A

Project Report Submitted

in Partial Fulfilment of the Requirements

for the Degree of

**BACHELOR OF TECHNOLOGY**

Under the supervision of

**Prof. Prasanta K. Jana**

(Professor Department of CSE)



**Department of Computer Science and Engineering**

**INDIAN INSTITUTE OF TECHNOLOGY**

**(INDIAN SCHOOL OF MINES)**

**DHANBAD**

By

**Avinesh Pratap Singh (20JE0219)**

**Brijesh Kumar (20JE0279)**

**Rahul Agrawal (20JE0744)**

**Acknowledgments**

We would like to express our deepest gratitude to our project supervisor,

**Prof. PRASANTA K. JANA.** Your expertise and encouragement were invaluable in shaping the direction and success of this project.

We are indebted to a number of people for their direct and indirect contributions that helped us in the completion of our project work.

First, I would like to thank our project supervisor, **Prof. PRASANTA K. JANA**, a Professor Department of Computer Science and Engineering IIT (ISM), Dhanbad for providing us the opportunity to work on this project under his supervision. It is always a pleasure to work on a project where you can develop your interest.

# Certificate

This is to certify that the project report titled **‘Credit Card Fraud Detection using Federated Learning’** submitted by **Avinesh Pratap Singh (20JE0219), Brijesh Kumar (20JE0279), and Rahul Agrawal (20JE0744)** to the Indian Institute of Technology (ISM), Dhanbad towards partial compliance with the requirements for obtaining the bachelor of technology in Computer Science and Engineering title is a record of the good faith work done by them under my Supervision and guidance during the Winter Semester (2023-24).

**Prof. PRASANTA K. JANA                                                           Prof. Chiranjeev Kumar**

**(Professor** **& IEEE Senior Member**)                                           **(Professor)**

Project Guide           CSE Head of Department

**Abstract**

Credit card fraud is a common problem for consumers and financial institutions all over the world. They are losing billions of dollars every year. Thus, an effective Fraud Detection System (FDS) is important to minimize the loss for financial institutions and cardholders.

A common solution to detect fraud is to use machine learning algorithms as they help to predict future outcomes and recognise patterns by analysing massive quantities of data. In order to get a well-performing model a large dataset is required, but an issue with credit card transaction datasets is that they are skewed, i.e. there are significantly fewer samples of fraudulent than legitimate transactions. Moreover, due to the data privacy and security associated with credit card transaction datasets, banks, and other financial institutions are usually not allowed to share their transaction data.

These problems together make it difficult for the centralized FDS to learn the patterns of fraud and, hence, to detect them. In this project, we propose a framework for training a fraud detection model with federated learning, i.e. a machine learning setting where multiple entities collaborate in solving a machine learning problem under the coordination of a central server or service provider. With this approach, financial institutions can collectively reap the benefits of a shared model, which has seen more fraud than each bank alone, without sharing the dataset with each other.

Additionally, we incorporate edge computing techniques to enhance scalability and eliminate the need for frequent reconfiguration of the central model when new nodes (banks) join the system. By leveraging edge computing, we mitigate straggler effects and improve the overall efficiency of the federated learning framework.

Hence, the sensitive information of the cardholders is protected. Our result of this project indicates that the federated model (Federated Averaging) can perform and even outperform the centralized model when trying to detect credit card fraud.

**Contents**

**1.Introduction**

**2.Background**

2.1 Fraud

2.2 Machine Learning for Fraud Detection

2.3 Fraud Areas

2.4 Issues and Challenges with Fraud Detection Systems

2.5 Skewed Data

2.6 Performance Measurement

2.7 Centralised Learning

2.8 Multi-Layer Perceptron

2.9 Federated Learning

2.10 Non-Identical Independent Distributed Data

2.11 Federated Averaging

2.12 Edge computing in Federated Learning

**3. Methodology**

3.1 Dataset

3.2 SMOTE for Balancing Dataset in fraud Detection

3.3 Bank Grouping

3.4 MLP and Federated Averaging

**4. Model Implementation**

4.1 Banks Grouping

4.2 Implementing SMOTE in Credit Card Fraud Detection

4.3 Multi-Layer Perceptron

4.4 Federated Averaging

**5. Results**

5.1 Banks Grouping

5.2 Performance Evaluation of Models with SMOTE

5.3 Multi-Layer Perceptron vs Decision Tree

5.4 Federated Learning with Edge Computing

**6. Conclusion**

**1. Introduction**

In today's financial landscape, combating fraud has become a critical priority for banks and financial institutions globally. The escalating complexity of fraudulent activities necessitates innovative approaches to detect and prevent financial fraud effectively. One of the most prevalent forms of fraud, especially in the banking industry, is credit card fraud, which costs institutions billions of dollars annually.

To address this pressing issue, our project focuses on developing an advanced fraud detection system leveraging cutting-edge technologies. Our approach integrates federated learning and edge computing techniques to create a robust and scalable solution for credit card fraud detection.

Federated learning enables multiple banks and financial entities to collaboratively train a fraud detection model without compromising the privacy of sensitive customer information. This distributed learning paradigm allows each participating bank to contribute to model training while keeping their data secure within their respective environments. By harnessing federated learning, we aim to leverage the collective knowledge and insights from diverse datasets to enhance the accuracy and effectiveness of fraud detection.

Additionally, we incorporate edge computing into our framework using Particle Swarm Optimization (PSO) for optimal assignments. This involves grouping banks based on geographical proximity and load balancing considerations. By minimizing the distance between banks and servers and optimizing the workload distribution, we ensure efficient model training and real-time fraud detection capabilities at the network edge. This approach not only improves scalability but also minimizes latency and computational overhead, making our fraud detection system agile and responsive to dynamic fraud patterns.

Our project's primary objective is to evaluate the performance of the Federated Averaging (FedAvg) model enhanced with edge computing capabilities using PSO for optimal assignments. We aim to assess the model's ability to detect credit card fraud accurately compared to traditional centralized approaches, such as the Multi-Layer Perceptron (MLP). Through this investigation, we seek to contribute insights into the potential of federated learning and edge computing in enhancing fraud detection systems, particularly in the context of credit card fraud.

This thesis unfolds with an exploration of fraud detection challenges, a review of related research, a detailed description of our dataset and preprocessing techniques, an explanation of our federated learning approach with PSO-based edge computing, experimental results, discussions on findings, recommendations for future work, and a conclusive summary of our contributions and insights. Our research question centralizes around evaluating the efficacy of Federated Averaging with edge computing enhancements for credit card fraud detection, positioning our work at the forefront of innovative solutions in financial fraud prevention.

**2. Background**

Understanding fraud, its prevalence in the banking sector, and the methodologies employed for its detection is crucial for developing effective fraud detection systems. This section delves into the definition of fraud, the role of machine learning in fraud detection, various types of fraud, and the challenges encountered when applying machine learning to detect credit card fraud. Additionally, it discusses the centralised and federated machine learning approaches for credit card fraud detection.

**2.1 Fraud**

Fraud is a criminal activity aimed at unlawfully obtaining money or assets. The Association of Certified Fraud Examiners (ACFE) defines fraud as:

“The use of one’s occupation for personal enrichment through the deliberate misuse or misapplication of the employing organization’s resources or assets.”

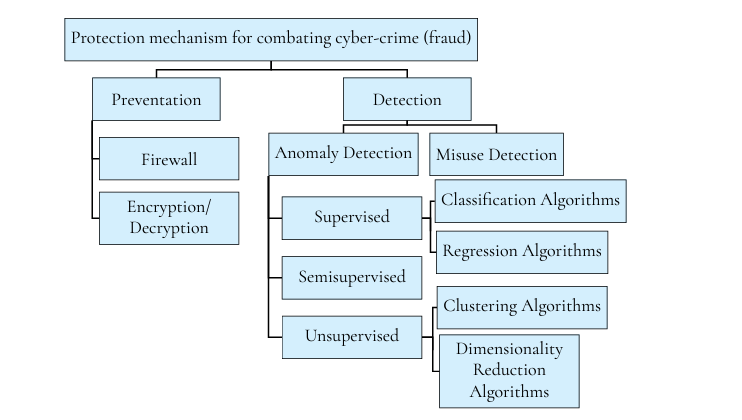
ACFE categorizes fraud into internal (committed by an employee against their organization) and external (committed against a company by external entities) [55]. This crime has significant economic, legal, and societal impacts, prompting extensive efforts in the financial industry to prevent and detect fraudulent behaviour through Fraud Prevention Systems (FPS) and Fraud Detection Systems (FDS).

FPS constitutes the initial layer of defence, securing technological systems against fraud through mechanisms like firewalls, encryption algorithms, and electronic signatures. On the other hand, FDS focuses on detecting fraudulent activities as they occur, employing two primary approaches: anomaly-based fraud detection and misuse-based fraud detection.

Anomaly-based fraud detection analyses individual client behaviour, raising alarms for suspected fraud when abnormal activities occur. This approach leverages data mining techniques, including statistical analysis, artificial intelligence, and machine learning, to extract valuable information from large datasets. In contrast, misuse-based fraud detection relies on pre-learned fraudulent behaviours, triggering alarms when clients exhibit such behaviours.

Pattern recognition plays a vital role in fraud detection, where behaviour patterns encompass transaction amounts, time gaps between purchases, day of the week for transactions, item categories, customer addresses, and more. Any deviation from a cardholder's established behaviour profile is flagged as suspicious, highlighting potentially fraudulent activity.

