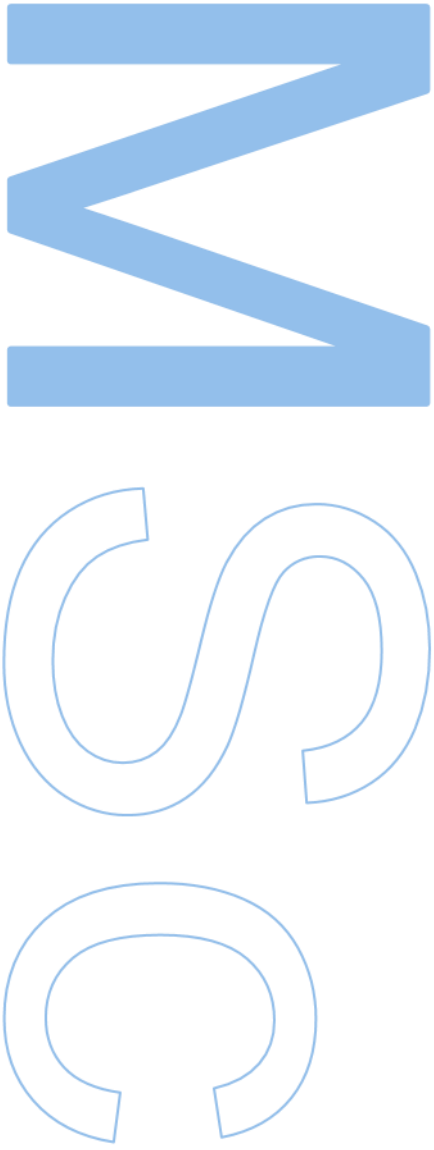
A Federated Learning Platform for High Speed Distributed Data Streams

João Pedro de Castro Ferreira
Mestrado Integrado de Engenharia de Redes e Sistemas Informáticos
Departamento de Ciências de Computadores
2023

Orientador
João Vinagre, Professor Auxiliar Convidado, Faculdade de Ciências da Universidade do Porto




U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Declaração de Honra

Eu, João Pedro de Castro Ferreira, inscrito(a) no Mestrado em Engenharia de Redes e Sistemas Informáticos da Faculdade de Ciências da Universidade do Porto declaro, nos termos do disposto na alínea a) do artigo 14.º do Código Ético de Conduta Académica da U.Porto, que o conteúdo da presente dissertação/relatório de estágio/projeto projeto "A Federated Learning Platform for High Speed Distributed Data Streams" reflete as perspetivas, o trabalho de investigação e as minhas interpretações no momento da sua entrega.

Ao entregar esta dissertação/relatório de estágio/projeto "A Federated Learning Platform for High Speed Distributed Data Streams", declaro, ainda, que a mesma é resultado do meu próprio trabalho de investigação e contém contributos que não foram utilizados previamente noutros trabalhos apresentados a esta ou outra instituição.

Mais declaro que todas as referências a outros autores respeitam escrupulosamente as regras da atribuição, encontrando-se devidamente citadas no corpo do texto e identificadas na secção de referências bibliográficas. Não são divulgados na presente dissertação/relatório de estágio/projeto "A Federated Learning Platform for High Speed Distributed Data Streams" quaisquer conteúdos cuja reprodução esteja vedada por direitos de autor.

Tenho consciência de que a prática de plágio e auto-plágio constitui um ilícito académico.

João Pedro de Castro Ferreira

Portugal, Porto,

07/05/2023

Abstract

In the era of digitalization, data has become an invaluable resource, encompassing various aspects of our daily lives. Machine Learning (ML) techniques enable us to extract meaningful patterns from data for diverse applications, including commercial and medical fields. However, the collection and analysis of data raise concerns about privacy and scalability.

This thesis focuses on the critical challenge of fraud detection in phone calls, faced by telecommunication companies. Traditional Centralized Learning (CL) approaches encounter privacy and scalability issues due to the sensitivity of customer data. To address these challenges, we propose a novel federated system for fraud detection in phone calls. Our approach leverages the power of Distributed Learning (DL) while preserving data privacy. The system enables multiple telecom service providers to collaboratively train ML models using their respective datasets, without sharing raw data.

Through extensive experiments on real-world call data, we demonstrate the competitive performance of our federated system compared to other learning approaches. The federated system effectively detects fraudulent phone calls while ensuring the privacy of sensitive customer information, which remains localized to each provider. Additionally, the scalability analysis reveals the efficiency of our federated system in handling large-scale datasets by distributing the computational load across providers.

The results highlight the potential of Federated Learning (FL) as a promising technique for fraud detection in the telecommunication industry. Furthermore, our findings open avenues for future research in this domain, exploring additional applications and addressing further challenges.

Keywords: Federated learning, fraud detection, privacy-preserving, scalability, comparative analysis.

Acknowledgements

I would like to express my heartfelt gratitude to all the contributors from the University of Coimbra who generously offered their assistance and guidance throughout the development of my project. Their unwavering support and valuable insights, stemming from their ongoing project, have been instrumental in shaping this thesis.

I am particularly indebted to my advisor and guide, João Vinagre, whose unwavering dedication and weekly guidance have played an indispensable role in the realization of this work. His expertise, patience, and encouragement have been invaluable in navigating the challenges and complexities of this research journey.

I would also like to extend a special thanks to my best friend, whose unwavering belief in me and constant encouragement provided the moral support and motivation to persevere until the completion of my work. Their unwavering support have been a source of strength and inspiration throughout this journey.

Lastly, I am deeply grateful to my partner, who has been by my side throughout this fleeting journey, offering unwavering affection and happiness. Your love and understanding have been my refuge, bringing joy and balance to my life amidst the demands of this academic endeavor.

To all those who have contributed to my growth, both academically and personally, I offer my sincere appreciation. This thesis would not have been possible without your unwavering support and belief in me. Thank you from the depths of my heart.

Contents

List of Tables

List of Figures

Listings

Acronyms

ACC Accuracy

AUC Area under the ROC Curve

CL Centralized Learning

DDoS Denial-of-Service

DL Distributed Learning

DNN Deep Neural Network

FedAvg Federated Averaging

FL Federated Learning

IRSF International Revenue Share Fraud

ML Machine Learning

MLP Multi-layer perceptron

NN Neural Networks

Chapter 1

Introduction

1.1 Motivation

Most of the devices that we use nowadays output a plethora of rich data that describe our behaviours as human beings and the trends of our society. Such data can be used to train ML models that analyze and are resorted to, for example, search for patterns or provide predictions of user's actions. These include for example the text auto-completion algorithms that our smart-phones have incorporated in them or the array of platforms that offer suggested content based on a user's likes and dislikes. Companies have been valuing now more than ever the need of good data to establish what a particular area looks like before you start any improvement, benchmarks, and goals to keep moving forward.

The main result of this project aims to implement and evaluate a DL platform based on results of the ongoing project AIDA (INESC TEC, Mobileum, Universidade de Coimbra, and Carnegie-Mellon University) which will focus on a set of algorithms able to learn local models at the edge, that can be combined in a FL scheme, and with the ability to run under privacy and computational constraints.

To carefully study high speed data distributed traffic while maintaining privacy, we need sets of computer instructions able to build models directly from multiple devices and make such models useful for the whole network. This includes appropriately detecting and protecting against Distributed Denial-of-Service (DDoS) attacks, phone fraud, or pirated/stolen content. For the creation attributed to these models, a study has been performed of the applicability and development of novel methods based on FL with state-of-the-art algorithms for novelty detection, anomaly detection, cost-sensitive learning, learning from imbalanced data, as well as general purpose stream-based algorithms for supervised learning (classification and regression).

1.2 Main goals

Fraud detection in phone networks is a critical challenge that requires innovative approaches for effective detection and prevention. This project aims to develop a federated approach to fraud detection, focusing on the main question of how learning separate fraudster detection models at different points in the network and aggregating those models compare to alternative methods. The objective is to build a system that surpasses the performance of individual models by leveraging the expertise of other participating entities.

Consider a scenario where multiple phone networks or country authorities collaborate to combat international fraud. Each participating entity possesses its own call graph, representing phone numbers as nodes and phone calls as edges. While inter-network calls exist, intra-network calls remain invisible to other networks. In this setup, each entity has the opportunity to develop a fraud detection model using its own network graph.

While completing these goals there a few lines of action that we must accomplish, each translated into its own respective task:

1. Focus on examining International Revenue Share Fraud data that has been provided to us to get a clear view on the foundation that has been set for this thesis.
2. Evaluate and analysis of such data through different kinds of ML techniques.
3. Inclusion of federated ML algorithms and their evaluation through the process of data streams.
4. Ensuring that the software framework with the necessary connections to the application side is rightly incorporated as its where our ML models will be mainly created from.

Additionally, an ensemble environment with a stacking algorithm where we implement different models that have been developed in the thesis shall be taken into question.

1.3 Contributions

The main contributions of this project are:

- A federated ML algorithm able to capture a stream of data and evaluate it, while retraining the privacy of the devices along with their users.
- A prototype that will be able to evaluate distributed algorithms for real-time detection of phone fraud and possibly other use cases.

1.4 Planning & Dissertation layout

The primary objective of this project is to investigate and evaluate the effectiveness of a federated approach to fraud detection in phone networks. Specifically, the focus is on comparing the performance of the following approaches:

Model learned from the complete graph: A centralized model trained using data from all participating entities.

Standalone models: Models developed independently by each participating entity, considering only their own network graphs.

Hybrid model: Model developed with the intent of using the CL with the standalone testing.

Federated model: Separate fraudster detection model learned at different points in the network, with partially disjoint who-calls-whom graphs, and then aggregated to enhance performance.

To achieve the project objective, the following tasks will be undertaken:

- Baseline models implementation and validation:
  - Centralized model: Implementing a fraud detection model using data from the complete graph.
  - Standalone models: Developing separate fraud detection models for each participating entity, based on their own network graphs.
  - Hybrid model: Developing a hybrid fraud detection model with centralized training and standalone testing after having centralized and standalone model completed.
  - Federated model: Developing federated fraud detection model with the intent of reaching comparable results to the previous models.
- Experiments with model aggregation:
  - Ensembles: Exploring ensemble techniques such as stacking and blending to aggregate models from different entities.
- Evaluation and fine-tuning:
  - Comparing the predictive performance of the centralized, standalone, hybrid and federated models.
  - Identifying limitations of the federated approach.
  - Conducting fine-tuning activities, including hyperparameter optimization and code optimization.

By developing a federated approach to fraud detection in phone networks, this project aims to address the challenge of detecting and preventing fraud across multiple entities. By comparing the performance of different approaches, including model aggregation, valuable insights can be gained to enhance fraud detection capabilities and facilitate collaborative efforts in combating international fraud.

Taking into account the planning this thesis is organized in 6 chapters:

1. The first chapter, referring to section 1, contextualizes the understudied project, specifies the project's objectives, identifies the problem addressed in this dissertation, introduces the project development plan, and summarizes the structure of this document.
2. Chapter 2 provides a concise introduction to the various types of ML techniques utilized in this study, along with an overview of Neural Networks (NN) and Fraud Detection.
3. In Chapter 3, we review the current state-of-the-art and description of the software tools that we will be using.
4. The fourth chapter encompasses the significant technical advancements accomplished during the course of this thesis.
5. Chapter 5 presents the results of our study, accompanied by a thorough analysis.
6. Finally, in Chapter 6, we draw conclusions by discussing the main findings and highlighting potential directions for future research.

Chapter 2

Background & State of the Art

In this section, we meticulously explore the foundational building blocks prior to their implementation. This entails a comprehensive analysis of each ML technique to be employed, including potential ensemble aggregation. Furthermore, we delve into the utilization of NNs with edge nodes, which form the basis of our dataset, and culminate with a focus on fraud detection.

2.1 Machine Learning

As humans, our capacity to learn stems from our ability to draw insights from past experiences, utilizing the knowledge and information we accumulate to shape our understanding and guide our decision-making. In contrast, machines traditionally rely on explicit instructions provided by humans. However, the advent of ML has revolutionized this paradigm, aiming to equip machines with the capability to autonomously learn and make decisions, often surpassing human speed and efficiency.

To elucidate this concept, let's consider the scenario of a person watching movies. This individual forms opinions about movies based on a multitude of factors, including genre, duration, and rating. By analyzing a collection of movies that the person has expressed a preference for or distaste towards, we can develop a model that facilitates predictions regarding their opinion of future movies based on these criteria. In essence, we establish a correlation between the person's preferences and the attributes of the movies.

This breakthrough in ML has paved the way for a powerful and transformative field that encompasses a wide range of techniques and methodologies. ML algorithms are designed to process vast amounts of data, automatically identifying patterns, extracting meaningful insights, and making accurate predictions or decisions.

One of the fundamental aspects of ML is the ability to learn from data without being explicitly programmed. Instead of relying on fixed rules or predefined instructions, ML algorithms iteratively analyze data, detect patterns, and adapt their models to improve performance over time. This

iterative learning process, known as training, allows machines to uncover hidden relationships, discover complex patterns, and generalize from the provided data to make predictions or take actions on new, unseen instances.

ML techniques can be broadly categorized into several subfields, each with its own set of algorithms and applications as we can see in figure. These include supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning, and deep learning. Each subfield addresses different types of learning tasks and utilizes distinct algorithms and approaches to tackle specific challenges. We can see some examples of this in figure 2.1

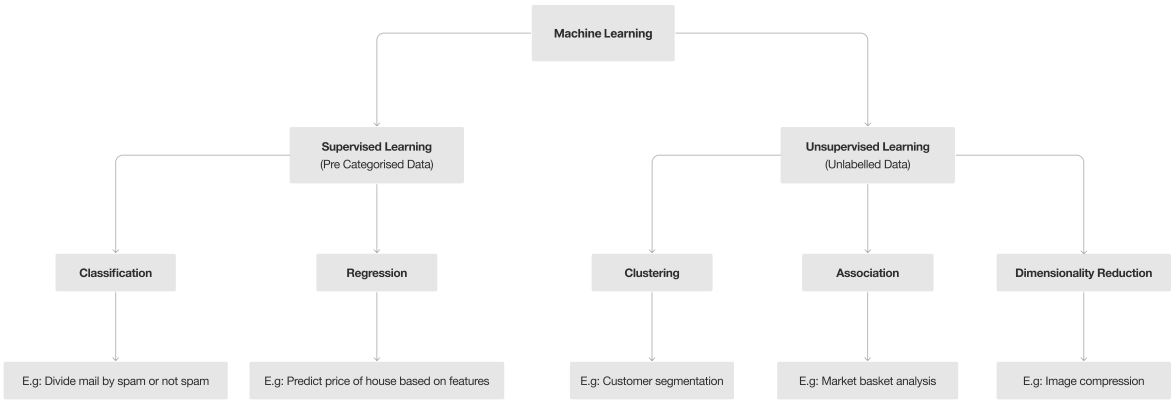


Figure 2.1: Types of machine learning with examples

We will also embark on a comprehensive exploration of various ML paradigms, including centralized, distributed, centralized-distributed, federated and ensemble learning. Each of these approaches offers unique advantages and trade-offs, presenting us with an opportunity to carefully select the most suitable methodology for our specific research objectives.

2.1.1 Centralized Learning

ML approaches have undergone significant evolution to accommodate diverse data settings and computational demands. Among these approaches, CL has emerged as a prominent paradigm, characterized by the centralized collection and processing of data.CL brings forth a range of benefits, including enhanced data availability and accessibility, improved computational efficiency, model consistency, and streamlined governance and decision-making. Nonetheless, it is essential to acknowledge that CL also carries certain limitations and considerations.

In terms of data availability and accessibility, CL offers the advantage of providing access to the complete dataset in one centralized location. This allows for comprehensive analysis and modeling by leveraging the entirety of the data distribution for training models. The centralized

storage infrastructure also facilitates efficient data management, maintenance, and preprocessing, which in turn streamlines the training process. By concentrating the computational burden on a central server or a cluster of high-performance machines, CL enables optimized resource allocation, resulting in faster training times and scalability albeit at a cost that will be mentioned later. [3].

