Federated Learning on Tabular Data

Weitong Huang

A thesis submitted for the degree of
Bachelor of Software Engineering (Honours)
The Australian National University

November 2021

Except where otherwise indicated, this thesis is my own original work under the supervision of Prof. Amanda Barnard.

Weitong Huang
6 November 2021

Acknowledgments

Firstly, I would like to pay my especial gratitude to my supervisor Prof. Amanda Barnard, without whom my vision and understanding of the field of data science and scientific research would be much narrower and shallower. Your constant patience and always proper guidance from many aspects supported me throughout my honours year on this project, which I enjoyed every moment of. The same gratitude goes to my examiners, Dr. Minh Bui and Dr. Yu Lin, who put extra efforts in guiding me in the middle of this project and provided interesting and valuable insights to this project.

Also, I would like to thank Mr. Kevin Clark, Prof. Jordan Louviere, and Prof. Richard Carson from **Choice***flows*. Thank you for the time and effort you put into our meetings and all the wonderful questions and feedbacks. The project would lack much more industrial insights without your input and support.

Next, I would also like to thank my parents for supporting me to study in such a great university, both financially and mentally. Any of my work would not have been possible without your endless encouragement, strongest backing and dearest love.

To my peer friends from the computer science cohort, your insights and brilliant ideas inspired me a lot on things within and beyond the scope of this project. Thank you for being there for me and it has been a great pleasure to have worked with all of you.

Finally, I would like to thank Vanessa, who has always stayed by my side and supported me. Your company and encouraging words are most precious to me. I sincerely appreciate the strength you gave me throughout my years of study.

Abstract

Data scarcity issues have been widely observed in real-world machine learning applications over the past decades. Even in the era of big data, it not only persists but also raised new challenges in efficiently integrating machine learning methods into fields that ever require data-driven insights. To mitigate negative impacts of data scarcity and to explore new possibilities in machine learning approaches, we adopted the federated learning technology to enable privacy-preserving collaborations between data owners, allowing shared models to be trained on a greater amount of information. We implemented the pipeline from scratch to validate its effectiveness, and compared (where possible) to established methods in centralised and decentralised learning. The source code is publically available on GitHub [Huang]. On this basis, we tested our innovative federated data preprocessing module and several aggregation metrics in two different stages of the pipeline, to retain information in a fair and equitable way. The result shows the effectiveness of building a model using this framework, and quantify the advantages and disadvantages. Improvements were observed upon applying our innovative methods and they proved the validity of our methods. After discussing findings during the process, we concluded that, firstly, the federated learning framework is an effective new way of performing machine learning tasks in a data scarce context and it represents opportunities for industries in future applications. Secondly, a number of future research directions emerged from several aspects that will help developing this new technology and applying it in real-world scenarios.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

This work considers a study into the federated learning techniques, specifically horizontal federated learning models, with predicting nanoparticles catalyst activities being our case study to explore the component relationships and to develop and examine new techniques in the system.

1.1.1 Data Scarcity

In machine learning tasks, one major obstacle people often run into is the lack of high quality training data, which is referred to as the data scarcity issue in this study. While machine learning techniques focus on extracting and learning patterns from a set of training data X, it can result in issues such as low accuracy and bad generality if X is limited and insufficient to capture general patterns in the domain.

It is no surprise that researchers have been trying to alleviate negative impacts that come with data scarcity as machine learning techniques develop. There are two broad categories of techniques that have been studied historically, statistics based and learning based. While both techniques are tested to be effective towards this issue, challenges arise in the era of big data when dataset sizes surpass the processing ability of traditional algorithms.

1.1.2 Federated Learning

Federated learning is a learning technique that enables multiple data owners to build machine learning models in a decentralised, collaborative, yet privacy-preserving way. Through collaboration between data owners, the underlying machine learning model is expected to learn patterns from all datasets with different local trends, and thus have a general, global view of the domain.

In order to perform machine learning tasks in a federated learning approach, there are several basic components that provide function supports from different aspects

of the framework, including cryptography, socket or communication, and the underlying machine learning algorithm itself. Currently, the framework is very flexible and does not have a standard implementation code base.

1.1.3 Cryptography

Cryptography broadly refers to studies or practical techniques of secure communication with the presence of adversarial behaviours during the communication process. With the computing power of digital devices growing and the cryptography algorithms improving, it has become a standard to follow security practices and use encrypted communication protocols as much as possible when communicating over web sockets.

1.1.4 Hyperparameter

Hyperparameters are special parameters that need to be set or defined before the execution of a specific technique. They are often very important to the underlying technique and can be a determinant factor of its behaviour. Hyperparameters are overwhelmingly used in the context of machine learning algorithms. For example, the learning rate, a factor that a gradient multiplies so as to decide how much the model should progress in a certain direction when optimising it, is a determinant factor to the model's convergence rate and accuracy. It is often the case that hyperparameters need to be tuned within a range or randomly for the model to reach an optimal behaviour.

1.1.5 Data Preprocessing

Data preprocessing refers to a series of data manipulation operations that are often used before machine learning, or broader categories of data mining tasks. Its main steps include data cleansing, data integration, data transformation, and data reduction. Data cleansing involves handling and removing inconsistent or noisy data (e.g., the combination of Gender attribute being Male and Pregnant being True is invalid). Data integration combines data from multiple sources into one dataset. Data transformation transforms data into a consistent format that is suitable for later processing (e.g., transforming complete date information into one that only contains year). Often in cases of machine learning, it is necessary to perform data reduction using feature extraction or feature selection techniques so as to have a relatively small number of features for the machine learning algorithms to deal with.