Machine learning techniques such as supervised learning, unsupervised learning, and semi-supervised learning underpin fraud detection systems. Supervised learning involves training models on labelled data, while unsupervised learning detects patterns in unlabelled data. Semi-supervised learning combines aspects of both supervised and unsupervised learning for enhanced fraud detection capabilities.



**Figure 2.1:** Protection system mechanisms for combating cyber-crime.

Figure 2.1 illustrates the protection mechanisms used to combat cyber-based fraud, emphasizing the interplay between prevention and detection strategies.

Moving forward, we will explore how these concepts and methodologies are applied in the context of centralised and federated machine learning approaches for credit card fraud detection.

**2.2 Machine Learning for Fraud Detection**

Machine learning plays a pivotal role in fraud detection by enabling computers to make predictions and recognize patterns without explicit programming. This capability is particularly effective in predicting risk and identifying abnormal behaviour in datasets, such as credit card fraud. The strength of machine learning lies in its ability to analysed vast amounts of data and derive accurate insights.

There are four main categories of machine learning techniques:

1. Supervised learning involves training the algorithm with labelled data, where each data point is associated with a specific outcome or class (fraudulent or non-fraudulent transactions, for example).

2. Unsupervised learning utilizes unlabelled data to identify patterns and structures within the dataset without predefined class labels.

3. Semi-supervised learning combines elements of both supervised and unsupervised learning by using a mix of labelled and unlabelled data for training.

4. Reinforcement learning is a learning method where the algorithm interacts with its environment, taking actions and learning from feedback in the form of rewards or errors.

In the context of credit card fraud detection, the classification problem—classifying transactions as fraudulent or non-fraudulent—can be approached using supervised, unsupervised, or semi-supervised learning techniques depending on the nature of the dataset.

According to previous study, the most common approach in building Credit Card Fraud Detection Systems (FDS) is through a classification model, which falls under supervised learning. This method involves training the model with labelled data to differentiate between fraudulent and legitimate transactions accurately.

**2.3 Fraud Areas**

Fraud encompasses various areas whenever monetary transactions are involved in technological systems. Previous studies have categorized the most common areas of fraud as bank fraud, insurance fraud, telecommunication fraud, and internet marketing fraud, as illustrated in Figure 2.2.

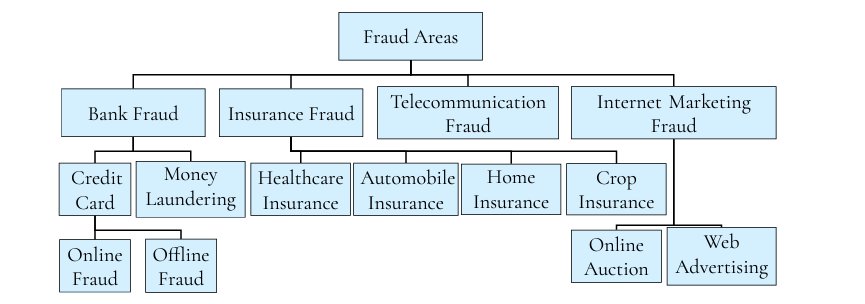


Figure 2.2: The most common areas of fraud according to

Among these areas, bank fraud is extensively researched, with credit card fraud being a widely recognized subset. Credit cards have gained widespread popularity due to their convenience and ability to track expenditures efficiently. Despite the implementation of authorization techniques such as signatures, credit card numbers, identification numbers, and cardholder addresses, fraudsters can still exploit vulnerabilities to perpetrate fraud.

Some studies categorize credit card fraud into two main types: **offline credit** **card fraud**, where the physical credit card is stolen and used by fraudsters, and **online credit card fraud**, where fraudsters obtain cardholder details through methods like skimming, site cloning, credit card generators, or phishing.

This project focuses specifically on bank fraud, with a primary emphasis on online credit card fraud within the banking sector.

**2.4 Issues and Challenges with Fraud Detection Systems**

Fraud detection systems encounter several challenging issues, with four main topics commonly discussed: concept drift, skewed class distribution, large amounts of data, and real-time detection support.

Concept drift, as defined by previous studies occurs when the model is trained on a specific pattern of customer or imposter behaviour but encounters change in behaviour afterward. This lack of adaptability can hinder the system's ability to efficiently distinguish between fraudulent and legitimate transactions, emphasizing the importance of dynamic learning mechanisms within Fraud Detection Systems (FDS).

Skewed class distribution, as described by previous studies, is one of the most critical issues faced by FDS and refers to heavily imbalanced data, where instances of fraudulent transactions are significantly fewer than legitimate ones. Various approaches exist to address this challenge, including data-level techniques and algorithmic adjustments, which will be detailed in Section 2.5.

Lastly, real-time detection is crucial for the system to detect fraud promptly and take necessary actions swiftly. Various techniques have been developed to enhance real-time detection capabilities, such as Very Fast Decision Tree (VFDT) and Self-Organization Map (SOM).

This project will focus specifically on addressing the challenges of skewed class distribution and handling large amounts of data within fraud detection systems.

**2.5 Skewed Data**

In credit card transaction datasets, the number of fraudulent transactions is significantly lower than non-fraudulent transactions, resulting in as a skewed dataset, characterized by a substantial imbalance in the number of data points for each class. This skewness is evident in the dataset analysed in this project.

Skewed datasets pose challenges for fraud detection, making the identification of fraudulent transactions difficult and imprecise. This challenge arises because classification models, or classifiers, may exhibit unbalanced performance across different classes or they might overlook the minority class entirely. To address this issue, there are two main approaches categorized by data-level approaches and algorithmic-level approaches.

Data-level approaches involve preprocessing the dataset to balance it before feeding it into the classifier. This balancing enhances the classifier's ability to correctly classify different classes. Data-level approaches can be further divided into two subcategories: under sampling and oversampling.

Undersampling involves reducing the size of the majority class by removing some of its observations until the dataset is balanced. Previous studies describe two under sampling techniques: Random Undersampling (RUS) and Direct Undersampling. RUS randomly removes data from the majority class, while Direct Undersampling selectively removes observations.

On the other hand, oversampling focuses on increasing the number of observations in the minority class to achieve balance. However, oversampling may lead to overfitting the model to the minority class. The Synthetic Minority Oversampling Technique (SMOTE) is a widely used oversampling technique that generates synthetic minority data points near existing ones in the feature space.

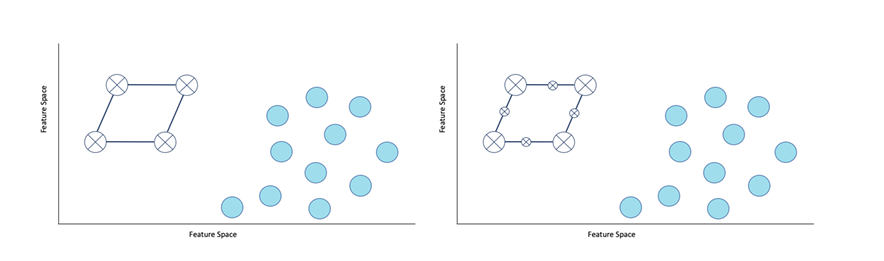


Figure 2.3: The filled data points belong to the majority class while the big points with a cross belongs to the minority class. In both the left- and right-hand side figure K = 2 since each data point in the minority class creates vectors to its two closest neighbours within the same class. Left: Before SMOTE has been applied to the dataset. Right: After SMOTE has been applied to the dataset and new samples are generated from the minority class on these K vectors, visualised as small data points with a cross.

Algorithmic-level approaches, in contrast to data-level approaches, do not preprocess the data. Instead, they modify the model itself. This can involve using cost-sensitive learning or adapting classification algorithms to handle minority class detection, such as One-Class Learner.

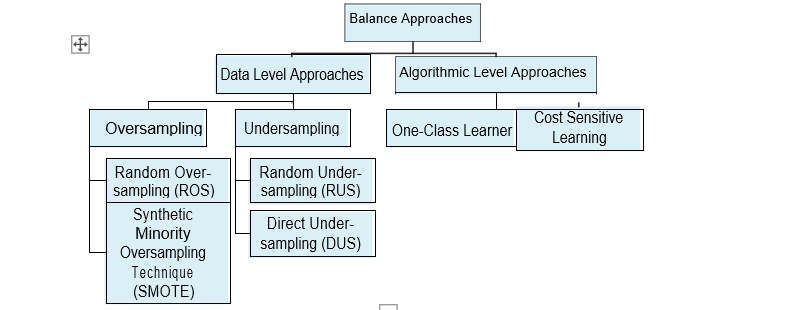


Figure 2.4: Balance approaches to handle skewed data.

Today, data-level approaches are favoured for addressing skewed data problems due to their ease of implementation and minimal impact on computing resources and training time. In alignment with these considerations, SMOTE has been chosen as a suitable data-level approach for this thesis.

**2.6 Performance Measurement**

Evaluating the performance of a machine learning model is crucial for assessing its accuracy and effectiveness, particularly in tasks like credit card fraud detection. While accuracy is a fundamental metric, it may not be suitable for imbalanced datasets common in fraud detection. Hence, alternative performance measurements are necessary to provide a comprehensive understanding of the model's capabilities.