In the context of CL, a significant benefit is that a single model is trained on the entire dataset as seen in figure 2.2, ensuring consistent and synchronized updates. This means that any modifications or improvements made to the model can be easily disseminated across the entire system, guaranteeing consistent behavior and predictions for all clients or users. Furthermore, CL provides a unified governance structure where a central authority is empowered to make decisions regarding crucial aspects such as data handling, model architecture, and updates. This centralized control simplifies the management and coordination of the ML process, enables the implementation of regulatory policies, facilitates quality control measures, and ensures compliance with data protection regulations. Ultimately, this streamlined governance enhances the overall reliability and accountability of the CL system.

Although CL significantly simplifies the training process, it can demand significant computational resources, especially with the expansion of datasets. As the size of the data increases, the computational and storage requirements also grow, potentially resulting in performance bottlenecks and impeding efficient processing. Additionally, the reliance on a centralized server introduces a vulnerability of a single point of failure, whereby system downtime or server malfunctions can cause disruptions to the entire learning process, impacting the availability and reliability of the system.

CL also encounters scalability challenges when confronted with large-scale datasets. As the dataset expands, the computational and storage requirements of the centralized server also increase proportionally, potentially impeding efficient processing and training.

In addition, the centralization of training data in CL can give rise to bias issues. Since the data may not adequately represent the diversity and distribution of information from different sources or user populations, the resulting models might lack generalizability to specific subgroups or fail to capture the intricacies present in decentralized data sources.

Furthermore, privacy and security concerns arise in the context of CL. The concentration of data in a central server raises worries about data privacy, security breaches, and unauthorized access. Concerns regarding privacy risks may cause users to be hesitant in sharing sensitive information, thereby limiting the availability of data for training centralized models.

To address these limitations, alternative approaches such as DL have emerged. DL tackles the scalability challenges of CL by facilitating distributed training on local devices while safeguarding data privacy in some schemes. By retaining data on local devices and aggregating model updates rather than raw data, DL mitigates privacy concerns and encourages collaboration among decentralized data sources.

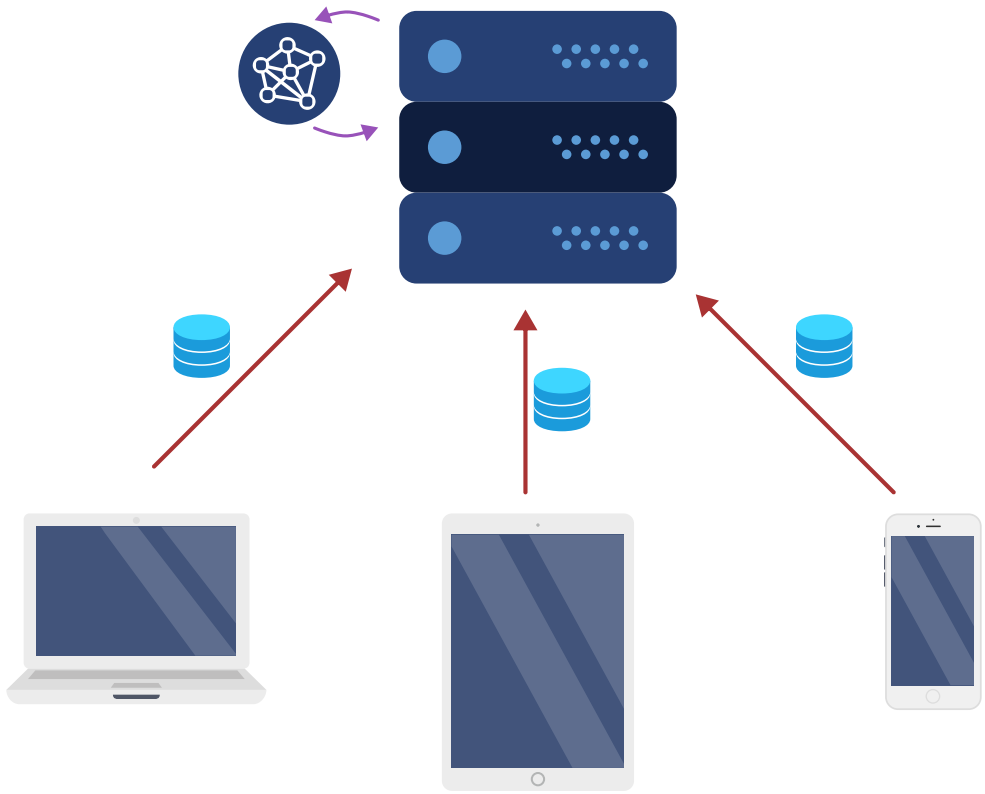


Figure 2.2: Centralized method

2.1.2 Distributed Learning

As data grows in scale and complexity, traditional CL approaches struggle to handle large volumes of data or distributed data sources. In response, DL has emerged as a viable alternative, offering advantages in scalability, privacy preservation, and decentralized decision-making.

DL involves training ML models across multiple geographically distributed machines or nodes. Each node holds a subset of the data and performs local computations. The models' parameters are iteratively updated and exchanged among the nodes, enabling collaborative model training. This approach enhances privacy preservation by keeping data locally at each node as demonstrated in figure 2.3, ensuring sensitive information remains secure. Only model updates are shared during the training process, promoting data privacy.

In DL, data is partitioned and distributed across multiple nodes, enabling parallel processing. Each node independently processes its local data, computes gradients, and updates the model parameters based on local computations. This parallelism, facilitated by a network of machines, ensures efficient utilization of computational resources, leading to faster training times and improved scalability and resource efficiency. Horizontal scaling enables DL to handle large-scale

datasets that surpass the capacity of a single machine. Proof of this can be seen in the paper "Decentralized decision making for networks of uncertain systems", where the results demonstrate that the proposed decentralized method efficiently computes solutions, even in scenarios where a significant number of agents are involved in the network. The effectiveness of the method in approximating the solutions of the centralized formulation is dependent on the structure of the underlying physical coupling network and the communication bounds [12].

To ensure resilience, DL systems are meticulously designed to be fault-tolerant. By distributing data and computations across multiple nodes, the impact of individual node failures on the overall training process is mitigated. For isntance, DL can be implemented with the adaptive search technique called "Nagging" which is intrinsically fault-tolerant [21]. Other fault tolerance mechanisms, such as redundancy and checkpointing, which refers to a mechanism that allows for the periodic saving of intermediate training states during the training process (these checkpoints serve as snapshots or checkpoints of the model's parameters and other relevant information at specific intervals), ensure that training progress is maintained even in the presence of failures.

Decentralized decision-making is another significant advantage of DL. Local nodes have the autonomy to make decisions based on their own models, reducing the reliance on a central authority. This decentralized approach facilitates edge computing, allowing for real-time decision-making at the edge devices. It is particularly valuable in scenarios with limited network connectivity or strict low-latency requirements [35].

Despite its benefits, DL presents challenges that require careful consideration. Synchronizing models and gradients across nodes, managing communication overhead, and coordinating distributed computations demand meticulous design and implementation. Addressing data heterogeneity, managing stragglers, and mitigating communication bottlenecks are critical aspects of DL systems [15, 31, 42].

In conclusion, DL offers a long range of advantages like scalability, privacy preservation, fault tolerance, and decentralized decision-making. It provides an effective solution for handling large-scale datasets, maintaining data privacy, and accommodating decentralized or edge computing environments. However, it comes at a cost of complexity, addressing coordination, communication, and heterogeneity challenges in data and computing resources requires thoughtful system design.

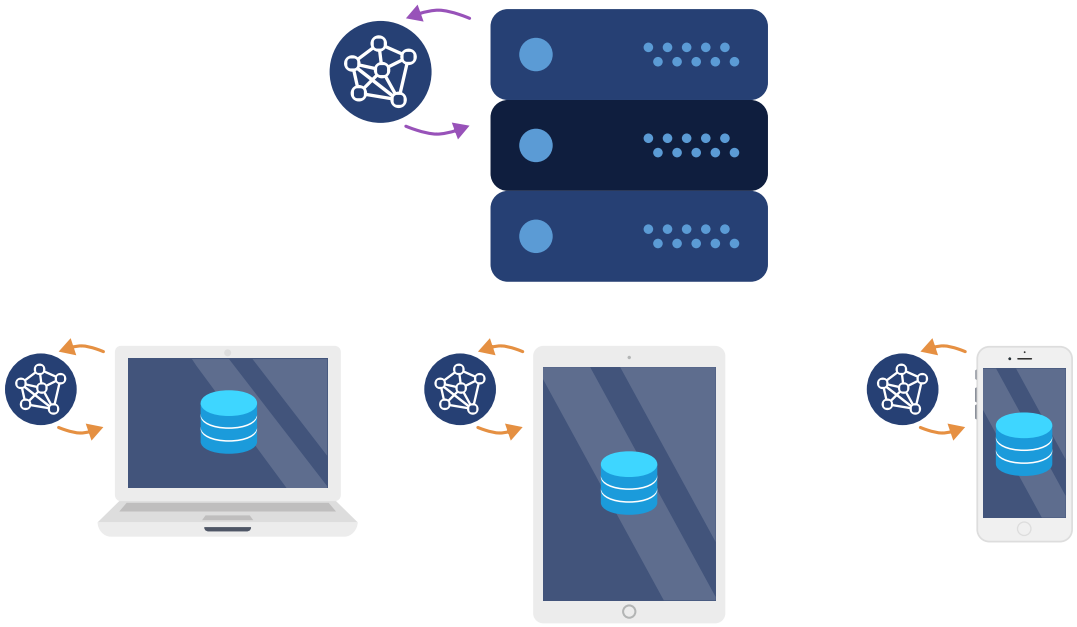


Figure 2.3: Distributed method

2.1.3 Federated Learning

FL is a distributed ML approach, first introduced by McMahan in 2016 [28]. It allows multiple devices to collaboratively train a shared model without sharing their individual data while maintaining high efficiency and low latency [25]. This approach addresses privacy concerns associated with centralized ML, where the data is sent and received by a central server, potentially compromising the privacy of the data owners. In FL, the data remains on the user's device, and only model updates are exchanged with a central server.

In FL, each participant (device or client) prepares their data locally. This data could include user behavior logs, sensor readings, or any other sensitive information. The data owners ensure that the data remains on their devices, preserving privacy [7]. Various other implemented countermeasures like the Krum aggregation rule have also proven to be effective in preventing attacks perpetrated by large coalitions of malicious clients [7]. Preprocessing steps such as feature extraction, data normalization, or encoding may also be performed locally as we will see later on the development of our FL platform.

The FL process begins with the initialization of a global model. This global model can be a pre-trained model or a newly created one specifically tailored for the task at hand. The initial model acts as a starting point for the collaborative training process. Each participant trains the global model using their local data. This training occurs on their respective devices without sharing the raw data with the central server or other participants (represented by the yellow arrows in figure 2.4). Local training can involve various ML techniques, such as gradient

descent, backpropagation, or as it is in our case, a deep learning algorithm with hidden layers. Participants independently update their local models based on their data, extracting valuable insights without compromising data privacy.

Once the local model training is complete, the participants send their model updates to the central server (represented by the red arrows in figure 2.4). The server aggregates these updates, often using an averaging technique, to generate an updated global model (represented by the purple arrows in figure 2.4). Aggregation methods can also consider the heterogeneity of the participants' devices, giving more weight to those with higher performance or more representative data. This collaborative model aggregation ensures that the knowledge gained from each participant's local training is captured and incorporated into the global model.

The aggregated model update is then applied to the global model on the server, refining and improving its performance. This process allows the global model to benefit from the collective knowledge of all participants without accessing their individual data directly. By aggregating model updates rather than raw data, FL ensures data privacy while still achieving model improvement.

The iterative nature of FL involves repeating these steps for multiple rounds or until convergence is reached (represented by the green arrows in figure 2.4 which finishes the cycle). This iterative training enables the global model to learn from diverse data sources and enhance its overall performance. Mathematically, the model aggregation step can be represented as follows:

$$\theta_{t+1} = \frac{1}{K} \sum_{k=1}^{K} \theta_{k,t+1}$$

,

where θ_{t+1} represents the global model,K is the set of selected clients, and $theta_{k,t+1}$ represents the model update from client k.

During the iterative training process, the local model updates θk can be computed using various optimization algorithms such as the ADAM optimizer (algorithm used for the development part of this thesis), which updates the model parameters based on adaptive moment estimation:

$$\theta_{k,t+1} = \theta_t - \eta \frac{m_t}{\sqrt{(v_t) + \varepsilon}}$$

,

where θ_t represents the local model parameters of client k at iteration t,η is the learning rate, m_t and v_t) are the first and second moment estimates respectively, and ε is a small constant for numerical stability [9].

These formulas capture the essence of the FL process, where model updates from multiple clients are aggregated to refine the global model while maintaining data privacy and collaborative learning.

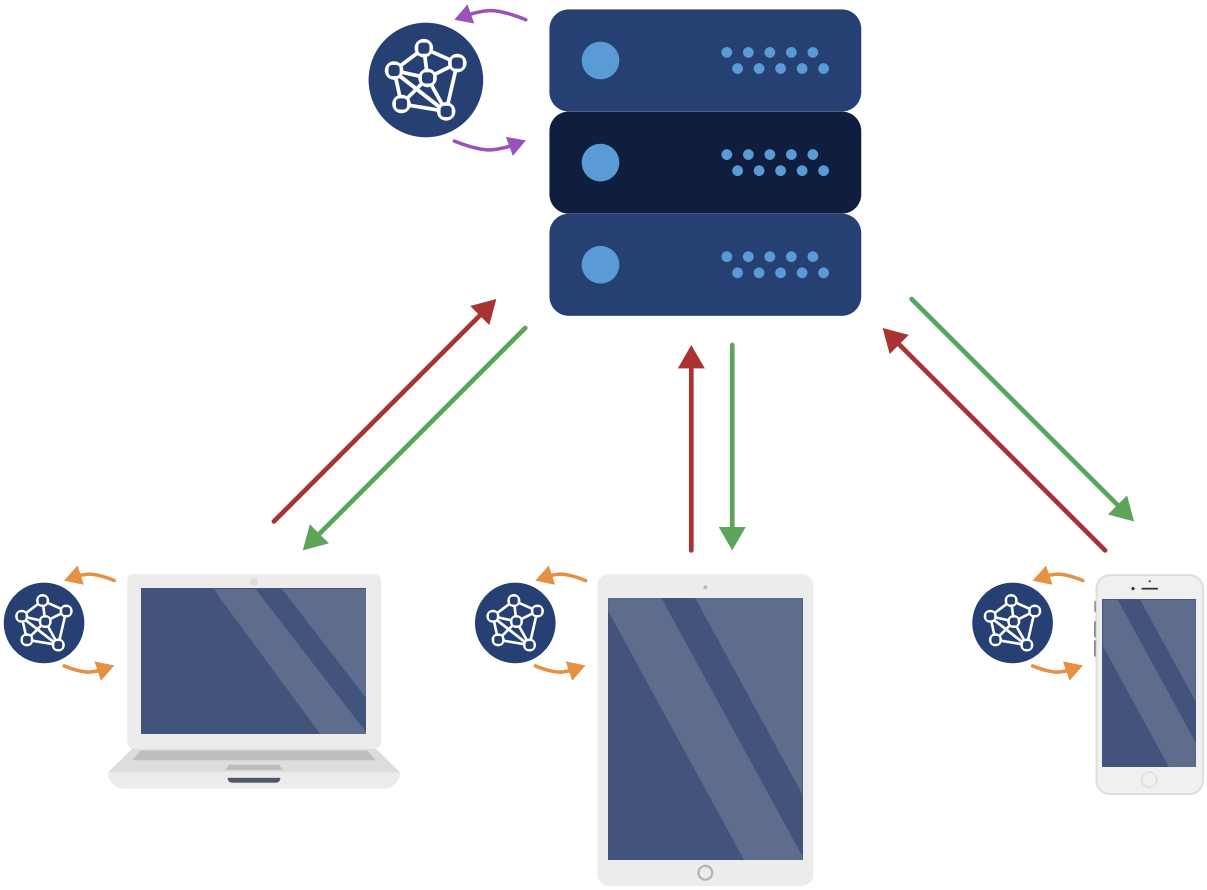


Figure 2.4: Federated method

Key differences from Distributed Learning

For all that its worth FL is very relatable to DL in the way that they both involve multiple entities collaborating to train a ML model. Both their approaches aim to address privacy concerns by keeping the data locally on the participating devices or entities even though distributed platforms only started taking this into account recently [23]. Being this the case we still can address some key differences to differentiate them.

