Accuracy	Precision	Recall	F1	AUC ROC	Avg Prec
Adam	0.9967	0.8161	0.5726	0.6730	0.8536	0.6336

Table 3: Evaluation metrics for the fully-connected MLP.

Table 3 shows that this model correctly identifies 57.26% of fraudulent cases, which is an improvement over the first structure, even after dropout. However, there is a slight decline in precision.

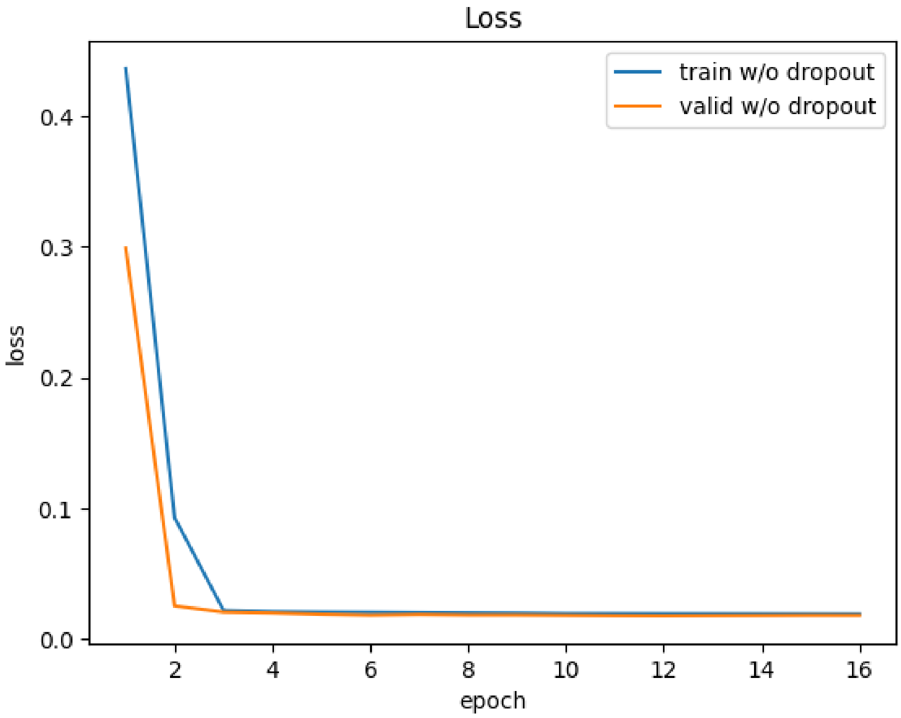

Figure 6: Loss progression without dropout for the fully-connected MLP.

As shown in Figure 6, both the training and validation losses converge steadily, with no apparent signs of overfitting. The network quickly finds a good representation of the features within the first few epochs. Next, we apply dropout to observe its effect.

Optimizer	Accuracy	Precision	Recall	F1	AUC ROC	Avg Prec
Adam	0.9969	0.9641	0.5054	0.6631	0.8525	0.6347

Table 4: Evaluation metrics with dropout for the fully-connected MLP

As shown in Table 4, precision significantly improved to 96%, but this came at the expense of recall. This structure demonstrates a better balance between precision and recall compared to previous models.