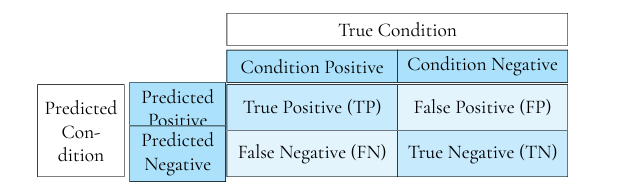
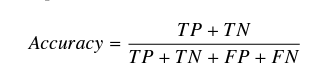
****

Figure 2.5: Confusion matrix with outcomes for a binary classifier.

Using these outcomes the performance measurement called Accuracy is de ned as



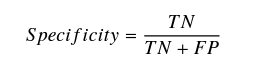
However, accuracy may not be suitable for credit card fraud detection due to data skewness. Instead, alternative performance metrics are recommended, such as cost-sensitive learning, Area Under the Curve Receiver Operating Characteristic (AUC-ROC), and Area Under the Precision-Recall Curve (AUPRC).

Cost-sensitive learning assigns misclassification costs to different classes or instances, aiming to minimize the total misclassification cost. While effective, obtaining accurate cost information from banks, can be challenging, leading to the adoption of other metrics like AUC-ROC and AUPRC.

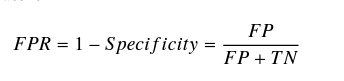
AUC-ROC evaluates a model's discrimination ability, measuring its capacity to differentiate between classes. It plots True Positive Rate (TPR) against False Positive Rate (FPR), with a higher AUC score indicating better predictive performance.



which measures the fraction of actual fraudulent transactions that are correctly classified as fraud. In order to define False Positive Rate, we first define Specificity as



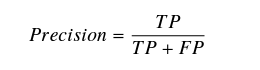
and now, False Positive Rate is



On the other hand, the Precision-Recall Curve plots Precision against Recall (TPR), where Precision measures the fraction of correctly classified fraud cases, and Recall represents the fraction of actual fraud cases correctly identified. A larger area under the precision curve signifies a more effective model.

In scenarios with skewed data, where true negatives are abundant, AUC-ROC may yield biased results, leading to the preference for AUPRC, which is more suitable for imbalanced datasets by excluding true negatives. Hence, this work will utilize the AUPRC measurement along with the confusion matrix for evaluating model performance in fraud detection.

Precision-Recall Curve as a plot of Precision versus Recall, i.e. TPR. Precision is defined as



**2.7 Centralised Learning**

Fraud detection systems have traditionally operated in a centralised manner, where machine learning algorithms are trained on locally stored data from a single server. This approach involves creating a model that learns patterns from the data to identify fraudulent transactions. Banks typically adopt this centralised approach independently due to privacy and security concerns associated with credit card transaction datasets.

**Choosing the Algorithm**

Various algorithms are available for detecting fraudulent transactions in a centralised system, each with its strengths and weaknesses. To select the most suitable algorithm, we referred to various research. This study compared different algorithms for credit card fraud detection, and the Multi-Layer Perceptron (MLP) emerged as the top-performing model. Hence, we've chosen MLP as the baseline algorithm for this thesis.

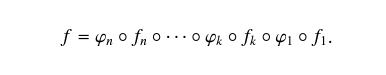
**Multi-Layer Perceptron (MLP)**

The Multi-Layer Perceptron is a type of artificial neural network that consists of multiple layers of nodes, each connected to the next layer. It's known for its ability to learn complex patterns and relationships in data, making it well-suited for tasks like fraud detection.

In the next section, we'll delve deeper into the workings of the Multi-Layer Perceptron and its application in detecting credit card fraud.

**2.8 Multi-Layer Perceptron**

Artificial neural networks (ANNs) are a collection name for many algorithms that almost all have in common that they are inspired by the construction and functioning of the human brain. Generally, ANNs can be described as a class of parameterised functions **f (x, w)** constructed by composing linear and non-linear functions as



Here, fn is a linear function parameterised by its weights wn while ϕn is a non-linear function. Usually, the function fn is referred to as a layer and ϕn is known as the activation function.

Each layer in ANN is composed by a set of nodes where the first set of nodes is referred to as the input layer, the last set of nodes is the output layer and if there are any set of nodes between these layers than they are referred to as hidden layers, see the right-hand side of Figure 2.7. Each weight is usually visualised as branches between the layers and depending on how they are connected to the nodes in the model the layers have different properties. For instance, a simple layer is referred to as a fully connected layer which means that the edges between every input node is connected to every output node, hence, the output layer is a linear combination of the input layer. Furthermore, the simplest ANN model is the single perceptron which only has an input and output layer, see Figure 2.6. This model takes a weighted sum of the input x and passes it through a beforehand determined activation function ϕn which then computes the output to be a probability between zero and one, i.e. a forward pass has been made.

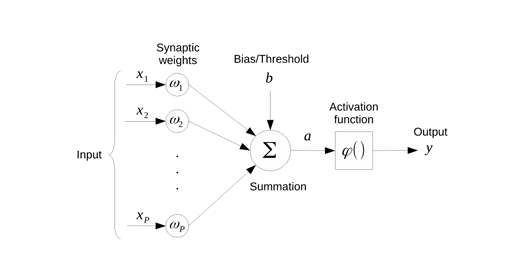
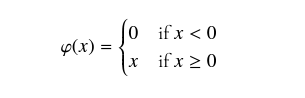


Figure 2.6: A simple perceptron with an input layer composed by the nodes x1–xp and an output layer with only one output y

A simple perceptron with p inputs and different weights performs well when the task is to detect linear separable patterns, though, this is rarely the case when aiming to detect credit card fraud. Instead, a more complex ANN model is used, namely, the Multi-Layer Perceptron, which has shown to perform well on credit card fraud detection in their article. In contrast to the simple perceptron, the MLP does not only have p input nodes and m nodes in its output layer, but the network also has hidden layers with an arbitrary number of nodes MLP as a feed-forward network, i.e. there are no recurrent connections in the network, composed of fully connected layers. Now, for each layer an activation function ϕ is applied and for a larger network the Rectifier Activation Function (ReLU) is popular, Also, when working with a binary classification problem the Logistic, also called Sigmoid, activation function is applied to the last layer of nodes which then results in the output being between zero and one. Below are the two activation functions ReLu and sigmoid respectively,



and



where ϕ(x) is the activation function which also appears on the left-hand side in Figure 2.7, where it can be seen that the function is applied to every node in each layer.

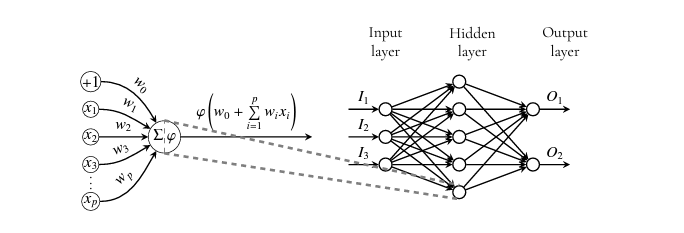
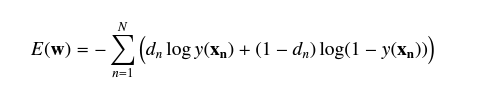
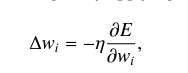


Figure 2.7: Left: A simple perceptron where the input is summed up and set as the argument of the activation function ϕ(x). Right: A Multi-Layer Perceptron with one input layer (three nodes), one hidden layer (five nodes) and one output layer (two nodes). The arrows between the nodes and neighbouring layer represents the weights, wk, which are pointed in a forward direction, i.e. it is a feed-forward network.

After a forward pass in the network the weights need to be optimised and this is referred to as training the model. The goal of the optimisation procedure is to minimise the error function, also referred to as the loss function. The main idea is that the network will eventually reduce its error as the network learns from the training dataset. A common error function for binary classification is the Cross-Entropy Error which is de ned as

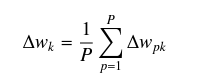


where w are all weights, dn is the target value for pattern n and y the prediction. The function can be minimised with respect to the weights by applying Gradient Descent



where the learning rate controls the size of the update. This way of updating the weights is called back propagation because, once a forward pass has been done to compare the actual output with the target then a backward pass is done to update the weights. The Gradient Descent algorithm is the most popular when training an MLP [.](#page63)

some studies states that Gradient Descent is not the most efficient optimiser algorithm to use since it only follows the gradient of an entire training set downhill and therefore can suffer from getting stuck in local minimum and regions of small gradients. Instead, they argue that one improvement might be to add a dynamical learning rate, in contrast to the previously fixed one which is set by default, to speed up the minimisation of the error function. The learning rate may be chosen by trial and error, but it is usually best to choose it by monitoring learning curves that plot the objective function as a function of time or training epochs. The optimiser we have chosen is Mini-Batch Gradient Descent (MBGD), which probably is one of the most used optimisation algorithms for machine learning in general and, in particular, for deep learning. Mini-Batch Gradient Descent refers to calculating the derivative from each training data instance and compute an immediate update. The MBGD optimiser updates the weights by using a small number of patterns, i.e. mini-batches. Formally, MBGD is written as