In today's world, the prevalence of personal technology devices for example health monitoring is high [8]. It has given rise to a significant advantage for FL over traditional DL approaches. This is because proliferation of technology brings about a substantial amount of heterogeneity, resulting in unbalanced and non-IID data challenges. By enabling training to occur on decentralized devices while ensuring the data remains local, FL directly tackles concerns pertaining to data privacy and distribution heterogeneity. In contrast, DL often necessitates the implementation of additional mechanisms to effectively address the complexities associated with such data [11].

Another key difference lies in their degree of decentralization. While FL aims for decentral-

ization by diluting the role of a central node, DL typically relies on a central server for data distribution and computation resource allocation. This centralized approach in DL can result in double communication overhead and impose additional requirements on computing power, storage, and bandwidth [47]. In contrast, FL allows each client to be autonomous, with data allocation not governed by a central server [23].

The 3 categories of Federated Learning

Federated Averaging (Horizontal Federated Learning or FedAvg) is the most common and widely used approach in FL. In this group, each client trains a local model using its own data and then sends the model updates to a central server. The central server aggregates the model updates from multiple clients by taking their weighted averages, resulting in an improved global model. This iterative process of model updates and aggregation is repeated until convergence is achieved.

One of the most novel horizontal federated optimization algorithms is the FedFa algorithm, which introduces the concept of a double momentum gradient, which incorporates historical gradient information from both clients and the server. This inclusion enhances the algorithm's convergence by leveraging past trends and updates in the gradients. Additionally, the algorithm employs a weighting strategy that aggregates weights based on training ACC and the number of participants. This approach promotes fairness and ACC in horizontal FL by appropriately weighing contributions based on individual training ACC and the participation count [17].

In **FL with Model Personalization (Vertical Federated Learning)** the focus is on personalizing the global model to individual clients while maintaining privacy. Clients not only send their model updates to the central server but also send additional information about their local data or gradients. The central server then adapts the global model based on this information, allowing the model to be tailored or personalized to specific clients or subsets of clients.

One of the latest methods from Tianyi Chen, Xiao Jin, Yuejiao Sun and Wotao Yin enables individual clients to independently execute stochastic gradient algorithms without the need for coordination with other clients. This approach is particularly well-suited for scenarios where clients experience intermittent connectivity, as it allows them to operate autonomously. Despite the lack of coordination, this method maintains comparable results to centralized and synchronous FL methods [10].

Federated Transfer Learning (FTL) techniques are applied in this group to leverage knowledge from pre-trained models and transfer it to the FL setting. Initially, a global model is trained on a large-scale dataset, typically in a centralized manner. This pre-trained global model is then fine-tuned or adapted using FL, where each client contributes its local data. The knowledge from the pre-trained model helps accelerate the learning process and improve the performance of the federated model.

In recent studies, contrast to existing secure deep learning approaches which suffer from

ACC loss, FTL has been as accurate as nonprivacy-preserving approaches, and is superior to nonfederated self-learning approaches [27].

These three groups provide different approaches within FL, catering to various requirements and scenarios. The choice of the group or variation depends on the specific objectives, data distribution, privacy concerns, and computational resources available in a given FL setup.

2.1.4 Centralized Training with Distributed Testing

In some ML scenarios, it may be desirable to train a model in a centralized manner while performing testing or evaluation in a distributed fashion. This approach combines the benefits of centralized training, where the model is trained on a central server using a complete dataset, with the advantages of distributed testing, where the evaluation process is performed across multiple distributed nodes.

In the centralized training phase the entire dataset is available for training, allowing for in-depth analysis, feature engineering, and model optimization. Centralized training benefits from efficient resource allocation, streamlined data preprocessing, and unified decision-making during model development. During this process, a model can be developed and optimized using advanced techniques such as hyperparameter tuning, cross-validation, and ensemble learning. The central server can leverage its computational resources to explore different model architectures, select optimal hyperparameters, and train the model using sophisticated algorithms.

Centralized training facilitates knowledge consolidation as all available data is utilized to train a single model. The model captures patterns, trends, and insights from the entire dataset, enabling comprehensive learning and generalization. The centralized server acts as a knowledge repository, accumulating insights from the training data and transferring them to the distributed testing phase.

In the distributed testing phase, the trained model is deployed to multiple distributed nodes or devices for evaluation. Each node receives a portion of the testing dataset and performs the evaluation locally. This distributed approach allows for concurrent testing, reduced latency, and efficient utilization of distributed resources. This method also enables scalability and resource efficiency by leveraging the computing power of multiple nodes. Each node independently performs inference on its allocated subset of the testing data, allowing for parallel processing and faster evaluation. This distributed setup is particularly beneficial when dealing with large-scale testing datasets or computationally intensive evaluation tasks.

Centralized training with distributed testing can address privacy and security concerns just like a full distributed environment can. During the testing phase, data remains localized on individual nodes, ensuring that sensitive information is not transferred across the network. And has mentioned before in the distributed setting this approach can be advantageous in scenarios where data privacy regulations or security considerations are paramount. Another advantage that comes from the distributed setting is that in the case of failures or disruptions in specific

nodes, other nodes can continue the evaluation process independently. This decentralized setup reduces the impact of node failures on the overall testing process and ensures that the evaluation can proceed even in the presence of individual node failures.

In summary, centralized training with distributed testing allows for comprehensive model development and optimization in a centralized setting while leveraging the scalability, resource efficiency, privacy preservation, and fault tolerance of distributed testing. This approach is particularly useful in scenarios where centralized model training is feasible but evaluation needs to be performed across distributed nodes or devices.

2.1.5 Ensemble learning

Ensemble learning in the realm of ML involves the art of amalgamating a multitude of distinct models, known as base learners, to conjure predictions that are more precise and resilient. The fundamental concept underlying ensembling lies in the collective wisdom exhibited by a diverse set of models, surpassing the capabilities of any individual model, and yielding enhanced prognostic potency and generalization aptitude.

Each base learner endeavors to apprehend dissimilar facets of the intrinsic patterns governing the data or counterbalance the biases inherent in the learning process. Following the training phase, the predictions of these base learners are consolidated, employing techniques such as voting (like in the example that we give in figure 2.5), averaging, or weighted amalgamation, thereby culminating in the ultimate ensemble prediction.

Ensemble learning confers a multitude of advantages. First and foremost, it efficaciously curtails overfitting by assimilating diverse models endowed with distinct biases and error tendencies [29]. This facilitates the assimilation of a more holistic representation of the underlying data, eventually culminating in improved generalization performance. Additionally, ensembles are generally more stable and impervious to the perturbations inflicted by noise or outliers in the input data, as any missteps committed by individual models are mitigated by the collective consensus of the ensemble. Lastly, ensembles are versatile, well-suited to tackle a diverse array of ML tasks encompassing classification, regression, and anomaly detection, thereby manifesting as a resilient and versatile framework for predictive modeling endeavors.

Overall, ensemble learning harnesses the prowess inherent in the assimilation of multiple models, thereby enhancing prediction ACC, fortitude, and generalization within the ambit of ML endeavors. This renders ensemble learning an invaluable and extensively employed technique within the field.

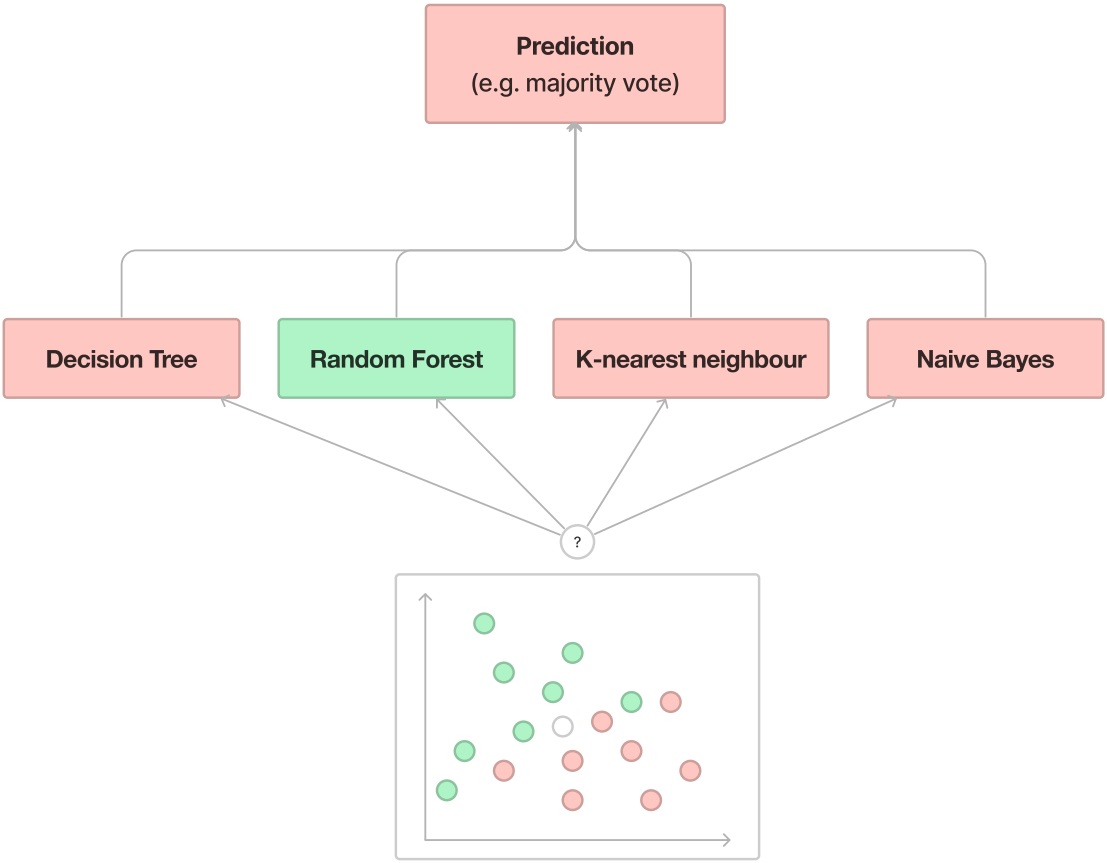


Figure 2.5: Ensemble method example with prediction decided by majority vote

2.1.5.1 Ensemble Diversity

Ensemble diversity refers to the degree of dissimilarity or variability among the individual models within an ensemble. It plays a crucial role in the performance and effectiveness of ensemble learning. By promoting diversity among the ensemble members, it is possible to enhance the ensemble's overall predictive ACC, generalization capability, and robustness. When diverse models are combined, they can compensate for each other's shortcomings, leading to improved overall performance. Diversity helps reduce the risk of overfitting by reducing the correlation among models and increasing the ensemble's ability to capture different aspects of the underlying data distribution.

Ensemble diversity can stem from various sources, for example using different learning algorithms or variations of the same algorithm with different parameters. By leveraging diverse algorithms, the ensemble can capture different decision boundaries or model different aspects of the data distribution [20]. Data diversity on the other hand is achieved by training individual

models on different subsets of the training data. This can be accomplished through techniques such as bagging, where each model is trained on a bootstrap sample of the original dataset. Another approach is cross-validation, where models are trained on different folds of the data. It has been found that ensembles comprising diverse classifiers through the use of stacking demonstrated that, at most, these ensembles perform similarly to the approach of selecting the top classifier from the ensemble using cross-validation [14].

Feature diversity arises from using different subsets of features for training the individual models. By selecting subsets of features randomly or based on some other criterion, the ensemble can focus on different subsets of the input space and capture diverse aspects of the data. Model diversity is related to the differences in the model architecture, parameters, or initializations. It can be achieved by training models with different hyperparameter settings, using different network architectures, or employing techniques like dropout or regularization.

Ensemble techniques, such as bagging, boosting, and stacking, are widely used in ML to improve predictive performance by combining the predictions of multiple models. These techniques can help mitigate overfitting, reduce bias, and increase overall ACC.

Bagging, short for Bootstrap Aggregating, involves training multiple base models independently on different subsets of the training data. The subsets are created by sampling the training data with replacement (bootstrapping). Each model is trained on a different subset, and their predictions are combined through averaging or voting to make the final prediction. A popular example of a bagging ensemble algorithm called random forests that utilizes decision trees as base learners can be seen in the next figure 2.6.

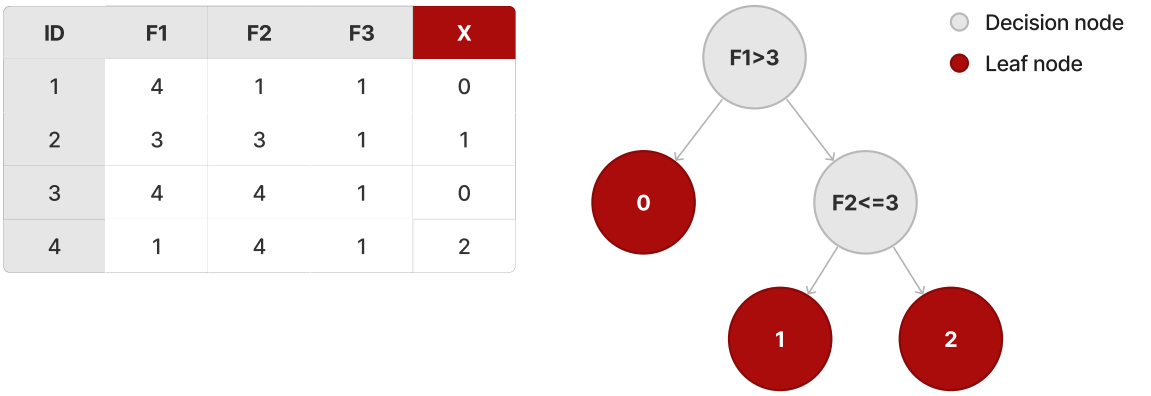


Figure 2.6: Decision tree method example

Boosting, on the other hand, focuses on iteratively improving the ensemble by giving more weight to misclassified instances. The base models are trained sequentially, with each subsequent

model emphasizing the instances that were misclassified by the previous models. This way, boosting focuses on the difficult instances in the data, leading to increased ensemble diversity. AdaBoost (Adaptive Boosting) is a well-known boosting algorithm that combines weak learners into a strong ensemble model/learner, an illustration of this concept is depicted in figure 2.7, which demonstrates the iterative process of identifying weak "star" learners and combining them to form a strong learner. After two rounds of training, the model successfully learns to differentiate between plus signs and stars.

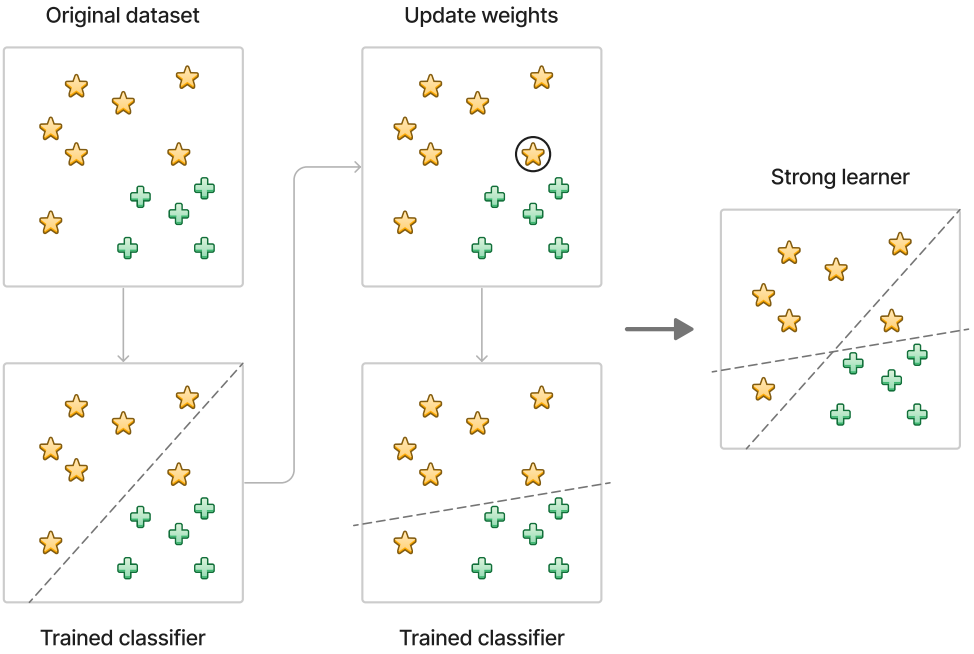


Figure 2.7: AdaBoost method example

Stacking, also known as stacked generalization, involves training multiple models and then training a meta-learner to make predictions based on the outputs of these models. The base models are trained on the training data, and their predictions are then used as input features for the meta-learner. The meta-learner is trained on this augmented dataset, learning to combine the predictions of the base models to make the final prediction. Stacking can be seen as a two-level ensemble, where the base models learn from the original data, and the meta-learner learns from the outputs of the base models.

