EXPLORING THE APPLICATION OF VERTICAL FEDERATED LEARNING FOR CLASSIFICATION PROBLEMS

by

Sehrish Asghar

In the partial fulfillment of the requirements
for the degree
Master of Philosophy

Department of Electronics
Quaid-I-Azam University Islamabad
Pakistan.
2021-2023

Certificate

It is certified that the work presented in this dissertation is accomplished by Sehrish Asghar under my supervision at Quaid-i-Azam University, Islamabad, Pakistan.

Supervisor:

____

Dr. Musarat Abbas
Associate Professor
Department of Electronics
Quaid-i-Azam University, Islamabad, Pakistan.

Submitted through:

____

Prof. Dr. Qaisar Abbas Naqvi
Chairman
Department of Electronics
Quaid-i-Azam University, Islamabad, Pakistan.

Acknowledgements

I sincerely thank Almighty Allah, the Most Gracious and the Most Merciful, Who has rewarded me in ways far beyond anything I could have imagined. I praise Prophet Muhammad (S.A.W.W), whose life inspires all humans.

I would like to extend my profound appreciation to my supervisor, Dr. Musarat Abbas, for his assistance, inspiration, and direction during the entirety of my M.Phil. studies. His expertise and experience have enhanced my knowledge and research-related abilities. This research would not have been possible without his gracious guidance and supportive suggestions. His guidance and belief in me have been instrumental in my academic growth and success.

I would like to express my appreciation to Prof. Dr. Antonio Coronato for his contribution to the successful completion of this research work. Sincere thanks also go to Dr. Giovanni Paragliola for his continuous support, advice, encouragement, and valuable suggestions during the entire period of my research work. Moreover, his timely assistance in providing necessary resources for research played a key role in the completion of this research work. I also appreciate Dr. Muddasar Naeem for his positive attitude and enthusiasm that kept me motivated throughout the research.

I offer my deepest gratitude to Prof. Dr. Qaisar Abbas Naqvi and all the teachers at the Department of Electronics who encouraged and taught me in my M.Phil coursework.

I want to express my sincere gratitude to my parents, and my entire family for their unwavering love and assistance during my education.

Sehrish Asghar

Dedicated To my beloved Parents

Abstract

The primary goal of occupancy detection is to determine whether the room or any specific place is currently in use by some individual or specific item. This capability holds immense potential for managing efficient electricity, heat, and ventilation in large buildings like hospitals, hotels, or industries. The main objective of this research work is to employ a machine learning technique called vertical federated learning for occupancy detection. In this research occupancy detection dataset from the UCI Machine Learning Repository is used. For occupancy detection, six classifiers are used for the prediction of the highest accuracy and deep learning model. The algorithm of vertical federated learning with categorical cross-entropy loss (vFedCCE) is used to deploy the gradient-based optimizer on the clients, instead of the centralized server using the occupancy detection dataset UCI. The proposed model consists of two clients: client1 and client2, both having the same samples with distinct sets of features. In the initial stage, client1 will develop its model by considering its distinctive features and sharing this model with client2. Client2 will utilize both its model and the model received from client1 to make predictions. After independently calculating its gradient values, client2 will then update its model weights accordingly. This collaborative effort aims to improve the overall model performance, as the client2 will incorporate the gradients obtained from the client1 into its model weights and subsequently return these updated weights to client1. After this client1 updates its model weights based on the received gradients from client2. This process continues until the desired results are achieved. It has been observed that the vFedCCE model exhibits low accuracy as compared to the centralized model while preserving computational cost, privacy, and data bandwidth.

Contents

List of Figures

List of Tables

List of Acronyms

Acronym	**Description**
AI	Artificial Intelligence
ANN	Artificial neural network
ML	Machine learning
FL	Federated learning
HFL	Horizontal Federated Learning
VFL	Vertical Federated Learning
FTL	Federated Transfer Learning
PCA	Principal component analysis
RF	Random Forest
LR	Logistic Regression
GNB	Gaussian Naive Bayes
LDA	Linear Discriminant Analysis
SVM	Support vector machine
KNN	K-nearest neighbor
DL	Deep learning
NLP	Natural language processing
IoT	Internet Of Things
vFedCCE	Vertical Federated Categorical Cross Entropy

Chapter 1

Introduction

1.1 Artificial Intelligence

In the present era, people experience a life marked by enhanced progress and convenient utilization of advanced technologies. AI pioneer John McCarthy coined the term as the discipline of engineering and scientific endeavors entailing the construction of intelligent devices, with a particular emphasis on computer programs. Artificial Intelligence involves the process of enabling computers, computer-controlled robots, or software to exhibit intelligent thinking akin to the human mind. This is achieved through a comprehensive study of human brain patterns and the analysis of cognitive processes. The knowledge gained from these studies is utilized to create intelligent software and systems. There has been an increase in artificial intelligence-related software in recent years. Figure 1.1 shows the brief history of AI.

1.1.1 Working of AI

In basic terms, AI systems operate by integrating extensive datasets with intelligent and iterative processing algorithms. This amalgamation enables AI to acquire knowledge from patterns and characteristics found in the analyzed data. With each iteration of data processing, an AI system evaluates its performance, measures the outcomes, and utilizes the results to enhance its expertise further.

1.1.2 Ways of implementing AI

There are several ways to implement AI, depending on the specific problem and the available resources.

1.1.2.1 Machine Learning

Machine Learning is described as a branch of AI that utilizes statistical techniques to empower computer systems in acquiring knowledge from data, ultimately working towards achieving specific objectives. The concept was originally coined by Arthur Samuel in 1959.

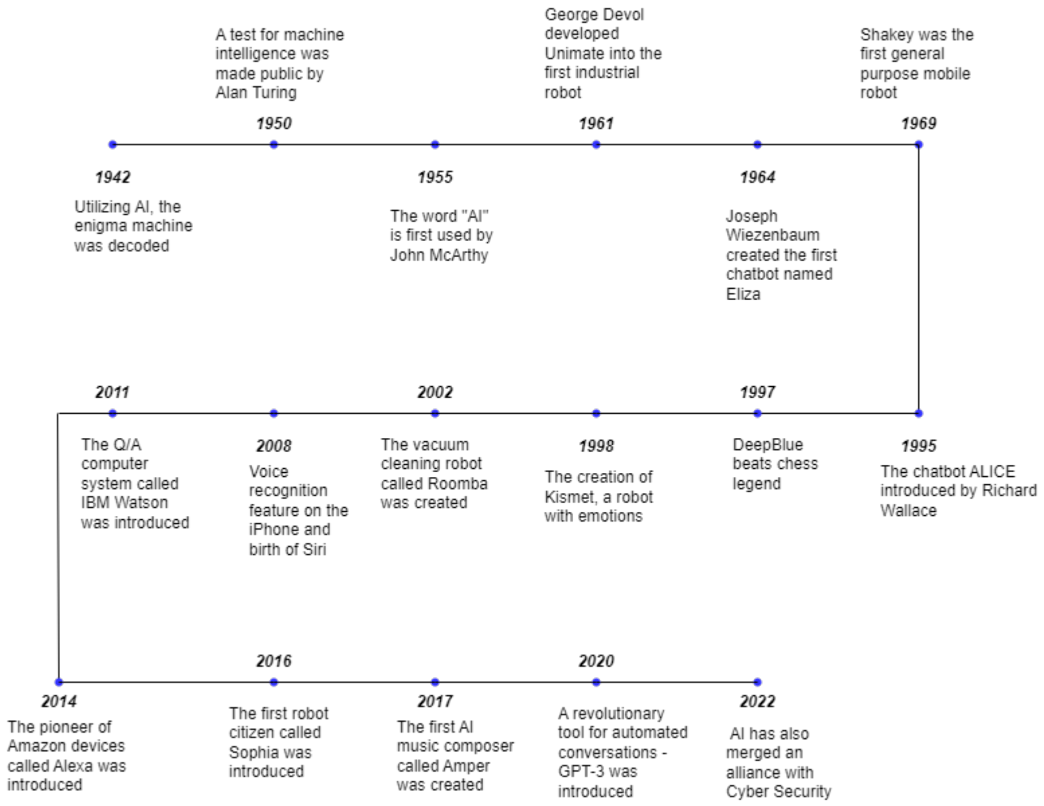

1942
Utilizing AI, the enigma machine was decoded
1950
A test for machine intelligence was made public by Alan Turing
1955
The word "AI" is first used by John McArthy
1961
George Devol developed Unimate into the first industrial robot
1964
Joseph Wiezenbaum created the first chatbot named Eliza
1969
Shakey was the first general purpose mobile robot
1995
The chatbot ALICE introduced by Richard Wallace
1997
DeepBlue beats chess legend
1998
The creation of Kismet, a robot with emotions
2002
The vacuum cleaning robot called Roomba was created
2008
Voice recognition feature on the iPhone and birth of Siri
2011
The Q/A computer system called IBM Watson was introduced
2014
The pioneer of Amazon devices called Alexa was introduced
2016
The first robot citizen called Sophia was introduced
2017
The first AI music composer called Amper was created
2020
A revolutionary tool for automated conversations - GPT-3 was introduced
2022
AI has also merged an alliance with Cyber Security

Figure 1.1: History of AI

1.1.2.2 Deep Learning

Deep learning is a subset of machine learning that enables AI to replicate the intricate neural network of the human brain. This sophisticated technique equips AI systems with the capacity to comprehend intricate patterns, identify relevant information amidst the noise, and effectively handle sources of ambiguity within the provided data. It is composed of artificial neurons or interconnected nodes, that transmit and process information to deal with challenging problems. It uses vast amounts of data to train these networks, allowing them to learn how to perform tasks such as autonomous decision-making, and speech and image recognition.

1.2 Learning Paradigms

The supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning are the learning paradigms of ML.

1.2.1 Supervised Learning

Supervised learning is a type of ML that involves training an algorithm with a labeled dataset whose outcomes have already been determined. For example, an image serves as the input, and the result is the image's class or category, such as cat, dog, or vehicle. Because the photographs are manually classified by humans, the outcome is already known. This labeled dataset trains the system on how to classify new, unseen images. Tasks including scene segmentation, object detection, and image recognition benefit from this methodology. The two dataset components that are split into two categories for supervised learning are the test and the training set. The algorithm uses the training set to figure out how the inputs and outputs are connected. The test set is then used to evaluate the algorithm's predictions' accuracy. Algorithms for supervised learning often fall into one of two categories: classification or regression. Classification and regression are the two basic subtypes of supervised learning algorithms. Classification algorithms are used to predict categorical outcomes, such as whether or not an email is spam, while regression algorithms are used to forecast continuous outcomes, such as the price of a stock.

1.2.2 Unsupervised Learning

Unsupervised learning is a kind of ML where the algorithm tries to find patterns or structures in the data without the help of labeled outputs. For example, in marketing, clustering can be used to segment customers based on their purchasing behavior, without having prior knowledge of which customers belong to which group. The algorithm explores the data to uncover hidden structures and relationships between variables. It is also used to reduce the dimensionality of the data. For example in dimensionality reduction, the algorithm reduces the number of features in high-dimensional data to make it easier to visualize and analyze. This can be beneficial in situations where high dimensional data is difficult to display and interpret, such as face recognition. Some algorithms used in unsupervised learning are PCA, Autoencoder, and many more depending upon requirements.

1.2.3 Semi-supervised Learning

Semi-supervised learning is a category of machine learning that falls somewhere between supervised learning and unsupervised learning. In semi-supervised learning, the training dataset consists of both labeled and unlabeled examples, allowing the model to learn from both sources of information. For example, Imagine you are working for a company that develops a spam email classifier. You have a limited budget to hire human experts to label emails as either "spam" or "not spam" for training. However, you have a much larger set of unlabeled emails available. In a typical supervised learning scenario, you would only use the labeled emails to train a model. However, in a semi-supervised learning approach, you can take advantage of the large pool of unlabeled emails to improve the model's performance.

1.2.4 Reinforcement Learning

Reinforcement learning is a machine learning approach wherein an agent acquires the skill of decision-making through a sequence of actions within an environment, aiming to optimize its performance based on a reward signal. The aim is to learn the best sequence of actions to take in order to maximize a reward signal. This occurs through a trial-and-error process in which the agent engages with its surroundings, receives feedback in the form of rewards or penalties, and adapts its approach accordingly. Over time, the agent's policy improves, leading to a more optimal approach to the task at hand. This method can be applied to real-world problems, like robotics, recommendation systems, and autonomous decision-making. A simple case of reinforcement learning can be seen in a robot attempting to navigate a maze. The robot is positioned in the maze and gets a reward for reaching the end and a penalty for hitting a wall. Through repeated trial and error, the robot gradually learns the best way to get to the end while avoiding the walls. The robot updates its strategy after each attempt by analyzing the rewards received, determining which actions lead to the highest rewards. Over time, the robot's approach to the maze becomes more refined, converging toward the optimal solution. This scenario showcases how reinforcement learning can be utilized to tackle problems that involve making sequential decisions and maximizing rewards.

1.3 Artificial Neural Network

The design of an ANN, including the number of nodes in each layer, the number of layers, and the connections between the nodes, is referred to as its architecture. The network's design has a critical role in how well it can identify the underlying patterns in the data. A standard artificial neural network comprises an input layer, one or more hidden layers, and an output layer.

- The input layer accepts input data in a variety of forms that the user provides.
- The hidden layers process the information and transmit it to the output layer from the input layer.
- The network's predictions are produced by the output layer.

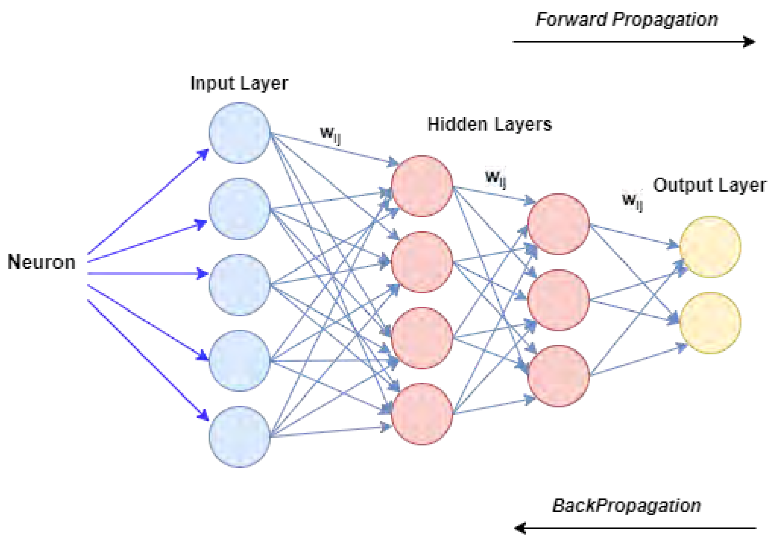


Figure 1.2: Artificial Neural Network

Each node in the network represents an artificial neuron, which performs a simple computation on the data it receives from input layers and transmits the result to the next layers. The connections linking the nodes are characterized by weights that undergo adjustments throughout the training process. These adjustments aim to minimize the disparity between the network's predictions and the real output. Weights determine the strength of the connection between two nodes, with larger weights indicating a stronger connection. Biases also play a crucial role in determining the network's predictions. This can help to ensure that the network is not stuck in a local minimum during the training process and can also help to improve the network's accuracy. Both weights and biases are updated during the training process, using algorithms such as backpropagation. The objective is to identify the values that minimize the error between the actual output and the network's predictions. The design of the architecture is often informed by the nature of the problem being solved and the available data. After receiving input signals and weights, the neuron performs a dot product and passes the result through the activation function. It maps the dot product to a new output value, which can be used by other neurons in the network. There are several popular activation functions used in ANNs, each with its own strengths and limitations. Some are

1.3.1 Sigmoid

The sigmoid function is widely used in the output layer of a binary classification problem, as it maps inputs to outputs in the range of 0 to 1, representing the probability of the positive class.

$$\mathrm{f}(x) = \frac{1}{1+e^{-x}} \tag{1.1}$$

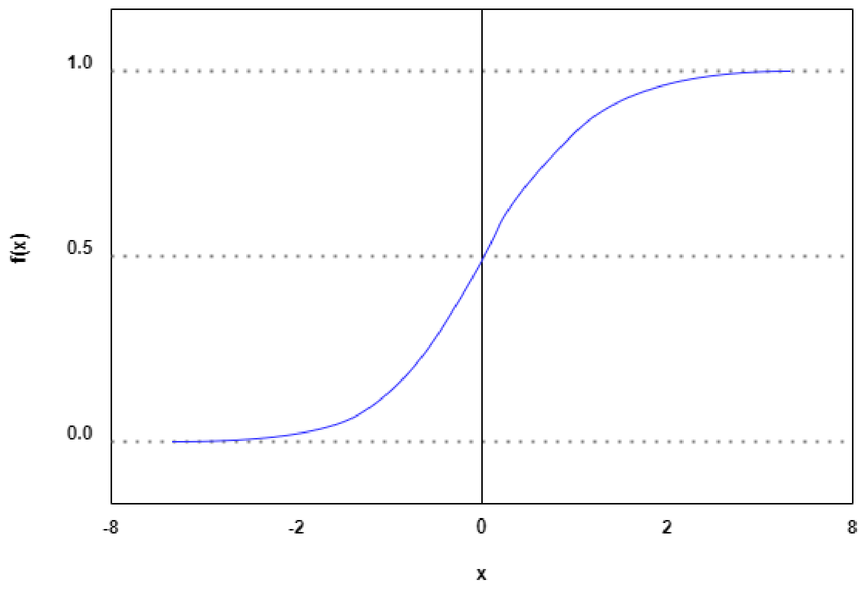


Figure 1.3: Sigmoid

The limitations of the sigmoid activation function include the saturation issue where it becomes difficult for the network to learn for large positive or negative inputs, as the derivative approaches zero. Additionally, the calculation of the sigmoid function can be slow, affecting the speed of network convergence during training.