1.1.6 Nanoparticles

Nanoparticles are a particles that are between 1 and 100 nanometers (nm) in diameter. They stand in the intersection of many natural sciences, such as physics, chemistry and biology, in many applications. To observe properties of a specific kind of nanoparticle, researchers often adopt methods such as Molecular Dynamics (MD) or Density Functional Tight Binding (DFTB) for measuring a nanoparticles' properties under certain kinetic and thermodynamic conditions.

Nanoparticle, due to its small size, often has completely different properties in many aspects with larger particles of the same substance. Also, being extremely small means they are very difficult to test on, not mentioning getting their accurate properties measured. One popular application field of nanoparticles is to use them as reaction catalysts, where precise measures of some nanoparticles properties are often required.

1.2 Thesis Statement

Data scarcity issues are especially common in two scenarios. Firstly, in commercials and economics where people wish to build domain models using machine learning techniques but blocked by lacking of training data due to privacy or financial causes. Secondly, in scientific research where data instances can be naturally small in number and hard (in techniques and expenses) to generate or collect elsewhere. Traditional approaches tend to solve the issue on individual cases where one tries to enlarge their database or enhance model prediction abilities on small datasets. Valid and effective as these methods are, we advocate for the alternative of using federated learning to solve the problem in the era of big data. We believe it is a better solution under the condition where people are willing to collaborate on machine learning tasks with their local data.

In this work, we firstly implemented a federated learning pipeline from scratch, where we studied the relationships between its components and its working mechanism. Then we introduced our innovative federated data preprocessing into the established pipeline and tested several new aggregation metrics in two stages of the pipeline. Different from the original studies carried out in previous literature where mostly the domain were computer vision or natural language processing tasks, we set our scope to be using tabular data, a form a lot of scarce data takes. We consider the main contribution of our work to be three aspects. Firstly, with little existing literature discussing the implementation details of the federated learning pipeline, our work has closely examined its basic structure, discovered difficulties in the process, and provided our relaxed alternatives with proper justification. Secondly, the innovative federated data preprocessing pipeline makes it possible to perform data preprocessing in a federated learning context. This technique has never been ad-

dressed in previous literature. Thirdly, the aggregation metrics we tested act as a guideline for further researches into the same area.

1.3 Thesis Outline

There are six chapters in this work.

In Chapter 2 we provide background of our study, including in-depth literature reviews discussing the technologies we used in our work. In Chapter 3 we provide mathematical expressions of the methodology we adopted. In Chapter 4 we provide details of our design and implementation of the system, covering all the components in the pipeline. In Chapter 5 we provide results and analysis on them. In Chapter 6 we conclude our work and discuss future possibilities in this area.

Chapter 2

Background

2.1 Outline

In this chapter, we will provide a detailed review of the data scarcity issue, existing methods towards solving it, our proposed alternative (i.e., federated learning), and several techniques that are extensively used in our method. Our testing plan and measurement metrics will also be included.

Section 2.2 discusses the general relationship between the big data and data mining techniques, specifically, machine learning methods are considered major practices within the data mining context in this study.

Section 2.3 introduces data scarcity issues present widely in machine learning tasks. We further discuss contributing factors to the problem in two different data availability situations, which can be summarised as privacy, financial, and natural factors. Section 2.4 then introduces several methods that have been used to mitigate the data scarcity issues and come to the conclusion that these methods could result in extra complexity and uncertainties in the designated machine learning task.

In Section 2.5, we first introduce our proposed method, federated learning, towards solving the aforementioned data scarcity issues with extensive addressing on its collaborative and privacy-preserving nature. We then talk about challenges within the framework and possible approaches towards solving them. Several concepts we used in our federated learning implementation are introduced, including data encryption, synchronisation, and hyperparameter tuning.

Section 2.6 introduces general data preprocessing actions in a complete pipeline and discusses the data preprocessing needs specifically in our study context. A brief introduction and discussion over the use of principle component analysis in our work is also included.

In Section 2.7 we introduce the background of our dataset, the eight synthetic nanoparticle datasets, and justify the reasonableness of using them as our case study based

on the three data scarcity contribution factors we introduced in Section 2.3.

Finally, in Section 2.8, we introduce two evaluation metrics we use for measuring the effectiveness of our results.

2.2 Big Data and Data Mining

With rapid development of web technology and hardwares that enabled faster network communications and larger storages, we are now in the era of big data [Sagiroglu and Sinanc, 2013]. Often described as datasets so big in size that they surpasses the ability of traditional softwares to process or manage, big data stimulates research and developments into new methodologies that aims to take advantages of its high volume and information density from non-traditional aspects as well as to solve the technical difficulties arose from its usage.

Data Mining (DM) is the process of extracting and capturing valid patterns from data using various methods, including classification, regression, clustering, and association [García et al., 2015]. Due to the usual presence of computer algorithms in these methods, most of them can be categorised as machine learning (ML) methods [Mitchell et al., 1997].

Over the past several decades, ML has always faced the problem of lacking high quality training data. An example of big data promoting ML development would be the famous language model, GPT-3 [Brown et al., 2020], which used 45 Terabytes of text data sourced from the Internet and reached state-of-the-art performance in natural language processing (NLP) tasks. In addition, with new hardware advances, ML models are now deployed on many mobile devices and prevalent in many aspects of people's lives, including economics, entertainment, science and medical research.