In summary, ensemble techniques like bagging, boosting, and stacking are powerful tools in ML. They leverage multiple models to improve predictive ACC and can handle various scenarios, including noise in the data. Case in point when there is minimal or no classification noise, randomization performs on par with, and possibly slightly better than, bagging, but falls short

of the ACC achieved by boosting. However, in scenarios with significant classification noise, bagging outperforms boosting and occasionally surpasses randomization as well [13].

By combining the predictions of diverse models, ensembles can capture a broader range of patterns and provide more robust predictions. The choice of a specific technique depends on the characteristics of the problem, available data, and the ensemble's goals.

2.1.5.2 Ensemble Selection

Ensemble selection is a critical phase in the process of ensemble learning, which involves reducing the size of an ensemble by selecting a subset of models to improve efficiency and predictive performance. The objective of ensemble selection is twofold: to minimize computational overhead by reducing the number of models in the ensemble and to enhance predictive performance by eliminating low-performing models while maintaining diversity among the remaining models [44].

One common approach to ensemble selection is based on a greedy search of the space of all possible ensemble subsets. Greedy search methods iteratively add or remove models from the ensemble to improve its performance. These methods employ different strategies for exploring the search space and various measures to evaluate the available actions at each state. Some greedy search-based ensemble selection algorithms utilize the training set for subset evaluation, while others employ a separate validation set [44].

The pruning will be helpful when having a large number of models in an ensemble which can introduce computational overhead, particularly in applications such as stream mining [18]. Minimizing run-time overhead becomes essential in such scenarios. Additionally, ensembles can consist of models with varying predictive performance. Pruning the ensemble by eliminating low-performing models while preserving diversity can significantly enhance overall predictive performance.

2.2 Neural Networks

The historical progression of NNs can be traced back to the era of mid-20th century, during which the exploration of artificial neurons was initiated, endeavoring to mimic the intricate information processing capabilities exhibited by the human brain. In 1943, McCulloch and Pitts presented their seminal work on the McCulloch-Pitts neuron model, which served as a simplified mathematical representation of biological neurons. Subsequently, notable researchers including Rosenblatt (1958) and Widrow-Hoff (1960) made significant contributions that laid the groundwork for the development of perceptrons, the earliest form of NNs endowed with the ability to learn [16].

The fundamental components of NNs revolve around a system of interconnected artificial neurons, referred to as nodes or units, arranged in a layered architecture. The key constituents

encompass the input layer, responsible for receiving and processing the initial data or features that are provided to the network for analysis and prediction. Situated between the input and output layers, the hidden layers conduct intricate transformations on the input data, enabling the NN to progressively acquire abstract representations of the data. Ultimately, the output layer generates the final predictions or outputs, which are predicated on the processed information from the hidden layers. The number of output nodes is contingent upon the problem at hand, such as classification or regression tasks [16].

Integral to the functionality of NNs are the weight parameters and biases associated with each connection linking neurons across different layers. Weights assign significance to the connections, representing their respective strengths. Biases, on the other hand, act as additional parameters that adjust the activation of neurons, imbuing them with the capacity to learn and adapt [16].

To effectively leverage NNs, it is crucial to comprehend a range of key concepts. Activation functions serve as pivotal components that introduce non-linearities to NNs, facilitating the modeling of intricate relationships between inputs and outputs. Common activation functions include sigmoid, tanh, and the one that is used in our case, rectified linear unit (ReLU).

During the process of forward propagation, the input data traverses through the network, layer by layer, with each layer conducting a weighted sum and activation function application to generate outputs. Backpropagation, a crucial concept, involves the propagation of error signals from the output layer back to the preceding layers, facilitating the adjustment of weights and biases to optimize the network's performance.

In summary, NNs have made significant strides since their inception, driven by historical advancements and breakthroughs. These intricate systems, comprising interconnected artificial neurons, employ fundamental components and adhere to key concepts to enable transformative information processing capabilities.

2.2.1 Edge Nodes

In recent years, the integration of edge computing with NNs has emerged as a prominent research area, revolutionizing the landscape of artificial intelligence. Edge nodes, situated at the edge of a network infrastructure, play a pivotal role in this paradigm by bringing computation and decision-making closer to the data source.

By deploying NNs on edge nodes, several benefits can be realized. First and foremost, edge computing reduces the latency associated with transmitting data to a centralized cloud server for processing. This is particularly advantageous in time-sensitive applications where real-time decision-making is crucial. Additionally, edge nodes alleviate the bandwidth requirements and network congestion by processing data locally, minimizing the need for data transfer over the network.

The deployment of NNs on edge nodes also enables privacy and security enhancements. Since

data is processed locally on the edge, sensitive information can remain within the confines of the local network, reducing the risk of unauthorized access and data breaches. Furthermore, edge nodes can facilitate offline operations, ensuring continued functionality even in scenarios where network connectivity is limited or intermittent.

In the context of NNs, edge nodes can be employed for a variety of tasks. For instance, edge nodes can serve as inference engines, performing real-time predictions and classifications based on trained NN models. This is especially beneficial in applications such as autonomous vehicles, where quick decision-making based on sensor data is crucial for ensuring safety. Additionally, edge nodes can be utilized for data preprocessing and feature extraction, reducing the amount of raw data that needs to be transmitted to the cloud for further analysis.

However, deploying NNs on edge nodes also poses certain challenges. Edge nodes typically have limited computational resources and power constraints compared to centralized cloud servers. Therefore, optimization techniques such as model compression and quantization need to be employed to ensure efficient utilization of resources without sacrificing ACC. Additionally, managing and updating NN models on distributed edge nodes requires robust synchronization and version control mechanisms.

In conclusion, the integration of edge nodes with NNs represents an exciting frontier in artificial intelligence. By bringing computation closer to the data source, edge computing empowers real-time decision-making, enhances privacy and security, and enables offline operations. While challenges exist, the potential benefits make edge nodes a promising avenue for deploying NNs in various applications.

2.2.2 Deep Neural Networks (DNN)

DNNs represent a powerful class of NNs that have gained significant attention and revolutionized various fields of artificial intelligence. Unlike traditional NNs with only a few hidden layers, DNNs are characterized by their ability to learn hierarchical representations from data through the use of multiple hidden layers. This depth allows them to capture intricate patterns and relationships in complex datasets, making them particularly effective in tasks involving high-dimensional input data.

They have achieved remarkable success in various domains. In computer vision, deep convolutional neural networks (CNNs) have surpassed human-level performance in tasks like image classification, object detection, and image segmentation. In natural language processing, deep recurrent neural networks (RNNs) and transformer-based architectures have shown exceptional results in machine translation, text generation, and sentiment analysis. DNNs have also made significant advancements in speech recognition, drug discovery, recommender systems, and we are now expecting in fraud detection as well [24] [43].

In addition to the remarkable success of CNNs and RNNs in various domains, multi-layer perceptron (MLP) models have also played a significant role in advancing deep learning and is

what we will be using in the development section.

MLPs, as a type of DNN, have been widely used for tasks such as classification, regression, and pattern recognition. While CNNs and RNNs have gained particular prominence in computer vision and natural language processing, MLPs have been valuable in other domains and applications.

For example MLPs excel in problems where the relationships between input features are complex and non-linear, and there may not be explicit temporal or spatial dependencies to capture. In areas such as fraud detection, they have proven effective in learning patterns and detecting anomalies in large datasets. MLPs can learn complex representations and capture intricate relationships within the data, allowing them to uncover subtle patterns and distinguish between normal and fraudulent behavior [32].

The architecture of a DNN consists of an input layer, one or more hidden layers, and an output layer. Each layer is composed of a collection of interconnected neurons, also known as units. The depth of the network refers to the number of hidden layers it possesses. The neurons within each layer receive inputs from the previous layer, perform a weighted sum of these inputs, apply an activation function, and propagate the results forward to the next layer. This process is known as forward propagation.

One of the key advantages of DNNs is their ability to automatically learn hierarchical representations of the input data. Each hidden layer extracts increasingly abstract and meaningful features from the input, enabling the network to model complex relationships between the input and output. This hierarchical representation learning is crucial in tasks such as image recognition, natural language processing, and speech analysis, where the input data exhibits rich structural dependencies.

Mathematically a general a DNN is defined using two main ingredients. The first ingredient is a set of linear functions represented as

$$\Theta(x) = W_x + b$$

where

$$W = (w_i j) \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m.$$

These linear functions map the input x from

$$\Theta : \mathbb{R}^n \to \mathbb{R}^m.$$

The second ingredient is a nonlinear activation function denoted as

$$\sigma : \mathbb{R} \to \mathbb{R}.$$

By applying this activation function element-wise, we can extend it to operate on vectors. To

construct a general DNN from $\mathbb{R}^d$ to $\mathbb{R}^c$, where d,c, and k are positive integers and $n_1, \cdots, n_k$ are positive integers with $n_0 = d$ and $n_{k+1} = c$, we use a sequence of k hidden layers [40].

Each layer applies a linear function followed by the activation function. The output of layer i is the result of applying Θ_i composed with σ to the output layer $i - 1$[40].

The notation

$$\Theta^k o\ \sigma\ o\ \Theta^{k-1}\ o\ \sigma\ \cdots\ o\ \Theta^1\ o\ \sigma\ o\ \Theta^0(x)$$

is commonly used in computer science literature to represent this computation[40].

The linear functions

$$\Theta_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i+1}}$$

are defined using the linear functions mentioned earlier[40].

In this context, a DNN with $k + 1$ layers is considered to have k hidden layers. The total size of the network is determined by the sum of the dimensions of the intermediate layers, which is $n_1 + \cdots + n_k$.

In this thesis, we primarily focus on a specific activation function that will be used in development called the rectified linear unit (ReLU), defined as

$$ReLU : \mathbb{R} \mapsto \mathbb{R},$$

$$ReLU(x) = max(0, x),\ x \in \mathbb{R}.$$

A ReLU DNN with k hidden layers constitutes the beating heart of our models and they might be written as:

$$f(x) = \Theta^k\ o\ ReLU\ o\ \Theta^{k-1}\ o\ ReLU \cdots\ o\ \Theta^1\ o\ ReLU\ o\ \Theta^0(x).$$

Now that we have every block in place, training DNNs typically involves an iterative process called backpropagation, which adjusts the network's weights and biases to minimize a defined loss or error function. Backpropagation calculates the gradients of the error with respect to the network parameters, and these gradients are used to update the parameters through gradient descent optimization algorithms.

However, training and optimizing DNNs come with their challenges. The vanishing gradient problem, where gradients diminish as they propagate backward through many layers, can hinder the training process. To alleviate this issue, activation functions like ReLU and initialization techniques like Xavier and He initialization are commonly used.

Overfitting, where the network becomes too specialized to the training data and fails to generalize well to new data, is another challenge. Techniques such as regularization, dropout, and, the one that is used in development, early stopping are employed to mitigate overfitting.

In summary, DNNs have revolutionized the field of artificial intelligence by enabling the learning of hierarchical representations from complex data. Despite the challenges associated with

training and optimization, DNNs continue to push the boundaries of AI and hold tremendous potential for solving complex real-world problems.

2.3 Fraud Detection

Although our focus is on fraud detection, it is essential to recognize that preventing fraud in the initial stages is of utmost importance. Mechanisms employed during this phase aim to proactively restrict, suppress, destruct, control, remove, or prevent cyber-attacks from occurring in computer systems, networks, or data. By implementing preventive measures, the need for detecting fraud in the first place is minimized [4].

However, even with robust prevention measures in place, fraud detection techniques continue to hold significant significance as a second line of defence in identifying and preventing fraudulent activities across diverse industries. These techniques serve as a vital component in the overall fraud management strategy, complementing the preventive measures by actively detecting any instances of fraud that may have bypassed the preventive layer.

Anomaly detection techniques aim to identify abnormal or unusual patterns in data that deviate from expected behavior. These methods establish a baseline of normal behavior and flag instances that significantly deviate from it. Statistical analysis, clustering algorithms, and ML models, such as unsupervised learning algorithms, are commonly employed in anomaly detection [33]. By training on historical data, these algorithms can learn patterns and detect anomalies in real-time. They offer scalability, adaptability, and the ability to detect evolving fraud patterns, making them suitable for identifying complex fraud schemes

AI techniques, including deep learning models like NNs, have shown promise in fraud detection. Deep learning models can automatically learn and extract complex features from vast amounts of data, enabling the detection of intricate fraud patterns. These models can analyze structured and unstructured data, such as text or images, to identify fraudulent activities. However, deep learning models often require significant computational resources, extensive training data, and careful parameter tuning to achieve optimal performance [36].

Network analysis techniques focus on detecting fraud by examining the relationships and interactions between entities or nodes within a network. By analyzing the connections, flow of information, and transactional links, network analysis can identify suspicious patterns, such as credit card transactions [45].

In conclusion, fraud detection techniques have evolved significantly, driven by advancements in technology and the need to combat increasingly sophisticated fraudulent activities. By employing a combination of rule-based systems, anomaly detection, ML algorithms, artificial intelligence, network analysis, and hybrid approaches, organizations can enhance their ability to detect and prevent fraudulent behavior, including the intricate schemes associated with IRSF.

International Revenue Share Fraud

International Revenue Share Fraud (IRSF) refers to a sophisticated form of telecommunications fraud where fraudsters exploit revenue sharing agreements between telecom service providers to generate illicit profits. By manipulating premium rate numbers and complex routing schemes, perpetrators exploit the international nature of telecommunications systems for financial gain.

It poses a significant challenge to the telecommunications industry. For example in 2021 the top fraud type reported was of IRSF, that cost telecom $6.69B [2]. IRSF involves the use of premium rate numbers that incur high charges. Fraudsters set up these numbers, often in collaboration with fraudsters, telecom insiders, and corrupt intermediaries.

These organized networks operate across borders, exploiting the complexity of international telecom systems and revenue sharing agreements to evade detection. Collusion enables them to set up fraudulent premium rate numbers, manipulate call routing, and share the illicit revenue by making fraudulent calls or SMS messages. These premium rate numbers may be associated with adult services, psychic hotlines, competitions, or other enticing offers [38].

Fraudsters employ intricate call routing schemes to bypass traditional anti-fraud measures. They exploit vulnerabilities in telecom networks, manipulate signaling protocols, and use international gateway operators to conceal the origin of fraudulent calls. By routing calls through multiple jurisdictions, they make it difficult to trace and track the fraudulent activities [39]. What makes thing worst is that they continuously evolve their schemes to counter detection efforts. Fraudsters adapt their methods to exploit emerging technologies, such as Voice over Internet Protocol (VoIP), virtual SIM cards, or SIM box fraud. They stay ahead of countermeasures by leveraging sophisticated tools, software, and knowledge of telecom systems [38].

However, it has turned into a competition to see who can stay ahead of the curve. On the victims' side, a wide range of advanced techniques has been developed to detect and prevent fraudulent activities. ML, for instance, as mentioned before,has emerged as a powerful approach for detecting IRSF due to its ability to uncover complex patterns and adapt to evolving fraud schemes. Recently, in April of this year, a paper titled "Wangiri Fraud: Pattern Analysis and Machine-Learning-Based Detection" was published, discussing and analyzing the aforementioned approaches. The authors discovered that the random forest and XGBoost algorithms outperformed the Naive Bayes algorithm in identifying all three Wangiri fraud patterns (defined by analyzing call records spanning over a year) based on the selected evaluation metrics. When comparing the performance of the random forest and XGBoost, the random forest exhibited a higher requirement for training and inference time [37].

Feature engineering plays a crucial role in extracting meaningful information from IRSF datasets. This section explores the process of identifying and engineering relevant features that capture the characteristics of fraudulent calls and distinguish them from legitimate ones. Features such as call duration, destination frequency, call volume patterns, geographic indicators, and time-based features are commonly used in IRSF detection models.