1.3.2 Hyperbolic Tangent (Tanh)

In a neural network, the tanh function is commonly used as an activation function. It maps any input value to a value between -1 and 1 and is centered around zero, making it a good choice for hidden layers in the network.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.2}$$

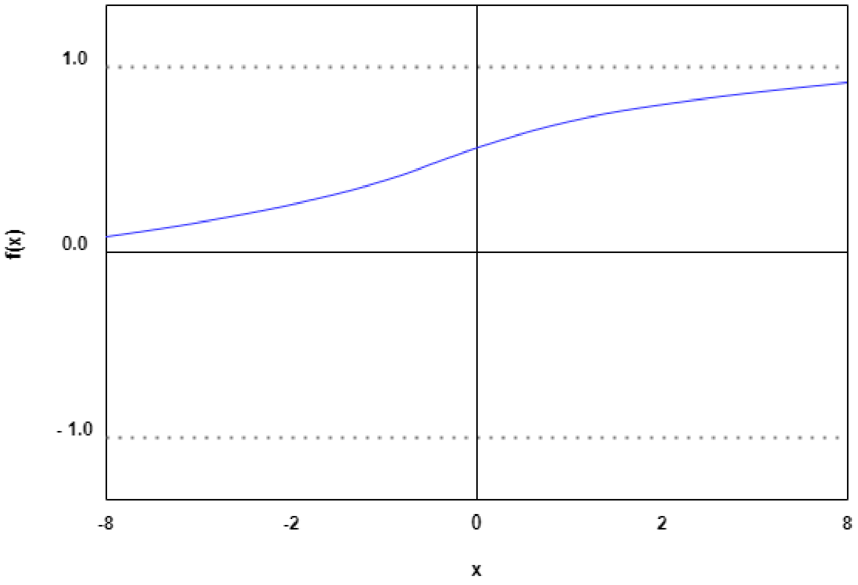


Figure 1.4: Tanh(x)

However, it also has some limitations. One of the major limitations is saturation, where the function reaches large positive or negative values and the gradient becomes close to zero, slowing down training. Since the range of tanh is not (0,1), it's not suitable for binary classification tasks where the output should be between 0 and 1. These limitations should be considered when using the tanh function in a neural network.

1.3.3 Softmax

In a neural network's output layer, the Softmax function is a mathematical function that is frequently used as an activation function. This function's objective is to predict the class that has the highest probability out of all the classes in multiclass classification issues. The Softmax function maps a set of real-valued inputs to a probability distribution across the classes.

$$\mathrm{f}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{1.3}$$

The limitation of the Softmax function is that it assigns non-zero probabilities to all classes, even when some of the classes are not relevant. This can result in poor performance in certain classification problems, such as multi-label classification, where it is important to differentiate between relevant and irrelevant classes.

1.3.4 Rectified Linear Unit(ReLU)

Deep neural networks or multi-layer neural networks both employ the non-linear activation function known as ReLu. It maps negative input values to zero and positive values remain unchanged, resulting in a piecewise linear function. The ReLU activation is simple and computationally efficient, making it a popular choice in deep networks. This helps improve the network's training speed and prevent the vanishing gradient problem. The mathematical expression is

$$\mathrm{f}(x) = \max(0, x) \tag{1.4}$$

If the input is negative, ReLU units can become inactive (output 0), which results in dead neurons that no longer contribute to the network's prediction.

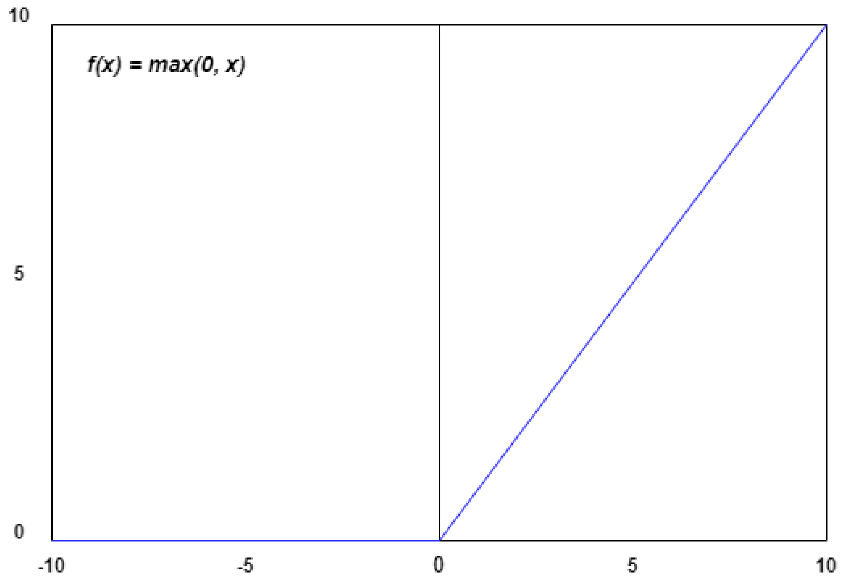


Figure 1.5: Relu

1.4 Limitations of AI, ML, and DL

Although AI, ML, and DL are transformative technologies that are reshaping numerous industries, they are not without limitations. They are found in many applications ranging from medicine, and engineering, to the Internet of Things (IoT), marketing, and business analytics tools. While the advancements in technology bring numerous benefits, it is important to note that these algorithms heavily rely on substantial volumes of data for the training and testing of their models. While data acquisition may be relatively straightforward in certain scenarios, there is a significant privacy concern when models require training on user data. This arises due to the sensitive nature of personal information and the need to protect individuals' privacy rights. To safeguard user privacy, several regulations and legal policies have been introduced, such as the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA). These measures aim to establish guidelines and requirements for the protection and secure handling of user data. The current ML model training process encounters several challenges. Firstly, gathering data and centralizing it in a single source can be costly and resource-intensive, making it a barrier to acquiring large and diverse datasets. Secondly, there is a concern regarding the security and privacy of sensitive data when it is concentrated in a single location. This concentration increases the risk of data breaches and unauthorized access, potentially compromising the confidentiality and integrity of the data. Additionally, ML model training demands extensive computing and power resources. The computational requirements can be significant, especially for deep learning models that involve complex neural networks. This places a burden on organizations to invest in a high-performance computing infrastructure to meet the computational demands of the training process. As users become more aware of privacy concerns and the regulations surrounding data protection, they exhibit a reluctance to share their data due to security and privacy apprehensions. Consequently, this lack of data availability negatively impacts the accuracy of ML models. Furthermore, the centralized training of ML and Deep Learning (DL) models necessitates substantial amounts of data and powerful computational resources. Acquiring the necessary energy and power to support the training and testing of these models becomes an additional challenge in their deployment. Moreover, the data collected from diverse sources often exist in different formats, requiring extensive preprocessing and cleaning efforts before they can be effectively utilized for model training. The classic method of training ML models, centralized training, is collecting and combining data from diverse sources into a single location where a global model is developed. This strategy has various disadvantages, including worries about privacy because it needs data exchange, which may expose sensitive information to unauthorized users. Decentralized training, on the other hand, is a method of training models locally on dispersed data without sharing the data itself. This technique resulted in the creation of FL, which enables collaborative training of machine learning models while ensuring data privacy.

1.5 Motivation

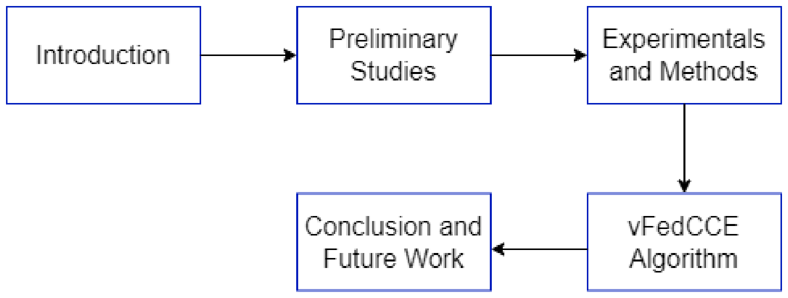
Recognizing the challenges of centralized learning and embracing the concept of vertical federated learning (VFL) applied to occupancy detection datasets, the motivation driving this thesis is to enhance the efficiency and precision of machine learning models tailored to this specific context. The fundamental goal remains rooted in safeguarding data security and privacy within vertical data sharing frameworks by using the vFedCCE algorithm. VFL emerges as a promising avenue, introducing decentralized learning that encourages seamless collaboration and data exchange among vertically segregated sources. This approach holds the realm of occupancy detection while upholding the confidentiality of sensitive occupancy-related data.

1.6 Objective

- The goal is to produce a model to classify and predict the occupancy detection dataset by using machine learning techniques.

- Improve the accuracy and efficiency of machine learning models used for occupancy detection.
- In the FL process, address the difficulties of centralized and decentralized learning.
- Once high accuracy is reached, implement VFL using the vFedCCE algorithm, enabling diverse participants to collaborate. It employs client-side gradient-based optimization with categorical cross-entropy, ensuring privacy in shared ID space and data alignment without disclosing sensitive information.

1.7 Thesis Structure



1.8 Summary

This chapter covers key aspects of AI, including various learning paradigms such as supervised, unsupervised, semi-supervised, and reinforcement learning. It introduces the concept of artificial neural networks as fundamental tools for learning complex patterns. Additionally, the content highlights the limitations of AI, ML, and DL. At the end, the chapter shows the research motivation, objectives, and thesis structure.

Chapter 2

Preliminary Studies

2.1 Federated Learning

In 2016, Google introduced Federated Learning (FL), and in 2017, they integrated it into Gboard for Android [35]. FL is a decentralized machine learning technique that protects user privacy and enables several parties to train a shared model without disclosing their data. The research on FL can be broadly categorized into three main aspects. Firstly, improving the efficiency and effectiveness of FL. Secondly, the security of FL is a significant concern. Finally, improving privacy preservation. FL is appropriate for applications where data confidentiality is important. The process of FL is as in figure 2.1. This approach is called federated averaging which is the baseline of FL in many researches. The process is explained in the following steps.

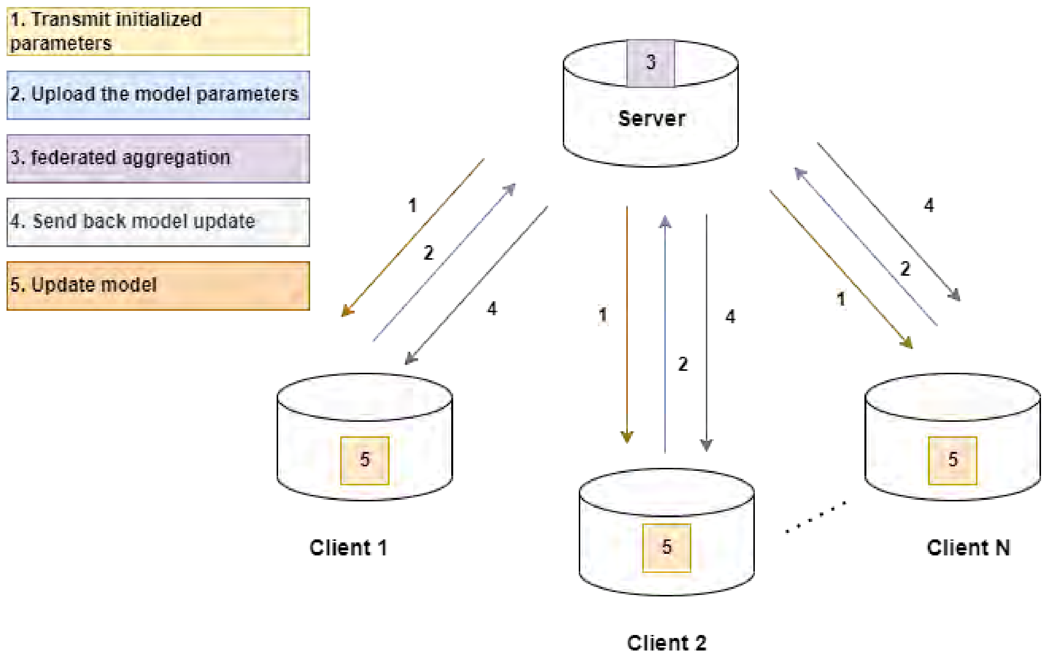

1. Transmit initialized parameters
2. Upload the model parameters
3. federated aggregation
4. Send back model update
5. Update model
3
Server
1
2
4
1
2
4
1
2
4
5
5
5
Client 1
Client 2
Client N

Figure 2.1: Process of Federated Learning

1. The server initializes the global model weights and hyperparameters. The hyperparameters include the number of FL rounds, the total number of participating clients, and the number of clients that will be selected during each training round. These hyperparameters are important as they govern the overall training process and can affect the performance and efficiency of the FL model. Once the global model weights and hyperparameters are initialized, the server activates the participating clients and broadcast the initialized global model to them. The server then selects a certain number of clients to participate in each training round based on certain criteria, such as device availability, device performance, and data quality. After the clients are selected, The server shares global information such as weights or gradients with the selected clients.

2. The selected clients then use their local datasets to train their local models and send their local information, such as weights or gradients, back to the server.

3. After receiving the local information from the selected clients, the server aggregates this information and update the global model. This updated information is then used by the clients for the next round of training.

4. The global server then sends the updated parameters of the global model to each client.

5. This process is repeated until the desired results are achieved.

2.2 Applications of Federated Learning

FL can be used in various applications. An overview of some of the applications is explained below.

2.2.1 Healthcare

FL has many potential applications in the healthcare industry due to the need to protect sensitive patient data while leveraging the benefits of machine learning. While medical institutions may have a considerable amount of patient data, this data alone may be insufficient for training their own prediction model. FL can be used to build predictive models to identify and prevent adverse health outcomes. With FL, healthcare organizations can collaborate and share data without compromising patient privacy. For example, multiple hospitals could contribute their EHR data to train a model for predicting sepsis risk, but each hospital would only send encrypted data to a central server for model aggregation. [5] proposed the framework “FedHealth” utilizes federated transfer learning to train a model for personalized prediction based on data gathered from wearable healthcare devices and it is tested for Parkinson’s disease. The project “Federated Mortality Prediction” by organizations such as AI Sweden utilizes federated learning to predict the survival rate of emergency care patients. [17] utilized federated learning to perform brain tumor segmentation using the BraTS dataset and they also incorporated differential privacy techniques to ensure that data privacy regulations were being followed.

2.2.2 Natural Language Processing

Natural language processing is one of the most common applications which uses machine learning models. It eliminates language complexity by enabling machines to communicate with it and comprehend human language in both written and spoken form. Training accurate language models for NLP requires a large amount of data, which be easily collected from mobile devices. However, centralized models that collect and store text data from individual devices may face privacy concerns as the data often contain user information. By leveraging FL, it is possible to build accurate NLP models without compromising the privacy and security of the training data as shown in [8]. Google's GBoard is a popular application that utilized FL to enhance query suggestions on Android phones keyboard [35]. Another application is predicting the next word [10]. In [25] FL utilized for the prediction of emojis.

2.2.3 Computer Vision

Computer vision is an important application area of federated learning that involves training models on visual data. For example, FL can be used in object recognition tasks, where the model is trained to identify specific objects within an image or video and FL can also be applied to facial recognition, where the model is trained to identify individuals in images or videos. Additionally, FL can be used in traffic sign recognition, where the model is trained to recognize different types of traffic signs, such as speed limit signs or stop signs. [27] introduced a method for detecting face presentation attacks using FL. [20] proposed a platform "FedVision" to develop computer vision-based safety monitoring solutions in smart city applications.

2.2.4 Autonomous Vehicles

Federated learning can provide two key benefits for self-driving cars. Firstly, it can ensure data privacy by using precise data rather than sharing and analyzing the complete user information on a central server. Secondly, it can reduce latency, allowing self-driving cars to respond quickly during safety incidents when there are many such cars on the roads. Conventional cloud-based machine learning requires transferring large amounts of data to a central server for training, which can be slow and inefficient for autonomous vehicles that require immediate decision-making. On the other hand, FL can allow vehicles to learn from decentralized data sources without transferring the raw data to a central location. This can result in faster learning and better performance, which is crucial for ensuring the safety of autonomous vehicles. FL can enable vehicles to respond more quickly and accurately to changing environments, reducing the risk of accidents and improving overall safety. In [23] the authors conducted a user study to evaluate the effectiveness of personalized federated learning for trajectory models in autonomous vehicles.

2.2.5 IoT

The Internet of Things (IoT) is a rapidly growing network of devices, vehicles, buildings, and other physical objects that are embedded with sensors, software, and network connectivity, enabling them to collect and exchange data. These devices are capable of sensing and computing, which allows them to monitor their environment, communicate with other devices, and make decisions based on the data they collect. The Internet of Things (IoT) has become increasingly prevalent in our daily lives as intelligent services and applications powered by AI have grown in popularity. AI techniques have been traditionally dependent on centralized data collection and processing, which could pose significant challenges in real-world IoT applications. Federated learning has emerged as a promising solution for IoT applications, where data privacy is of utmost importance. IoT-based federated learning has numerous application areas, including smart healthcare, smart transportation, smart city, and smart banking.

2.3 Categorization of Federated Learning

Federated learning is categorized based on the following factors: the distribution of data, models learning, and architecture [16].