Where ∆wk is the kth weight to be updated and P is the size of the mini-batch. Every time one mini-batch has been used to update the weights one iteration has been performed. When all mini-batches have been iterated it is said that one epoch has been performed. MBGD has the important property that computation time per update does not grow with the number of training examples, which allows convergence even when the training dataset becomes very large. Finally, it is important to keep in mind that, when building a large network, it might get too specific, i.e. the network gets over trained, as a consequence of having too many layers and nodes to find features in the training dataset. Therefore, when building the MLP the architecture needs to be carefully chosen and thoroughly tested to avoid both over- as well as underfitting. An implementation of our baseline model, the Multi-Layer Perceptron, is explained in Section 4.3

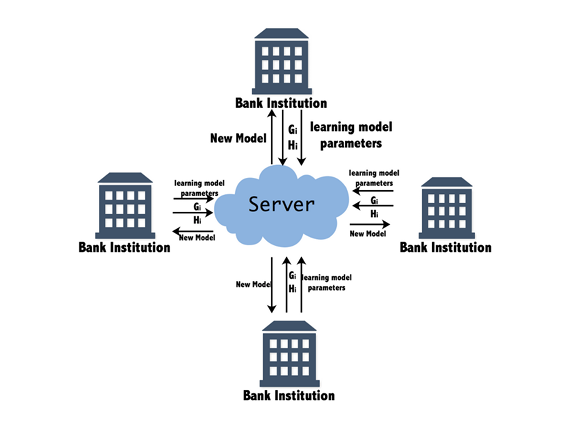
**2.9 Federated Learning**

Traditionally, centralised learning methods face challenges in detecting credit card fraud due to dataset insufficiency, skewed data distribution, and limitations in detection time. To address these issues and protect user privacy, federated learning (FL) emerges as a promising alternative.

FL involves multiple clients collaboratively training a model under the coordination of a central server. The key principle is that clients' raw data remains locally stored and is not exchanged; instead, only model parameter updates are communicated to the server.

FL is not limited to edge devices but also extends to collaborations among multiple organizations, termed cross-silo FL. We can define FL as a setting where clients collaborate under a central server's coordination, with each client's raw data stored locally and focused updates used for aggregation.

The typical FL process involves client selection, broadcast of model updates, client computation, aggregation of updates, and model updating. This process repeats until convergence is achieved.



Applications of FL include credit card fraud detection, enabling banks to collaboratively learn a shared model while maintaining data privacy locally. This approach allows the central model to learn from diverse datasets, potentially improving fraud classification accuracy.

Privacy and communication are paramount in FL, with techniques for secure computation and privacy-preserving disclosure being crucial. Machine learning algorithms like Federated Averaging are common in FL, offering neural network-based solutions suitable for federated environments.

For this project, Federated Averaging is chosen as it aligns with the baseline MLP model, facilitating performance comparison and adaptation for building a federated Fraud Detection System (FDS).

**2.10 Non-Identical Independent Distributed Data**

In traditional machine learning, the assumption of Independent and Identically Distributed (IID) data is common, implying that samples are independent and originate from the same generative process. However, in federated learning (FL), datasets are typically non-IID. For example, datasets in FL are generated locally in different contexts and by different clients, leading to non-identical distributions across participants.

**2.10.1Types of Non-IID Data**

Previous studies identify several sources of non-identicalness in FL datasets:

Feature Distribution Skew: Marginal distributions of features may vary across clients, even when the conditional distributions of labels given features are the same.

Label Distribution Skew: Marginal distributions of labels may vary across clients, despite the same conditional distributions of labels given features.

Same Label, Different Features: Different clients may have different feature sets for the same label due to various factors like cultural differences, geographical regions, or time effects.

Same Features, Different Label:Similar features may result in different labels across clients due to personal preferences or regional variations.

Quantity Skew or Unbalanced Ness**:** Clients may have vastly different amounts of data.

**2.10.2 Impact on Federated Learning**

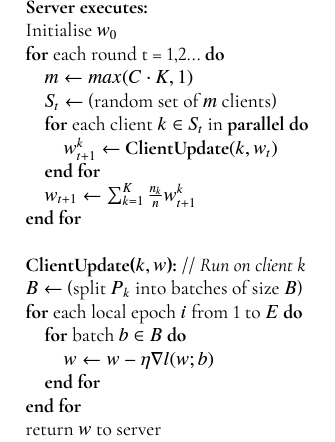
The presence of non-IID data poses challenges in FL, particularly for algorithms relying on Gradient Descent optimization. These algorithms assume IID data to ensure unbiased gradient estimates. However, non-IID data violates this assumption.

Despite these challenges, certain FL algorithms like Federated Averaging can still work with non-IID data to some extent, these algorithms adapt to the non-IID nature of the data to enable collaborative learning across distributed clients.

**2.11 Federated Averaging**

The neural network algorithm referred to as Federated Averaging is similar to the centralised MLP. The main idea for the Federated Averaging algorithm is that for each global epoch of the training a central server chooses a fraction of the participating clients. The central server then sends its model parameters to each of the selected clients. When the training is done locally the clients make one (or more) local weight update, i.e. a local epoch, by applying Mini-Batch Gradient Descent (explained in Section 2.8) and eventually their updated weights are sent back to the central server. Now, the central server updates the federated model by averaging all the weight updates from the participating clients and then the process is repeated until convergence or satisfying results are achieved. Usually, when working with machine learning algorithms the aim is to have as few epochs as possible but still obtain a model that performs well. This is because the number of epochs is directly correlated to computer power and memory needed. Also, more epochs effect the privacy issue by adding to the privacy budget. previous studies propose two approaches to limit the global epochs needed for convergence, i.e. limit the rounds of communications between the central server and the clients, namely, increase parallelism, i.e. increase the fraction of clients used, and increase local computation, i.e. increase the number of local epochs. When some studies tested these two approaches in their experiments increase local computation showed the best effect of limiting communication while still keeping the same performance. Background Moreover, the averaging of the weight updates in the central server could, be a problem when solving non-convex problems. They argue that in general, averaging models in parameter space can create an arbitrarily bad neural network model. This is because training several models on a non-convex problem can create significantly different models as they find different settings for the optimal weight parameters. Taking the average of models that have big disparity creates a poor central model. However, one method to minimise this problem is, according to the article, to use shared initialisation for the local models. In other words, each client is initialised with the same weights for every global epoch. This is how Federated Averaging is implemented in our thesis since, at the start of each global epoch the central server initialises the clients with the same weights as its own. In Algorithm 1 the pseudo code for Federated Averaging is shown.

**Algorithm 1** Federated Averaging. Here, C is the fraction of clients used, K are the clients indexed by k, n is the total number of data points, nk is the number of data points in client k, Pk is the local dataset, B is the local mini-batch size, E is the number of local epochs and lastly η is the learning rate



**2.12 Edge Computing in Federated Learning**

In the context of federated learning (FL), edge computing plays a pivotal role in enabling efficient and distributed model training while maintaining data privacy and security. Edge devices such as smartphones, IoT devices, and local servers act as endpoints for data processing and model training, contributing to the federated learning paradigm.

**Benefits of Edge Computing in Federated Learning:**

**Data Localization:** Edge devices store and process data locally, reducing the need for centralized data storage and minimizing data transfer, which enhances privacy and reduces latency.

**Real-time Inference:** Edge devices can perform real-time inference and decision-making based on locally trained models, leading to faster response times and improved user experience.

**Bandwidth Efficiency:** By processing data and training models locally, edge computing reduces the amount of data transmitted over networks, optimizing bandwidth usage.

**Privacy Preservation:** Since data remains localized on edge devices, sensitive information is not exposed to external networks, enhancing data privacy and security.

**Scalability:** Edge computing allows for distributed computing across a network of devices, enabling scalable and parallelized model training without overloading centralized servers.

**Challenges and Considerations:**

Resource Constraints: Edge devices may have limited computational power, memory, and battery life, which can impact the complexity and size of models used in federated learning.

**Heterogeneity:** Edge devices in FL environments may vary in terms of hardware capabilities, network connectivity, and data distribution, requiring adaptive and robust algorithms.

**Security Risks:** Edge devices are susceptible to security threats such as data breaches, tampering, and malicious attacks, necessitating robust encryption, authentication, and access control mechanisms.

**Data Synchronization:** Ensuring data consistency and synchronization across edge devices during model training and aggregation poses challenges in FL systems.

Edge Computing Techniques in Federated Learning:

Model Compression: Techniques like model pruning, quantization, and distillation are used to compress models before deployment on edge devices, optimizing resource utilization.

**On-device Training:** Edge devices perform local model training using subsets of data, leveraging techniques like federated learning, transfer learning, and online learning for continuous model updates.

Data Filtering and Preprocessing: Edge devices preprocess and filter data locally before transmitting relevant information to central servers, reducing data transfer overhead.

**Edge Server Coordination:** Edge servers act as intermediaries between edge devices and central servers, coordinating model updates, aggregations, and communication protocols in FL systems.

In summary, edge computing plays a crucial role in federated learning by enabling distributed data processing, model training, and inference at the network edge, thereby improving efficiency, privacy, and scalability in machine learning applications.

**3. Methodology**

**3.1 Dataset**

In Figure 3.1 it can be seen that the time for non-fraudulent transactions have a rather clear periodical pattern that follows day and night cycles, while fraudulent transactions do not.