Although ML methods affect our daily lives in many ways, in an ML researcher's opinion [Sarker, 2021], it currently has only very limited applications and there is a general lack of pipelines of efficient integration, particularly in cases where the scarcity of data, instead of the high volume of data, limits the applicability of conventional ML approaches.

2.3 Data Scarcity

Data scarcity refers to the scenario where high quality data in large volume is often unavailable for training a model. Generally, it results in low prediction accuracy in machine learning models.

Compared to data sparsity, which is often confused with our topic here, data scarcity means the number of data instances is small. In the case of data sparsity, the problem is often having too many empty values (i.e., values that do not contribute to the model, often zero or null) between data entries. According to the size of the source of data, data scarcity issues could be classified into two categories.

- Data is abundant but only a little is suitable for other systems, such as machine learning models, to use directly. We call this situation data rich but information poor (DRIP) [Bernus and Noran, 2017]; and
- Little data is available from sources. We call this situation data poor (DP). Note that this is different from the data poverty, which Lucas et al. defined for the situation where financial poverty contributes to low Internet data access. We specifically introduce and use DP, meaning the lack of data from sources, in our work;

There are many examples of the two situations in different areas and each with different causes. Ker et al. pointed out that there is a lack of publicly available data in medical research in general, and even less high quality data with labels available. This phenomenon has already been regarded as the substantial problem of implementing ML in clinical practices [Willemink et al., 2020].

Privacy regulations and concerns are the main reasons for data scarcity. Shown by Brankovic and Estivill-Castro in 1999, the public was already very concerned about privacy data in many aspects of their lives, including the most critical medical and banking records. Laws and regulations regarding patient data, such as the Health Insurance Portability and Accountability Act (HIPAA) by the US congress in 1996, were established to explicitly restrict the usage of patient medical records by insurance companies and healthcare organisations. Many efforts in algorithms and system design are also made from different levels of implementation to preserve user privacy [Domingo-Ferrer and Torra, 2005].

Similar trends are observed in business and economics contexts, where regulations on user privacy are ever strengthened in the era of big data. Regulations such as the European General Data Protection Regulation (GDPR) [Voigt and Von dem Bussche, 2017; Albrecht, 2016] leaves business owners little space to learn about their customers and the whole market using customer records while some data might be essential to one's business.

Aside from privacy concerns, financial factors also bring about data scarcity issues. An example given by Perlich et al. shows that in the case of advertisement, it is very expensive to sample qualified data from advertising practices and positive outcomes are even more rare. In this case, the limitation makes it hard for companies with the financial power to collect data, not to mention small and medium-sized enterprises

(SMEs) that could not afford to. This could very possibly put smaller entities in a situation where their limited knowledge of the market eventually leads to monopoly by large conglomerates.

In a DP scenario, some typical types of data can be naturally hard to collect. Results from natural science laboratories (e.g., nanoparticles or chemical reactions) can take a long time to generate through experiments, which can be costly, and potentially dangerous to researchers. Aside from the fact that one lab would not want others to know about their confidential results, even within the lab the results might be extremely scarce, which prevents anyone from building a model to accurately predict desired properties.

To overcome the data scarcity impediment, we can either encourage researchers to combine their data and take the risk, or develop new technologies that allow them to collaborate while respecting their privacy.

2.4 Traditional Methods

2.4.1 Overview

Before going into details of our proposed solution, we would like to introduce some traditional methods people use to mitigate the data scarcity issues. Based on the amount of the data from sources (i.e., DRIP or DP), researchers have formulated different approaches. The general intention can be categorised as discovering more values from existing data or increasing the amount of data so as to increase the amount of information.

Data Availability	DRIP	DP
Main Challenge(s)	1. Data duplication 2. Find related data 3. Data type, formatting, and structure can be random	1. Not enough data for training
Solutions	1. Data cleansing methods 2. Use indicator to help discover relations	1. Synthetic data using datatransformation techniques 2. Domain adaptation usingtransfer learning techniques
Solution Drawbacks	1. No generic cleansing pipeline 2. Indicator not necessarily exist 3. Process can be highly customised and labour intense	1. Synthetic data can be hard to generate generally 2. Transfer learning may not be feasible in may cases

Table 2.1: Traditional methods and their drawbacks in both data availability scenarios

2.4.2 Case DRIP

In a DRIP situation, the difficulty often lies in choosing data that is of high quality as well as being highly related to the goal a learning task wishes to achieve. Such problems can be studied from a data wrangling perspective, where the main goal is to identify, remove and replace inconsistent or duplicated data from large datasets. Hernández and Stolfo closely compared sorted-neighbourhood and data-cluster based methods in a data merge/purge task and came to the realisation that both methods require multi-pass data cleansing processes to reach high accuracy, which could be computationally intense. On the other hand, Maletic and Marcus focused more on identifying potential errors in large datasets using a variety of methods, with the conclusion being many data cleansing tasks require highly customised operations to reach a satisfying outcome, which is especially true when more data sources being available in a big data context. We believe as the volume and types of data continue to grow, it will become harder to come up with any data cleansing standard or generic pipeline that suits all tasks.

Similar cases are also studied by Ridzuan and Zainon in 2019. The proposed methods often look into statistical properties of the data and usually require high computing power due to the large volume and highly complex data. Additionally, there are no existing generic tools for all kinds of data cleansing tasks. Most methods still require a huge amount of human interaction and input.