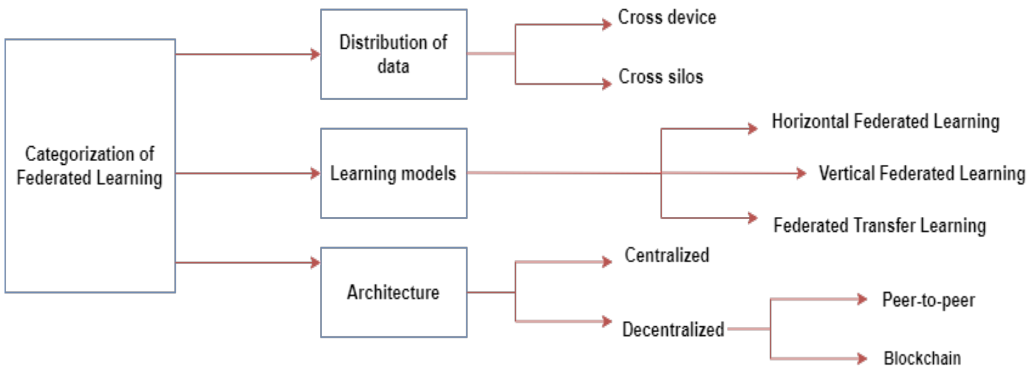

Categorization of Federated Learning
Distribution of data
Learning models
Architecture
Cross device
Cross silos
Horizontal Federated Learning
Vertical Federated Learning
Federated Transfer Learning
Centralized
Decentralized
Peer-to-peer
Blockchain

Figure 2.2: Categorization of Federated Learning

2.3.1 Distribution of Data

When training the models using federated learning, it is important to consider the distribution of data across the data sources. There are two main types of data distribution: cross-device and cross-silos.

In cross-device, the model is distributed among edge devices, such as smartphones or IoT devices, and is trained locally on each device using the data available on that device. The local models are then aggregated into a global model that represents the knowledge learned across all devices. This approach allows for privacy-preserving machine learning, as the data never leaves the device and is only used to update the local model. It is particularly useful in scenarios where data is generated at the edge, such as in healthcare or IoT applications.

On the other hand, cross-silos involve training a local model at data centers or silos and then aggregating those models into a global model at a centralized point. It is useful in scenarios where the data is soiled, and it is not possible to train the model on all the data sources due to regulatory or privacy concerns. It allows for collaboration across multiple organizations or data silos, while still maintaining the privacy and security of the data.

2.3.2 Learning Models

Another way to categorize federated learning is based on the learning model used. There are several types of FL learning models, including Horizontal Federated Learning, Vertical Federated Learning, and Federated Transfer Learning.

2.3.2.1 Horizontal Federated Learning

Horizontal federated learning is a type where an ML model is trained using data from multiple organizations or entities. In this category, the dataset shares the same features with different instances.

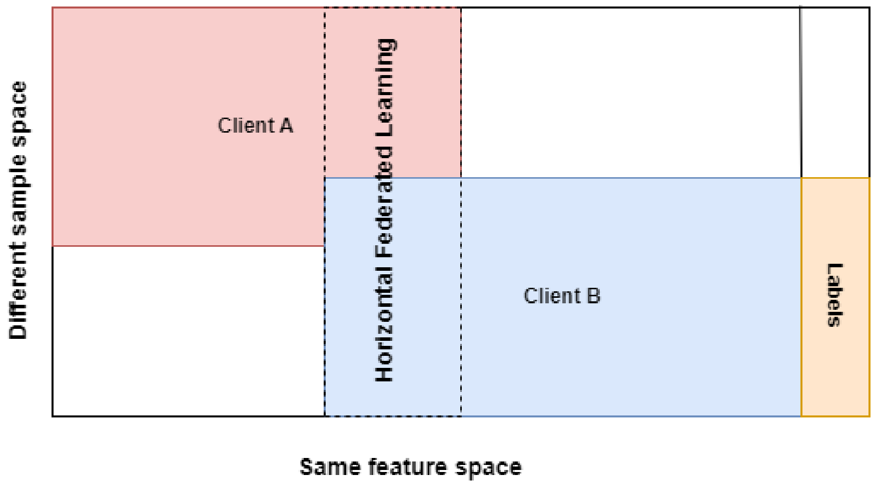


Figure 2.3: Horizontal Federated Learning

This scenario usually occurs in the same fields. Google's federated model solution for Android mobile phone is a kind of horizontal federated learning since the data used in this scenario has the same feature dimension [22]. For example, a healthcare organization intends to use a federated learning model to predict heart disease. To achieve this goal, the organization could collect data from multiple hospitals, each with patient data related to heart disease diagnosis. This data is considered a horizontal subset as it belongs to the same class (heart disease) but comes from different sources (hospitals). By pooling this data together, the organization can leverage the diversity of the dataset to train a more robust machine learning model. Another example of HFL would be two regional banks operating in different areas with different customer bases. The user group of these banks are largely unique to their respective areas, and the overlap between their customer is very minimal. However, despite catering to different customer segments, their core business functions and operations are quite similar. This means that the features and attributes used to describe and analyze their customers are essentially the same.

2.3.2.2 Vertical Federated Learning

Vertical federated learning is a type where different organizations or entities can collaborate to train a machine learning model while keeping their data private and secure. In this category, each party has a different feature space but the same sample space.

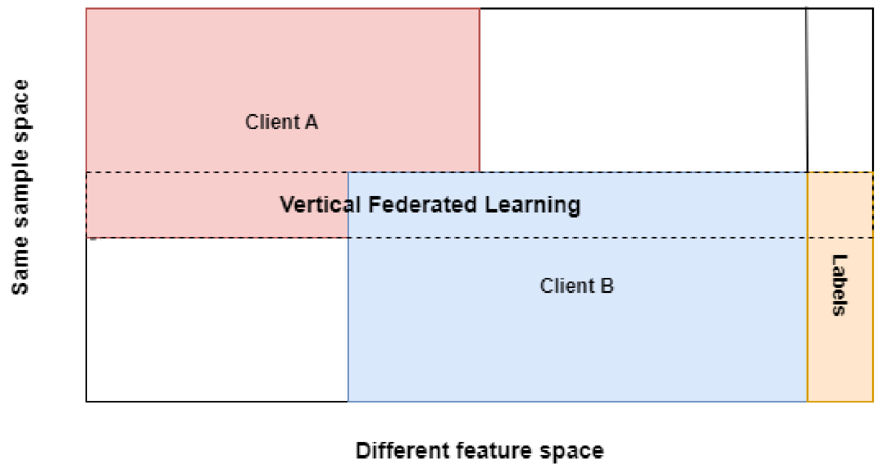


Figure 2.4: Vertical Federated Learning

This scenario usually occurs in different fields. The FATE (Federated AI Technology Enabler) project is an example of vertical federated learning. For example, let's imagine two distinct companies operating in the same city, a bank, and an e-commerce company. Given their presence in the area, it is highly probable that their user bases encompass a significant portion of the local residents. However, the types of data they collect from their users differ substantially. The bank primarily focuses on capturing information related to user financial activities, including revenue, expenditure behavior, and credit ratings. On the other hand, the e-commerce company primarily retains data associated with user browsing habits, purchase histories, and preferences. In the case of airlines and hotels, they possess distinct data belonging to the same customer such as flight details and lodging records. Hence, for vertical federated learning, it is crucial to synchronize the data samples and encrypt the model. Vertical federated learning during the training process ensures that the other participants are unaware of each other's data and attributes. This approach enables the global model to data insights from all the participants, eliminating the risk of model loss.

2.3.2.3 Federated Transfer Learning

Federated transfer learning is an example of federated learning which is applicable in a scenario where the dataset varies not only in feature space but also in sample space. For example, consider two institutions, a bank located in China and an e-commerce company based in the United States. Due to geographical limitations, the user based of these companies has minimal overlap. Additionally, their distinct business operations contribute to only a small portion of their feature spaces being shared. In such a scenario, transfer learning techniques can be employed to address the entire sample and feature space within a federated setting. These techniques involve learning a shared representation that bridges the gap between the two feature spaces, utilizing the limited set of common samples available. The objective is to establish a common understanding and representation that can be applied to make predictions for samples that possess features exclusively from one side of the institution. By learning a common representation, the shared knowledge from the limited common sample sets can be effectively utilized to enhance the predictive capabilities of both institutions. This approach allows for leveraging insights gained from the shared feature space to make predictions and draw conclusions for samples that possess features unique to one institution. FTL research is still in its early stages and has significant room for growth, particularly in its ability to handle diverse data structures. However, FTL is a successful method for removing data island boundaries while also securing user data security and privacy.

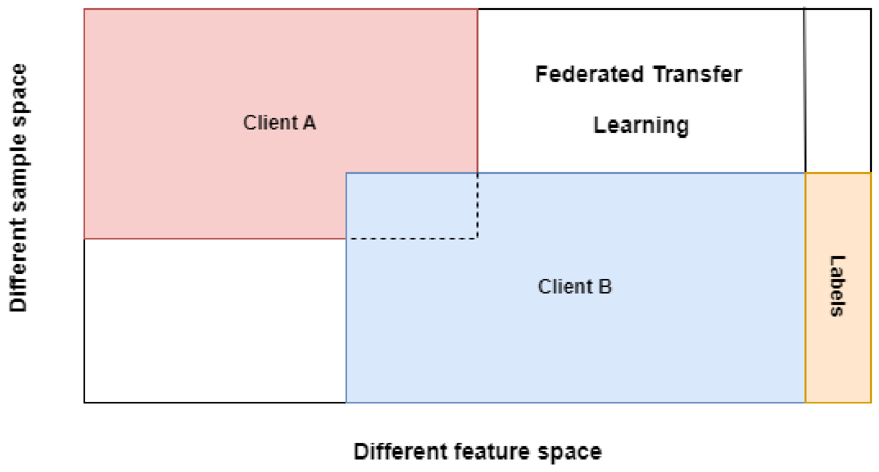


Figure 2.5: Transfer Federated Learning

2.3.3 Architecture

Federated learning allows the training of machine learning models across a distributed network of devices while preserving user data privacy. FL models can be trained in different modes, depending on the architectural design and interactions between the elements of the system. FL can be implemented using either centralized or decentralized approaches to train a model.

In a centralized approach, a central server collects and stores all user data and coordinates the training of the FL model. The central server is in charge of choosing the clients who will participate in training the model, combining the local models into a global model, and transmitting the global model to all the participating devices. For example, a centralized social media platform where all user data, posts, and interactions are stored on a single central server. Users log in to the platform through the server, and all communication between users, content moderation, and data storage is handled by this central server. The platform controls the algorithms that determine the content shown in users' feeds, manages privacy settings, and enforces community guidelines. Any updates, changes, or new features are implemented and deployed from this central server to all users.

In a decentralized approach, there is no central server. Data is distributed among multiple devices, and each device trains its own local model on its own local data. For model sharing and aggregating the local updates, peer-to-peer or blockchain can be used. For a peer-to-peer decentralized approach consider a P2P file sharing network. In this decentralized approach, users directly connect to each other's devices to share files. Each user has both the role of a consumer and a contributor. When a user downloads a file, they also become a source for others to download from. There is no central server; instead, the network relies on the collective resources of all participants. Users can search for and download files from multiple sources simultaneously, leading to efficient and distributed file sharing. For a blockchain decentralized approach consider the use of a blockchain-based cryptocurrency like Bitcoin. In this decentralized approach, transactions are verified and recorded in a distributed and immutable ledger known as the blockchain. Multiple participants, called nodes, contribute computing power to validate transactions using consensus mechanisms. Transactions do not rely on a central bank or authority; instead, they are verified by the decentralized network. This approach ensures transparency, security, and resistance to censorship or manipulation.
In this thesis, vertical federated learning is employed, and the following sections elaborate on its precise definition, its architectural framework, applications across various domains, and challenges.

2.4 Vertical Federated Learning

Existing studies primarily concentrate on horizontal federated learning (HFL), where participants have the same attribute space but different sample spaces. One instance of HFL is when a group of hospitals collaborates to develop an ML model that predicts health risks for their patients. This is achieved by leveraging agreed-upon data and sharing knowledge across the participating hospitals. However, HFL often faces limitations in practical scenarios, particularly when it comes to fostering collaboration among organizations that have competing interests. Business considerations make it highly unlikely for organizations to willingly cooperate with their competitors [6]. In contrast, vertical federated learning (VFL) is well-suited for situations in which organizations possess the same set of data samples but vary in terms of their feature space. VFL facilitates collaboration among organizations that do not compete with each other and possess vertically partitioned data. In such instances, it is common for one organization to have access to the ground truth or labels associated with certain features of a set of samples. The remaining participants contribute to the federation by providing additional feature information within the same sample space. However, they take measures to ensure that their data is not directly disclosed to other participants. In exchange for their contributions, these participants can receive compensation in the form of monetary or reputational rewards.

Let `D = (I; X; Y)` represent a comprehensive dataset comprising three components: `I`, `X`, and `Y`, which respectively represent the sample ID space, the feature space, and the label space. According to the definition provided in [33] VFL is performed on two distinct datasets: `D1 = (I1; X1; Y1)` and `D2 = (I2; X2; Y2)`. These datasets satisfy the conditions

$X1 \neq X2$, $Y1 \neq Y2$, and I1 = I2.

There exist two fundamental architectures for VFL: with a third party or without a third party.

2.4.1 VFL Architecture With A Third-party

[11] [33] proposed a specific architecture consisting of a trusted third party called the coordinator and two parties. The primary responsibilities of the coordinator were: computing the training loss and generating key pairs for homomorphic encryption to ensure privacy. Suppose that two clients, A and B, join forces to collaboratively train a machine learning model using their respective local datasets. Client A possesses the label data required for training the global model. However, since Client A and Client B are honest but curious about each other, ensuring data privacy is essential. To address this concern, a trustworthy third-party coordinator, C, is introduced into the process. This coordinator, such as governmental authorities, plays a pivotal role in overseeing and managing the FL procedure to safeguard data privacy. The VFL system can be broken down into five key steps:

1. The VFL system involves aligning the IDs between the datasets of Client A and Client B. Since the datasets may have different IDs, privacy-preserving techniques based on encryption such as private set intersection (PSI) [21], are employed to ensure the confidentiality of Client A and Client B's data. These techniques allow the identification of shared data instances between clients without compromising their private information, which is then used for training the VFL model.

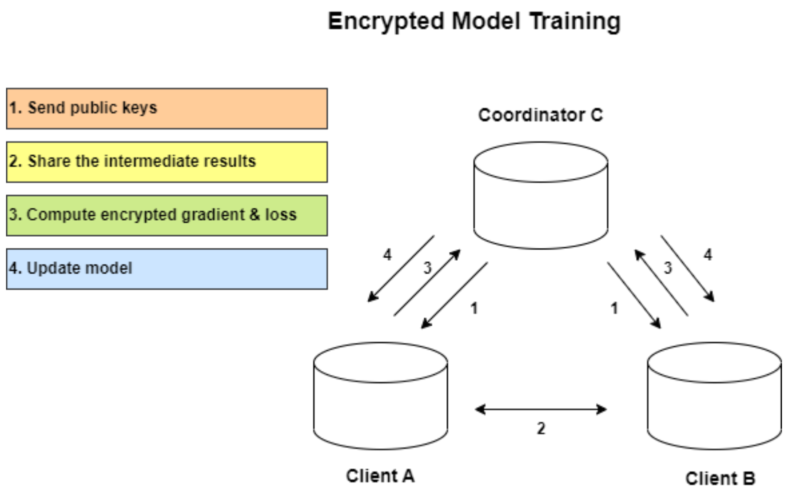


Figure 2.6: Vertical federated learning with coordinator

2. The Coordinator C creates an encryption key pair and shares the public key with Client A and Client B.

3. Both clients, A and B, employ encryption to secure their intermediate results and then proceed to exchange this encrypted information with one another.

4. Both clients, A and B individually compute encrypted gradients while incorporating a mask, and A additionally calculates an encrypted loss. Subsequently, the encrypted results, including gradients and the loss, are sent by both clients, A and B to C.

5. C decrypts the results received from Client A and Client B, allowing access to the decrypted gradients and loss, which are then sent back to both clients, A and B. Following this, Client A and Client B unmask the gradients and utilize them to update their respective model parameters.

2.4.2 VFL Architecture Without A Third-party

Involving the third party has significant drawbacks in terms of computational or communication burdens and potential data insecurity for the involved parties. Subsequent research studies, such as those conducted by [34] [13] [29], introduced a two-party architecture that effectively eliminated the requirement for a trusted coordinator. Suppose that two clients, A and B collaboratively train a machine learning model using their respective local datasets, and Client A possesses label data required for training the global model. Both clients A and B are trustworthy and are curious about each other's data. To safeguard against privacy breaches, the VFL system implements seven steps:

1. An ID alignment technique such as private set intersection (PSI) [21] is employed to verify the shared IDs between Client A and Client B. Subsequently, the data instances that are common to both datasets are utilized to train a VFL model. This approach ensures that only overlapping data is used for model training, promoting effective collaboration and privacy preservation.

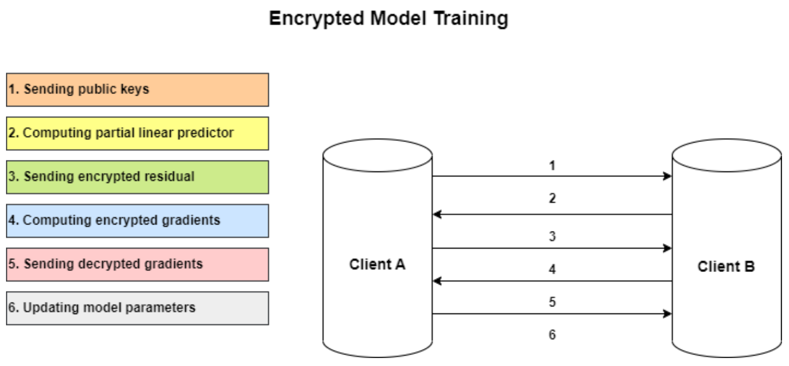

Encrypted Model Training
1. Sending public keys
2. Computing partial linear predictor
3. Sending encrypted residual
4. Computing encrypted gradients
5. Sending decrypted gradients
6. Updating model parameters
Client A
Client B
1
2
3
4
5
6

Figure 2.7: Vertical federated learning without coordinator

2. Client A creates an encryption key pair and shares the public key with Client B.

3. Both clients, A and B set up their model weights.

4. Client A and Client B individually compute their partial linear predictors, and afterward, Client B transmits its predictor's outcome to Client A.