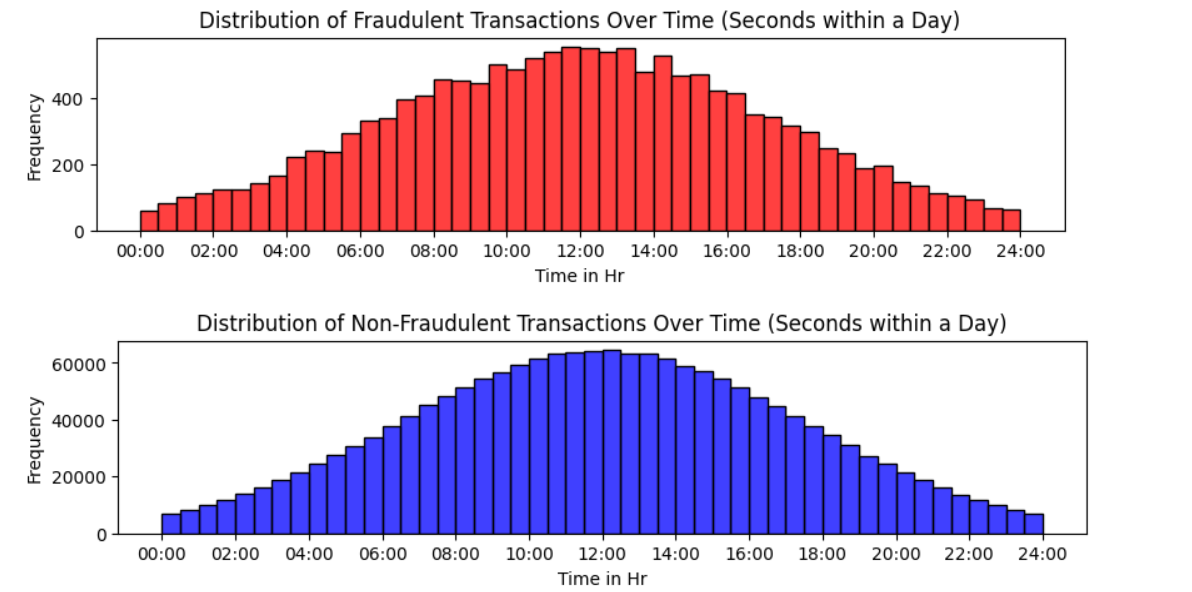
****

Figure 3.1: Number of transactions against time for the different classes.

Further, in Figure 3.2 it can be seen that the fraudulent transactions are usually small while the non-fraudulent transactions occur within a larger span of money.

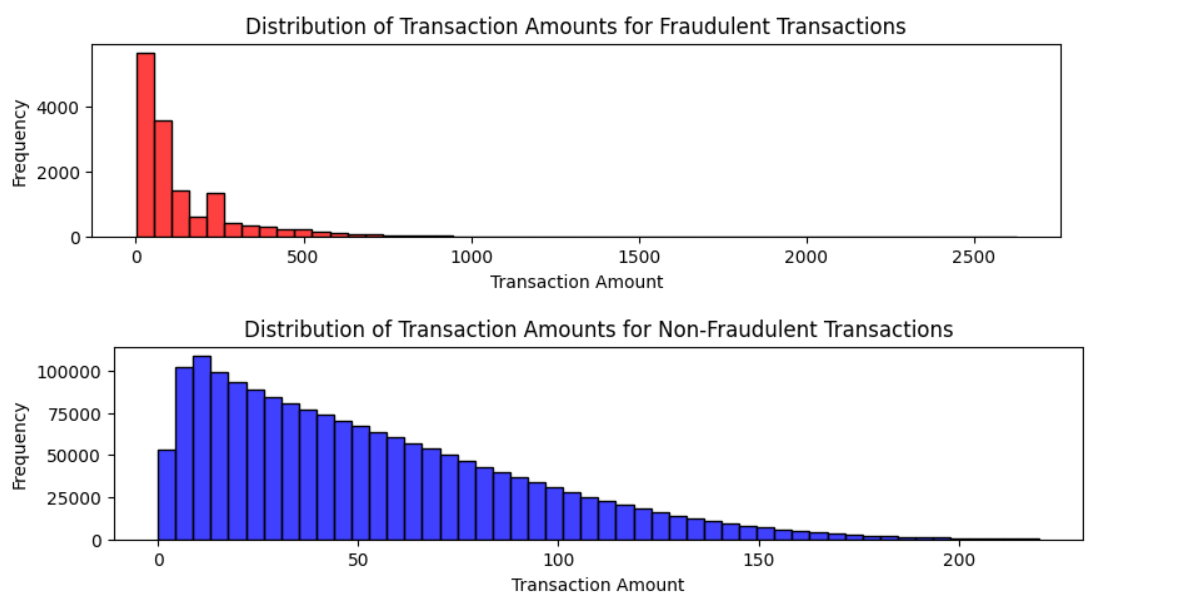
****

Figure 3.2: Number of transactions against amount for the different classes.

As a final data analysis, the distributions for fraudulent and non-fraudulent transactions in time and amount have been carried out for all features. It can be seen that for some features the curves for fraud and not fraud overlaps to a greater extent than for others. The interpretation of this is that, the more overlap of the curves the more difficulties to extract differences between fraudulent and non-fraudulent transactions. When the distributions do not overlap at all or very little it is easier to and these differences. This means that, for a model to be able to detect the differences between a fraudulent and non-fraudulent transaction the distributions should have as little overlap as possible.

**3.2 SMOTE for Balancing Dataset in Fraud Detection**

Synthetic Minority Over-sampling Technique (SMOTE) is an innovative oversampling method used to address the class imbalance problem in datasets used for training machine learning models. It works by synthesizing new examples from the minority class (fraudulent transactions in this case) to balance the class distribution, thus providing better training conditions for predictive models.

**Mathematical Formulation**

SMOTE algorithm generates synthetic samples by operating in the feature space rather than the data space. For each minority class sample, it selects its

k nearest neighbours from the minority class. Synthetic samples are then created by choosing one of the

K nearest neighbours and connecting the line segment between the minority class sample and its chosen neighbour. Samples are generated randomly along this line, according to the formula:

New Sample = Existing Sample + λ × (Neighbour Sample − Existing Sample)

where λ is a random number between 0 and 1.

**Algorithm Steps**

Neighbour Selection: For each sample in the minority class, identify its k nearest neighbours in the feature space.

Synthesis of New Samples: Generate synthetic samples by interpolating between each minority class sample and its selected neighbours.

Integration into Dataset: Combine the original dataset with the synthetic samples, resulting in a balanced dataset that improves the learning process for subsequent models.

**3.3 Bank Grouping:**

**Particle Swarm Optimization (PSO) for Node Assignment**

Particle Swarm Optimization (PSO) is a computational method that optimizes a problem by iteratively improving a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best-known position, but is also guided toward the best-known positions in the search space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

**Mathematical Formulation**

The problem at hand is to optimally assign N bank nodes to M server nodes, minimizing the total cost defined by:

**Cost =𝛼 × Distance + 𝛽 × Load**

where Distance is the Euclidean distance between assigned bank and server nodes, and Load is the number of bank nodes assigned to a particular server, influencing server capacity strain. Here,

𝛼 and 𝛽 are weighting factors that balance the importance of distance minimization and load balancing.

**PSO Algorithm Application**

Initialization: Generate an initial population (swarm) of particles. Each particle represents a potential solution to the node assignment problem, encoded as an array of server indices for each bank node.

Velocity and Position Update: At each iteration, update the velocity of each particle toward its personal best and global best solutions. Adjust each particle's position based on its velocity, respecting the problem's constraints.

Fitness Evaluation: Calculate the fitness of each particle using the cost function, which integrates both the distance and the load.

Update Bests: If a particle finds a new personal best solution, update its personal best. If this personal best is better than the global best, update the global best as well.

**3.4 MLP and Federated Averaging**

We opted for a Multi-Layer Perceptron (MLP) integrated with Federated Averaging (FedAvg) due to its robust ability to model complex, non-linear relationships inherent in financial transaction data. The MLP is ideal for this task, given its flexibility in architecture which allows it to learn from a multitude of features that characterize fraudulent activities. Utilizing two hidden layers enables the MLP to extract and refine a wealth of subtle patterns from the data, which are critical for accurately detecting fraud.

Incorporating Federated Averaging enhances the model's capability by leveraging decentralized datasets across various nodes (e.g., different banks) without compromising data privacy. This approach is vital in environments where data cannot be shared due to privacy regulations or security policies. Federated Averaging allows for the collective intelligence of diverse datasets to be harnessed, improving the generalization of the model while adhering to privacy concerns. By training locally and averaging model updates centrally, FedAvg effectively broadens the model's exposure to varied data patterns, enhancing detection efficacy and making it superior to traditional centralized learning methods that lack robustness against diverse or skewed data distributions.

**4. Model Implementation**

**4.1 Banks Grouping**

The PSO algorithm was implemented to manage the assignment of 600 bank nodes to a subset of 90 server nodes, each with a specific communication range. The process began with the random initialization of particle positions within the feasible space, ensuring no server assignment exceeded the communication range of 1000 units. Each particle's position represents a specific assignment of bank nodes to servers, and its velocity represents the potential changes to this assignment.

**Visualization of Assignments**

To visually verify the algorithm's efficacy, in fig. 4.1 we plotted the final assignments of bank nodes to servers. This visualization depicted bank nodes and server nodes in different colours, with lines connecting each bank node to its assigned server, illustrating how effectively the PSO algorithm optimized the placement to reduce overall costs and balance server loads.