Second, any data collected might be enormous in size while random in format, quality, and topics [Alharthi et al., 2017]. Accurately identifying the right data to collect can be challenging. To gather relevant data from a large source in the first place, methods such as using data from a related context as indicators are studied. An example would be Souza et al. using social media data as indicators to augment the dengue-related data in Brazil in 2014. However, these methods might not always be applicable because such indicators do not always exist. Also, it can also be hard to validate the relationship between a candidate indicator and the actual event, which adds extra uncertainty to the original problem of interest.

2.4.3 Case DP

In a data poor situation, a straightforward way of dealing with low data availability is to increase its volume by creating synthetic data. One could achieve this using data transformation techniques [Krizhevsky et al., 2012; Simard et al., 2003; Le et al., 2017] to create new data with existing labels. Such data is then used as if it was authentic and trained together with the original data. However, these techniques are not suitable for all tasks. For new instances to retain the same label after the transformation, they must be as meaningful as the original data. This is the reason why this method is mostly used in computer vision (CV) tasks for image augmentation rather than on

general tabular data.

Another solution is to use transfer learning techniques [Pan and Yang, 2009; Hutchinson et al., 2017]. If a generic model in a similar context exists, we could perform domain adaptation using our limited data and hopefully an acceptable model can be produced for our task. In this case, domain adaptation generally means "adapting" the original domain of the existing model to the new, but related, domain that we are interested in [Pan et al., 2010]. However, this method can be optimistic for two reasons. First, there is no guarantee of a generic model for every research field. Second, even if one exists, the final result depends highly on how well the model describes the domain from a high level and how much training data for the transferring is available. Therefore, although generic base models exist in many topical research areas such as CV or NLP, this is not a feasible approach for many other ML tasks.

2.4.4 Summary

In both data availability scenarios, drawbacks are quite obvious. Although all methods attempt to solve the data scarcity issue, additional complexity always comes with the proposed solutions. While the original learning task could be easy and straightforward to solve given enough data, these methods make it harder to implement and add extra uncertainty to the system, which can affect the performance of the final model. While those methods might be the optimal options when there is a strict limit of data available, it might not be the case when a more straightforward and cost-efficient method exists.

As the name suggests, data scarcity issues essentially refers to problems raised by lack of data. Given that data is overall abundant but owned in small amounts by individual clients, a collaborative model training framework should solve most of the problem. Federated learning, a collaborative, privacy-preserving model training framework, has been invented to address such issues.

2.5 Federated Learning

2.5.1 Overview

Federated learning (FL) is a learning technique that allows users to train models in a collaborative, decentralised, and privacy-preserving way using their local data [McMahan et al., 2017a]. Being a learning technique rather than an algorithm targeting a specific field, FL focuses on applying existing ML models on the decentralised data regardless of its domain. McMahan et al. invented the training structure and tested it using two famous datasets and models: a convolutional neural network (CNN) on the Modified National Institute of Standard and Technology (MNIST) dataset, and a long short-term memory (LSTM) on Shakespeare works dataset, to

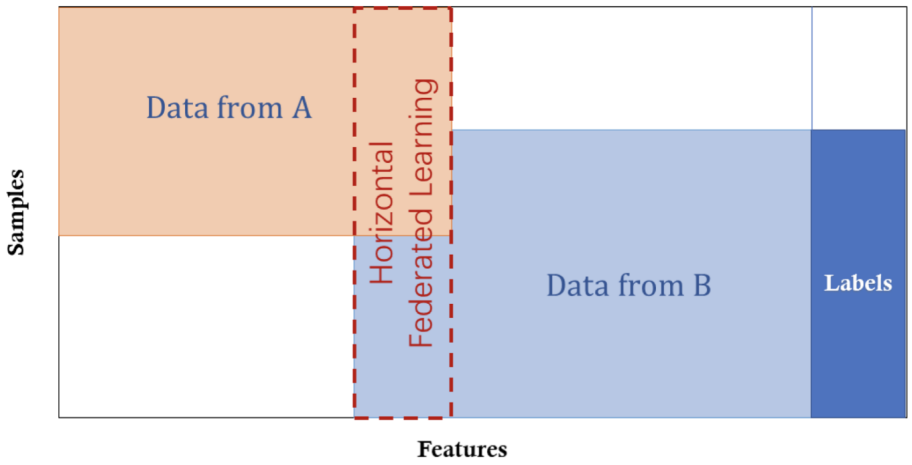
show its universality to different models. Both tasks have shown expected performance from the framework.

The framework aims to promote collaborations from decentralised data owners (clients). It relies on client hardwares for the training process and uses a central coordinator to aggregate those sub-models into a shared one. Calculations are done locally on client devices and the aggregation process can be different depending on what type of clients we work with. Kairouz et al. separated FL applications into two categories, cross-device and cross-silo. The former assumes clients to be edge devices such as mobile phones, Internet of Things (IoT) devices, sensors, and etc. These devices are usually large in number, while their local data can vary in type, format, and quality and can be very biased. On the other hand, cross-silo assumes clients to be reliable data centres with stable Internet connection and generally higher data quality. An example of a cross-silo setup would be hospitals with reliable patient data collaborating on the training process. Depending on which setting we work with, different issues should be taken care of. For example, edge devices might face higher rates of dropping out due to less stable Internet connection, while data centres might have huge overhead in training and communication due to larger amount of data.