5. After calculating the model residual, Client A encrypts it and subsequently shares it with Client B.

6. Upon computing the encrypted gradient, Client B forwards the masked gradient to Client A.

7. Once Client A decrypts the masked gradient, it returns the decrypted gradient to Client B. Subsequently, both Client A and Client B perform local model updates.

As a result, this architectural approach significantly reduced the system's complexity.

2.4.3 Application of Vertical Federated Learning

Vertical Federated Learning (VFL) has gained significant traction in both academia and industry due to its ability to facilitate data collaboration across industries and institutions. This has led to a surge in interest and exploration of VFL applications. Prominent examples of successful implementations include Fedlearner in ByteDance, Angel PowerFL in Tencent, FATE in Webank, Fedlearn in JD, and Paddle in Baidu. These companies have leveraged VFL's potential to enable efficient and secure collaborative learning, opening up new avenues for cross-institutional data analysis and knowledge sharing. In this section, an overview of some of the applications is explained below.

2.4.3.1 Healthcare

VFL in healthcare has the potential to revolutionize patient care and advance medical research significantly. For example, by securely combining and analyzing distributed healthcare data from various sources, such as hospitals, research institutions, and wearable devices. VFL empowers healthcare providers to leverage diverse patient data while prioritizing stringent privacy safeguards. [30] introduces vertical asynchronous federated learning (VAFL), a novel approach that validates its efficiency using the publicly available healthcare dataset MIMIC-III. [26] utilized VFL for cancer survival analysis, aiming to forecast the probability of patients surviving a certain duration after being diagnosed. [28] utilized VFL to establish a collaborative model between mobile network operators (MNOs) and healthcare providers (HP).

2.4.3.2 Finance

New approaches to VFL have seen significant development in finance, offering promising applications in this domain. For example, VFL in finance allows banks with different sets of customer data to collaboratively train an ML model for fraud detection while ensuring data privacy. By securely combining their unique features, the banks can improve the accuracy of the model without sharing sensitive information. [38] introduces a method for training traditional scorecard models using a gradient-based approach. [4] developed a secure algorithm for large-scale sparse logistic regression, which was then applied to enhance financial risk control. In [24], a combination of homomorphic encryption and coordination framework was utilized to construct a non-split vertical federated learning system that seamlessly integrates with other Bayesian systems for risk management purposes. [14] proposed an innovative adversarial domain adaptation algorithm explicitly designed for finance. The algorithm focuses on tackling the challenges of limited labeled data, commonly known as label deficiency issues in financial applications.

2.4.3.3 Recommendation Systems

Recommendation systems are extensively utilized in various applications, including online shopping and news reading, enhancing users' daily experiences. Recommendation systems aim to predict user interest in specific items by leveraging user and item-related data, as well as their historical behaviors. Increasing the volume of training data can improve the performance of these predictions. However, privacy concerns and data silo issues necessitate a viable solution, and VFL emerges as a promising approach to address these challenges. As an example, an online library service provider that possesses users' reading history can collaborate with an online movie streaming platform, which holds information about users' viewing history. By combining these datasets, valuable data features can be enriched, resulting in improved recommendation accuracy [32]. [28] proposed an innovative solution that leverages SecureBoost to augment recommendation precision when working with data from mobile network operators and healthcare providers. [3] developed a novel framework for social recommendations that utilize Multi-Party Computation (MPC) and follow the vertical federated approach.

2.4.3.4 Computer Vision and Natural Language Processing

VFL is also being actively investigated in the fields of computer vision and natural language processing. For example, in computer vision, organizations with diverse datasets can collaborate using VFL to collectively improve object detection models by securely exchanging gradient updates while preserving data privacy, enabling accurate detection of objects from various viewpoints. In natural language processing, VFL allows organizations to collaboratively train language models by leveraging the unique linguistic characteristics of separate datasets, enhancing the models' capabilities for tasks like sentiment analysis and language translation while maintaining data privacy. [18] introduced a VFL CNN algorithm designed for image classification, addressing the scenario where multiple parties possess incomplete portions of the same set of images. The VFL method developed demonstrated accuracy comparable to that of a centralized approach.

2.4.3.5 Telecommunication

In the field of telecommunication, traditional centralized learning methods necessitate the sharing of data from various sources, posing risks of privacy breaches for each party involved. As a result, VFL is being embraced to provide support for telecommunication in various domains, addressing the need for enhanced data privacy and security. [12] introduced a VFL algorithm specifically designed for collaborative model training in disaggregated networks. The algorithm is applied to classify the quality of transmission, addressing the need for efficient and secure data analysis in such networks. [37] proposed a low-latency VFL algorithm aimed at enhancing the accuracy of spectrum sensing in the domain of wireless communication. The algorithm was specifically designed to address the need for improved data analysis in this area while minimizing latency. [19] utilized VFL to effectively leverage multi-view observation data from distinct wireless sensing devices in the domain of motion recognition. The application of VFL enabled them to capitalize on diverse data sources and improve the accuracy of the motion recognition task.

2.4.4 Challenges of Vertical Federated Learning

This section offers valuable insights into the current approaches employed to enhance various aspects of VFL. It also identifies gaps in the existing literature, highlights unresolved challenges, and discusses recent advancements in the field [15]. Let's explore some of them.

2.4.4.1 Communication Efficiency

As the size of training data continues to grow across various platforms, the challenges posed by VFL become more pronounced. The total communication and computation cost in VFL scales proportionally with the data size, resulting in significant increases in local model computations, updates, and communication overhead. However, researchers have been working on addressing this issue by developing computation and communication efficient methods that aim to reduce complexity. Despite these efforts, the ever-increasing volume of data poses a persistent challenge for VFL, especially considering that the computing resources available to individual participants do not grow at the same rate as the data. The goal of achieving low communication rounds and minimizing computation costs remains an ongoing challenge in making VFL more efficient in this data-explosive environment.

2.4.4.2 Privacy and Security

Privacy and security have always been major concerns in FL since ensuring the confidentiality of raw data is crucial for data owners to participate in the collaborative model training process. Researchers have made significant efforts to identify potential privacy vulnerabilities and malicious attacks in FL setups. However, the focus on security aspects in VFL is relatively limited compared to traditional FL. Many privacy-preserving protocols, such as homomorphic encryption, secret sharing, and differential privacy, are commonly used in HFL scenarios. In contrast, their application and effectiveness in VFL settings have not been thoroughly explored. Additionally, there is a lack of research on defense mechanisms against poison attacks, where adversaries introduce malicious data during the training phase, and backdoor attacks, which involve inserting malicious functionality into targeted models through poisoned updates from malicious clients. Given these challenges, there is still ample room for improvement in addressing privacy and integrity leakage issues in VFL.

2.4.4.3 Feature Selection

For many decades, feature selection has been a subject of extensive research and application in various fields, such as text mining, image recognition, fault diagnosis, and intrusion detection.

VFL holds great promise in numerous feature selection applications due to its ability to maintain privacy while collaborating across different organizations. By combining feature selection with VFL, businesses can jointly conduct feature selection tasks without the risk of exposing their sensitive data. Although a few solutions for privacy-preserving feature selection in VFL have been presented, but this area remains largely unexplored. Therefore, there is a compelling opportunity for future research to focus on developing efficient and effective privacy-preserving feature selection protocols specifically tailored for VFL scenarios. This would advance the field and facilitate secure collaboration in feature selection across organizations.

2.4.4.4 Limited Training Data

As a critical preprocessing step in VFL, identifying overlapping samples among the clients is essential. However, in many cases, the number of overlapping samples may be insufficient to achieve optimal performance for VFL models. One possible solution is to expand the training data, but this approach must carefully address privacy concerns since sharing local raw data is not permissible. On the other hand, disregarding the non-overlapping samples among participants would mean wasting valuable data resources. Given the expensive and challenging nature of data acquisition, there is a need for innovative approaches that can infer relevant information from non-overlapping data as well. Such methods could help enhance the performance of VFL models without compromising data privacy, making more effective use of available data resources.

2.5 Challenges of Federated Learning

Since federated learning is a relatively new field, it faces several challenges that must be addressed. Let's explore some of them.

2.5.1 Expensive Communication

One major challenge faced by federated learning is the difficulty of reducing communication overhead during the process of training across multiple devices. To train a model on the data produced by devices in a federated network, it is crucial to use communication-efficient techniques that send messages or model updates during the training process instead of transmitting the entire dataset over the network. This is because federated networks include a vast number of devices and the communication within the network can be significantly lower than the local computation due to limited resources like bandwidth, energy, and power [31]. To minimize communication overhead in a federated network, the two aspects to consider are: reducing the number of communication rounds and minimizing the size of the messages that are transmitted during each round. By decreasing the number of communication rounds and sending smaller messages, the overall communication rounds, the overall communication cost can be significantly reduced, leading to more efficient federated learning.

2.5.2 System Heterogeneity

To effectively implement federated learning in networks with multiple devices, it is crucial to consider the varying storage, computational, and communication capabilities of each device. Such differences can arise due to hardware variations such as differences in CPU and memory, network connectivity issues such as type of network, and power-related limitations such as battery levels. To reduce the burden on federated networks, only a small proportion of devices are usually active at any given time. This is due to limitations in network size and systems-related constraints on each device, resulting in only a fraction of the devices being operational at once. For instance, in a network with millions of devices, only a few hundred devices may be active at any given time. In federated networks, active devices may drop out during an iteration due to connectivity or energy constraints, which is not uncommon. Such system-level characteristics can significantly worsen challenges such as fault tolerance and straggler mitigation. This can make it difficult to standardize the training process and ensure that the final model will work well across all participating devices.

2.5.3 Statistical Heterogeneity

In federated learning, each client device training is determined by its usage pattern. Therefore, the local data set of a single client is not expected to be indicative of the total data distribution across all clients. This means that the data distribution across the clients is likely to be highly non-identical, which violates the assumption of independent and identically distributed (i.i.d.) data in traditional machine learning settings. The non-i.i.d. data distribution in federated learning poses a challenge to developing effective and accurate models. This is because the models trained on one client data may not generalize well to other client data due to differences in the local data sets.

2.5.4 Privacy Risk

Privacy is a crucial aspect of federated learning since it involves the sharing of model updates rather than raw data. Although this method offers some level of data protection, privacy concerns remain. Transmitting model updates during training can still reveal sensitive information to the central server or a third party. New methods have been proposed to address this issue, such as secure multiparty computation and differential privacy. These methods can improve privacy but often come with a cost to model performance or system efficiency. Balancing these tradeoffs is a significant challenge in creating an effective and private federated system.

2.6 Summary

This chapter explores Federated Learning, discussing its applications in healthcare, natural language processing, computer vision, autonomous vehicles, and IoT. It categorizes Federated Learning based on data distribution, learning models, and architecture. The concept of Vertical Federated Learning is introduced, along with its architectures and applications, while also addressing the challenges it presents. Challenges of Federated Learning in general, such as communication costs, system and statistical heterogeneity, and privacy risks, are also discussed.

Chapter 3

Experimental and Methods

3.1 Introduction

Occupancy detection refers to the process of determining whether a particular room or building is currently occupied by individuals or not. It is an important aspect of smart buildings, home automation systems, energy management, and security applications. By accurately detecting occupancy, various systems can be efficiently controlled and optimized based on real-time occupancy information.
There are several benefits to occupancy detection. Firstly, it enables energy conservation by allowing automated systems to adjust heating, cooling, and lighting based on occupancy patterns. For instance, if a room is unoccupied, the system can automatically turn off the lights and adjust the temperature to save energy. This can significantly reduce electricity consumption and lower utility costs. Secondly, occupancy detection is crucial for enhancing security. By knowing whether a room or building is occupied, security systems can be activated or adjusted accordingly. For example, in a smart home, if an unoccupied room is detected to be occupied, the security system can send an alert to the homeowner, or cameras can be activated to monitor the situation. Thirdly, occupancy detection enables organizations to optimize space utilization by analyzing occupancy data. For instance, in a large office building, an occupancy detection system tracks real-time occupancy in different areas. Through data analysis, the organization discovers that certain meeting rooms are consistently underutilized while individual offices and collaborative spaces are in high demand. Based on this information, they can convert underutilized meeting rooms into flexible workspaces or implement shared desks, ensuring workstations are occupied when needed. This results in cost savings, increased productivity, and a more balanced work environment. Fourthly, occupancy detection systems play a vital role in ensuring safety and effective emergency response within buildings. For example, in a commercial high-rise building equipped with occupancy detection sensors, a fire alarm is triggered. The occupancy detection system immediately identifies the areas where occupants are present based on real-time data. This information is quickly relayed to emergency responders, enabling them to prioritize their response and evacuate the affected areas efficiently. Additionally, the system can provide accurate occupancy information to guide individuals to the nearest safe exit routes, minimizing confusion and ensuring the safety of everyone inside the building. The timely and precise data provided by occupancy detection systems enhances emergency preparedness, accel-

erates response times, and maximizes the chances of a successful evacuation in critical situations. Lastly, occupancy detection systems offer significant cost-saving benefits across various areas of operation. By optimizing energy consumption, these systems help reduce utility costs by automatically adjusting lighting, heating, and cooling systems based on real-time occupancy data. These savings can have a significant positive impact on operational budgets and financial sustainability. By harnessing the power of occupancy detection, organizations can create smarter, more efficient, and safer environments, benefiting both occupants and the environment.
Several types of sensors can be utilized for occupancy detection.

1. Passive Infrared (PIP) Sensors: These sensors detect the presence of humans by measuring the infrared radiation emitted by their bodies. PIR sensors are commonly used in lighting control systems and security applications. They are cost-effective and work well in detecting occupancy in a defined area.

2. Ultrasonic Sensors: Ultrasonic sensors emit high-frequency sound waves and measure the time it takes for the sound waves to bounce back after hitting objects in a room. By analyzing the reflected sound waves, occupancy can be determined. These sensors are effective in detecting occupancy in large areas or rooms with obstacles.

3. Microwave Sensors: Microwave sensors emit low-power microwaves and detect changes in the reflected waves caused by moving objects, including humans. They are capable of detecting occupancy through walls and are often used in security systems.

4. Image-based Sensors: These sensors use cameras or image sensors to capture visual information and analyze it to detect human presence. By employing computer vision algorithms, these sensors can identify human shapes or movement patterns. Image-based sensors are commonly used in surveillance systems and advanced occupancy detection applications.

5. Environmental Sensors: These sensors monitor various environmental factors such as temperature, humidity, air quality, and light levels. By integrating environmental sensors with occupancy detection systems, a more comprehensive understanding of the overall conditions within a space can be obtained. This information can subsequently be harnessed to enhance energy efficiency, enhance occupant comfort, and improve the overall performance of the building.

3.2 Dataset

The Occupancy Detection dataset has been referred for analysis from the UCI Machine Learning Repository [2]. The dataset consists of 20,560 instances and includes 7 attributes. The dataset is divided into a training set with 8143 instances and two test sets with instances 2665 and 9752. The attributes are

1. Date

2. Temperature in Celsius

3. Relative Humidity in
4. Light in Lux
5. CO2 in ppm
6. Humidity Ratio, Derived quantity from temperature and relative humidity, in kg water-vapor per kg-air
7. Occupancy, 0 or 1, 0 for not occupied, 1 for occupied status

3.3 Methodology

3.3.1 Implementing Occupancy Detection Data Analysis on Google Colab

Google Colab is a widely used cloud-based platform offered by Google. It empowers users to create and run Python code within a Jupyter Notebook interface, while also granting complimentary access to robust GPUs and TPUs, making it suitable for various machine learning and data analysis tasks. Utilizing Google Colab offers several benefits to students engaged in data science and machine learning projects:

1. A free Jupyter Notebook environment accessible for applications requiring data science and ML.
2. Utilization of a cloud-based platform reduces the necessity for students to possess extensive computing resources.
3. Data processing and analysis skills with user-friendly interface access to strong computational tools, such as GPUs, to facilitate intricate data analysis.
4. ML and data analysis integration with well-known Python libraries and tools.
5. An notable capability of Google Colab is its seamless connection with Google Drive, enabling direct access and effortless saving of Colab notebooks into your Google Drive storage.

```
from google.colab import drive
drive.mount('/content/drive')
```

3.3.2 Import Libraries

The libraries utilized in the code are extensively employed across data science and machine learning domains to perform various tasks, including data manipulation, numerical computations, development and training of machine learning models, data visualization, and more.

1. **NumPy**: NumPy is a highly efficient Python library designed for numerical computing. It offers optimized data structures and a comprehensive set of mathematical functions that enable seamless handling of large arrays and matrices.

2. **Pandas**: Pandas is a versatile Python library specifically designed for data manipulation and analysis. It offers convenient data structures, such as DataFrames, that are particularly well-suited for efficiently working with structured data.

3. **Matplotlib**: Matplotlib is a Python library primarily used for plotting and visualization purposes. It offers an extensive collection of functions that enable the creation of diverse visual representations, including line plots, bar plots, histograms, and many other types of visualization.

4. **Seaborn**: Seaborn, a data visualization library, extends the capabilities of Matplotlib and provides a user-friendly interface for crafting visually engaging statistical graphics.

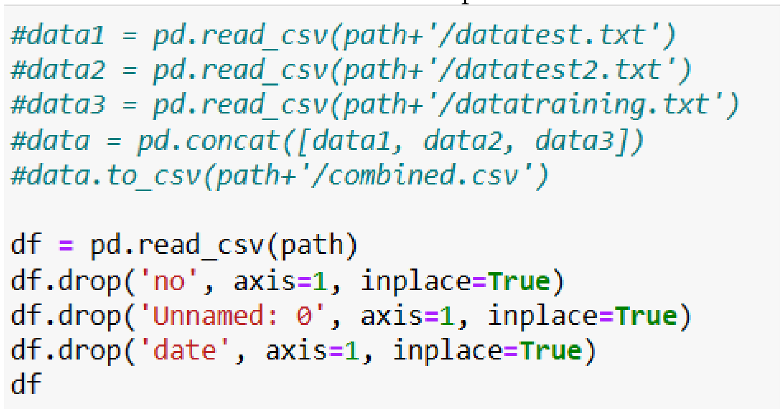
5. **TensorFlow**: TensorFlow is a widely-used open-source framework for machine learning that offers a versatile environment for constructing and deploying various types of machine learning models, including those based on deep learning techniques.