In conclusion, IRSF is a sophisticated form of telecommunications fraud that exploits revenue sharing agreements and complex call routing schemes for financial gain. Fraudsters continuously adapt their methods to evade detection, making it challenging for the telecommunications industry to combat IRSF. However, advanced techniques like ML, particularly the random forest and XGBoost algorithms, have shown promise in detecting and preventing IRSF [5, 30]. Feature engineering plays a vital role in distinguishing fraudulent calls from legitimate ones. Efforts to combat IRSF are ongoing as industry professionals strive to stay ahead of evolving fraud schemes.

Chapter 3

Software Tools

3.1 Federated Frameworks

To initiate our project, we require a federated framework, which has gained significant attention and undergone notable advancements in recent years due to its ability to facilitate collaborative ML and data analysis while ensuring data privacy. Although there exists a relevant paper from 2022 that provides a comprehensive analysis of these frameworks [26], in which our decision in choosing will be influenced by, we conducted our own thorough throughput study with more federated frameworks to identify and evaluate the various options available to us. Our evaluation considered factors such as open-source availability and the accompanying descriptions. Based on [41], we have compiled and analyzed the identified frameworks in table 3.1.

An ideal platform for horizontal learning should include native testing, be open source, provide rich optimizers, and have low training time. Considering these factors, FLOWER (Federated Learning Over Wireless Mesh Networks) was chosen as a FL framework specifically designed for resource-constrained environments like wireless mesh networks. FLOWER focuses on optimizing communication and collaboration between edge devices while maintaining data privacy. It utilizes decentralized optimization algorithms and secure aggregation protocols to enable efficient and privacy-preserving FL. Additionally, FLOWER's compatibility with TensorFlow, a well-known library, further solidified the decision to choose it.

Table 3.1: Frameworks for Federated Learning

Name	Language	OpenSource	Description
FATE	Python	Yes	Contains features such as federated sampling, feature scaling, and logistic regression
TensorFlow Federated	Python	Yes	Based on TensorFlow, supporting models like ID3, Swivel, word2vec, and RNN
SyferText	Python	No	Privacy-preserving natural language processing (NLP) using federated learning
Leaf	Python	Yes	Support for models like CNN, Stacked-LSTM, and Logistic Regression
DataSHIELD	R	Yes	Basic statistical computations, Generalized Linear Models, KNN, Data transformations
IBM FL	Python	No	Support for various models and algorithms including neural networks, logistic regression, and ridge regression
FL Pytorch	Python	Yes	PyTorch-based framework with support for models like MLP and CNN
FedML	Python	Yes	Support for various models such as CNN, ResNet18, and RNN
Harmonia	Python	Yes	Designed for radiology and medical imaging tasks, supporting the RSNA dataset
Sherpa.ai	Python	Yes	Supports linear regression, SVM, and TensorFlow deep learning
Flower	Python	Yes	Support for models like MXNet, LSTM, Transformer, and BERT
Paddle	Python	Yes	Based on PaddlePaddle, supporting models like logistic regression, neural networks with MPC, and federated averaging
Federated Android Trainer	Java	Yes	For Android devices, utilizing average gradient aggregation
Photolabeller	Java	Yes	For learning in image labeling tasks, utilizing federated averaging
Fair_flearn	Python	Yes	Agnostic federated learning, designed to work with multiple frameworks and models
dAgora Chain	Java Script	Yes	Based on blockchain technology that provides a distributed infrastructure for federated learning tasks
Federated Xgboost	Python	Yes	Framework specifically for Federated XGBoost, an implementation of the popular gradient boosting algorithm
distcomp	R	Yes	Framework for R, providing support for various statistical computations and models
FedData	R	Yes	Deals with Geospatial Data
ppmHR	R	Yes	Supports weighted Cox models

Key Features of FLOWER

Network Communication: FLOWER handles the communication between the server and the participating devices over wireless networks. It optimizes the communication protocol to account for limited bandwidth, high latency, and intermittent connections. FLOWER ensures efficient synchronization of model updates while minimizing the network overhead.

An example of this can be seen in the paper "Flower: A friendly Federated Learning Framework" believing this platform will be a *game-changer*. They put the framework on five ML workloads with up to 15 million clients and still efficiently made use of available resources in single-machine simulations and ran experiments with millions of clients whilst sampling thousands in each training [6].

Heterogeneous Device Support: FLOWER supports heterogeneous devices with varying computational capabilities and network conditions. It allows devices with different hardware configurations, such as smartphones, IoT devices, and edge servers, to participate in the FL process. FLOWER incorporates mechanisms to handle device heterogeneity, including adaptive learning rates and model compression techniques.

A cutting-edge advancement of this, specifically within the FL framework Flower, is exemplified by Protea. Protea serves as a lightweight and adaptable client profiling component, enabling the automatic collection of system-level statistics and resource estimation for each client. This resource-aware approach facilitates simulations, optimizing performance within federated systems [46].

The outcomes of their research demonstrate the successful impact of Protea's design. The implementation leads to a notable improvement in parallelism, resulting in a 1.66 times faster wall-clock time and a significant 2.6 times enhancement in GPU utilization. These advancements enable the execution of large-scale experiments on heterogeneous clients, opening new possibilities in FL research and practice [46].

Fault-Tolerance and Robustness: FLOWER addresses the challenges of device failures and dropouts during the training process. It implements fault-tolerant mechanisms to handle device disconnections, device failures, and data transmission errors. FLOWER ensures that the FL system remains resilient and continues training effectively, even in the presence of unreliable network conditions.

Just in 2021, a significant breakthrough in was achieved through the introduction of Salvia. Salvia is an implementation of Secure Aggregation (used to ensure that the server is unable to inspect individual trained models as it aggregates them) specifically designed for Python users within the Flower FL framework. Building upon the SecAgg(+) protocols, which are tailored for a semi-honest threat model, Salvia ensures robustness against client dropouts [22].

One notable advantage of Salvia is its flexible and user-friendly API, which seamlessly integrates with diverse ML frameworks. This compatibility empowers Python users to effortlessly

incorporate Salvia into their existing workflows, leveraging its advanced privacy and security features within the Flower FL framework [22].

Privacy and Security: FLOWER prioritizes privacy and data security in FL. It employs techniques like differential privacy, secure aggregation as mentioned before, and encryption to protect sensitive user data during the training process. FLOWER ensures that individual device data remains private and secure, promoting user trust and compliance with privacy regulations.

Model Evaluation and Selection: FLOWER provides mechanisms for model evaluation and selection in FL. It allows the server to aggregate model updates from different devices, perform model evaluation on validation data, and select the most suitable model for deployment. FLOWER incorporates techniques for model quality assessment and selection based on performance metrics and predefined criteria.

Wireless Network Adaptability: FLOWER's focus on FL over wireless networks makes it well-suited for scenarios where devices communicate over Wi-Fi or cellular networks. It optimizes the communication protocol and network utilization, ensuring efficient training even in challenging wireless network conditions.

In 2022, a comprehensive study was conducted to analyze the performance of FLOWER in wireless networks. The results indicated that the ACC of the FL system within a simulated heterogeneous network environment was notably dependable and comparable to the centralized approach. Additionally, the study revealed that the amount of data exchanged between the clients and the server using the FL system was significantly reduced compared to traditional ML methods. This reduction in data transfer is particularly advantageous for wireless networks, as it addresses concerns related to limited resources and privacy [19].

3.2 TenserFlow

TensorFlow is a widely adopted open-source framework that excels in building and training DNNs. It provides a comprehensive ecosystem of tools and libraries that facilitate the development and deployment of ML models. TensorFlow's versatility and scalability make it an ideal choice for our case that includes deep learning and FL.

TensorFlow offers extensive functionality for constructing, training, and evaluating DNNs. It provides a high-level API, TensorFlow Keras, which simplifies the process of building and training DNN models. Keras offers an intuitive interface for defining model architectures, handling optimization algorithms, and managing the training process [34].

When it comes to FL, TensorFlow can be seamlessly integrated with the FLOWER framework to enable distributed training on heterogeneous devices. FLOWER, as a FL framework, builds upon TensorFlow's capabilities to simplify the implementation of FL systems. By leveraging TensorFlow within the FLOWER framework, we can harness the power of DNNs in our FL tasks.

The Sequential model for example from TensorFlow's Keras API allows us to construct a sequential stack of layers in a linear manner like the one we will be working on figure 4.3. By utilizing the tensorflow.keras.models.Sequential import, we can define the architecture of our MLP model and configure its layers, activation functions, and optimization algorithms. This integration of TensorFlow's DNN capabilities with the FLOWER framework empowers us to train complex models, such as MLPs, in a FL setting.

3.3 Docker

Docker has emerged as a cutting-edge platform, revolutionizing application packaging, deployment, and management. With a plethora of innovative features and capabilities, Docker has become the preferred choice for developers and organizations alike.Docker enables the packaging of applications and their dependencies, ensuring seamless portability across diverse environments. Docker containers transcend platform-specific constraints, facilitating effortless deployment on various operating systems and cloud providers, they are also renowned for their lightweight nature, imposing minimal overhead.

Leveraging the host operating system's kernel allows Docker to achieve swift startup times and optimize resource consumption. This agility makes Docker an excellent choice for rapidly scaling applications.

The built-in support for container orchestration frameworks, such as Kubernetes and Docker Swarm, empowers organizations to effortlessly scale and manage containerized applications. These frameworks equip developers with powerful tools to deploy and scale applications across multiple hosts or clusters, streamlining operations.

Docker streamlines the creation and management of container images through Dockerfiles. These files empower developers to define the application's environment and dependencies, ensuring reproducibility and version control. Additionally, Docker offers a range of image registries, including the widely-used Docker Hub, facilitating efficient image storage and distribution.

Why Docker for Federated System

Docker's versatility lies in its seamless integration with federated code, enabling enhanced collaboration and facilitating distributed computing. Let's explore how Docker contributes to the integration and deployment of federated code[1]:

- Docker containers provide a reliable runtime environment that ensures isolation between federated entities. By encapsulating federated code within separate containers, Docker restricts access to system resources, maintaining data security and privacy.
- Docker eliminates inconsistencies in the execution environment across federated entities.

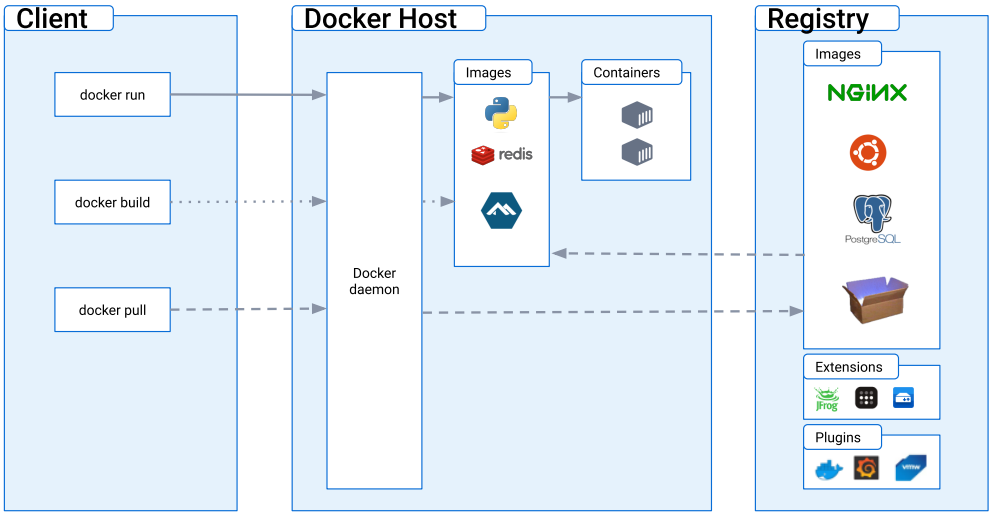


Figure 3.1: Docker framework

By encapsulating dependencies and libraries within Docker containers, the federated code enjoys a consistent runtime environment, regardless of the participating devices.

- Docker's container orchestration frameworks, such as Kubernetes, efficiently manage federated computing clusters. These frameworks excel at scheduling and deploying containers across distributed devices, managing load balancing, and automating scaling to meet computational demands.
- Docker simplifies the deployment of federated code by providing a standardized containerization format. This format enables easy deployment on various devices, regardless of their underlying operating systems or hardware architectures. Docker's capability to clone and replicate containers further simplifies the distribution of federated code among multiple participants.

In summary, Docker's state of the art features, including improved application portability, lightweight deployment, scalable container orchestration, and robust isolation, make it an ideal platform for integrating and deploying federated code. Docker ensures consistent execution environments and secure isolation, enabling efficient collaboration and seamless execution of federated computing scenarios.

Chapter 4

Development

This section describes the methodology employed to develop and evaluate the FL framework for the given dataset using Python. It outlines the steps taken to achieve the objectives of the research, including the design, implementation, and evaluation of the framework.

4.1 Data

The data that we will be working with is of decent size, containing 78,125 entries, each representing a different call that was made and recorded. These calls encompass a wide range of telecommunications events, including mobile originated calls (MOC), SMS mobile originated (SMSMO), mobile terminated calls (MTC), and SMS mobile terminated (SMSMT). The dataset provides various information for each call, such as the duration of the call, event type, dates, and additional relevant features. A more detailed overview can be seen in table 4.1

Most importantly in the dataset used in this work, it contains a target column "APN" that classifies phone calls as either "normal" or "fraud" (IRSF). In this study, the "APN" attribute is modeled as a binary classification problem in our supervised learning. To have a better understanding of our target column, Figure 4 depicts the distribution of the two classes in "APN" within the complete dataset and, subsequently, for the edge nodes when operating in a distributed or federated setting.

Across all phone calls, the majority were classified as "normal," with only a small percentage categorized as "fraud." In fact, the abundance of "normal" calls was observed in most batches, with the "fraud" calls accounting for only around 15% to 18% of the total records. This percentage translates to approximately 11,718 to 14,062 calls, representing the instances of fraud within each batch.

To further deepen our analysis, we will utilize the Pandas library in Python to read the CSV file that contains the raw data. Pandas offers powerful data manipulation capabilities and provides a convenient way to work with structured data. By utilizing Pandas data frames, we

can easily explore, manipulate, and preprocess the dataset in preparation for the FL framework.

The Pandas library will be instrumental in performing data preprocessing operations (which will be brought into further detail in the next chapter), such as handling missing values, data cleaning, and transforming categorical variables into numerical representations. By utilizing Pandas data frames, we can efficiently navigate and manipulate the dataset, ensuring it is in a suitable format for subsequent analysis and model development.

To further enhance the functionality of the code, several key steps were implemented. The preprocessed data was divided into a training set and a testing set using a function from the scikit-learn library. This splitting process ensured that the testing set comprised 25% of the data, allowing for an effective evaluation of the trained model's performance.

The training process utilized a batch size of 64, which facilitated efficient computation and parameter updates. The early stopping callback was employed during training to monitor the loss and halt the process if necessary, thereby preventing unnecessary iterations and saving computational resources.

Field	Description	Values
EVENT_TYPE	Type of event	MOC/MTC/SMSMO/SMSMT
SERVED_ENTITY_ID	A Number	Integer
IMSI	International Mobile Subscriber Identity	Integer
IMEI	International Mobile Equipment Identity	Integer
A_NUMBER	The calling number	Integer
B_NUMBER	The called number	Integer
C_NUMBER	Used in redirected calls	Empty
START_DATE	Event start date	String
UTC_TIME_OFFSET	The UTC offset	String
DURATION	Event duration	Float
CALL_TYPE	Call type	0/1/2
CHARGE_AMOUNT	Charged amount	Float
CAUSE_FOR_TERMINATION	Termination causes	Integer
GGSN_ADDRESS	IP address assigned to the GGSN	Empty
SGSN_ADDRESS	IP address assigned to the SGSN	Empty
UP_VOLUME	Data usage	Empty
DOWN_VOLUME	Data usage	Empty
APN	Type of fraud	String/Empty
CELL_ID	Cell Identifier	09980-09999
ROAMING_INDICATOR	tbg	Integer
VISITED_COUNTRY_CODE	Visited country code	Integer
LOCATION_AREA_CODE	Area code	Integer
IN_TRUNK_GROUP	Incoming trunk group	String
IN_TRUNK_NUMBER	Incoming trunk number	String
OUT_TRUNK_GROUP	Outgoing trunk group	String
OUT_TRUNK_NUMBER	Outgoing trunk number	String
SERVED_PDP_ADDRESS	IP address assigned to the PDP	Empty
MOBILE_SESSION_SERVICE	Mobile session	Integer
MESSAGING_EVENT_SERVICE	Messaging event	Integer
CALL_FORWARDING	Call fowarding	Integer
CALL_CONFERENCE	Call conference	Integer
CALL_HOLD	Call hold	Integer
VOICEMAIL	Voicemail	Integer
VOICEMAIL_CALLBACK	Voicemail callback	Integer
EMERGENCY_CALL	Emergency call	Integer

Table 4.1: Data columns of fraud detection dataset

Data Preprocessing

In order to streamline the analysis and focus on the most relevant features, a preprocessing step was performed by identifying specific columns for removal from the Data Frame. The purpose of dropping these columns was to eliminate any information that was deemed irrelevant or redundant for the analysis.