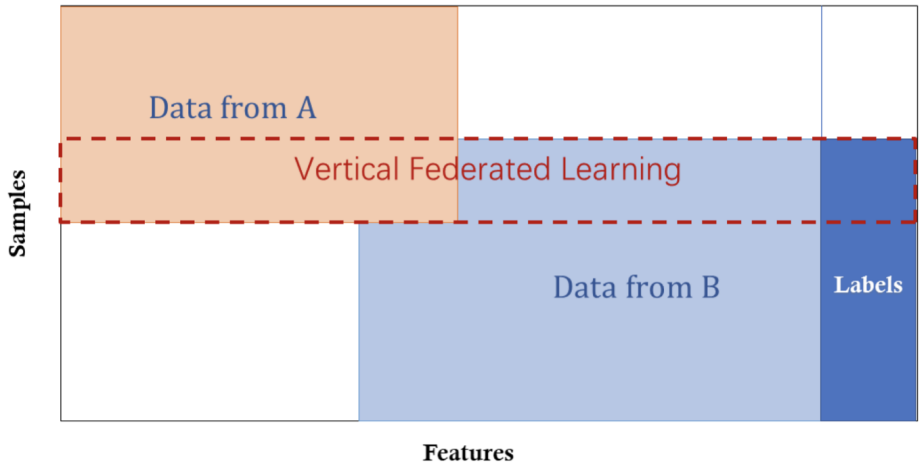
While having similarities with traditional distributed machine learning techniques, where large datasets are deliberately distributed to different computing cores or devices by the central coordinator for faster parallel calculation, an FL central coordinator (referred as a *federator* in later contexts) does not have access to any data at all and is designed for totally different purposes. During the training process, data on clients' devices is never shared to either peer participants or the federator. This guarantees data integrity, retains confidentiality, and builds a basis for trusted collaborations between clients. In fact, the name federator emphasizes the role of the central coordinator being one that federates and unites the clients instead of an overlord that owns everything in the process.

2.5.2 Federated Learning Models

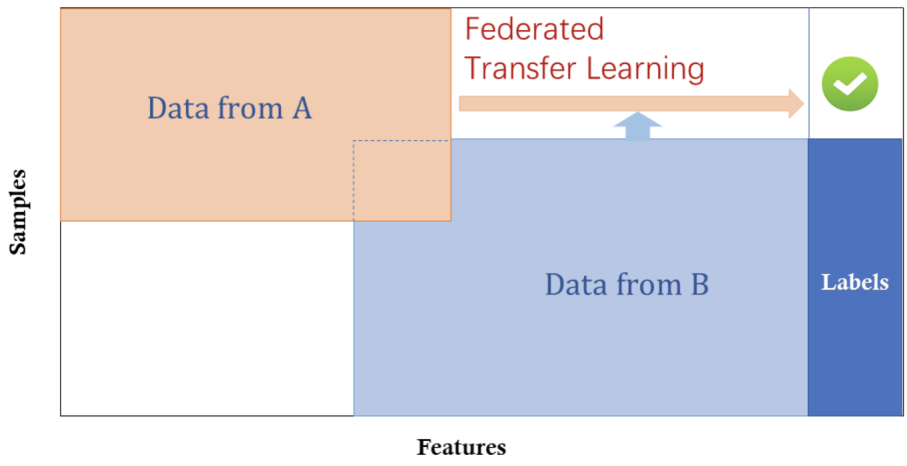
Given a specific field of interest and several clients with related local data, there are differences in sample space (i.e., the identity of the measured target of a record) and feature space (i.e., the measured attributes of a record) among all datasets. An FL model can be classified as a horizontal, vertical, or federated transfer learning model (see Figure 2.1) according to the overlapping relationship between the sample and feature spaces and the underlying learning strategy.



(a) Horizontal FL



(b) Vertical FL



(c) Federated Transfer Learning (TFL)

Figure 2.1: Three FL Models [Yang et al., 2019]

2.5.2.1 Horizontal FL

Horizontal FL takes advantage of the identical or largely overlapped feature space of the datasets. In this case, records should have a consistent set of features across all the datasets while records from different clients are not required to be generated from the same test subjects. In this case, the federator is needed for initialising training, collecting model updates, and arranging communication between the clients. Although a potentially malicious coordinator is often imagined for the purpose when developing better security and privacy schemes, in our study, we assume the coordinator is a trustworthy entity. This Federator-Clients structure is shown in Figure 2.2.