6. **Keras**: Keras, an API for neural networks, functions at an elevated level of abstraction and is constructed upon the framework of TensorFlow. It delivers an intuitive interface that streamlines the steps of constructing, training, and deploying deep learning models.

7. **Scikit-learn**: Scikit-learn stands out as a widely used machine learning library, offering a range of capabilities for tasks such as data preprocessing, model selection, and assessment. In this library, the `train_test_split`, found in the `model_selection` module, is employed to divide datasets into training and testing subsets. Additionally, for feature scaling, the `StandardScalar` from the `preprocessing` module is utilized.

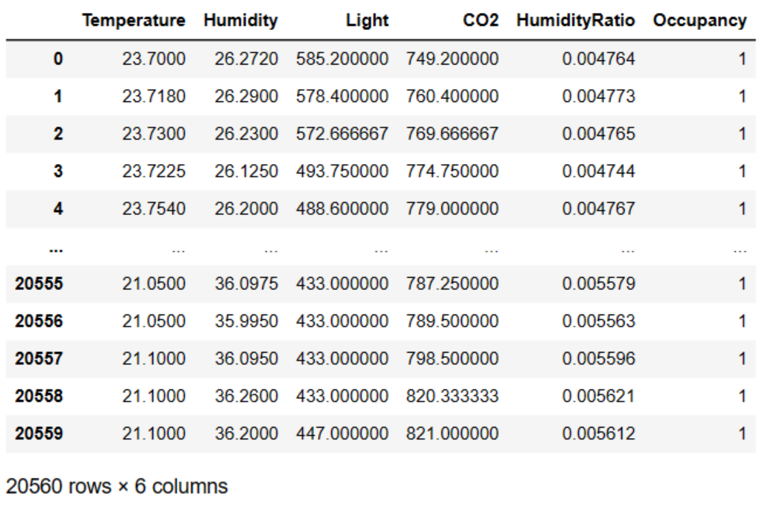
3.3.3 Data Analysis

The dataset contains one main folder which contains three txt files. The code starts by reading three separate CSV files named **datatest.txt**, **datatest2.txt**, and **datatraining.txt** located at the specified `path`. Each file's data is read into separate DataFrames: `data1`, `data2`, `data3`.



```
#data1 = pd.read_csv(path+'/datatest.txt')
#data2 = pd.read_csv(path+'/datatest2.txt')
#data3 = pd.read_csv(path+'/datatraining.txt')
#data = pd.concat([data1, data2, data3])
#data.to_csv(path+'/combined.csv')

df = pd.read_csv(path)
df.drop('no', axis=1, inplace=True)
df.drop('Unnamed: 0', axis=1, inplace=True)
df.drop('date', axis=1, inplace=True)
df
```

Next, the code combines these three DataFrames using the `pd.concat()` function, which concatenates the DataFrames based on their row indices. The resulting merged DataFrame is assigned to the variable `df`. This operation combines the data from the three DataFrames into a single DataFrame. Afterward, the merged DataFrame `df` is saved to a new CSV file named **combined.csv** at the specified path using the `to_csv()` function. Following that, the code reads the "combined.csv" file into a new DataFrame `df` using the `pd.read_csv` function. The data from the CSV file is loaded into the DataFrame, assuming the `path` variable contains the correct path to the "combined.csv" file. Finally, the code proceeds to drop three columns from the DataFrame `df` using the `drop()` function. The columns **no**, **Unnamed: 0**, and **date** are removed from `df` by specifying their names and setting `axis=1` to indicate that columns should be dropped. The `inplace=True` parameter ensures that the changes are applied directly to the DataFrame `df`, modifying it in place and then the final DataFrame is printed.

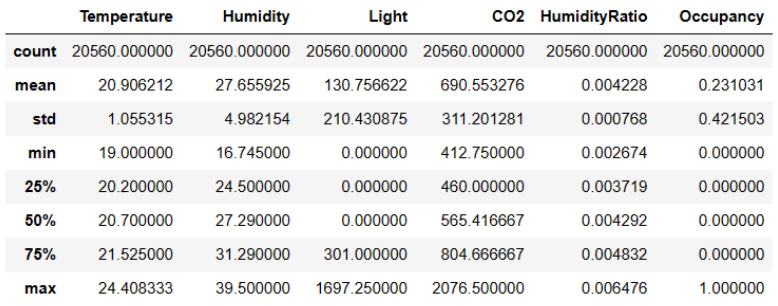


	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
0	23.7000	26.2720	585.200000	749.200000	0.004764	1
1	23.7180	26.2900	578.400000	760.400000	0.004773	1
2	23.7300	26.2300	572.666667	769.666667	0.004765	1
3	23.7225	26.1250	493.750000	774.750000	0.004744	1
4	23.7540	26.2000	488.600000	779.000000	0.004767	1
...	...	...	...	...	...	...
20555	21.0500	36.0975	433.000000	787.250000	0.005579	1
20556	21.0500	35.9950	433.000000	789.500000	0.005563	1
20557	21.1000	36.0950	433.000000	798.500000	0.005596	1
20558	21.1000	36.2600	433.000000	820.333333	0.005621	1
20559	21.1000	36.2000	447.000000	821.000000	0.005612	1

20560 rows × 6 columns

3.3.3.1 df.describe()

Following that, the `df.describe()` function in pandas serves the purpose of generating comprehensive statistical insights regarding a DataFrame or Series. It furnishes a synopsis encompassing fundamental statistical aspects such as the dataset's centrality, dispersion, and distribution shape for numerical data. The produced output comprises the count, indicating the quantity of non-missing values; the mean, which signifies the data's average; the standard deviation, a metric for data dispersion; the minimum and maximum data values; and quartiles, which partition the data into four equivalent segments. Additionally, it provides information about the data distribution, such as the 25th, 50th, and 75th percentiles. The `describe()` function is a useful tool to quickly assess the overall characteristics and basic statistics of a dataset, allowing users to gain insights into the data's range and distribution. Figure 3.1 shows the statistics of the attributes.



	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
count	20560.000000	20560.000000	20560.000000	20560.000000	20560.000000	20560.000000
mean	20.906212	27.655925	130.756622	690.553276	0.004228	0.231031
std	1.055315	4.982154	210.430875	311.201281	0.000768	0.421503
min	19.000000	16.745000	0.000000	412.750000	0.002674	0.000000
25%	20.200000	24.500000	0.000000	460.000000	0.003719	0.000000
50%	20.700000	27.290000	0.000000	565.416667	0.004292	0.000000
75%	21.525000	31.290000	301.000000	804.666667	0.004832	0.000000
max	24.408333	39.500000	1697.250000	2076.500000	0.006476	1.000000

Figure 3.1: Attributes statistics

3.3.3.2 Distribution of Target Variable

The Seaborn library to utilized to create a bar plot showing the count of each unique value in the 'Occupancy' column of the DataFrame `df` with the y-axis representing the count and the x-axis representing the unique values.

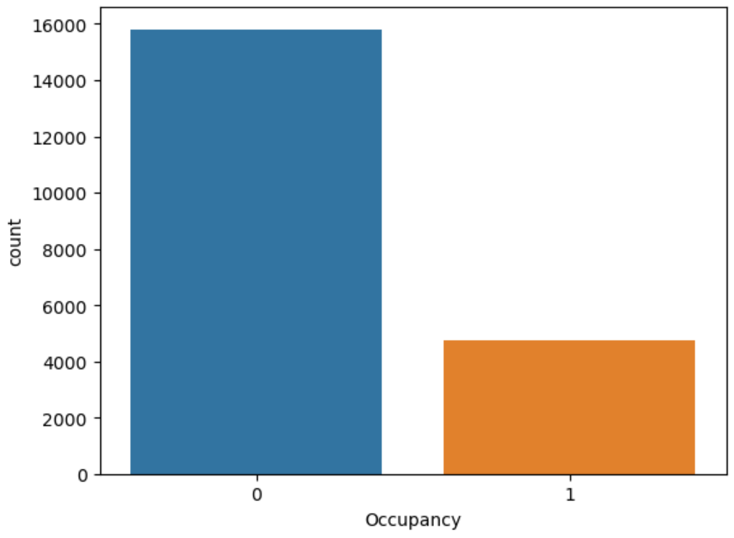
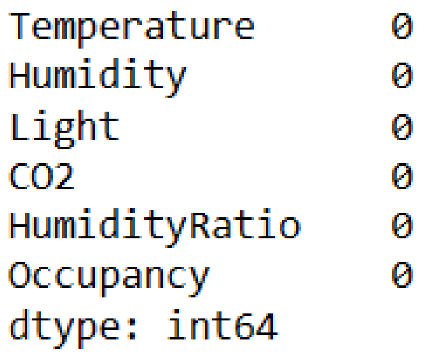


Figure 3.2: Count of each value

3.3.4 Data Preprocessing

3.3.4.1 Check Missing Values

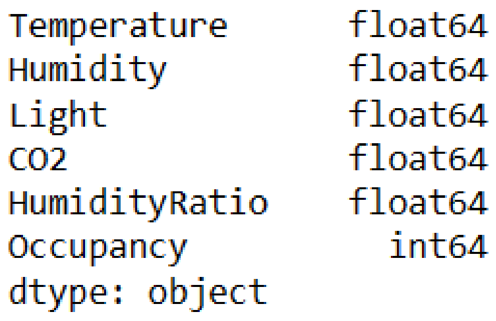
The code checks for missing values in a Pandas DataFrame using the `isna()` and `sum()` functions. The `isna()` function returns a DataFrame that marks missing values(NaN) as True and non-missing values(NaN) as False. The `sum()` function then adds up the number of True values in each column to give a total of the missing values. As you can see from the output below, there are no missing values in the dataset.



```
Temperature      0
Humidity         0
Light            0
CO2              0
HumidityRatio    0
Occupancy        0
dtype: int64
```

3.3.4.2 Check Data Type

The code checks the data type using the `dtypes` function in pandas and returns the data types of each column in a DataFrame, providing information about the type of data stored in each column.



```
Temperature      float64
Humidity         float64
Light            float64
CO2              float64
HumidityRatio    float64
Occupancy          int64
dtype: object
```

3.3.4.3 Split features and target

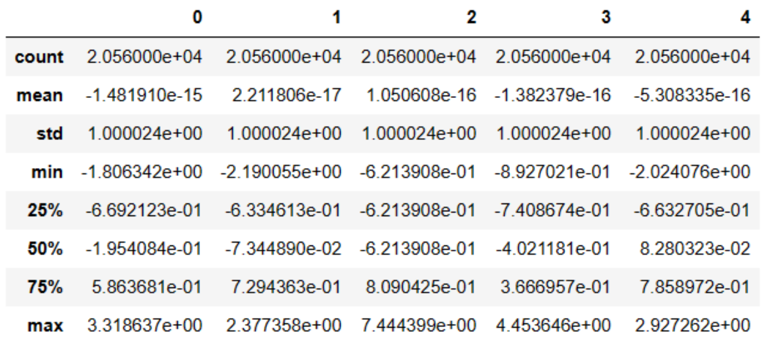
In this step, we're dividing the dataset into two separate parts: one containing the input variables (features) that serve as inputs to the model, and another containing the output variable (target) that the model aims to predict based on those features. This division allows us to train the model on the features and evaluate its performance in predicting the target variable.

3.3.4.4 Normalization

The code utilized the `StandardScaler` which is a preprocessing technique from `scikit-learn` library that is being applied to the feature dataset. The `StandardScaler` is fitted on the data and then transforms the values of the feature dataset such that they are standardized, i.e., rescaled to have zero mean and unit variance. This normalization process helps in improving the performance and stability of certain machine learning algorithms that are sensitive to the scale of the input features. Figure 3.3 shows the scaled dataset.

3.3.5 Splitting the Dataset into Training and Testing Sets

The `train_test_split` function is used to split the feature dataset and the target dataset into separate training and testing sets. 20% of the data is allocated for testing, while the remaining



	0	1	2	3	4
count	2.056000e+04	2.056000e+04	2.056000e+04	2.056000e+04	2.056000e+04
mean	-1.481910e-15	2.211806e-17	1.050608e-16	-1.382379e-16	-5.308335e-16
std	1.000024e+00	1.000024e+00	1.000024e+00	1.000024e+00	1.000024e+00
min	-1.806342e+00	-2.190055e+00	-6.213908e-01	-8.927021e-01	-2.024076e+00
25%	-6.692123e-01	-6.334613e-01	-6.213908e-01	-7.408674e-01	-6.632705e-01
50%	-1.954084e-01	-7.344890e-02	-6.213908e-01	-4.021181e-01	8.280323e-02
75%	5.863681e-01	7.294363e-01	8.090425e-01	3.666957e-01	7.858972e-01
max	3.318637e+00	2.377358e+00	7.444399e+00	4.453646e+00	2.927262e+00

Figure 3.3: Scaled dataset

80% is used for training. This division allows us to evaluate the performance of a machine learning model on unseen data by training it on the training set (`X_train` and `y_train`) and then assessing its predictions on the testing set (`X_test` and `y_test`).

3.3.6 Classifiers

This thesis examines six classifiers for the analysis.

3.3.6.1 K-Nearest Neighbours (KNN)

KNN is a supervised Machine Learning algorithm capable of tackling both classification and regression problems. It operates on the principle that items sharing similarities tend to be located near each other.

3.3.6.2 Random Forest (RF)

RF is an ensemble learning technique that falls under the category of machine learning algorithms. It involves generating numerous decision trees during the training phase, incorporating randomness in the construction process. The final output is determined by taking the average prediction from all the individual trees or identifying the class that represents the mode. By doing so, the algorithm prevents overfitting or simply memorizing the training data.

3.3.6.3 Gaussian Naive Bayes (GNB)

A probabilistic algorithm called GNB uses strict independence assumptions between features to create classifications based on the Bayes theorem.

3.3.6.4 Support Vector Machines (SVM)

SVM, a classification algorithm, utilizes a hyperplane to create a separation between two classes of data. Once trained, the SVM model can effectively classify unseen data.

3.3.6.5 Logistic Regression (LR)

LR is a classification method that represents the logit function as a linear combination of features.

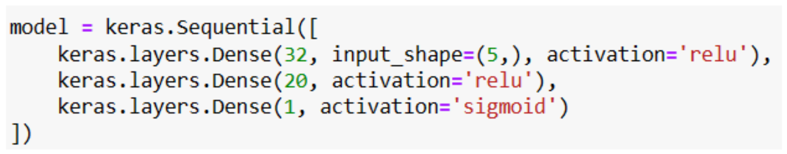
3.3.6.6 Linear Discriminant Analysis (LDA)

LDA is a linear algorithm that uses Bayes' theorem and assumes normal distributions for the classes and equal covariance for each class.

3.3.7 Deep Learning Model Using Neural Network

3.3.7.1 Model Definition

The code snippet defines a neural network model using the Keras library. The provided code defines a basic neural network architecture with three layers using the `keras Sequential` class.



```
model = keras.Sequential([
    keras.layers.Dense(32, input_shape=(5,), activation='relu'),
    keras.layers.Dense(20, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

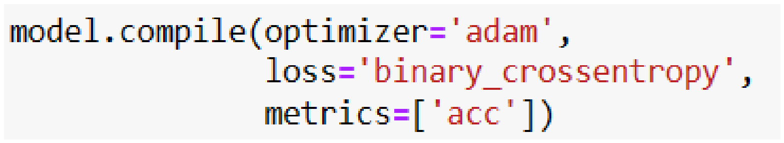
1. The first layer is the input layer. The dense layer is a fully interconnected layer, wherein each neuron establishes connections with all the neurons in the preceding layer. The layer has 32 units/neurons, which means it will output a vector of length 32. The `input_shape` parameter is set to (5,), which means the input to this layer should have 5 features. Within this layer, the activation function employed is the rectified linear unit (ReLU), denoted by `activation='relu'`. The `ReLU` function introduces non-linearity to the model by converting negative values to zero while preserving positive values unchanged.

2. The second layer is the hidden layer. It has 20 units/neurons and applies the `ReLU` activation function. This layer accepts the output from the preceding layer as input and generates a vector with a dimensionality of 20.

3. The third layer is the output layer. It has 1 unit/neuron, making it suitable for binary classification problems. The activation function used in this layer is the `sigmoid` function, specified by `activation='sigmoid'`. The `sigmoid` function squashes the output between 0 and 1, allowing the model to output probabilities for the positive class.

3.3.7.2 Model Compilation

Once the model is defined, the next step involves compiling it using the `model.compile()` function. This step is crucial as it configures the model for the training phase, and it accepts various arguments to define the optimizer, loss function, and metrics to be applied during training.