By removing these columns from the dataset, the preprocessing step helped to simplify and optimize the subsequent modeling process. It reduced the dimensionality of the data, eliminating

noise and potential confounding factors that may hinder the model's performance. This selection process ensured that the remaining columns in the dataset contained the most meaningful and informative attributes for the analysis of the targeted fraud type. In this manner a python list of these chosen dropped columns can be seen in listing 4.1.

```
cols_to_drop = ["C_NUMBER", "UTC_TIME_OFFSET", "GGSN_ADDRESS",
    "SGSN_ADDRESS", "UP_VOLUME", "DOWN_VOLUME", "LOCATION_AREA_CODE",
    "IN_TRUNK_GROUP", "IN_TRUNK_NUMBER", "OUT_TRUNK_GROUP",
    "OUT_TRUNK_NUMBER", "SERVED_PDP_ADDRESS",
    "MOBILE_SESSION_SERVICE","MESSAGING_EVENT_SERVICE", "CALL_FORWARDING",
    "CALL_CONFERENCE", "CALL_HOLD","VOICEMAIL", "VOICEMAIL_CALLBACK",
    "EMERGENCY_CALL"]
```

Listing 4.1: Columns dropped from data frame

Now that the irrelevant columns haven been pruned we need to make it so that every kind of data is readable and can be analysed. Therefore in order to utilize categorical information effectively in subsequent ML models, another preprocessing step related to the "EVENT_TYPE" column was conducted. The encoding process involved converting the textual categories in the "EVENT_TYPE" column into corresponding numerical values.

In our case as can be seen in listing 4.2, categories such as "MOC" (Mobile Originated Call), "SMSMO" (SMS Mobile Originated), "MTC" (Mobile Terminated Call), and "SMSMT" (SMS Mobile Terminated) were replaced with numeric codes (1, 2, 3, and 4, respectively). By performing this encoding, the categorical values were transformed into a numerical representation, which is more suitable for ML algorithms. This allows the "EVENT_TYPE" column to be treated as a feature in subsequent modeling processes.

```
df["EVENT_TYPE"] = df["EVENT_TYPE"].replace("MOC", 1)
df["EVENT_TYPE"] = df["EVENT_TYPE"].replace("SMSMO", 2)
df["EVENT_TYPE"] = df["EVENT_TYPE"].replace("MTC", 3)
df["EVENT_TYPE"] = df["EVENT_TYPE"].replace("SMSMT", 4)
```

Listing 4.2: Encoding of "EVENT_TYPE" column

Further processing is applied to the target column "APN", which is the target column and the one that tells us if that call was a fraud or not. Missing values in this column signifies that the call was not a fraud and are filled with 0 to ensure no data points are left empty. Additionally, a binary counterpart representation is applied, where non-missing values are set to 1, while the remaining 0 values are retained. This transformation converts the column into a numeric format and just like before allows to be treated as a feature.

Any remaining missing values or empty string values ("") in the Data Frame are filled with 0, ensuring a complete and consistent dataset for analysis. This step prevents missing values from causing issues in subsequent computations or modeling steps.

With these steps the initial Data Frame is transformed into a cleaned and standardized format, ready for further analysis or ML tasks, a cleaner table after the preprocessing step can be seen in table 4.2.

Field	Description	Values
EVENT_TYPE	Type of event	1/2/3/4
SERVED_ENTITY_ID	A Number	Integer
IMSI	International Mobile Subscriber Identity	Integer
IMEI	International Mobile Equipment Identity	Integer
A_NUMBER	The calling number	Integer
B_NUMBER	The called number	Integer
START_DATE	Event start date	String
DURATION	Event duration	Float
CALL_TYPE	Call type	0/1/2
CHARGE_AMOUNT	Charged amount	Float
CAUSE_FOR_TERMINATION	Termination causes	Integer
APN	Type of fraud	1/0
CELL_ID	Cell Identifier	09980-09999
ROAMING_INDICATOR	tbg	Integer
VISITED_COUNTRY_CODE	Visited country code	Integer

Table 4.2: Data columns of fraud detection dataset after preprocessing state

4.2 Model

A sequential model was created using the Sequential API provided by Keras. This allowed for the systematic addition of layers to the model and the design of a neural network architecture. As demonstrated in table 4.3 various dense layers were added to the model. These layers employed different numbers of units and utilized the rectified linear unit (ReLU) activation function. The ReLU activation function introduces non-linearity into the model, enabling it to capture complex patterns and relationships in the data.

As part of using the TensorFlow library, an early stopping callback was implemented. This callback actively monitored the loss during the training process and halted the training if the loss did not improve over a specified number of epochs (in this case, 3). The early stopping mechanism aimed to prevent overfitting and improve the efficiency of model training.

The model was compiled by specifying the loss function, optimizer, and metrics to be utilized during both training and evaluation. The chosen loss function was binary cross-entropy, which is commonly used for binary classification tasks. The Adam optimizer was employed to optimize the model's performance. Various metrics such as AUC, ACC, precision, recall and F1 score,

were selected to assess the model's performance from different perspectives

Layer	Input	Output
dense_input	[(None,14)]	[(None,14)]

→

Layer	Input	Output
dense_0	[(None,14)]	[(None,14)]

↓

Layer	Input	Output
dense_1	[(None,14)]	[(None,32)]

←

Layer	Input	Output
dense_2	[(None,32)]	[(None,64)]

↓

Layer	Input	Output
dense_3	[(None,64)]	[(None,14)]

→

Layer	Input	Output
dense_4	[(None,14)]	[(None,1)]

Table 4.3: Multilayer Perceptron scheme

4.3 Metrics

In the context of ML evaluation, several performance metrics are used to assess the effectiveness and quality of classification models. These metrics provide insights into different aspects of the model's performance, including its ability to discriminate between classes, overall ACC, precision, recall, and the balance between precision and recall. In this subsection, we will discuss the significance and interpretation of the following key performance metrics that will be used in this thesis results: Area Under the ROC Curve (AUC), Accuracy (ACC), Precision, Recall, and F1-Score.

The AUC is a widely used metric that measures the ability of a classifier to discriminate between positive and negative instances. It represents the probability that a randomly selected positive instance will be ranked higher than a randomly selected negative instance. A higher AUC value, closer to 1, indicates superior classifier performance in terms of correctly ranking positive and negative instances.

ACC is a fundamental metric that measures the overall correctness of a classifier's predictions. It calculates the ratio of correctly classified instances to the total number of instances in the dataset. While ACC is commonly used, it may not be suitable for imbalanced datasets where the class distribution is skewed, as it can be influenced by the majority class.

Precision quantifies the proportion of correctly predicted positive instances out of all instances predicted as positive. It focuses on the ACC of positive predictions, indicating how precise the classifier is when labeling instances as positive. Precision is particularly important in scenarios

where false positives can have significant consequences, such as medical diagnostics or fraud detection.

Recall measures the proportion of correctly predicted positive instances out of all actual positive instances. It captures the classifier's ability to identify positive instances, indicating how sensitive the classifier is to detecting the positive class. Recall is crucial in situations where false negatives (missed positive instances) are critical and need to be minimized, such as disease diagnosis or anomaly detection.

The F1-Score combines precision and recall into a single metric, providing a balanced measure of a classifier's performance. It is the harmonic mean of precision and recall, considering both metrics simultaneously. The F1-Score gives equal weight to precision and recall, and a higher value indicates a better balance between the two, reflecting a more robust classifier.

These performance metrics collectively offer a comprehensive evaluation of a classifier's performance, taking into account different aspects such as discrimination ability, overall ACC, precision, recall, and the balance between them. When interpreting these metrics, it is important to consider the specific requirements, characteristics of the problem domain, and the trade-offs associated with precision and recall. Selecting the most appropriate metrics depends on the nature of the problem and the desired performance goals of the ML system.

4.4 Centralized & Distributed Approach

After creating the model, preparing the dataset and choosing the metrics to evaluate each model, the implementation of the centralized approach is relatively straightforward. By directly inputting the entire dataset into the model, we can consider it as a centralized approach.

On the other hand, for the distributed approach, we aim to distribute the data in a meaningful way. To achieve this, we have identified the "cell_id" column, which signifies specific locations within the phone operator's coverage zone. Leveraging this column, we can strategically distribute the data based on the corresponding cell IDs. By doing so, we can ensure that data from different locations is distributed among different clients in the FL setup.

This approach allows us to incorporate geographical information and utilize the unique characteristics of each location in the training process. By distributing the data based on cell IDs, we can enable the clients to learn location-specific patterns and improve their performance in handling events related to different areas within the phone operator's network.

Locations

To divide the dataset into different locations, we can leverage the range of cell IDs, which spans from 09980 to 09999. We can simulate a city or county by dividing the dataset into four

different locations. For instance, considering the range from 09980 to 09985, we can define this as one city zone. Continuing this logic, we can allocate the remaining zones to different locations.

Using this approach, we assign the following ranges to each location:

- Location 1: Zones 09980 to 09984
- Location 2: Zones 09985 to 09989
- Location 3: Zones 09990 to 09994
- Location 4: Zones 09995 to 09999

By distributing the data in this manner, we create a representation of different locations within the simulated city. This division allows the FL process to capture location-specific patterns and variations in the data.

Analyzing the distribution of the target variable across the locations, as shown in figure 4.1, we observe some imbalance in regards to the target value "APN". However, it is important to note that the original dataset itself was unbalanced. Thus, the observed imbalance in the distributed dataset was expected. The crucial aspect was to avoid further exacerbating the imbalance during the distribution process, as this could lead to future issues. Overall, we believe the data distribution was successful in maintaining the initial balance and ensuring the representation of different locations in the FL setup.

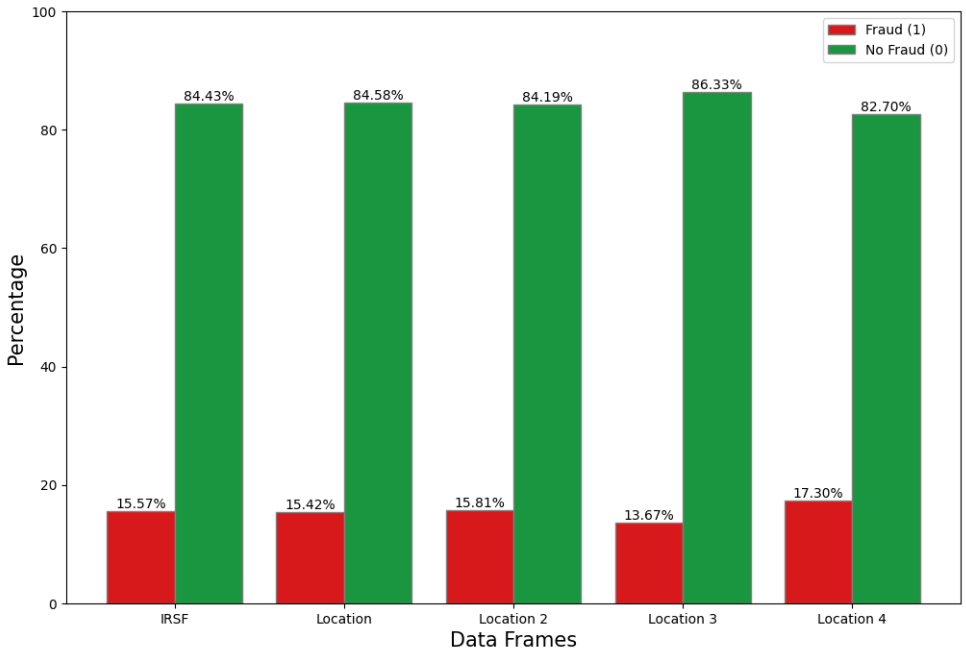


Figure 4.1: Distributed visualization of the edge nodes for the target value "APN"

4.5 Centralized Training with Distributed Testing

After completing the DL approach, we can now proceed with a hybrid approach that combines centralized training and distributed testing. In the previous steps, we performed a 25% train-test split for each location in the FL setup. However, in this hybrid approach, we will aggregate all the training data from different locations while preserving the respective test cases for each location.

To implement this, we append the previously split training data from all locations into a single centralized training dataset. This centralized training dataset consists of data from all locations, providing a more diverse and comprehensive representation of the overall dataset. We then utilize this centralized training dataset to train the model.

Once the model is trained using the centralized training data, we can employ it to predict the test cases that were originally split for each location. This approach leverages the combined knowledge and patterns learned from the diverse data sources across different locations, leading to potentially improved prediction performance.

Figure 4.2 provides a visual representation of the process, illustrating the flow from centralized training using aggregated data to distributed testing on location-specific test cases.

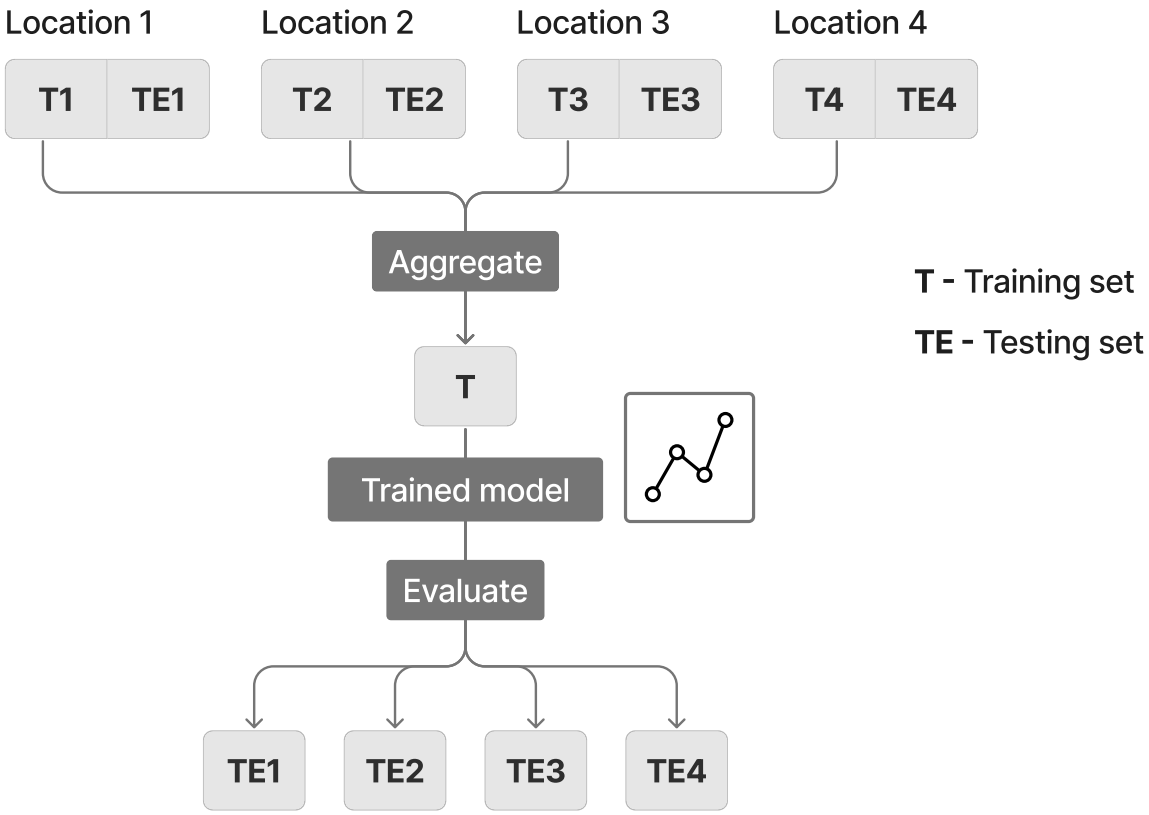


Figure 4.2: Steps taken for our Centralized Training with Distributed Testing environment

The hybrid approach enhances the model's ability to capture a broader range of patterns and variations present in the overall dataset, resulting in potentially more accurate predictions.