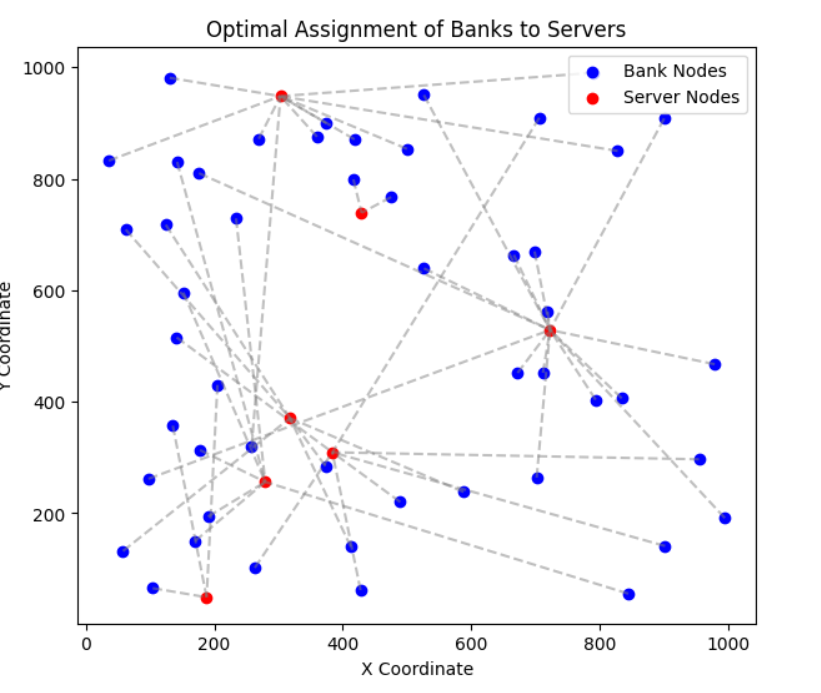
****

Figure 4.1 optimal assignment of bank to the server using PSO

**4.2 Implementing SMOTE in Credit Card Fraud Detection**

To address the significant class imbalance typically present in credit card transaction data, where fraudulent transactions are much rarer than legitimate ones, SMOTE was applied to the training data before model training. The implementation involved the following steps:

**Preprocessing Steps**

Data Cleaning: Initial preprocessing to handle missing values and outliers, ensuring data quality for effective SMOTE application.

Feature Selection: Identification of relevant features that influence the prediction of fraudulent transactions, based on domain knowledge and exploratory data analysis.

Normalization: Scaling of features to standardize the data, ensuring that the SMOTE algorithm operates effectively in the feature space.

Application of SMOTE

Parameter Tuning: Choosing the appropriate number of nearest neighbours (k) to use for generating synthetic samples based on validation performance.

Synthetic Sample Generation: Applying SMOTE to the minority class in the dataset to create a balanced class distribution.

**4.3 Multilayer perceptron**

In our fraud detection system, we employ a Multi-Layer Perceptron (MLP), a type of feedforward artificial neural network, to classify transactions as fraudulent or non-fraudulent. The MLP is particularly suited for this task due to its ability to learn non-linear models in complex datasets involving numerous input features.

**Model Configuration**

The MLP model is configured with the following specifications:

Input Layer: The number of neurons corresponds to the number of features in our dataset, excluding the target feature ('TX\_FRAUD'). Each neuron in this layer represents a unique input feature such as transaction amount, time, customer information, etc.

Hidden Layers: The first hidden layer contains 64 neurons. This layer is crucial for capturing the complex patterns in the data, which are not immediately obvious to the human eye or basic analytical techniques.

The second hidden layer consists of 32 neurons and serves to further refine the features processed by the first hidden layer, enhancing the model’s ability to distinguish between fraudulent and non-fraudulent transactions.

Output Layer: Comprises a single neuron using a sigmoid activation function. This setup is typical for binary classification tasks, where the output is the probability of the input transaction being fraudulent.

Activation Function: The ReLU (Rectified Linear Unit) activation function is used for the neurons in the hidden layers due to its efficiency and effectiveness in helping neural networks converge faster. The output layer uses a sigmoid activation function to output probabilities.

**Training Process**

Data Preprocessing: Before training, the data is pre-processed using a Standard Scaler to normalize the features, ensuring that the model does not become biased towards variables on a larger scale.

Optimization Algorithm: The 'adam' optimizer, known for its adaptive learning rate capabilities, is used to minimize the binary cross-entropy loss during training. This choice helps in quickly converging to the optimal model weights and biases.

Iterations: The model is set to run for a predefined number of iterations (max iteration =100 for demonstration purposes, but typically a higher number is chosen based on the convergence behaviour observed during training).

**Model Evaluation**

Performance Metrics: Post-training, the model's performance is evaluated on a separate test set using metrics such as accuracy, precision, recall, and the area under the ROC curve. These metrics provide insights into the model's effectiveness in identifying fraudulent transactions.

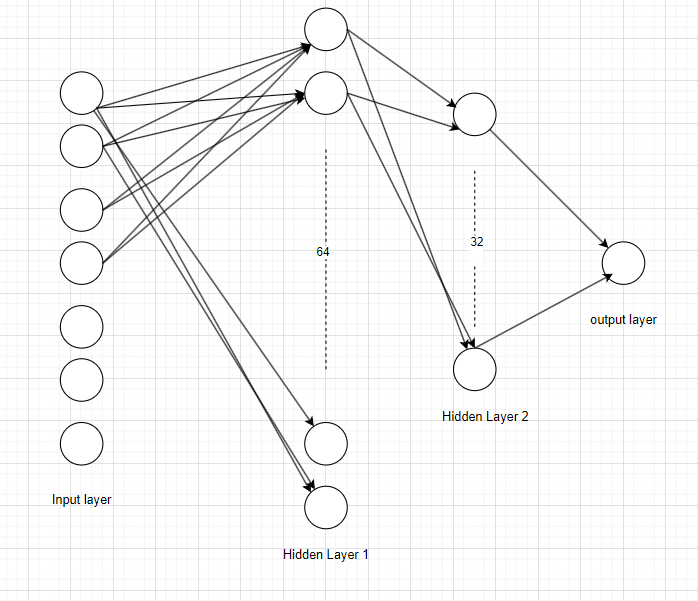


Fig. 4.2 MLP Architecture

**4.4 Federated Averaging**

**4.4.1. Aggregation at the Edge Node Level**

**Purpose:**

Each edge node aggregates model updates locally from multiple bank branches under its jurisdiction. This step reduces the variability of the updates before they are sent to the central server, enhancing overall model stability and performance.

**Process:**

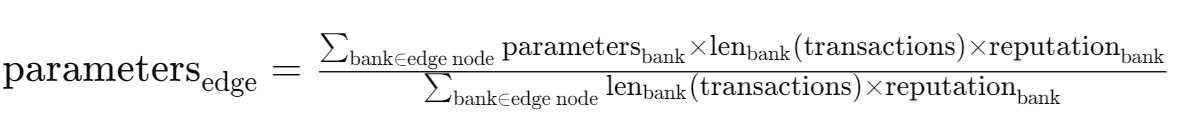
Initialization: Arrays for temporary storage of weights and biases are created for each edge node, initialized to zero. These arrays are sized to match the global model's architecture (three layers in this example).

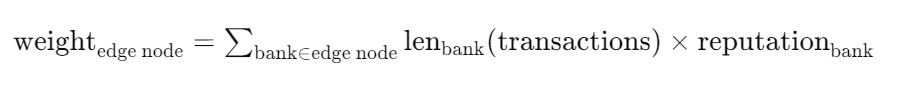
Weighted Sum Calculation: For each bank associated with an edge node, the model weights and biases are summed, weighted by the edge weight of the bank. This weighting factor might represent the bank's data volume or a reliability score, contributing to the overall influence of each bank's update.

The sums incorporate contributions from all relevant banks to produce a comprehensive set of weighted average parameters at each edge node.

Normalization: After computing the weighted sums, the aggregated weights and biases are normalized by dividing them by the total weight. This normalization accounts for the different sizes or importances of the banks, ensuring that no single entity disproportionately influences the model.

Updating Local Models: The normalized weights and biases are then used to update the local models at each edge node, preparing them for the next iteration of local training or for further aggregation at the central server.





**4.4.2. Aggregation at the Central Server Level**

**Purpose:**

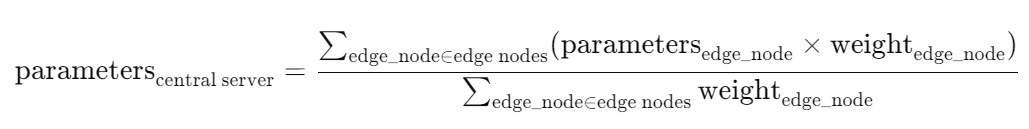
The central server further aggregates the updates received from various edge nodes to update the global model. This aggregation aims to harness the broader insights gathered across different regions or demographics, refining the model's ability to generalize across diverse scenarios.

**Process:**

Collection and Weighted Sum: Similar to the edge node process, the central server computes weighted sums of the weights and biases received from each edge node. The weights applied here can also reflect the volume of data or the strategic importance of the edge nodes.

Normalization: The aggregated weights and biases are normalized by the total weight contributed by all edge nodes. This step ensures that the global model update fairly represents the collective learning achieved across the network.