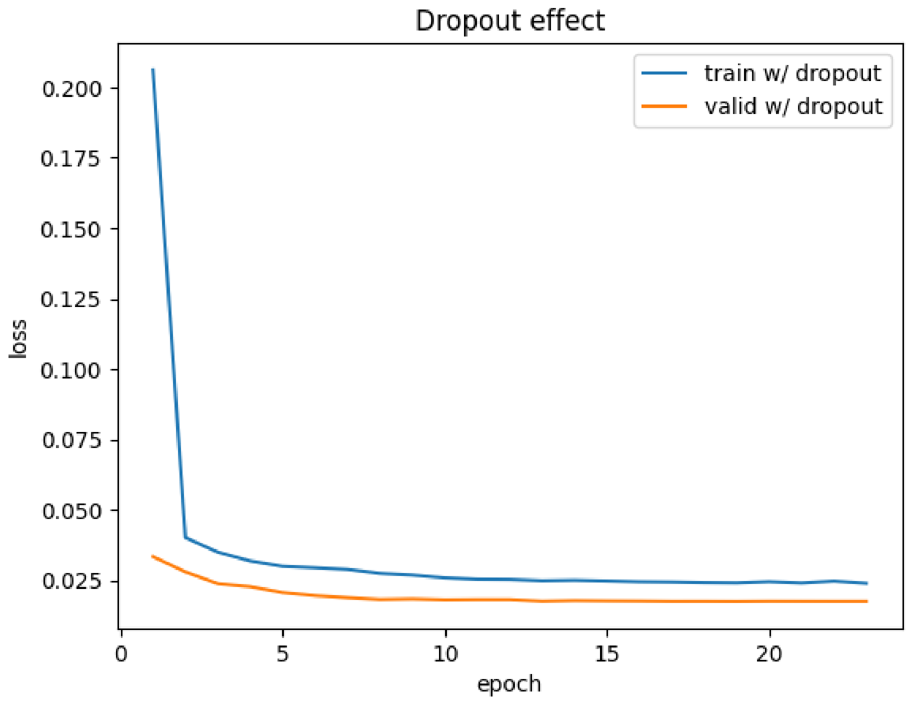

Figure 7: Loss progression with dropout for the fully-connected MLP.

Figure 7 demonstrates how dropout significantly affects validation loss, allowing the network to converge and minimize it within a few epochs. This indicates the network has captured useful patterns from the features, effectively identifying fraudulent transactions.

6.3 Funnel-like Structure MLP

Finally, we test the data on a larger model to assess whether it improves the previous results. It's important to note that larger models have a higher tendency to overfit, requiring additional regularization layers to compensate. First, the network is trained and evaluated without dropout.

Optimizer	Accuracy	Precision	Recall	F1	AUC ROC	Avg Prec
Adam	0.9967	0.7774	0.6290	0.6954	0.8709	0.6595

Table 5: Evaluation metrics for the funnel-like MLP without dropout.

As expected, precision dropped to 77%, but recall improved to 62% compared to previous results, suggesting potential overfitting during training. Let us now examine the loss progression.

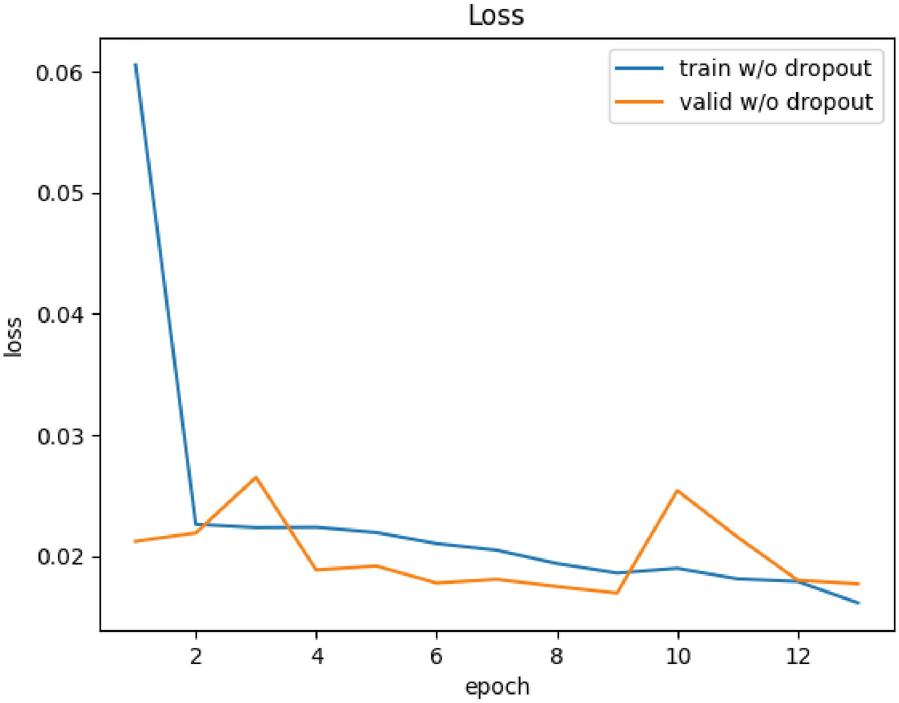

Figure 8: Loss progression without dropout for the funnel-like MLP

Figure 8 confirms that overfitting occurred, especially in the later epochs, due to the large network size. Dropout is necessary to achieve better results.