```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['acc'])
```

1. **Optimizer:** The optimizer determines how the model will update its parameters during training to minimize the loss function. In this case, the `Adam` optimizer is used, which is a popular and effective gradient-based optimization algorithm.

2. **Loss:** The loss function quantifies the disparity between the model's predicted results and the actual labels. In binary classification scenarios, it's common to utilize binary cross-entropy loss. It is suitable when the model is trying to estimate probabilities for two mutually exclusive classes.

3. **Metrics:** The metrics argument is a list of evaluation metrics that you want to monitor during training and evaluation. In the code, accuracy is used as the metric.

3.3.7.3 Model Training

The `model.fit()` function is used to train the model. The parameters passed to the fit method include the training data `(X_train, y_train`, the number of `epochs` (10), `batch_size` (32), `validation_split` this parameter specifies the fraction of the training data to be used for validation. In this case, 20% of the training data will be used for validation during training, and `validation_data=(X_train, y_train)` the same training data is used for validation.

3.3.7.4 Training and Validation Loss and Accuracy plots

Figure 3.4 shows the plot of training and validation loss and figure 3.5 shows the plot of training and validation accuracy of the neural network.

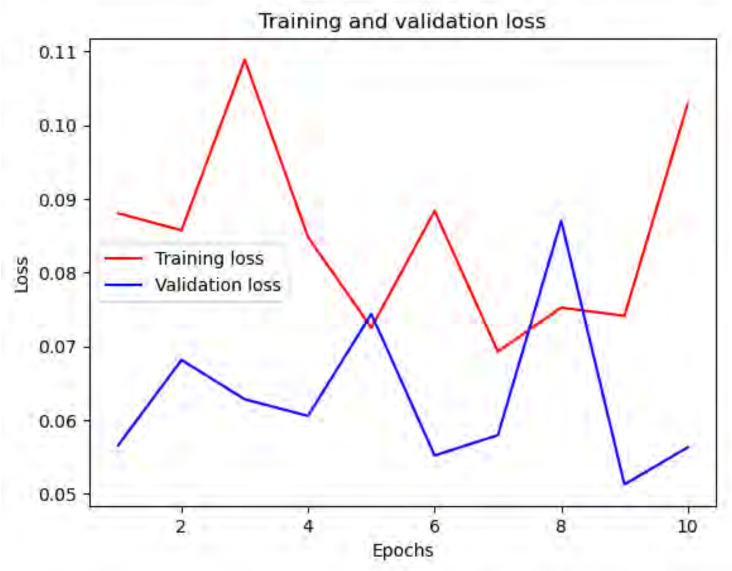


Figure 3.4: Plot of Training and Validation loss

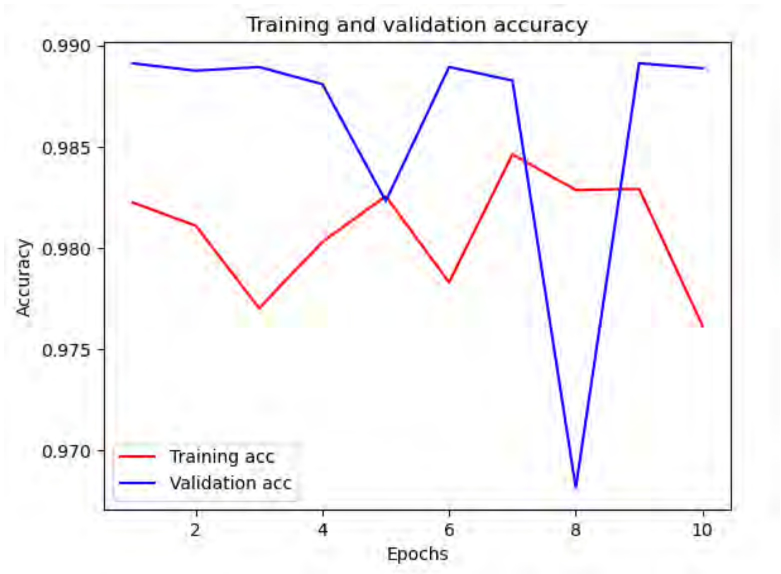


Figure 3.5: Plot of Training and Validation accuracy

3.3.8 Performance Metrics

To assess the effectiveness of different machine learning algorithms and neural network, four performance metrics were employed. The metrics include

Table 3.1: Performance Metrics of Classifiers

Classifier	**Accuracy**	**Precision**	**Recall**	**F1 Score**
KNN	0.9917	0.9918	0.9917	0.9917
RF	0.9927	0.9927	0.9927	0.9927
GNB	0.9683	0.9721	0.9683	0.9690
SVM	0.9895	0.9898	0.9895	0.9896
LR	0.9902	0.9905	0.9902	0.9903
LDA	0.9868	0.9875	0.9868	0.9869

3.3.8.1 Accuracy

The accuracy metric signifies the fraction of accurate predictions relative to the overall number of predictions conducted.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{3.1}$$

3.3.8.2 Precision

Precision is a measure that assesses the ratio of accurate positive predictions to the total number of positive predictions generated by the model.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3.2}$$

3.3.8.3 F1 score

The F1 score represents the harmonic mean of a model's precision and recall, measuring the model's accuracy on a specific dataset, specifically in the context of binary classification tasks.

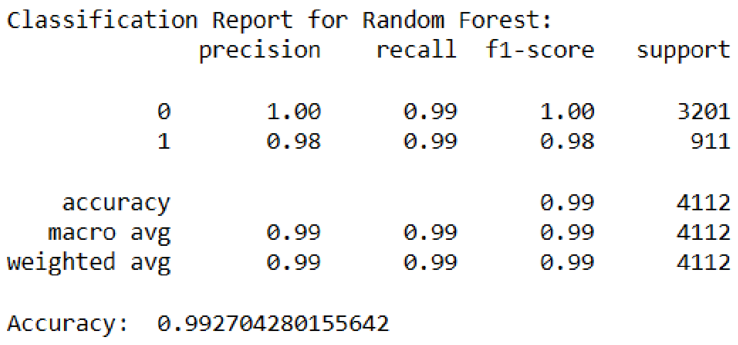
$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.3}$$

3.3.8.4 Recall

Recall, which can also be referred to as sensitivity or the true positive rate, measures the fraction of positive cases (Occupancy) that the model accurately detects.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

The table shows that the RF model achieved the highest accuracy of 99.27 %. The classification report of RF is obtained for further analysis.



```
Classification Report for Random Forest:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      3201
           1       0.98      0.99      0.98       911

    accuracy                           0.99      4112
   macro avg       0.99      0.99      0.99      4112
weighted avg       0.99      0.99      0.99      4112

Accuracy:  0.992704280155642
```

Figure 3.6: Classification report of RF

3.4 State of the Art Comparison

Table 3.2 shows the state-of-the-art comparison of occupancy detection compared to previous studies that had been published.

Reference	**Sensors**	**Techniques**	**Accuracy (%)**
[1]	Temperature sensor; sound press ure sensing, light sensor; humidity, pressure, temperature sensor; a PaPIR motion sensor	NN	94.6% and 91.5% for the binary and multi-class problems
[9]	CO2 sensors, PIR RH sensors, air speed sensors and globe thermometer	Machine Learning/An Autoregressive Hidden Markov Model (ARHMM)	the ARHMM technique has an average accuracy of Retracted 80.78%.
[7]	unprecedented carbon dioxide sensing system	SVM, ANN, and HMM algorithms	HMM showed the best result providing an accuracy of 73%.
[36]	BLEMS box - each sensor box hosts a number of sensors including a light sensor, a sound sensor, a motion sensor, a CO2 sensor, a temperature sensor, a relative humidity sensor, an infrared sensor, and a door switch sensor	six machine-learning techniques are evaluated in both single-occupancy and multi-occupancy offices.	Of the six, the decision-tree technique yielded the best overall accuracy (i.e. 96.0% to 98.2%)
This study	Temperature, Humidity, Light, CO2, HumidityRatio	RF, NN, vFedCCE	99.27%, 98.5%, 77%

Table 3.2: A Comprehensive Overview of Occupancy Detection

3.5 Summary

This chapter compares several categorization models on the occupancy detection dataset. The dataset utilized in the study is described in the chapter. The section also describes the technique used to compare the performance of various classifiers. The chapter finishes by giving the comparative analysis results, including accuracy and precision metrics, as well as discussing the state of art comparison with other papers.

Chapter 4

vFedCCE Algorithm

Traditional federated learning approaches typically involve sending data from clients to the server for aggregation and model updates, which can pose privacy concerns. The vFedCCE algorithm differs from traditional federated learning approaches, which is a vertical federated learning algorithm designed for classification problems with gradient-based optimization. It utilizes the categorical cross entropy (CCE) loss function to deploy the gradient-based optimizer on the clients instead of the centralized server, which helps to preserve privacy while training on a shared ID space. The algorithm involves clients performing local computations on their own data using the CCE loss function and then sharing the model updates with other clients. The use of a shared ID space allows for the alignment of datasets without revealing sensitive information. The vFedCCE algorithm reduces communication costs by minimizing the amount of data transmitted and can incorporate differential privacy techniques to further enhance privacy preservation during training. The algorithm vFedCCE is as follows:

Algorithm 1 :`vFedCCE`. m-class classification. Shared ID space I. Epoch number E. Batch size N. Batch examples indexed as $x_i^1, x_i^2, y_i,\ i \in \{1, \ldots, N\}$. w_1, w_2 are the client weight vectors. Client 2 stores $y_i \in \mathbb{R}^m$ as one hot arrays of length m.

```
server executes:
for each epoch $e = 1, 2, \ldots, E$ do
    $\mathcal{B} \leftarrow$ divide $I$ into batches of size $N$
    for batch in $\mathcal{B}$ do
        client 1 sends $a \leftarrow \begin{bmatrix} w_1x_1^1 & w_1x_2^1 & \ldots & w_1x_N^1 \end{bmatrix}^T$ and $\frac{\partial a_{ij}}{\partial w_1}$ to client 2
        server logs $\mathcal{L} \leftarrow$ client2.CalculateLossAndUpdate($a$)
        $g \leftarrow$ client2.AssembleGradient($\frac{\partial a_{ij}}{\partial w_1}$)
        client1.UpdateWithGradient($g$)
    end
end

CalculateLossAndUpdate($a$):   // client 2 executes
$b \leftarrow \begin{bmatrix} w_2x_1^2 & w_2x_2^2 & \ldots & w_2x_N^2 \end{bmatrix}^T$
shared model output $p \leftarrow (a+b)/2$
$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p_i) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m} y_{ij}\log(p_{ij})$
$c \leftarrow \begin{bmatrix} y_1 & y_2 & \ldots & y_N \end{bmatrix}^T \oslash p$
client2.UpdateWithGradient$\left(\frac{\partial}{\partial w_2}\left(-\frac{1}{N}\langle c, b\rangle_{\mathrm{F}}\right)\right)$
return $\mathcal{L}$

AssembleGradient($\frac{\partial a_{ij}}{\partial w_1}$):   // client 2 executes
return $-\frac{1}{N}\sum_{i=1}^{N}\sum_{i=1}^{m} c_{ij}\frac{\partial a_{ij}}{\partial w_1}$

UpdateWithGradient($g$):   // client 1 or 2 executes
update client weights via a specified optimizer with gradient $g$
```

4.1 Code Implementation

Using Google Colab for coding and machine learning tasks can bring several challenges for students. Limited resources like CPU and memory might hinder performance, while session timeouts can lead to lost work. Reliable internet connectivity is crucial, as disruptions can affect code execution and data access. Uploading and managing large datasets on Google Drive might prove cumbersome, and installing dependencies could lead to compatibility issues. Security concerns might arise when dealing with sensitive data, and the learning curve can be steep for newcomers. Additionally, occasional service unavailability and downtime might disrupt workflow. Google Colab initially served as the platform for the development and refinement of the research code. However, the above mentioned challenges necessitated the exploration of an alternative approach. Consequently, the codebase was migrated to a dedicated server, leveraging a system distinguished by its robust specifications. The server, identified as Device DTP-ELE-137, encompasses an Intel Xeon Gold 5220 CPU operating at a frequency of 2.20GHz with dual processors, complemented by a total of 16.0 GB of installed RAM. Additionally, the infrastructure is underpinned by a 64-bit operating system, housing an x64-based processor architecture. This strategic transition effectively mitigated the issue of session timeouts, thereby ensuring uninterrupted and extended periods of code execution. Importantly, this transition was seamlessly facilitated by utilizing Jupyter Notebook within the Anaconda Navigator, thereby augmenting operational efficiency and optimizing workflow management in alignment with the overarching objectives of the thesis.

4.1.1 Import Libraries

The libraries utilized in the code are

1. **Pandas**: Supplies Python with tools for both data manipulation and analysis, frequently employed when dealing with structured data like tabular datasets represented as DataFrames.
2. `pd.set_option('display.max_colwidth', None)`: Sets a pandas option to display the full contents of each column in a DataFrame, without truncating the data.
3. **Numpy**: It offers assistance for extensive, multi-dimensional arrays and matrices, in addition to a range of mathematical functions designed for manipulating these arrays.
4. `np.set_printoptions(precision=3, suppress=True)`: Sets the printing options for NumPy arrays. It configures the precision to 3 decimal places and suppresses the printing of small floating-point values in scientific notation.
5. **Seaborn**: Constructed atop matplotlib, it delivers a user-friendly interface for crafting informative and visually appealing statistical graphics and visualizations.
6. `sns.set(style='whitegrid')`: Sets the default style for seaborn plots to 'whitegrid', which displays a white background with gridlines.
7. **Matplotlib**: Offers a set of functions for crafting static, animated, and interactive visualizations within the Python programming language.
8. **Tensorflow**: Imports the TensorFlow library, which is an open-source machine learning framework created by Google. TensorFlow offers a range of tools for creating and training diverse machine learning models.
9. **Keras**: This is an advanced deep learning API that offers a streamlined interface for constructing and training neural networks.
10. `from keras import layers`: Contains a variety of pre-built layers that can be used to construct neural networks.

11. `from keras import regularizers`: Provides functions for applying regularization techniques to neural networks, such as L1 and L2 regularization.

12. `from sklearn.model_selection import train_test_split`: Imports the `train_test_split` function from the `model_selection` module in `scikitlearn`, a widely used Python machine learning library. This function is employed to divide a dataset into training and testing subsets.

13. `from keras.layers import preprocessing`: Provides functions for preprocessing input data, such as scaling or normalizing features.

14. `from keras.layers import Normalization`: This can be used to normalize the input data during training.

15. **Math**: Provides various mathematical operations and functions.

16. **uuid**: Provides functions for generating universally unique identifiers (UUIDs). UUIDs are commonly used to identify objects uniquely and unambiguously.

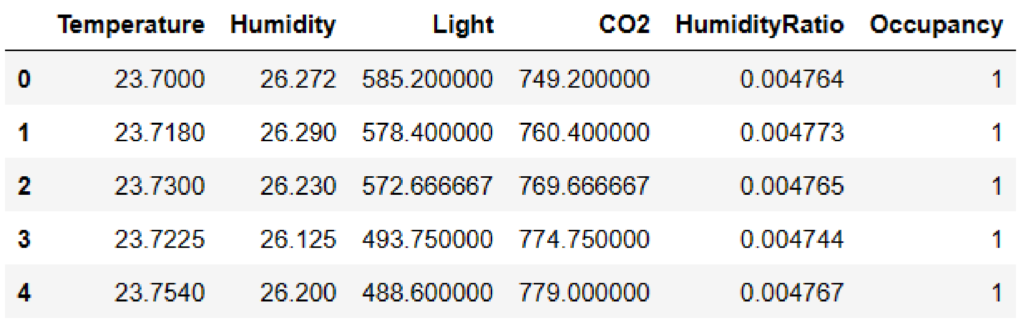
17. **Random**: Provides functions for generating random numbers, shuffling sequences, and making random choices.

4.1.2 Data Analysis

The dataset contains one main folder which contains three txt files. The code starts by reading three separate CSV files named **datatest.txt**, **datatest2.txt**, and **datatraining.txt** located at the specified `path`. Each file's data is read into separate DataFrames: `data1`, `data2`, `data3`. Next, the code combines these three DataFrames using the `pd.concat()` function, which concatenates the DataFrames based on their row indices. The resulting merged DataFrame is assigned to the variable `df`. This operation combines the data from the three DataFrames into a single DataFrame. Afterward, the merged DataFrame `df` is saved to a new CSV file named **combined.csv** at the specified path using the `to_csv()` function. Following that, the code reads the "combined.csv" file into a new DataFrame `df` using the `pd.read_csv` function. The data from the CSV file is loaded into the DataFrame, assuming the `path` variable contains the correct path to the "combined.csv" file. Finally, the code proceeds to drop three columns from the DataFrame `df` using the `drop()` function. The columns **no**, **Unnamed: 0**, and **date** are removed from `df` by specifying their names and setting `axis=1` to indicate that columns should be dropped. When the `inplace=True` parameter is used, it guarantees that the alterations are directly made to the DataFrame `df`, thereby modifying it on the spot, and the resulting DataFrame is subsequently displayed.

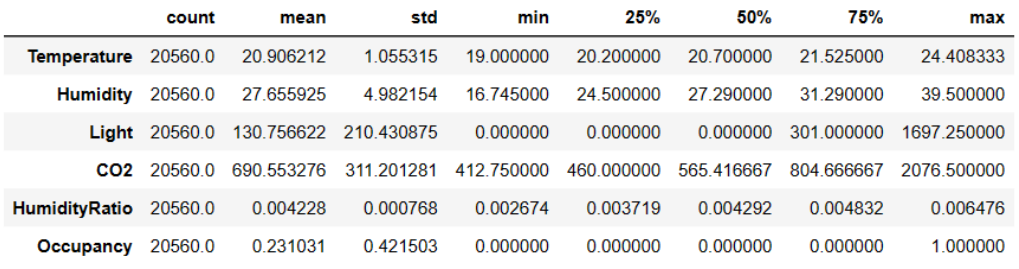
4.1.2.1 df.describe()

The `df.describe()` function in pandas is employed to produce statistical summaries for a DataFrame or Series. It offers an overview of the central characteristics, spread, and distribution shape of numerical data. The output includes count, which represents the number of



	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
0	23.7000	26.272	585.200000	749.200000	0.004764	1
1	23.7180	26.290	578.400000	760.400000	0.004773	1
2	23.7300	26.230	572.666667	769.666667	0.004765	1
3	23.7225	26.125	493.750000	774.750000	0.004744	1
4	23.7540	26.200	488.600000	779.000000	0.004767	1

non-missing values; mean, the average value of the data; standard deviation, which measures the spread of the data; minimum and maximum values; and quartiles, which divide the data into four equal parts. Additionally, it provides information about the data distribution, such as the 25th, 50th, and 75th percentiles. The `describe()` function is a useful tool to quickly assess the overall characteristics and basic statistics of a dataset, allowing users to gain insights into the data's range and distribution.



	count	mean	std	min	25%	50%	75%	max
Temperature	20560.0	20.906212	1.055315	19.000000	20.200000	20.700000	21.525000	24.408333
Humidity	20560.0	27.655925	4.982154	16.745000	24.500000	27.290000	31.290000	39.500000
Light	20560.0	130.756622	210.430875	0.000000	0.000000	0.000000	301.000000	1697.250000
CO2	20560.0	690.553276	311.201281	412.750000	460.000000	565.416667	804.666667	2076.500000
HumidityRatio	20560.0	0.004228	0.000768	0.002674	0.003719	0.004292	0.004832	0.006476
Occupancy	20560.0	0.231031	0.421503	0.000000	0.000000	0.000000	0.000000	1.000000

4.1.3 Data Shuffling and UUID Matching for Dataset Synchronization

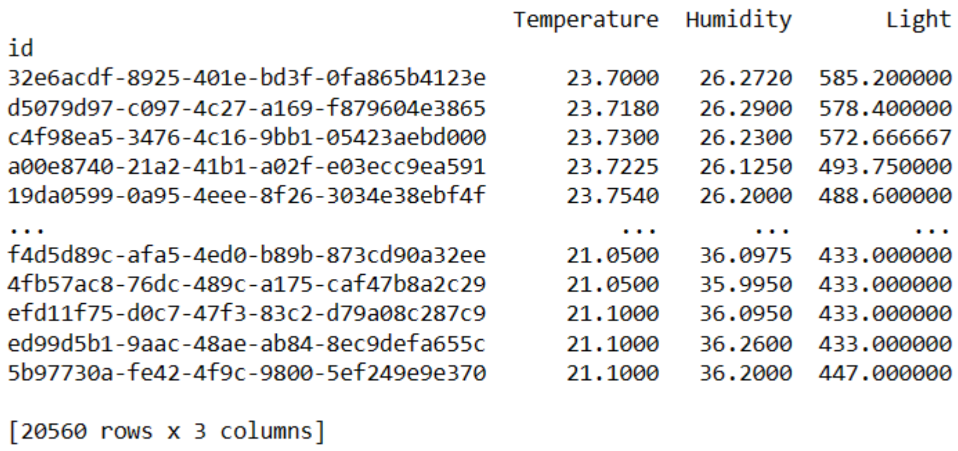
Assign a unique identifier (UUID) to each entry in the two datasets, and then randomize the order of the entries. This way, it will be possible to later identify matching entries by referring to their respective UUIDs.

To create unique identifiers for each entry in the dataset, a series of sequential numbers ranging from 0 to 20559 is generated. Using the `uuid.uuid4()` function, each number is transformed into a universally unique identifier (UUID) by converting it into a string format. The original DataFrame, df, is duplicated into a new DataFrame called `df_with_id`. An additional column named 'id' is added to `df_with_id`, containing the generated UUIDs. The 'id' column is then set as the index of the DataFrame.

The original `df` is vertically partitioned and `client1_data` and `client2_data`, are created from `df_with_id`. `client1_data` includes the features 'Temperature', 'Humidity', and 'Light', while `client2_data` includes the features 'CO2', 'HumidityRatio', and a label 'Occupancy'. These datasets are extracted from `df_with_id` based on the specified column names.

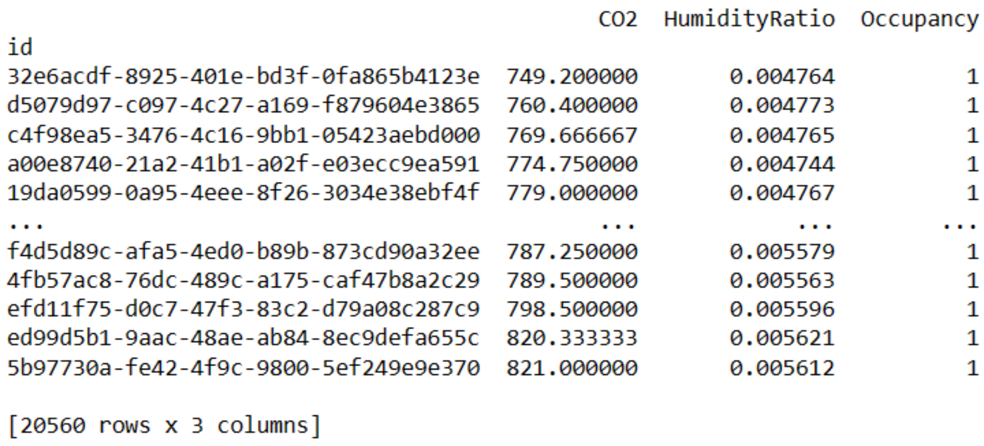
By associating the same UUIDs with corresponding entries in `client1_data` and `client2_data`, it becomes possible to establish connections or matches between the datasets using the unique identifiers.

The first output, `client1_data`, represents a DataFrame with three columns: 'Temperature', 'Humidity', and 'Light'. Each row in this DataFrame corresponds to a specific entry identified by its ID (indexed by the 'id' column). The values in these columns indicate the respective measurements of temperature, humidity, and light for each entry. For example, in the first row, we have a temperature of 23.7, humidity of 26.272, and light level of 585.2.



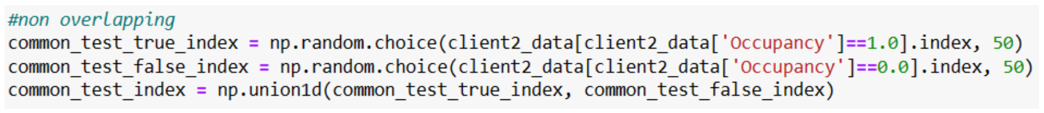
```
                                      Temperature  Humidity       Light
id
32e6acdf-8925-401e-bd3f-0fa865b4123e      23.7000   26.2720  585.200000
d5079d97-c097-4c27-a169-f879604e3865      23.7180   26.2900  578.400000
c4f98ea5-3476-4c16-9bb1-05423aebd000      23.7300   26.2300  572.666667
a00e8740-21a2-41b1-a02f-e03ecc9ea591      23.7225   26.1250  493.750000
19da0599-0a95-4eee-8f26-3034e38ebf4f      23.7540   26.2000  488.600000
...                                           ...       ...         ...
f4d5d89c-afa5-4ed0-b89b-873cd90a32ee      21.0500   36.0975  433.000000
4fb57ac8-76dc-489c-a175-caf47b8a2c29      21.0500   35.9950  433.000000
efd11f75-d0c7-47f3-83c2-d79a08c287c9      21.1000   36.0950  433.000000
ed99d5b1-9aac-48ae-ab84-8ec9defa655c      21.1000   36.2600  433.000000
5b97730a-fe42-4f9c-9800-5ef249e9e370      21.1000   36.2000  447.000000