Global Model Update: The global weights and biases are updated with these normalized values, resulting in a new version of the global model that incorporates learning from across the network.



Distribution: Finally, the updated global model is distributed back to the edge nodes for another cycle of local training, thus closing the federated learning loop.

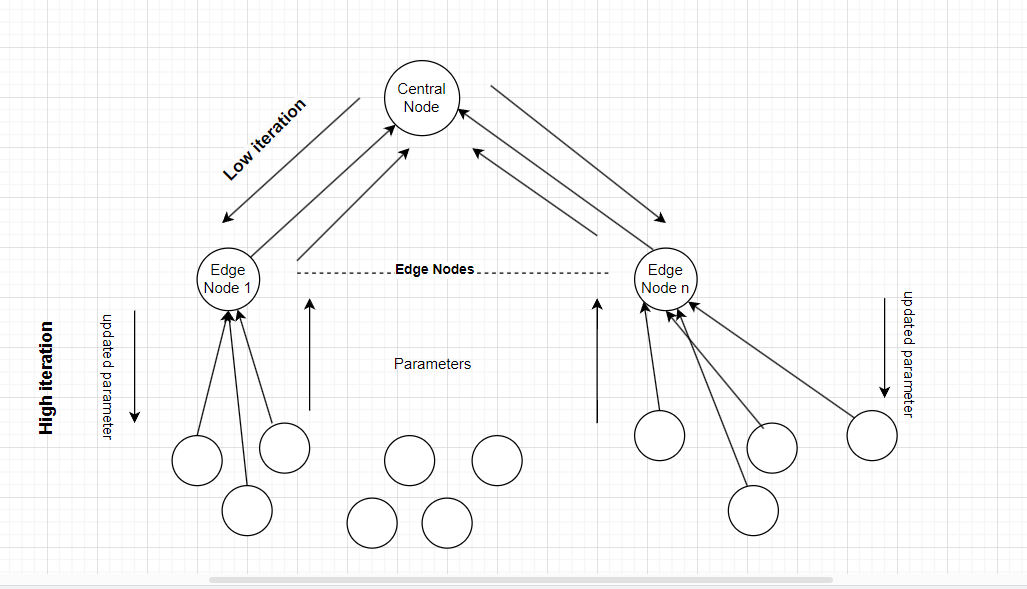


Fig. 4.3 Architecture of federated learning with edge computing as implemented

**5. Results**

In this section the results from the simulations will be presented. In addition to that the results from the Multi-Layer Perceptron and Federated Averaging are presented for the three different dataset settings, namely, skewed data, non-IID data and SMOTE non-IID data respectively.

**5.1 Banks Grouping**

**Performance Evaluation of PSO**

The performance of the PSO algorithm was evaluated based on its ability to minimize the cost function. The results were compared against other optimization models such as the Genetic Algorithm (GA) and Jaya Algorithm. Each model's effectiveness was quantified by the final value of the cost function, convergence speed, and the load distribution among servers.

**Comparative Analysis**

The PSO algorithm demonstrated a robust ability to quickly converge to an optimal or near-optimal solution, outperforming GA and Jaya in terms of cost minimization and computational efficiency. The load balance achieved by PSO was also more uniform, indicating its suitability for scenarios demanding both proximity considerations and capacity management.

**Graphical Representation**

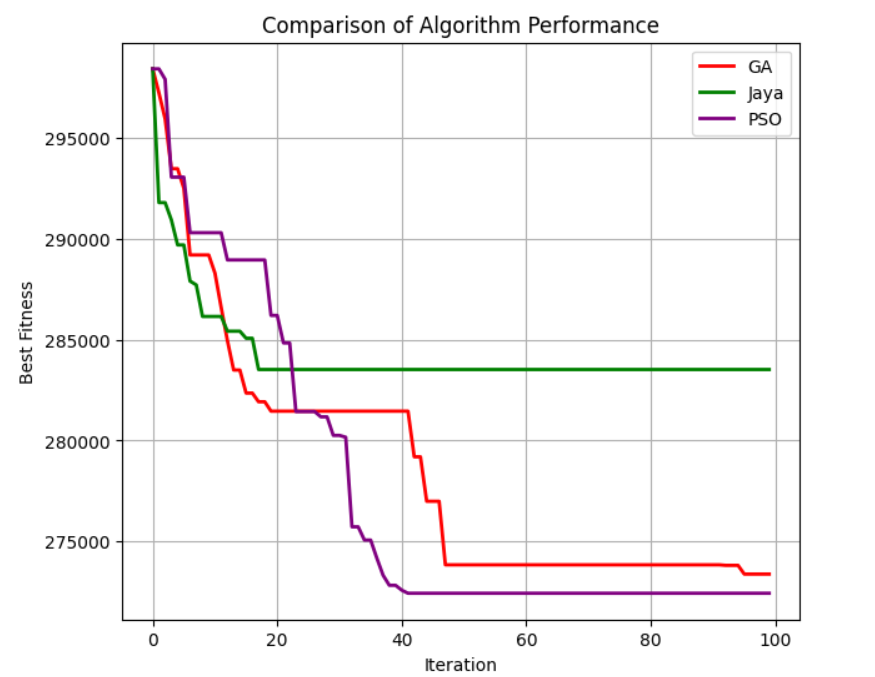
****

Fig 5.1 Graphs comparing the convergence of the different algorithms illustrated the iterations required for PSO to reach the lowest cost, juxtaposed against the trajectories of GA and Jaya Algorithm. This provided a clear visual representation of PSO's superior performance in handling the dynamic and complex problem of node assignment.

**5.2 Performance Evaluation of Models with SMOTE**

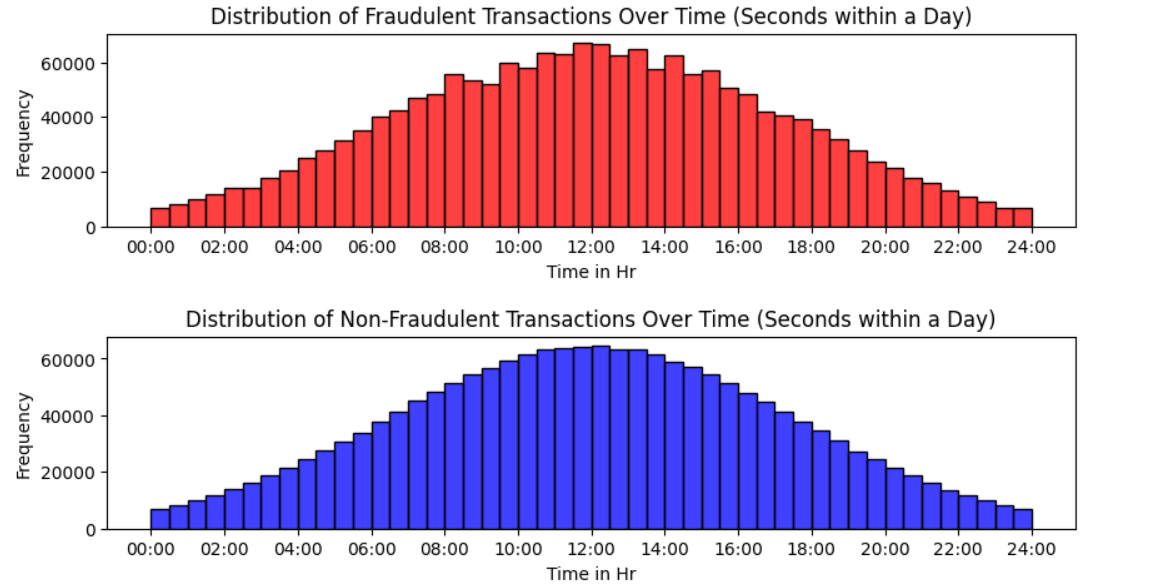
The impact of SMOTE on model performance was evaluated by training various classification models on both the original and the SMOTE-enhanced datasets. Key performance metrics such as accuracy, precision, recall, and the area under the ROC curve (AUC-ROC) were used to assess improvements.

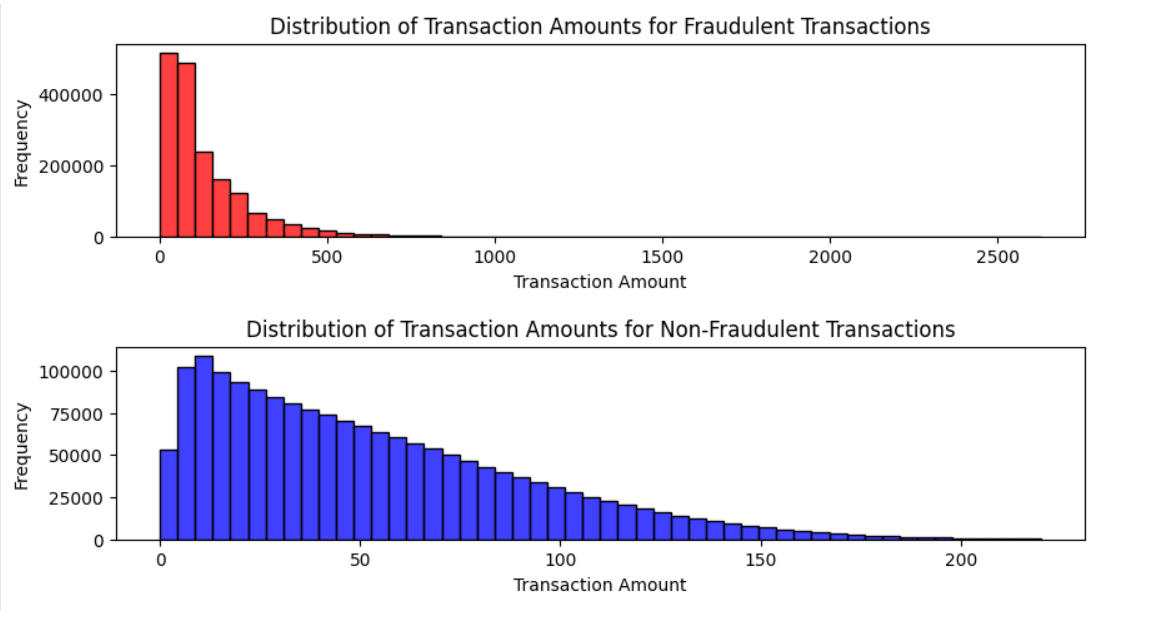
**Comparative Analysis**

Models trained on the SMOTE-enhanced dataset generally showed improved recall and precision for detecting fraudulent transactions, indicating the effectiveness of SMOTE in providing a more generalizable and robust predictive model. The improvement was particularly noticeable in complex models such as neural networks and ensemble methods that are sensitive to class distribution.