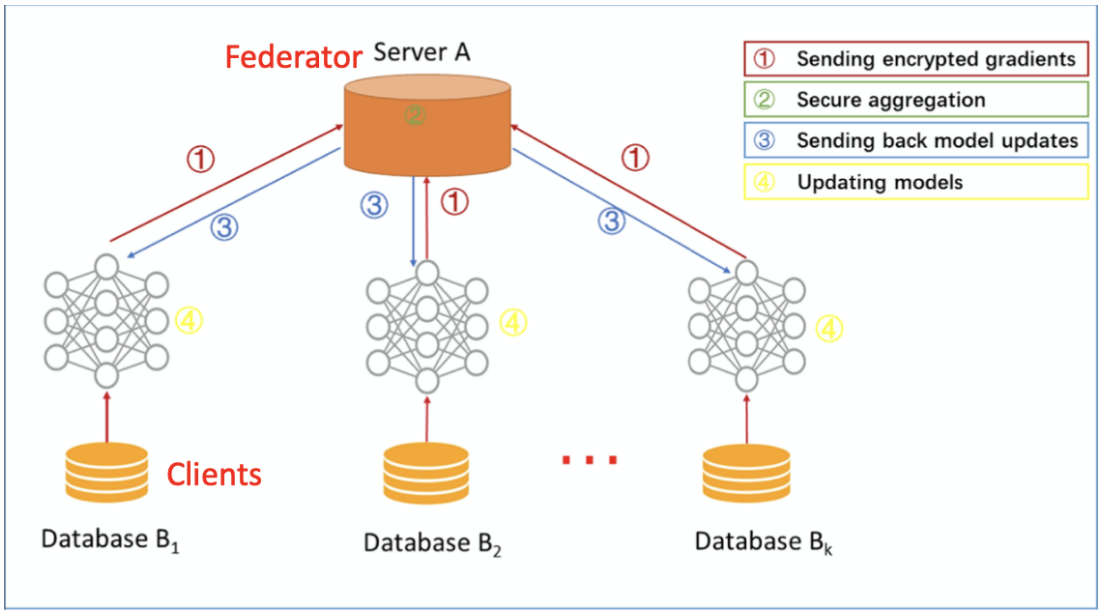


Figure 2.2: Architecture for a horizontal FL system (adapted from [Yang et al., 2019])

2.5.2.2 Vertical FL

Vertical FL focuses on overlaps in the sample space. In other words, data from different owners should be generated by the same group of testing subjects. This can be same people or same chemical material. An example would be a person having a bank account also having an account on a shopping website. Although their data might be measured on different attributes in a bank or on the website, collaboration between the clients would be able to find useful patterns when combining these two pieces of records. To achieve this, steps different from those in a horizontal model, such as encrypted entity alignment (i.e., anonymously matching record sources) and intermediate inter-client communication, are required.

2.5.2.3 Federated Transfer Learning

When dealing with situations where little overlapping is observed in both sample and feature space, FTL is used to transfer general patterns of the larger context to a client-specific level. FTL has a structure similar to that of vertical FL. The difference lies in the intermediate results that get exchanged. One client can use the architecture to gain broader knowledge of the entire domain and promote the model's ability of predicting labels from the target-domain using those in the source-domain [Yang et al., 2019].

2.5.3 FedAvg and FedSGD

Aggregating models from different client is a major step in federated learning. It refers to the case where the federator combines results that are generated locally on clients. The local updates can be training more than one epochs using mini-batches, which is called FedAvg (federated averaging), or, on the other hand, aggregating gradients, which means communicating with the federator and sending gradient updates for aggregation every time after a local model update. This approach is called FedSGD (federated stochastic gradient descent).

As described by McMahan et al., there are two hyperparameters that decides which update algorithm a FL is using, namely batch-size B and number of epochs E. Given a set of participating clients, we call the update algorithm FedSGD if and only if $B = +\infty$ and $E = 1$, where $+\infty$ stands for the size of the training set. Otherwise, we call the algorithm FedAvg. We can view FedSGD as a special case of FedAvg with corresponding parameters.

Since large-batch synchronous stochastic gradient descent (SGD) algorithm has been proven state-of-the-art setting in a data centre, or cross-silo, context due to little overhead caused by including new clients for an update round [Chen et al., 2016], the corresponding adapted algorithm in a federated context (i.e., FedSGD) has been chosen as a benchmark for many FL tasks [Chai et al., 2020; Liu et al., 2020; McMahan et al., 2017a].

2.5.4 Advantages

While it is infeasible to collect or combine a huge amount of data due to privacy and confidentiality concerns as well as financial limitations, most entities usually have the ability to collect some small amount of data, which, if used collectively with others of similar interests, would be sufficient for training ML models that have better prediction ability.

The main advantage of FL is its collaboration mechanism that allows clients of different sizes to contribute and share the result, with privacy protection being the basis

and utmost priority.

Collaboration between clients of similar interests has a positive implication on data quality (i.e., completeness, accuracy, consistency, integrity, timeliness, and validity [Laranjeiro et al., 2015]). Under the assumption that none of the participants are malicious and wants to corrupt the learning outcome, each client is likely to have higher quality data that is related to the learning task, compared to random and multi-sourced data. This is very important for ML models to train properly as the better training data captures patterns in a domain, the more accurate a trained model could be.

Collaboration, and having data from clients of different backgrounds, means data can be diversed within the same field of study, which, when combined, can provide a high-level overview of the larger area of study. An intuitive way to understand this is by thinking about data distributed to different computing devices in traditional distributed machine learning practices. Each of the data is expected to capture a portion of patterns from all data. The same is expected in a federated scenario, that data from different participants capture specific patterns within one's specific domain. In a real-world scenario, for example, a travel specific book store would have different buyer profiles than that of a cooking book store, while both of them have data in the book buyers category. Through building a shared model using diversed data, purchasing patterns within both travel and cooking books can be captured, providing both owners broader visions and data-driven insights [Qian and Zhang, 2021; Xu et al., 2021] over a bigger book-selling context that is usually beyond their own scope.

An unintended but positive side-effect is that distributed computation on client devices lifts the heavy burden of training off the central server. The training process can be fast and accurate thanks to the advanced hardwares present in all kinds of computing devices, from mobile phones and IoT devices to dedicated data servers.

Another important advantage of FL is that it is simple to implement. Although the traditional methods solve part of the data scarcity issues from different perspectives, they always come with extra complexity and uncertainty due to introducing new technologies into the system whose effects are highly related to the final result. FL only requires the proposed model to be applied decentralised without any other modification. The communication infrastructure is provided by the FL framework and no special changes on the model itself are required. An example would be the TensorFlow Federated (TFF) package in Python. It only requires users to add one line of packaged decoration to make an existing machine learning model federated. In a real-world scenario, such simplicity would help clients to adapt to a federated context very quickly.