4.6 Federated

In our example, we have a FL setup involving four clients: client1.py, client2.py, client3.py, and client4.py. Each client possesses its own dataset, namely location1.csv, location2.csv, location3.csv, and location4.csv. The local models on these clients are initialized with identical architectures and hyperparameters, ensuring consistency and enabling fair comparisons across the clients. This setup allows for distributed training, where each client independently trains its model using its respective dataset.

4.6.1 Server & Client execution

Client side:

- The code begins by reading the data from the "location1.csv" file using pandas and applies the preprocessing steps defined earlier. It splits the dataset into training and testing sets using the *train_test_split* function from scikit-learn. Usually a test size of 25% will be chosen.
- The model architecture is defined using the Sequential API from TensorFlow and compiled with the desired loss function, optimizer, and evaluation metrics as showed in table 4.3.
- The *CifarClient* class is defined as a subclass of *fl.client.NumPyClient*. This class represents a client participating in the FL process. It includes methods for retrieving and updating model parameters, fitting the model locally on the client's dataset, and evaluating the model's performance. Running the FL process the *fl.client.start_numpy_client* function is called to start the FL. It connects the client to the server specified by the server address and utilizes the *CifarClient* class to handle communication and interactions with the server.

Server side:

- Starting the Flower server the *fl.server.start_server* function is called to start the Flower server.
- The address and port on which the server will listen for client connections will be estrablished.
- A config setup for the server is established with it being a *fl.server.ServerConfig* object.
- The strategy object to be used for training and aggregation on the server. In this case, the FedAvg strategy object created earlier is passed. Inside it we have parameters like the *min_available_clients* parameter which is set to 4.

By running this code, it initiates a Flower server that awaits client connections to engage in the FL process. On the server side, comprehensive reports detailing the data received and any failures encountered by each client are generated. On the client side, a reasonably detailed account of the training progress is recorded. This cycle persists until the specified number of training rounds, as defined in the config parameter, are completed.

4.6.2 Docker Implementation

In order to facilitate the deployment and execution of the FL setting, a Docker-based solution has been implemented. The setup consists of multiple containers, including a server container and client containers. Each client container represents a participant in the FL process.

The server container is responsible for coordinating the FL process using the Flower framework. The Dockerfile for the server container is as follows:

```
FROM python:3.9

ADD server.py .

RUN pip install --upgrade pip \
 && pip install flwr

COPY entrypoint.sh .
RUN chmod +x entrypoint.sh
ENTRYPOINT ["./entrypoint.sh"]
```

Listing 4.3: Dockerfile for Python server environment

The Dockerfile starts with a base Python 3.9 image. It copies the server.py file, which contains the code for the server using the Flower framework. The necessary dependencies, including the flwr package, are installed using pip. An entrypoint.sh script is also copied to execute the python file.

The client containers represent individual participants in the FL process. Each client container has its own client.py file, which contains the other part of the FLOWER code for the client's interaction with the server. Additionally, each client container has access to its own dataset in the form of a location.csv file.

The Dockerfile for a client container is as follows:

```
FROM python:3.9

ADD client1.py .
ADD location1.csv .

RUN pip install --upgrade pip
&& pip install flwr numpy tensorflow scikit-learn pandas

COPY entrypoint.sh .
RUN chmod +x entrypoint.sh
ENTRYPOINT ["./entrypoint.sh"]
```

Listing 4.4: Dockerfile for Python client environment

Similar to the server container, the client container starts with a base Python 3.9 image. It copies the client1.py file, which contains the code for the client's interaction with the server, as well as the location1.csv dataset file. The necessary dependencies for ML model building, including the flwr package and other libraries such as numpy, tensorflow, scikit-learn, and pandas, are installed using pip.

After following the Docker build process in figure 3.1 and using the provided Dockerfiles, we successfully created the necessary Docker images, as depicted in figure 4.3.

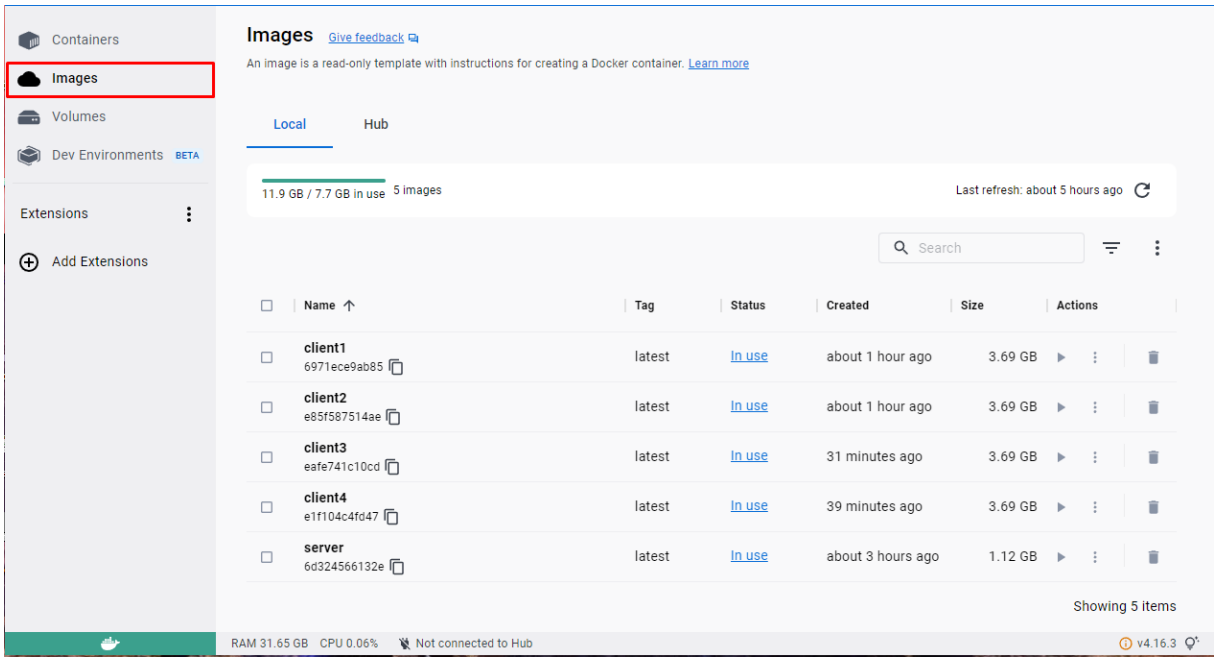


Figure 4.3: Docker images after executing Dockerfile

These images were then used to instantiate the corresponding Docker containers, as shown in figure 4.4.

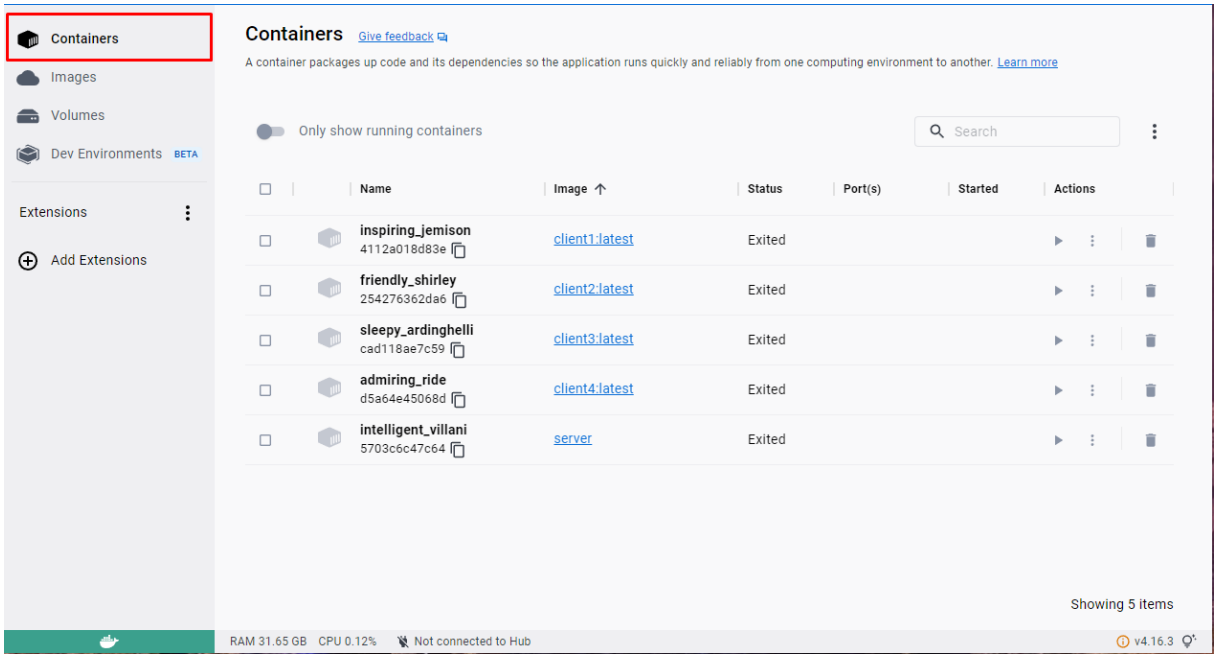


Figure 4.4: Docker containers after executing images

In the given FL setup, it is important to note that the code execution starts as soon as the containers are launched. Therefore, it is crucial to start the server container first, followed by

the client containers. This sequential order is necessary because the server, implemented using the Flower framework, waits for the clients to join in.

If a client container attempts to connect to the server before it is up and running, a disconnection or failure to establish a connection may occur. To prevent such issues, it is recommended to ensure that the server container is started before initiating the client containers. This guarantees proper synchronization and communication between the server and the clients, allowing for a smooth FL process. This approach leverages the benefits that Docker brings to the FL setting, providing advantages such as the ones mentioned in the software tools section.

4.7 Ensemble

To initiate the process of creating an ensemble, the first step involves selecting a suitable ensemble tactic or algorithm from the options discussed in the background. After conducting thorough research, it was determined that stacking would be an ideal choice due to its ease of implementation, particularly considering the compatibility with our existing programming environment.

To begin with, we will enhance the preprocessed data by adding four new columns, each representing the predictions of different locations and their respective models. We have taken care to ensure that the specific trainings and tests of each location are used, just like in the distributed section.

The purpose of creating these additional columns/models is to leverage the concept of ensemble learning. As said before ensemble learning involves combining multiple models to improve the overall prediction and performance. By having multiple models trained on different subsets of the data, we can capture diverse perspectives and potentially achieve better predictive ACC.

Therefore, by incorporating the predictions of multiple models from different locations into our dataset, we aim to create an ensemble approach that combines the strengths of individual models and enhances the overall prediction capabilities.

Chapter 5

Results and Analysis

This chapter presents a detailed examination of the results obtained through various ML techniques applied to the dataset, aiming to shed light on the effectiveness and performance of each approach.

The primary objective of this chapter is to compare and contrast the different types of ML models employed in our study. By examining the results and employing rigorous analytical techniques, we can gain valuable insights into the strengths, weaknesses, and overall performance of each model.

5.1 Centralized Results

For the centralized ML approach, the following results were obtained when analyzing the data:

Type (w/ items)	**AUC**	**ACC**	**Precision**	**Recall**	**F1Score**
All data	0.9777	0.9932	1	0.9565	0.9699

Table 5.1: Centralized results

Here are several important facets concerning each of the values:

- AUC (Area Under the Curve): The AUC value of 0.9777 indicates that the model exhibits a high level of discriminatory power in distinguishing between normal and fraudulent telecommunications events. A higher AUC suggests that the model has a strong ability to correctly classify instances from both classes.
- ACC (Accuracy): With an ACC of 0.9932, the model demonstrates an impressive ability to correctly classify the telecommunications events overall. It correctly predicts the class

labels in 99.32% of the cases.

- Precision: The precision score of 1 indicates that when the model predicts a call as fraudulent, it is highly accurate. The model avoids false positives, minimizing the number of normal calls misclassified as fraudulent.
- Recall: The recall value of 0.9565 highlights the model's ability to identify the majority of fraudulent calls from the dataset. It indicates that 95.65% of the actual fraudulent calls were correctly identified by the model.
- F1Score: The F1Score, which is the harmonic mean of precision and recall, is 0.9699. This metric provides a balanced evaluation of the model's performance by considering both precision and recall. The F1 score indicates that the model achieves a good balance between identifying fraudulent calls and minimizing false positives.

Overall, the results suggest that the centralized ML approach, when applied to the analyzed dataset, shows promising performance in detecting and classifying fraudulent telecommunications events. The high ACC, precision, recall, and F1 score indicate that the model is effective in identifying fraudulent calls while maintaining a low rate of false positives.

5.2 Distributed Results

For the DL approach with four edge nodes, the following results were obtained when analyzing the data at each location:

Type (w/ items)	**AUC**	**ACC**	**Precision**	**Recall**	**F1Score**
Location 1	0.9598	0.9924	0.9903	0.9597	0.9511
Location 2	0.9821	0.9947	1	0.9649	0.9765
Location 3	0.9398	0.991	0.976	0.9562	0.9485
Location 4	0.9805	0.993	0.9951	0.9634	0.9772

Table 5.2: Distributed results

Let's delve into the essential elements pertaining to each of the values:

- AUC (Area Under the Curve): The AUC values across locations range from 0.9398 (location 3) to 0.9821 (location 2).
- ACC (Accuracy): All ACC values surpass 0.9924 (location 1), with location 4 achieving the highest ACC among the evaluated models.
- Precision: Each location achieves precision values above 0.9903, with location 2 standing out as the only one to achieve a perfect precision score of 1.

- Recall: Recall values show variation between locations, with location 2 achieving a value of 0.9649, surpassing the value of 0.9562 achieved by location 3.
- F1Score: The F1Score ranges from 0.9485 (location 3) to 0.9772 (location 4), highlighting location 3's lower performance compared to the others.

5.3 Centralized Training with Distributed Testing Results

In the centralized training and distributed testing approach, the models were trained on a centralized machine using a combined dataset from all locations. The trained models were then tested on individual datasets from each location. We were also careful enough to use the same testing data as in the distributed model. The following results were obtained:

Type (w/ items)	**AUC**	**ACC**	**Precision**	**Recall**	**F1Score**
Location 1	0.9833	0.9934	1	0.9725	0.972
Location 2	0.9787	0.9947	1	0.9668	0.9802
Location 3	0.9673	0.9902	1	0.9424	0.9386
Location 4	0.9819	0.9908	1	0.9522	0.9696

Table 5.3: Centralized Training with Distributed Tests results

Once again we examine the essential elements pertaining to each of the values:

- AUC (Area Under the Curve): The AUC values vary across locations, ranging from 0.9673 (location 3) to 0.9833 (location 1).
- ACC (Accuracy): All ACC values exceed 0.9902 (location 3), with location 2 achieving the highest ACC among the evaluated models.
- Precision: Each location achieves a perfect precision value of 1.
- Recall: Recall values show variation across locations, with location 1 achieving the highest value of 0.9725 and location 3 reaching the lowest value of 0.9424.
- F1Score: The F1Score ranges from 0.9386 (location 3) to 0.9802 (location 2).

5.4 Federated Results

We now present the results obtained from applying the FL approach to the fraud detection task. The FL process involved training ML models on distributed edge nodes located in different locations. The following table summarizes the performance metrics for each location:

Type (w/ items)	AUC	ACC	Precision	Recall	F1Score
Location 1	0.9797	0.9909	0.9831	0.9562	0.9652
Location 2	0.9811	0.9943	1	0.9652	0.9705
Location 3	0.9712	0.9899	0.9853	0.9394	0.9536
Location 4	0.982	0.9912	0.9902	0.9575	0.9706

Table 5.4: Federated results

Now we finally witness the results of our federated system:

- AUC (Area Under the Curve): The AUC values across locations range from 0.9712 (location 3) to 0.9820 (location 4).
- ACC (Accuracy): All ACC values surpass 0.9899 (location 3), with location 4 having the highest ACC among the evaluated models, reaching 0.9912.
- Precision: Location 2 achieves perfect precision, while the other locations exhibit closely similar values, with location 1 having the lowest precision of 0.9831.
- Recall: Recall values show variation across locations, with location 2 achieving the highest value of 0.9652 and location 3 reaching the lowest value of 0.9394.
- F1Score: The F1Score ranges from 0.9536 (location 3) to 0.98706 (location 4).