Optimizer	Accuracy	Precision	Recall	F1	AUC ROC	Avg Prec
Adam	0.9968	0.9048	0.5108	0.6529	0.8546	0.6247

Table 6: Evaluation metrics for the funnel-like MLP with dropout.

As shown in Table 6, dropout resulted in significant overall improvement, achieving a precision score of 90%. Although precision improved, recall suffered. The model is less affected by overfitting compared to the non-dropout configuration.

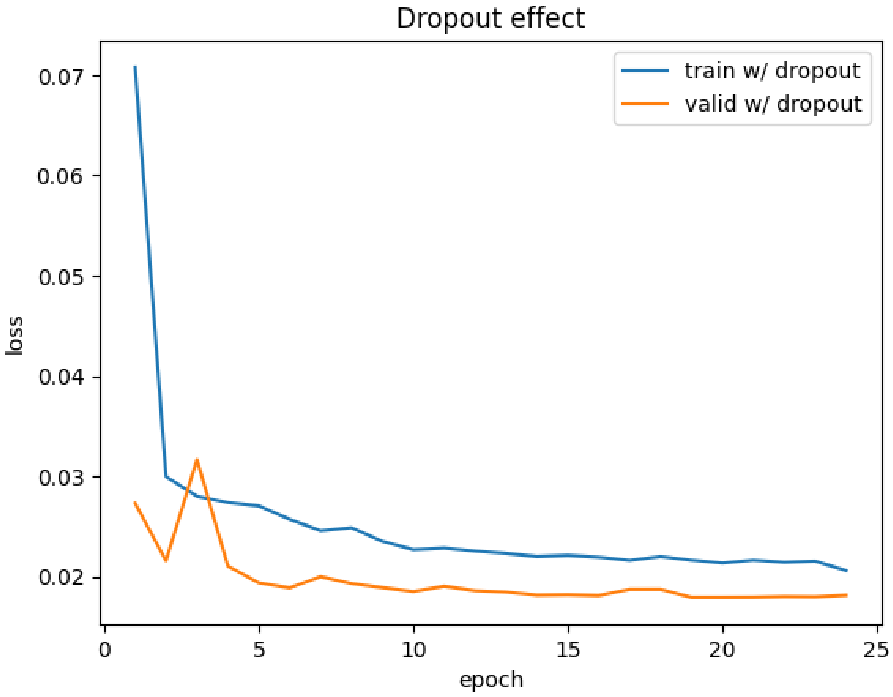

Figure 9: Loss progression with dropout for the funnel-like MLP.

Figure 9 shows that while overfitting was reduced, the model still experienced it in the early epochs, stabilizing afterward. The combination of the Learning Rate Scheduler and dropout produces promising results.

6.4 High Precision, Low Recall

At the conclusion of this analysis, all the models demonstrated strong performance on the data, most showing a high AUC ROC metric, indicating their ability to correctly classify a randomly presented fraudulent instance. However, the average precision, which hovers around 0.6 in most cases, suggests that the balance between precision and recall could still be improved. The most notable issue was the imbalance between precision and recall. Table 7 provides a clear visualization of this imbalance, showcasing the precision and recall values for each model with dropout applied:

Model	Precision	Recall
Downsampling MLP	**0.9766**	0.4489
FC MLP	0.9641	0.5054
Funnel MLP	0.9048	**0.5108**

Table 7: Precision and Recall with dropout for all three MLPs.

As shown, there is a clear trade-off between precision and recall: as precision increases, recall tends to decrease. This indicates that the models are very conservative, avoiding false positives but missing many actual fraud cases. The few fraudulent cases that are detected tend to be classified correctly. This behavior arises from the MLP's limited complexity in identifying key correlations between the behavioral features and the fraud labels. While the low false-positive rate might seem beneficial, the significant number of missed fraud cases is a critical flaw, making these systems unreliable for practical use.