2.5.5 Challenges

Challenges in the FL context mainly come from four aspects: the expensive communication, system heterogeneity, statistical heterogeneity, and privacy concerns [Li et al., 2020b]. Expensive communication concerns with communication costs resulted from repeated large message exchange over the networks. This problem gets worse as the number of communication rounds and size of models increase and is more obvious in a cross-device scenario. The problem is also related to system heterogeneity issues where the main concern is hardware differences that can result in problems including straggler and dropping-outs. Again, the problem is mostly prominent in cross-device scenarios.

Statistical heterogeneity refers to the problems caused by data not independently identically distributed (non-i.i.d.) among datasets in FL. It is an intuitive and widely acknowledged issue by the federated learning research community [Zhao et al., 2018; Sattler et al., 2019]. Such issues are related to different dataset sizes and features being used collaboratively in training. Although FL aims to learn a global representation of data patterns through diverse data sources, biases introduced by such imbalances negatively affect the model's prediction ability. One way of dealing with this problem is through meta-learning, as mentioned by Li et al.. However, under certain conditions [Li et al., 2019; Sahu et al., 2018], FedAvg algorithm is guaranteed to converge even on non-i.i.d. data, only that an accuracy loss is expected compared to centralised training [Yang et al., 2019].

Privacy concerns are mostly related to data or gradient information being leaked in either the communication or model aggregation processes [McMahan et al., 2017b]. For communication encryption, it is a standard to use homomorphic encryption which is covered in Section 2.5.5.2. However, for higher levels of security due to concerns beyond a general trusted coordinator assumption, loss of model performance or pipeline efficiency is often observed [Bonawitz et al., 2017; McMahan et al., 2017b]. This remains an open question to be solved.

2.5.6 Data Encryption

2.5.6.1 Summary

For transferring data securely from one place to another over a web socket, data encryption is often required as a precaution such that even data is hijacked during transportation, attackers would not be able to know the content without the correct decryption credentials.

Modern cryptography is usually divided into two categories based on how keys, special strings of characters or numbers used in a cryptographic algorithm, are used in the encryption and decryption process. Symmetric-key cryptography shares

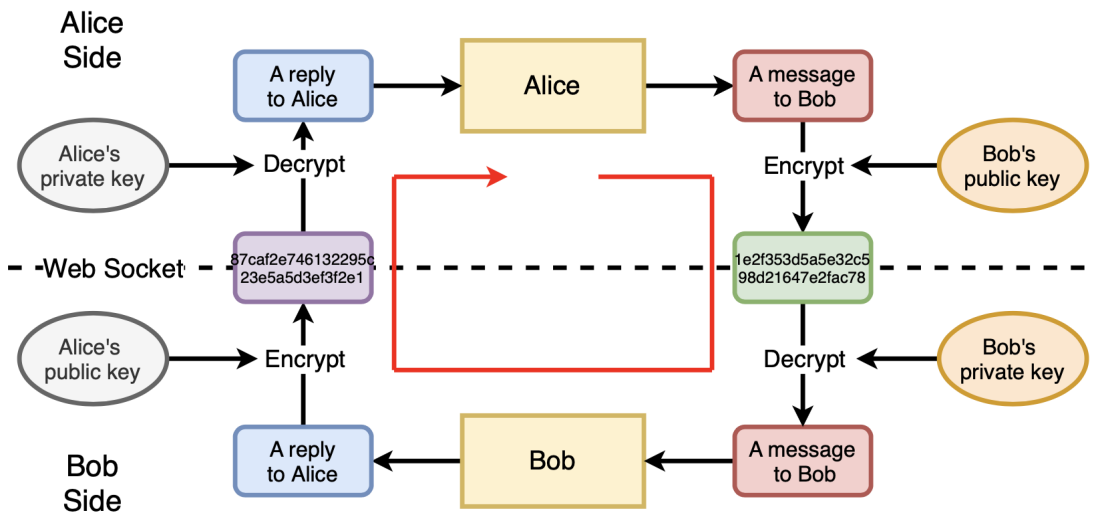


Figure 2.3: Public-key Encryption and Two-way Communication Pipeline

the same key between the sender and receiver, while public-key cryptography, or asymmetric-key cryptography, uses a unique public-private key pair to encrypt and decrypt a message respectively. Each public key has a corresponding private key, making it extremely hard to decrypt an encrypted message without using the designated private key. An example of sending and receiving messages securely using public-key encryption is shown in Figure 2.3. Asymmetric methods are generally slower than symmetric methods due to having less trivial mathematical computations [Fontaine and Galand, 2007].

2.5.6.2 Homomorphic Encryption

Homomorphic encryption (HE) is a special public-key encryption algorithm that allows computations to be done on encrypted values without decrypting them first [Fontaine and Galand, 2007; Rivest et al., 1978]. An encryption scheme is said to be homomorphic if the relationship in Figure 2.4 is satisfied.

Formally, given an operation domain D, a valid operation F defined on D, we call an encryption scheme E an HE scheme if

$$\forall a, b \in D, F'(E(a), E(b)) = E(F(a, b))$$

is satisfied, where F' is the definition of operation F on encrypted values.