**Graphical Representation**

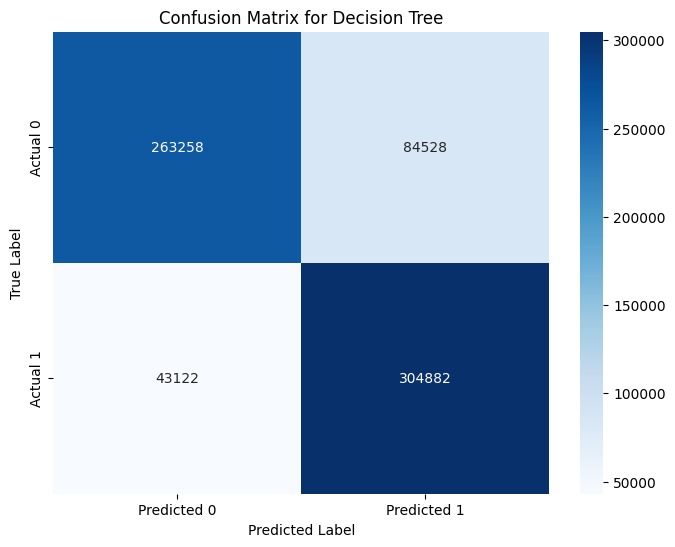
Graphs and charts comparing the performance metrics before and after applying SMOTE illustrated the benefits in terms of increased detection rates of fraudulent transactions and reduced false negatives. The improvements in the ROC curve and precision-recall curves provided visual evidence of the models' enhanced discriminatory power when trained on balanced data.

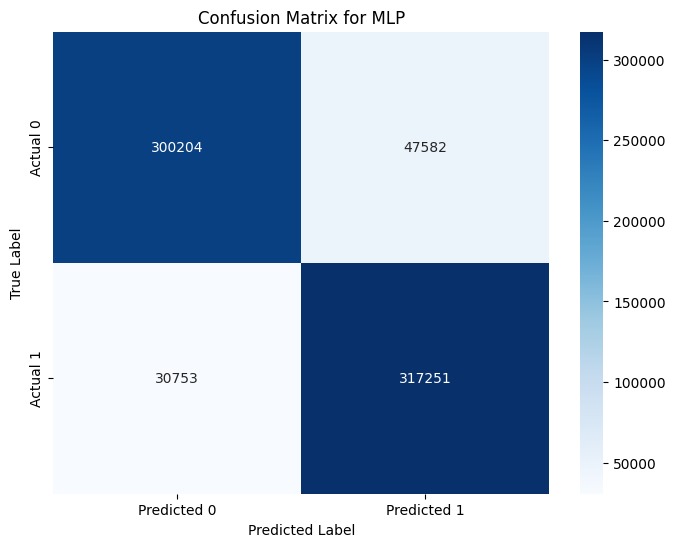


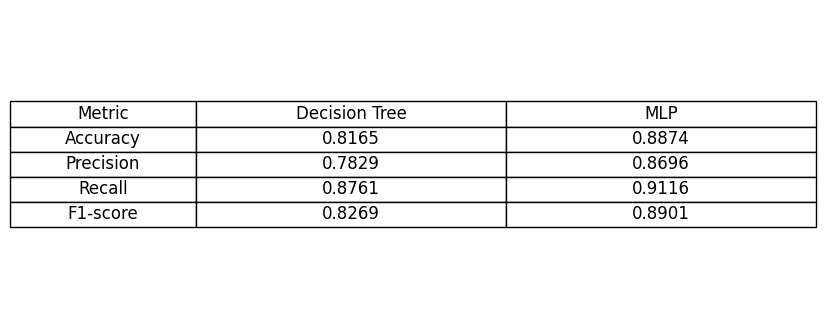


**5.3 Multilayer perceptron vs Decision Tree**

The performance of MLP is much better than the decision tree model, hence the MLP (DL) is selected for final implementation. the result metric is shown below:

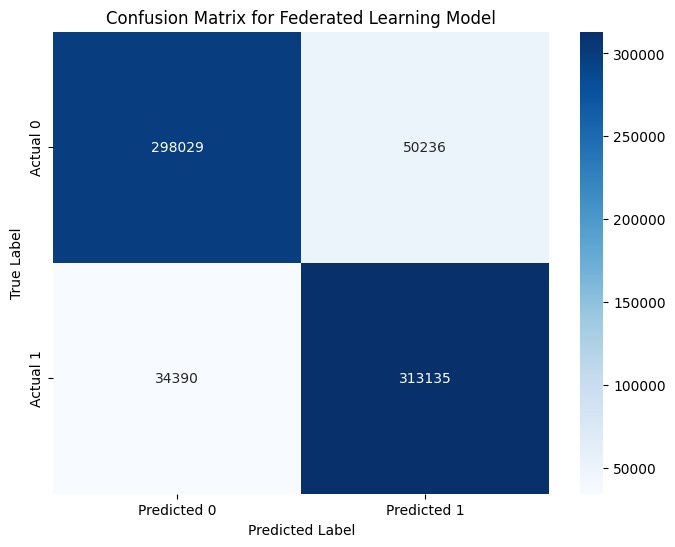






**5.4 Federated Learning with Edge Computing**

The result for overall federated learning model is as shown below:



|  |  |
| --- | --- |
| Performance Metric | Value |
| Accuracy | 0.8784 |
| Precision | 0.8618 |
| Recall | 0.9010 |
| F1 Score | 0.8810 |

**6. Conclusion**

In the ever-evolving landscape of cybersecurity and fraud detection, our exploration into federated learning (FL) coupled with advanced machine learning techniques has unearthed promising avenues for combating credit card fraud while preserving data privacy. Collaborating with IBM and Lund Technical University, we embarked on a journey to harness the power of FL, leveraging edge computing, advanced algorithms like Federated Averaging (FedAvg), and data-level techniques such as Synthetic Minority Oversampling Technique (SMOTE). Our endeavour was not just about building models; it was about paving the way for innovative, privacy-preserving solutions in the realm of financial security.

At the heart of our project lies Federated Averaging, a revolutionary approach to collaborative model training. Unlike traditional centralized models that require data aggregation in a central server, FedAvg allows multiple entities, such as banks, to train a shared model collaboratively without sharing raw data. This paradigm shift addresses the inherent challenges of data privacy and security, making it an ideal framework for financial institutions striving to combat fraud while safeguarding sensitive customer information.

The integration of edge computing into our federated learning environment played a pivotal role in optimizing model training and inference. Edge devices, acting as localized data processors, minimized data transfer and enhanced real-time inference capabilities. This not only improved response times but also mitigated privacy risks associated with data transmission over networks. The synergy between federated learning and edge computing signifies a paradigm shift in machine learning architectures, emphasizing privacy-preserving methodologies without compromising on model performance.

Our exploration into advanced machine learning algorithms, particularly Multi-Layer Perceptron (MLP) as a baseline model, underscored the potential of Federated Averaging in fraud detection. Despite limitations in training data, Federated Averaging demonstrated commendable performance, outperforming centralized models in certain scenarios. The incorporation of SMOTE further enhanced our model's ability to detect fraudulent transactions accurately, addressing challenges posed by skewed data distributions commonly encountered in fraud detection systems.

Looking ahead, our project opens doors for continued research and innovation in federated learning, edge computing, and privacy-preserving methodologies. Collaborative partnerships among financial institutions, coupled with advancements in algorithmic optimizations and secure computations, will drive the evolution of robust and scalable fraud detection systems. The impact of our research extends beyond credit card fraud detection; it lays the foundation for a future where data privacy and model performance coexist harmoniously, fostering trust and security in digital transactions.

In conclusion, our journey through federated learning, edge computing, and advanced machine learning techniques has not only shed light on the potential of these technologies but also underscored the imperative of privacy-preserving methodologies in modern cybersecurity. As we navigate the complexities of financial security, collaborative efforts, technological innovations, and a steadfast commitment to data privacy will continue to shape the landscape of fraud detection, paving the way for a more secure and resilient digital ecosystem.