6.5 Proposed Solutions

Among the models studied, the funnel-like structure provided the best balance between performance and simplicity. This architecture, characterized by a gradual increase in the number of neurons per layer, followed by a progressive reduction, allows for more efficient data representation as it passes through the network. While effective, one potential extension is to further deepen the network by adding more layers and increasing the number of neurons per layer in the initial stages. A deeper, more complex network would have greater capacity to capture intricate patterns within the data, potentially increasing the number of detected fraudulent transactions. By increasing the depth and complexity, the model could better learn complex feature interactions, improving its ability to distinguish between fraudulent and non-fraudulent transactions. This could enhance the model's confidence in its classifications, especially in edge cases where the distinction is less clear. However, increasing model complexity comes with significant trade-offs, such as a heightened risk of overfitting, particularly on the validation set. This could degrade the model's performance and reduce its reliability in real-world scenarios. To counteract this, regularization techniques such as dropout, L2 weight penalties, or early stopping should be carefully applied. Additionally, fine-tuning hyperparameters like the learning rate and adjusting the learning rate scheduler to control the learning pace will be essential. Beyond expanding a complex MLP, a more promising approach is the use of autoencoders. Autoencoders, an unsupervised type of neural network, are well-suited for learning representations of high-dimensional data. By compressing input data into a lower-dimensional latent space and reconstructing it, an autoencoder can learn key features of the input data without over-reliance on labels. This can significantly aid in detecting subtle patterns in fraudulent transactions, making the model more robust to variations in the data.

6.5.1 Autoencoders as a Solution for Transaction Data

Autoencoders are particularly well-suited for detecting fraudulent transactions for several reasons:

- **Feature Extraction in High-Dimensional Data:** Transaction data often involves many variables, making it high-dimensional. Autoencoders excel at reducing this data into a compressed form (latent space) while retaining essential features. This enables the model to detect subtle patterns, such as unusual interactions between variables, which can indicate fraud.

- **Unsupervised Learning Advantage:** Fraudulent transactions are rare compared to legitimate ones, meaning that labeled data is often limited. Autoencoders, as unsupervised models, do not rely on labeled data. They learn to reconstruct the input data and detect anomalies (such as fraud) based on how poorly the input is reconstructed. This makes autoencoders ideal for scenarios with limited labeled data.

- **Anomaly Detection:** Autoencoders detect anomalies by measuring reconstruction error. When trained on normal transactions, the autoencoder will have low reconstruction error for legitimate transactions. Fraudulent transactions, which deviate from normal patterns, will result in higher reconstruction errors, making them easier to flag as suspicious.

- **Noise Resilience:** Transaction data can be noisy, with irrelevant or erroneous information. Autoencoders naturally filter out noise by focusing on the core features of the data during compression. This enhances the model's ability to detect fraud even when the data includes noisy inputs, ensuring robustness in real-world transaction datasets.

- **Pretraining for Downstream Tasks:** Autoencoders can also be used for pretraining more complex models. By initializing the model with weights that capture the underlying distribution of transaction data, the autoencoder improves the performance of subsequent classifiers tasked with identifying fraud, leading to better overall accuracy.

- **Scalability:** Autoencoders are highly scalable for large transaction datasets. They can learn compact representations of vast amounts of data, making them suitable for efficiently processing large transaction volumes. Their scalability ensures that performance is maintained even with growing data sizes, a crucial feature for fraud detection in large financial systems.

In summary, autoencoders offer a robust approach to fraud detection by learning compressed representations of transaction data, detecting anomalies via reconstruction errors, and filtering out noise. Their unsupervised learning capability and scalability make them an ideal solution for identifying fraudulent transactions in large-scale, noisy datasets.

7 Conclusions

In this study, we explored various techniques across three different MLP architectures, aiming to achieve robust results while evaluating their suitability for practical applications. Each architecture displayed distinct strengths and weaknesses, offering valuable insights into multiple design approaches. Despite the differences in structure, a consistent pattern emerged: while these networks demonstrated high precision, they struggled with reliably identifying all fraudulent transactions. This observation highlighted a critical shortcoming—although capable of reducing false positives, the models lacked sufficient recall to capture all cases of fraud. This insight led us to identify the most appropriate architecture as a baseline for future experiments. Furthermore, it prompted us to explore the Autoencoder as a promising alternative to tackle these limitations. With its more complex architecture, the Autoencoder holds the potential to better detect subtle patterns within the data, offering improved overall performance, particularly in identifying edge cases of fraud that traditional MLP models might miss. The project has been published in the following repository on Github: `https://github.com/claudiunderthehood/transactions_mlp`.

References

[1] Véronique Van Vlasselaer et al. "APATE: A novel approach for automated credit card transaction fraud detection using network-based extensions". In: *Decision Support Systems* 75 (2015), pp. 38–48. DOI: `10.1016/j.dss.2015.04.013`.