[20560 rows x 3 columns]
```

The second output, `client2_data`, is also a DataFrame consisting of three columns: 'CO2', 'HumidityRatio', and 'Occupancy'. Similar to `client1_data`, each row in this DataFrame represents an entry identified by its ID. The columns contain measurements of CO2 levels, humidity ratio, and occupancy status for each corresponding entry. For instance, in the first row, we have a CO2 level of 749.2, humidity ratio of 0.004764, and an occupancy status of 1.



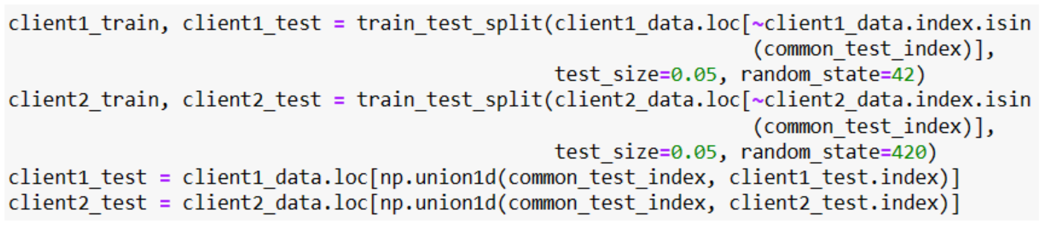
```
                                             CO2  HumidityRatio  Occupancy
id
32e6acdf-8925-401e-bd3f-0fa865b4123e  749.200000       0.004764          1
d5079d97-c097-4c27-a169-f879604e3865  760.400000       0.004773          1
c4f98ea5-3476-4c16-9bb1-05423aebd000  769.666667       0.004765          1
a00e8740-21a2-41b1-a02f-e03ecc9ea591  774.750000       0.004744          1
19da0599-0a95-4eee-8f26-3034e38ebf4f  779.000000       0.004767          1
...                                          ...            ...        ...
f4d5d89c-afa5-4ed0-b89b-873cd90a32ee  787.250000       0.005579          1
4fb57ac8-76dc-489c-a175-caf47b8a2c29  789.500000       0.005563          1
efd11f75-d0c7-47f3-83c2-d79a08c287c9  798.500000       0.005596          1
ed99d5b1-9aac-48ae-ab84-8ec9defa655c  820.333333       0.005621          1
5b97730a-fe42-4f9c-9800-5ef249e9e370  821.000000       0.005612          1

[20560 rows x 3 columns]
```

Both `client1_data` and `client2_data` have the same number of rows (20560), and their entries are aligned based on the shared ID values.



```
#non overlapping
common_test_true_index = np.random.choice(client2_data[client2_data['Occupancy']==1.0].index, 50)
common_test_false_index = np.random.choice(client2_data[client2_data['Occupancy']==0.0].index, 50)
common_test_index = np.union1d(common_test_true_index, common_test_false_index)
```

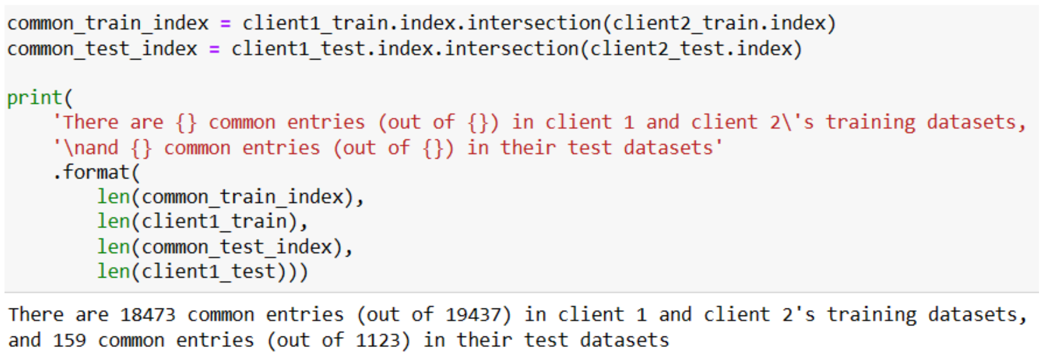
In the provided code, two sets of indices are randomly selected from the `client2_data` DataFrame. The first set, `common_test_true_index`, consists of 50 indices corresponding to rows where the ‘Occupancy’ column has a value of 1.0 (indicating occupancy is true). The second set, `common_test_false_index`, also contains 50 indices, but this time they correspond to rows where the ‘Occupancy’ column has a value of 0.0 (indicating occupancy is false). The `common_test_index` is then created by taking the union of these two sets of indices. This resulting index array, `common_test_index`, contains a mixture of indices representing both occupied and unoccupied instances from `client2_data`.



```
client1_train, client1_test = train_test_split(client1_data.loc[~client1_data.index.isin
                                                                (common_test_index)],
                                               test_size=0.05, random_state=42)
client2_train, client2_test = train_test_split(client2_data.loc[~client2_data.index.isin
                                                                (common_test_index)],
                                               test_size=0.05, random_state=420)
client1_test = client1_data.loc[np.union1d(common_test_index, client1_test.index)]
client2_test = client2_data.loc[np.union1d(common_test_index, client2_test.index)]
```

In this code snippet, the dataset for each of the two clients (client1 and client2) is being split into training and testing subsets. The training dataset is utilized to train the model, whereas the testing dataset is set aside for assessing the model’s performance. The split is done using the `train_test_split` function, where a small portion (5%) of each client’s data is set aside as the test set, ensuring randomness and reproducibility using specified random seeds (42 for client1 and 420 for client2). To ensure consistency in testing, data points that are common to both clients are identified using `common_test_index`. The testing subsets for both clients are then updated to include these common data points along with their respective individual test sets. This process ensures that both clients’ models are evaluated on a shared set of data points.

4.1.4 Train and Test Datasets Information



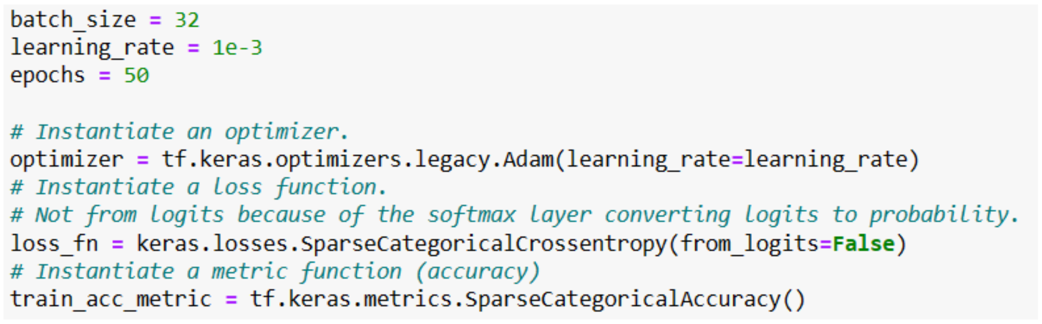
```
common_train_index = client1_train.index.intersection(client2_train.index)
common_test_index = client1_test.index.intersection(client2_test.index)

print(
    'There are {} common entries (out of {}) in client 1 and client 2\'s training datasets,
    '\nand {} common entries (out of {}) in their test datasets'
    .format(
        len(common_train_index),
        len(client1_train),
        len(common_test_index),
        len(client1_test)))
```

```
There are 18473 common entries (out of 19437) in client 1 and client 2's training datasets,
and 159 common entries (out of 1123) in their test datasets
```

The provided code calculates the common indices between the training and test datasets of two clients, referred to as “client 1” and “client 2”. In the code, `client1_train` and `client2_train` represent the training datasets of client 1 and client 2, respectively. Similarly, `client1_test` and `client2_test` represent their respective test datasets. The `intersection()` function is used to find the common indices between the two datasets. It returns a new index that consists of only the shared entries between the two datasets. These common indices indicate the entries that exist in both client 1 and client 2’s datasets. The code then prints the number of common entries found in the training datasets (`common_train_index`) and the number of common entries found in the test datasets (`common_test_index`). The lengths of these index arrays represent the number of shared entries between the respective datasets. The purpose of calculating these common indices is to determine the overlap between the data samples used for training and testing in client 1 and client 2. Having common entries ensures that the evaluation of models trained on these datasets can be compared and analyzed properly, as they are tested on the same set of samples.

4.1.5 Initialization of Training Components

As the dataset is analyzed and vertically partitioned into `client_1` and `client_2`, and a unique ID is assigned, it's now time to initialize the parameters for `VFedCCE`.