5.5 Ensemble Results

After training and evaluating the models for each location, we performed an ensemble by combining the predictions from all four locations as explained before. The results of the ensemble are summarized in table 5.5.

Type (w/ items)	AUC	ACC	Precision	Recall	F1Score
Location 1	0.9804	0.9917	0.9823	0.9639	0.964
Location 2	0.98	0.9939	1	0.9599	0.9662
Location 3	0.9698	0.9899	0.9821	0.9423	0.9382
Location 4	0.9819	0.993	0.9953	0.9648	0.9608

Table 5.5: Ensemble results

What was condired in the beguining as a bonus objective we also have the results of our ensemble:

- AUC (Area Under the Curve): The AUC values vary across locations, ranging from 0.9698 (location 3) to 0.9819 (location 4).
- ACC (Accuracy): All ACC values exceed 0.9899 (location 3), with location 4 achieving the highest ACC of 0.9930 among the evaluated models.
- Precision: Location 2 achieves perfect precision, while the other locations display close values, with location 1 reaching a minimum of 0.9823.
- Recall: Recall values show variation across locations, with location 4 achieving the highest value of 0.9648 and location 3 reaching the lowest value of 0.9423.
- F1Score: The F1Score ranges from 0.9382 (location 3) to 0.9662 (location 2).

5.6 Analysis of the Results

To begin with, the results demonstrate the effectiveness of the CL approach in detecting and classifying fraudulent telecommunications events. It achieves high scores in AUC, ACC, recall, F1 score, and perfect precision, indicating its strong performance in identifying fraudulent calls while minimizing false positives.

In terms of precision, the CL approach outperforms the DL approach, as expected. The centralized model benefits from access to a broader range of phone call data, allowing it to gain a more comprehensive understanding of fraudulent call patterns. Conversely, the DL models implemented at individual edge nodes may have limited data access, resulting in slightly lower precision. Nonetheless, it is important to note that the precision achieved by the distributed models remains high, indicating their effectiveness in detecting fraudulent calls.

Moreover, when considering other evaluation metrics such as AUC, ACC, recall, and F1Score, the results exhibit remarkable similarity between the centralized and DL approaches. This suggests that the DL approach can achieve comparable performance to the centralized approach, even with the limitations of working with smaller data subsets at each edge node.

In the case of the centralized training with distributed testing approach, it is expected that precision values of 1 would be attained across all nodes. This expectation arises from the utilization of the centralized model, which benefits from comprehensive training data and the ability to make precise predictions regarding fraudulent calls.

In our FL system, although the results may not always rank as the absolute best among different machine learning approaches, it is crucial to emphasize that achieving the highest scores is not the sole objective. Instead, the focus is on obtaining competitive values while leveraging the unique advantages offered by a federated platform. The slight variations observed in the FL results are outweighed by the benefits it offers, such as data privacy, reduced data transfer, and the ability to leverage local knowledge and patterns within each location. Despite that, the overall results were outstanding, even yielding some perfect scores.

Furthermore, the ensemble approach exhibits remarkable performance across various evaluation metrics. However, it slightly falls short of our expectations in achieving improved results, particularly in the F1 score and precision, in certain scenarios. Strategies for enhancing these aspects will be discussed in the "Future Work" section of chapter 6.

Additionally, it is worth noting that location 3 consistently demonstrates comparatively lower results across all methods. It consistently exhibits the lowest values for AUC, ACC, Recall, and F1Score when compared to other locations. Nevertheless, in the FL setting, location 3 reaches its highest performance peak, surpassing other ML approaches in terms of AUC and F1Score.

Chapter 6

Conclusions

In this thesis, a comprehensive investigation was undertaken to develop and evaluate ML methodologies for fraud detection in telecommunications events. The primary objective was to compare and contrast different approaches and gain insights into their effectiveness and performance against our FL system.

The FL approach was mainly explored, enabling each client to train a model locally on their own data while the server coordinated the learning process by aggregating the models' updates by FedAvg. This approach facilitated collaborative model training without the need to share raw data, thereby enhancing privacy and reducing communication costs.

Throughout the development phase, Docker containers were employed to encapsulate and manage the execution environment, ensuring reproducibility, ease of deployment and privacy insurance. The Docker implementation provided a streamlined and efficient approach to package the necessary components and dependencies for the experimentation process.

Following the development of the methodologies, an extensive evaluation of their performance was conducted. The evaluation encompassed multiple metrics such as AUC, ACC, precision, recall, and F1 score. These metrics provided a comprehensive assessment of the models' discriminatory power, classification accuracy, and ability to balance precision and recall.

This thesis has made significant strides in the development of a FL methodology for fraud detection in telecommunications events. Through a comprehensive evaluation and comparison with various ML methodologies, the thesis has achieved exceptional results, highlighting the effectiveness of the FL approach in detecting fraudulent activities. The investigation shed light on the strengths and weaknesses of different approaches, facilitating an informed selection based on specific requirements such as data privacy, communication costs, and the utilization of local knowledge and how they hold up against our own methodology. The findings presented herein contribute to the advancement of a FL fraud detection technique in the telecommunications industry, providing valuable insights for further research and practical applications. The developed methodology and its evaluation outcomes with other methods lay a solid foundation for future enhancements and refinements in the field of fraud detection.

Future Work

While this thesis has successfully developed and evaluated various ML methodologies for fraud detection in telecommunications events, there are several avenues for future exploration and enhancement.

One promising direction for future work is the development of an ensemble approach that combines the strengths of all the ML models developed in this thesis. By leveraging the diversity and complementary nature of these models, an ensemble can potentially enhance the overall fraud detection performance. Ensemble techniques, such as model averaging, voting, can be employed to combine the predictions of individual models and produce a final consolidated output. This ensemble approach could benefit from the high ACC, precision, and recall demonstrated by the CL approach, the data privacy and reduced communication costs offered by FL, and the potential for leveraging local knowledge in the DL approach.

Another promising avenue to explore would be the implementation of this code on real devices, rather than relying solely on docker containers. This practical implementation would provide valuable insights into the performance and effectiveness of the developed methodology, validating the prominent results obtained in this thesis.

Bibliography

[1] https://docs.docker.com.

[2] Communications fraud control association – fraud loss survey report 2021, 2021.

[3] Sawsan Abdulrahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Mohsen Guizani. A survey on federated learning: The journey from centralized to distributed on-site learning and beyond. *IEEE Internet of Things Journal*, 8(7):5476–5497, 2021. doi:10.1109/JIOT.2020.3030072.

[4] Gary W Adams, David R Campbell, Mary Campbell, and Michael P Rose. Fraud prevention. *The CPA Journal*, 76(1):56, 2006.

[5] Mais Arafat, Abdallah Qusef, and George Sammour. Detection of wangiri telecommunication fraud using ensemble learning. In *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, pages 330–335, 2019. doi:10.1109/JEEIT.2019.8717528.

[6] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2022.

[7] Alberto Blanco-Justicia, Josep Domingo-Ferrer, Sergio Martínez, David Sánchez, Adrian Flanagan, and Kuan Eeik Tan. Achieving security and privacy in federated learning systems: Survey, research challenges and future directions. *Engineering Applications of Artificial Intelligence*, 106:104468, 2021. ISSN: 0952-1976. doi:https://doi.org/10.1016/j.engappai.2021.104468.

[8] Theodora S. Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch. Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International Journal of Medical Informatics*, 112:59–67, 2018. ISSN: 1386-5056. doi:https://doi.org/10.1016/j.ijmedinf.2018.01.007.

[9] Jeongmin Chae and Songnam Hong. Multiple kernel-based online federated learning, 02 2021.

[10] Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin. Vafl: a method of vertical asynchronous federated learning, 2020.

[11] Te-Chuan Chiu, Yuan-Yao Shih, Ai-Chun Pang, Chieh-Sheng Wang, Wei Weng, and Chun-Ting Chou. Semisupervised distributed learning with non-iid data for aiot service platform. *IEEE Internet of Things Journal*, 7(10):9266–9277, 2020. doi:10.1109/JIOT.2020.2995162.

[12] Georgios Darivianakis, Angelos Georghiou, and John Lygeros. Decentralized decision making for networks of uncertain systems, 2021.

[13] T. G Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization, machine learning, 2000.

[14] B. Dzeroski, S.and Zenko. Is combining classifiers with stacking better than selecting the best one?, machine learning, 2004. ISSN: 1573-0565. doi:https://doi.org/10.1023/B:MACH.0000015881.36452.6e.

[15] Vipul Gupta, Avishek Ghosh, Michal Derezinski, Rajiv Khanna, Kannan Ramchandran, and Michael Mahoney. Localnewton: Reducing communication bottleneck for distributed learning, 2021.

[16] Betty Edelman Herve Abdi, Dominique Valentin. Neural networks, 1999.

[17] Wei Huang, Tianrui Li, Dexian Wang, Shengdong Du, Junbo Zhang, and Tianqiang Huang. Fairness and accuracy in horizontal federated learning. *Information Sciences*, 589:170–185, 2022. ISSN: 0020-0255. doi:https://doi.org/10.1016/j.ins.2021.12.102.

[18] Botao Jiao, Yinan Guo, Dunwei Gong, and Qiuju Chen. Dynamic ensemble selection for imbalanced data streams with concept drift. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2022. doi:10.1109/TNNLS.2022.3183120.

[19] Abdurraheem Joomye, Mohammad Tahir, Muhammad Aman Sheikh, Mee Hong Ling, and Yap Kian Meng. Performance analysis of federated learning in wireless networks. In *2022 IEEE 6th International Symposium on Telecommunication Technologies (ISTT)*, pages 68–73, 2022. doi:10.1109/ISTT56288.2022.9966534.

[20] Ludmila I. Kuncheva. Diversity in multiple classifier systems. *Information Fusion*, 6(1):3–4, 2005. ISSN: 1566-2535. Diversity in Multiple Classifier Systems. doi:https://doi.org/10.1016/j.inffus.2004.04.009.

[21] Wai Lam and A.M. Segre. A distributed learning algorithm for bayesian inference networks. *IEEE Transactions on Knowledge and Data Engineering*, 14(1):93–105, 2002. doi:10.1109/69.979975.

[22] Kwing Hei Li, Pedro Porto Buarque de Gusmão, Daniel J. Beutel, and Nicholas D. Lane. Secure aggregation for federated learning in flower. In *Proceedings of the 2nd ACM International Workshop on Distributed Machine Learning*, DistributedML '21, page 8–14, New York, NY, USA, 2021. Association for Computing Machinery. ISBN: 9781450391344. doi:10.1145/3488659.3493776.

[23] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers Industrial Engineering*, 149:106854, 2020. ISSN: 0360-8352. doi:https://doi.org/10.1016/j.cie.2020.106854.

[24] Qing Li, Weidong Cai, Xiaogang Wang, Yun Zhou, David Dagan Feng, and Mei Chen. Medical image classification with convolutional neural network. In *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*, pages 844–848, 2014. doi:10.1109/ICARCV.2014.7064414.

[25] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 22(3): 2031–2063, 2020. doi:10.1109/COMST.2020.2986024.

[26] Xiaoyuan Liu, Tianneng Shi, Chulin Xie, Qinbin Li, Kangping Hu, Haoyu Kim, Xiaojun Xu, Bo Li, and Dawn Song. Unifed: A benchmark for federated learning frameworks, 2022.

[27] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. A secure federated transfer learning framework. *IEEE Intelligent Systems*, 35(4):70–82, 2020. doi:10.1109/MIS.2020.2988525.

[28] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2023.

[29] David Mease and Abraham Wyner. Evidence contrary to the statistical view of boosting. *J. Mach. Learn. Res.*, 9:131–156, jun 2008. ISSN: 1532-4435.

[30] Yoram J. Meijaard, Bram C. M. Cappers, Josh G. M. Mengerink, and Nicola Zannone. Predictive analytics to prevent voice over ip international revenue sharing fraud. In Anoop Singhal and Jaideep Vaidya, editors, *Data and Applications Security and Privacy XXXIV*, pages 241–260, Cham, 2020. Springer International Publishing. ISBN: 978-3-030-49669-2.

[31] Srujana Merugu and Joydeep Ghosh. A distributed learning framework for heterogeneous data sources. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, page 208–217, New York, NY, USA, 2005. Association for Computing Machinery. ISBN: 159593135X. doi:10.1145/1081870.1081896.

[32] Aji Mubalaike Mubarek and Eşref Adalı. Multilayer perceptron neural network technique for fraud detection. In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 383–387, 2017. doi:10.1109/UBMK.2017.8093417.

[33] E.W.T. Ngai, Yong Hu, Y.H. Wong, Yijun Chen, and Xin Sun. The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision Support Systems*, 50(3):559–569, 2011. ISSN: 0167-9236. On quantitative methods for detection of financial fraud. doi:https://doi.org/10.1016/j.dss.2010.08.006.

[34] Bo Pang, Erik Nijkamp, and Ying Nian Wu. Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics*, 45(2):227–248, 2020.

[35] Taehyeun Park and Walid Saad. Distributed learning for low latency machine type communication in a massive internet of things. *IEEE Internet of Things Journal*, 6(3): 5562–5576, 2019. doi:10.1109/JIOT.2019.2903832.

[36] Pradheepan Raghavan and Neamat El Gayar. Fraud detection using machine learning and deep learning. In *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, pages 334–339, 2019. doi:10.1109/ICCIKE47802.2019.9004231.

[37] Akshaya Ravi, Mounira Msahli, Han Qiu, Gerard Memmi, Albert Bifet, and Meikang Qiu. Wangiri fraud: Pattern analysis and machine-learning-based detection. *IEEE Internet of Things Journal*, 10(8):6794–6802, 2023. doi:10.1109/JIOT.2022.3174143.

[38] Merve Sahin and Aurélien Francillon. Understanding and detecting international revenue share fraud. In *NDSS*, 2021.

[39] Merve Sahin, Aurélien Francillon, Payas Gupta, and Mustaque Ahamad. Sok: Fraud in telephony networks. In *2017 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 235–250, 2017. doi:10.1109/EuroSP.2017.40.

[40] Juncai He sci. Relu deep neural networks and linear finite elements. *Journal of Computational Mathematics*, 38(3):502–527, jun 2020. doi:10.4208/jcm.1901-m2018-0160.

[41] Paula Raissa Silva, João Vinagre, and João Gama. Towards federated learning: An overview of methods and applications. *WIREs Data Mining and Knowledge Discovery*, 13(2):e1486, 2023. doi:https://doi.org/10.1002/widm.1486.

[42] Rashish Tandon, Qi Lei, Alexandros G. Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3368–3376. PMLR, 06–11 Aug 2017.

[43] Kanchan M Tarwani and Swathi Edem. Survey on recurrent neural network in natural language processing. *Int. J. Eng. Trends Technol*, 48(6):301–304, 2017.

[44] Grigorios Tsoumakas, Ioannis Partalas, and Ioannis Vlahavas. A taxonomy and short review of ensemble selection, 2008.

[45] Massimiliano Zanin, Miguel Romance, Santiago Moral, Regino Criado, et al. Credit card fraud detection through parenclitic network analysis. *Complexity*, 2018, 2018.

[46] Wanru Zhao, Xinchi Qiu, Javier Fernandez-Marques, Pedro P. B. de Gusmão, and Nicholas D. Lane. Protea: Client profiling within federated systems using flower. In *Proceedings of the 1st ACM Workshop on Data Privacy and Federated Learning Technologies for Mobile Edge Network*, FedEdge '22, page 1–6, New York, NY, USA, 2022. Association for Computing Machinery. ISBN: 9781450395212. doi:10.1145/3556557.3557950.

[47] Anders Øland and Bhiksha Raj. Reducing communication overhead in distributed learning by an order of magnitude (almost). In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2219–2223, 2015. doi:10.1109/ICASSP.2015.7178365.