```
batch_size = 32
learning_rate = 1e-3
epochs = 50

# Instantiate an optimizer.
optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=learning_rate)
# Instantiate a loss function.
# Not from logits because of the softmax layer converting logits to probability.
loss_fn = keras.losses.SparseCategoricalCrossentropy(from_logits=False)
# Instantiate a metric function (accuracy)
train_acc_metric = tf.keras.metrics.SparseCategoricalAccuracy()
```

In the provided code, we see the initialization of various components necessary for training a model. The `batch_size` variable is set to 32, which determines the number of training examples processed in a single iteration. The `learning_rate` is set to 1e-3, determining the step size for the optimizer's updates during training. The `epochs` variable is set to 50, indicating the number of complete passes through the entire training dataset. An instance of the `Adam` optimizer is created with the specified learning rate. The role of the optimizer is to adjust the model's parameters by using the computed gradients, to enhance the model's performance. The `loss_fn` variable is initialized with the `SparseCategoricalCrossentropy` loss function. This loss function computes the variance between the model's predicted outcomes and the actual labels. In this case, the model's outputs are assumed to be probabilities after passing through a `softmax` layer. Additionally, the `train_acc_metric` variable is instantiated as the `SparseCategoricalAccuracy` metric. This metric is used to evaluate the accuracy of the model's predictions by comparing them to the true labels.

4.1.6 Function Descriptions for Plotting

For plot, `plot_loss(loss, accuracy)` function takes two lists, loss, and accuracy, as input. It uses Matplotlib to plot the loss and accuracy values over epochs. The loss values are plotted as a line with the label "loss," and the accuracy values are plotted as another line with the label "accuracy." The function also adds a legend and grid to the plot. `plot_accuracy(predictions, answers)` function calculates various evaluation metrics based on the predictions and true answers provided. It computes the true positives `(tp)`, true negatives `(tn)`, false positives `(fp)`, and false negatives `(fn)` by comparing the predictions to the answers. Then, it calculates metrics such as accuracy, precision, recall, specificity, and F-measure based on these counts. The function prints the values of these metrics. `convert_to_non_sparse(sparse)` function takes a sparse array as input and converts it to a non-sparse array. It creates a new array, `vector_list`, with the same length as the input array. It iterates over each element in the input array and converts it into a two-element array. The first element of the new array represents the complement of the corresponding element in the input array, and the second element represents the value of the element in the input array. The function returns the `vector_list`.

4.1.7 Collaborative Model Training Framework for Client Collaboration

In the given scenario, there are two clients, Client 1 and Client 2. Both clients possess a subset of the features required for the task. Additionally, Client 2 has access to the corresponding labels for the data points. The objective is to collaboratively update a model using information from both clients. To achieve this, Client 1 can send its partial predictions and any other intermediate data to Client 2. Client 2, with access to the labels, can calculate the total loss by comparing the partial predictions with the true labels. This total loss represents the overall performance of the model on the combined data from both clients. After computing the total loss, Client 2 can update the global model using this information. The updated model is then shared back with Client 1, enabling both clients to have access to the improved model.

In the code, a class named “Client” with various methods for training and updating a model collaboratively is created. Here’s a summary of each method:

1. `__init__(self, train_data, test_data, labelled)`: This is the constructor method of the Client class. It initializes the client with training and testing data, and a boolean flag indicating whether the data is labeled or not. It also creates a sequential model using TensorFlow Keras, consisting of normalization layers, dense layers with activation functions, dropout layers, and a softmax layer.

2. `next_batch(self, index)`: This method prepares the next batch of training data for the client based on the provided index. If the data is unlabelled, it calculates and returns the gradients of the model’s output concerning the trainable weights for each example in the batch. If the data is labeled, it computes the model’s output for the batch.

3. `cal_model(self)`: This method returns the model’s output for the current batch.

4. `predict(self, test_index)`: This method predicts the output for the specified test data index using the model.

5. `test_answers(self, test_index)`: This method returns the true labels for the specified test data index if the data is labeled.

6. `batch_answers(self)`: This method returns the true labels for the current batch if the data is labeled.

7. `loss_and_update(self, a)`: This method is only called by Client 2. It calculates the updated probabilities by averaging the partial predictions with the existing model output.

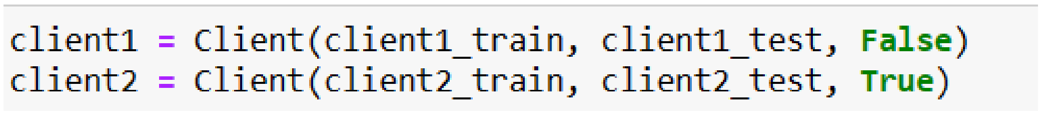
It then computes the coefficient and updates the model based on the calculated probabilities and the true labels of the batch. The method returns the updated probabilities and the loss.

8. `coefficient_and_update(self)`: This method is only called by Client 2. It computes the coefficient used to update the model based on the updated probabilities and true labels of the batch. It calculates the gradients and applies them to the model's trainable weights.

9. `update_with(self, grads)`: This method updates the model using the provided gradients for Client 1.

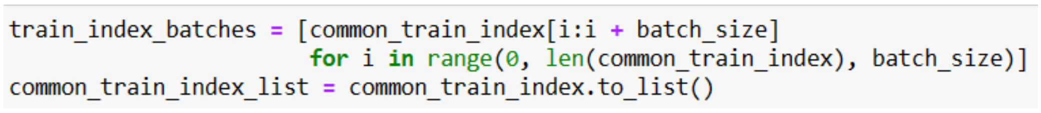
10. `assemble_grad(self, partial_grads)`: This method is only called by Client 2. It assembles the gradients for Client 1 by multiplying each partial gradient with the corresponding coefficient. It returns the assembled gradients. These methods enable the two clients to collaborate by exchanging partial predictions, gradients, and model updates to collectively train the model.

4.1.8 Instantiation of Client Objects



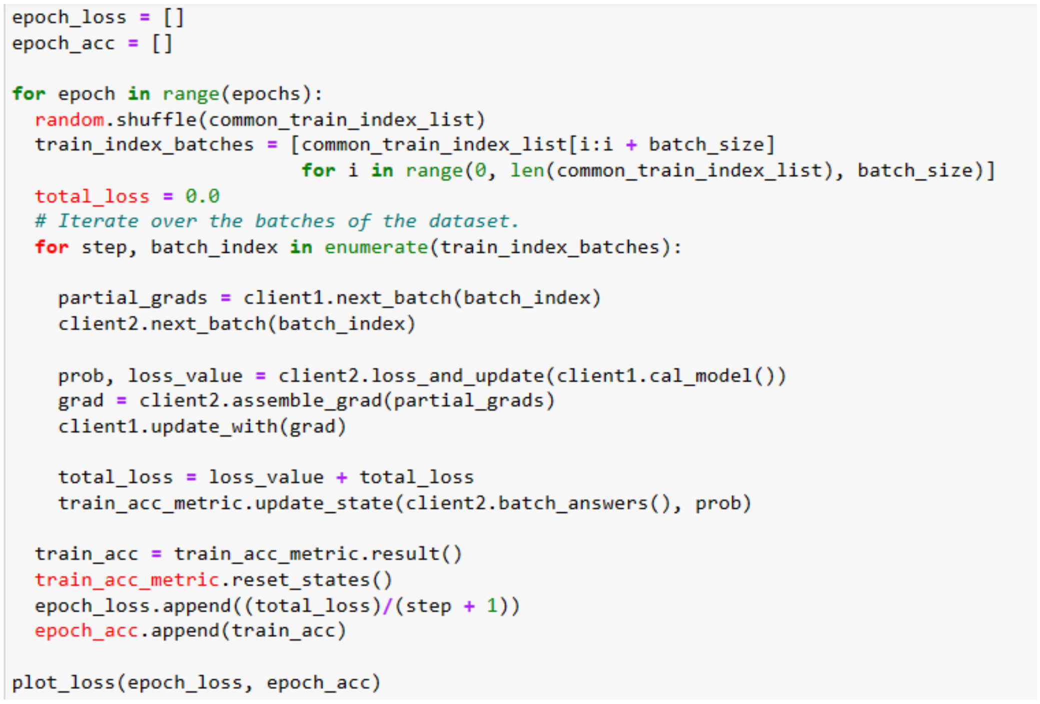
```
client1 = Client(client1_train, client1_test, False)
client2 = Client(client2_train, client2_test, True)
```

The provided code creates two instances of a "Client" object, namely "client1" and "client2". Each instance is initialized with different sets of training and testing data. The "client1" object is instantiated with the training data from `client1_train` and the testing data from `client1_test`. Additionally, the third argument passed to the "Client" constructor is set to "False", indicating that it does not require a specific setting. On the other hand, the "client2" object is initialized with the training data from `client2_train` and the testing data from `client2_test`. The third argument is set to "True", suggesting that this client requires a specific configuration.



```
train_index_batches = [common_train_index[i:i + batch_size]
                       for i in range(0, len(common_train_index), batch_size)]
common_train_index_list = common_train_index.to_list()
```

The first line uses list comprehension to create batches of data. It assumes that `common_train_index` is a sequence-like object (such as a list, pandas DataFrame, or pandas Series) containing the training indices. The variable `batch_size` represents the desired size of each batch. The code generates a sequence of indices using range and iterates over them. For each index `i`, it slices `common_train_index` from `i` to `i + batch_size` to extract a batch of data. These batches are stored in the `train_index_batches` list. The second line, `common_train_index_list` = `common_train_index.to_list()`, converts the `common_train_index` object, assumed to be a pandas DataFrame or Series, into a Python list using the `to_list()` method. The resulting list is assigned to the variable `common_train_index_list`.



```
epoch_loss = []
epoch_acc = []

for epoch in range(epochs):
  random.shuffle(common_train_index_list)
  train_index_batches = [common_train_index_list[i:i + batch_size]
                         for i in range(0, len(common_train_index_list), batch_size)]
  total_loss = 0.0
  # Iterate over the batches of the dataset.
  for step, batch_index in enumerate(train_index_batches):

    partial_grads = client1.next_batch(batch_index)
    client2.next_batch(batch_index)

    prob, loss_value = client2.loss_and_update(client1.cal_model())
    grad = client2.assemble_grad(partial_grads)
    client1.update_with(grad)

    total_loss = loss_value + total_loss
    train_acc_metric.update_state(client2.batch_answers(), prob)

  train_acc = train_acc_metric.result()
  train_acc_metric.reset_states()
  epoch_loss.append((total_loss)/(step + 1))
  epoch_acc.append(train_acc)

plot_loss(epoch_loss, epoch_acc)
```

The code implements a distributed learning approach, where two clients, client1 and client2, collaborate to train a model on a common dataset. The code begins by initializing empty lists to store the loss and accuracy values for each epoch. It then enters a loop over the specified number of epochs. Within each epoch, the training data indices are shuffled to introduce randomness, and the indices are divided into batches. The code then iterates over these batches, obtaining partial gradients from `client1` and providing the same batch of data to `client2`. The model's predicted probabilities and loss value are calculated using client2, and the gradients from both clients are combined. Client1's model is then updated using the combined gradients. The total loss and training accuracy is tracked for each epoch, and these values are appended to their respective lists. The vFedCCE algorithm achieves 77% accuracy.

4.2 Summary

The vFedCCE algorithm, discussed in this chapter, involves implementing a specific code. This algorithm focuses on federated learning in a vertical setting. This approach enables collaborative model training across different entities while keeping their data decentralized. The algorithm's details and implementation are outlined, allowing entities to jointly train a model while preserving data privacy and security.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This comprehensive thesis thoroughly explores various machine learning (ML) techniques, including Artificial Intelligence (AI), Deep Learning (DL), and Federated Learning (FL), shedding light on their merits and limitations. The analysis is centered around the Occupancy Detection dataset obtained from the UCI Machine Learning Repository. Six distinct machine learning classifiers and deep learning model were employed for the study, with the Random Forest model exhibiting the highest accuracy at 99.27%. Recognizing the constraints of traditional machine learning models, the research employs the vFedCCE algorithm, illustrating the vertical division of the dataset between two clients. This algorithm entails the partitioning of the dataset with shared samples but distinct features, and the collaborative construction of a shared model, resulting in an accuracy of 77%. Furthermore, it compares various categorization models on the Occupancy Detection dataset, presenting comprehensive analysis results and engaging in a discourse on the state-of-the-art comparisons with related research. This thesis stands as a significant contribution to the advancement of ML and AI research, offering valuable insights into cutting-edge techniques, emerging trends, and practical applications.

5.2 Future Work

VFL holds tremendous potential across diverse domains, but several challenges warrant attention for its successful deployment. In this thesis, the neural network is used in the vFedCCE algorithm for training the model, in the future the effectiveness of the proposed approach on other machine learning models, such as random forest or decision trees could be explored. Future research work can also be extended by increasing the number of clients for the vFedCCE algorithm.

Bibliography

[1] Ramoni Adeogun, Ignacio Rodriguez, Mohammad Razzaghpour, Gilberto Berardinelli, Per Hartmann Christensen, and Preben Elgaard Mogensen. Indoor occupancy detection and estimation using machine learning and measurements from an iot lora-based monitoring system. In *2019 Global IoT Summit (GIoTS)*, pages 1–5. IEEE, 2019.

[2] Luis Candanedo. Occupancy Detection . UCI Machine Learning Repository, 2016. DOI: https://doi.org/10.24432/C5X01N.

[3] Chaochao Chen, Liang Li, Bingzhe Wu, Cheng Hong, Li Wang, and Jun Zhou. Secure social recommendation based on secret sharing. *arXiv preprint arXiv:2002.02088*, 2020.

[4] Chaochao Chen, Jun Zhou, Li Wang, Xibin Wu, Wenjing Fang, Jin Tan, Lei Wang, Alex X Liu, Hao Wang, and Cheng Hong. When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2652–2662, 2021.

[5] Yiqiang Chen, Xin Qin, Jindong Wang, Chaohui Yu, and Wen Gao. Fedhealth: A federated transfer learning framework for wearable healthcare. *IEEE Intelligent Systems*, 35(4):83–93, 2020.

[6] Yong Cheng, Yang Liu, Tianjian Chen, and Qiang Yang. Federated learning for privacy-preserving ai. *Communications of the ACM*, 63(12):33–36, 2020.

[7] Bing Dong, Burton Andrews, Khee Poh Lam, Michael Höynck, Rui Zhang, Yun-Shang Chiou, and Diego Benitez. An information technology enabled sustainability test-bed (itest) for occupancy detection through an environmental sensing network. *Energy and Buildings*, 42(7):1038–1046, 2010.

[8] Daniel Garcia Bernal. Decentralizing large-scale natural language processing with federated learning, 2020.

[9] Zhenyu Han, Robert X Gao, and Zhaoyan Fan. Occupancy and indoor environment quality sensing for smart buildings. In *2012 IEEE international instrumentation and measurement technology conference proceedings*, pages 882–887. IEEE, 2012.

[10] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[11] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

[12] Nazila Hashemi, Pooyan Safari, Behnam Shariati, and Johannes Karl Fischer. Vertical federated learning for privacy-preserving ml model development in partially disaggregated networks. In *2021 European Conference on Optical Communication (ECOC)*, pages 1–4. IEEE, 2021.

[13] Daojing He, Runmeng Du, Shanshan Zhu, Min Zhang, Kaitai Liang, and Sammy Chan. Secure logistic regression for vertical federated learning. *IEEE Internet Computing*, 26(2):61–68, 2021.

[14] Yan Kang, Yuanqin He, Jiahuan Luo, Tao Fan, Yang Liu, and Qiang Yang. Privacy-preserving federated adversarial domain adaptation over feature groups for interpretability. *IEEE Transactions on Big Data*, 2022.

[15] Afsana Khan, Marijn ten Thij, and Anna Wilbik. Vertical federated learning: A structured literature review. *arXiv preprint arXiv:2212.00622*, 2022.

[16] Mashal Khan, Frank G Glavin, and Matthias Nickles. Federated learning as a privacy solution-an overview. *Procedia Computer Science*, 217:316–325, 2023.

[17] Wenqi Li, Fausto Milletarì, Daguang Xu, Nicola Rieke, Jonny Hancox, Wentao Zhu, Maximilian Baust, Yan Cheng, Sébastien Ourselin, M Jorge Cardoso, et al. Privacy-preserving federated brain tumour segmentation. In *Machine Learning in Medical Imaging: 10th International Workshop, MLMI 2019, Held in Conjunction with MICCAI 2019, Shenzhen, China, October 13, 2019, Proceedings 10*, pages 133–141. Springer, 2019.

[18] Yuanzhang Li, Tianchi Sha, Thar Baker, Xiao Yu, Zhiwei Shi, and Sikang Hu. Adaptive vertical federated learning via feature map transferring in mobile edge computing. *Computing*, pages 1–17, 2022.

[19] Peixi Liu, Guangxu Zhu, Wei Jiang, Wu Luo, Jie Xu, and Shuguang Cui. Vertical federated edge learning with distributed integrated sensing and communication. *IEEE Communications Letters*, 26(9):2091–2095, 2022.

[20] Yang Liu, Anbu Huang, Yun Luo, He Huang, Youzhi Liu, Yuanyuan Chen, Lican Feng, Tianjian Chen, Han Yu, and Qiang Yang. Fedvision: An online visual object detection platform powered by federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13172–13179, 2020.

[21] Linpeng Lu and Ning Ding. Multi-party private set intersection in vertical federated learning. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 707–714. IEEE, 2020.

[22] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.

[23] Manabu Nakanoya, Junha Im, Hang Qiu, Sachin Katti, Marco Pavone, and Sandeep Chinchali. Personalized federated learning of driver prediction models for autonomous driving. *arXiv preprint arXiv:2112.00956*, 2021.

[24] Wei Ou, Jianhuan Zeng, Zijun Guo, Wanqin Yan, Dingwan Liu, and Stelios Fuentes. A homomorphic-encryption-based vertical federated learning scheme for rick management. *Computer Science and Information Systems*, 17(3):819–834, 2020.

[25] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329*, 2019.

[26] Thomas Rooijakkers. Convinced—enabling privacy-preserving survival analyses using multi-party computation. 2020.

[27] Rui Shao, Pramuditha Perera, Pong C Yuen, and Vishal M Patel. Federated face presentation attack detection. *arXiv preprint arXiv:2005.14638*, 2020.

[28] Yong Song, Yuchen Xie, Hongwei Zhang, Yuxin Liang, Xiaozhou Ye, Aidong Yang, and Ye Ouyang. Federated learning application on telecommunication-joint healthcare recommendation. In *2021 IEEE 21st International Conference on Communication Technology (ICCT)*, pages 1443–1448. IEEE, 2021.

[29] Huizhong Sun, Zhenya Wang, Yuejia Huang, and Junda Ye. Privacy-preserving vertical federated logistic regression without trusted third-party coordinator. In *Proceedings of the 2022 6th International Conference on Machine Learning and Soft Computing*, pages 132–138, 2022.

[30] Chen Tianyi, Jin Xiao, Sun Yuejiao, and Yin Wotao. Vafl: a method of vertical asynchronous federated learning. *arXiv preprint arXiv:2007.06081*, 2020.

[31] CH Van Berkel. Multi-core for mobile phones. In *2009 Design, Automation & Test in Europe Conference & Exhibition*, pages 1260–1265. IEEE, 2009.

[32] Liu Yang, Di Chai, Junxue Zhang, Yilun Jin, Leye Wang, Hao Liu, Han Tian, Qian Xu, and Kai Chen. A survey on vertical federated learning: From a layered perspective. *arXiv preprint arXiv:2304.01829*, 2023.

[33] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[34] Shengwen Yang, Bing Ren, Xuhui Zhou, and Liping Liu. Parallel distributed logistic regression for vertical federated learning without third-party coordinator. *arXiv preprint arXiv:1911.09824*, 2019.

[35] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[36] Zheng Yang, Nan Li, Burcin Becerik-Gerber, and Michael Orosz. A systematic approach to occupancy modeling in ambient sensor-rich buildings. *Simulation*, 90(8):960–977, 2014.

[37] Zezhong Zhang, Guangxu Zhu, and Shuguang Cui. Low-latency cooperative spectrum sensing via truncated vertical federated learning. In *2022 IEEE Globecom Workshops (GC Wkshps)*, pages 1858–1863. IEEE, 2022.

[38] Fanglan Zheng, Kun Li, Jiang Tian, Xiaojia Xiang, et al. A vertical federated learning method for interpretable scorecard and its application in credit scoring. *arXiv preprint arXiv:2009.06218*, 2020.