In our work, we especially focus on additive HE for numerical values. That means, domain D is the real number domain $\mathbb{R}$ and operation F in this case is addition, such

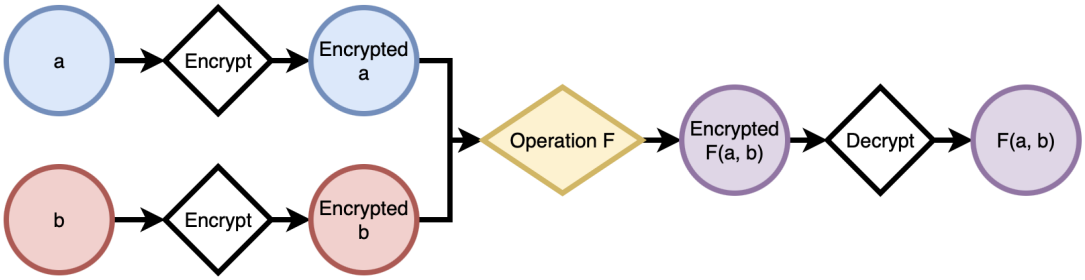


Figure 2.4: Homomorphic Encryption Condition

that $\forall a, b \in \mathbb{R}, E(a) + E(b) = E(a + b)$.

2.5.6.3 RSA Encryption

RSA encryption is a wildly used asymmetric encryption scheme that is based on the difficulty of factoring the product of two large prime numbers. For generating an RSA keypair, a key length is required. It decides the security level and the maximum length of a message that it can encrypt. Usually, the longer the key, the harder a message could be decrypted using brute force.

RSA is one of the oldest asymmetric encryption algorithms and is quite slow compared to symmetric algorithms [Fontaine and Galand, 2007]. In a good cryptography practice, RSA is not used directly to encrypt the bulk communication message due to its low speed and limit on message size. Instead, it is used for transferring a symmetric key securely to the receiver when initialising an online communication. After that, encryption and decryption are done using the symmetric key for higher speed. However, we did not follow this convention and we will talk about our reasons in the Design Chapter.

2.5.6.4 BatchCrypt

In situations where large number of repeated encryption or decryption actions are needed, it is very likely that this security requirement becomes the largest overhead of a system. This issue has been discussed in many FL literature when people are repeatedly using HE for securely transferring large gradient matrices [Liu et al., 2019; Zhang et al., 2020]. Therefore, BatchCrypt [Zhang et al., 2020] was introduced to reduce this overhead.

Theoretically, BatchCrypt combines numbers in a matrix into a new variable based on some rules, which, when applied, makes it possible for two purposes. Firstly,

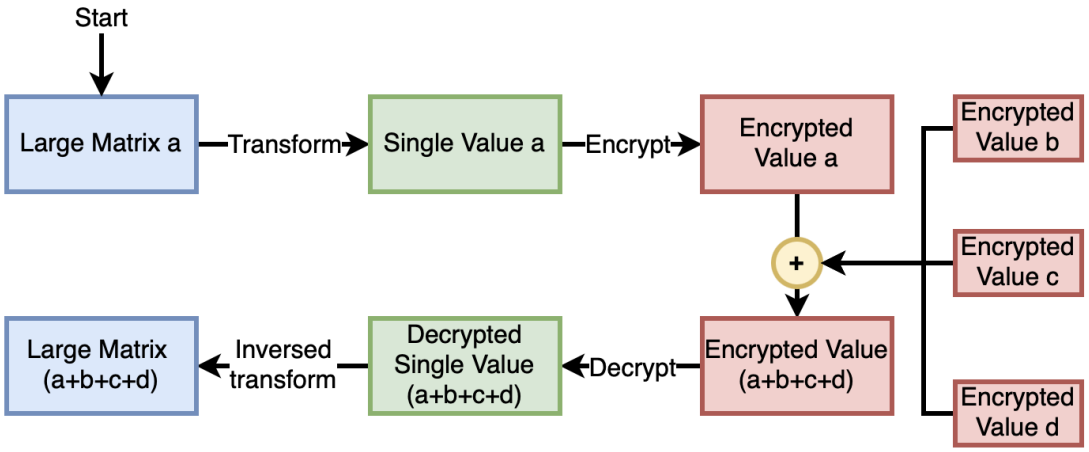


Figure 2.5: BatchCrypt Process

the new variable should be able to be reversely calculated to regenerate the original matrix. Secondly, the new variable should also satisfy the homomorphic calculation rules described in Section 2.5.5.2. When satisfying both rules, we could first convert a large matrix to a single value and perform the encryption once on that value. After adding up with other encrypted values, the new value can be covert back to a matrix for further operations. An illustration of the process is shown in Figure 2.5. It is much faster to perform the encryption and decryption computation once compared to doing it repeatedly for every entry of the matrix.

Another advantage of using BatchCrypt is that the size of a single encrypted value is much smaller than that of a matrix of encrypted values. It reduces the network traffic and amount of bandwidth required for communication.

2.5.7 Synchronisation & Asynchrony

Synchronisation refers to process synchronisation, which is the idea that multiple processes join at a certain point to perform a set of actions or to reach agreements before proceeding again. Such actions or agreements include the famous forks-and-joins [McCool et al., 2012] and producer-consumer [Arpaci-Dusseau and Arpaci-Dusseau, 2018] patterns in a parallel system design. Alternatively, it could be used to limit concurrent access to a shared variable. In our work, we mainly focuses on its usage of processes joining to perform an action before proceeding.

Opposite to synchronisation, asynchrony refers to the behaviour where the occurrences of each individual event does not affect the main flow of a program. An example is a message server receiving new connections does not block existing connections from their usual workflow. Asynchrony is often seen in server setups where

new connections are handled using asynchronous function calls (i.e., in a separate thread or sub-process) to lift the heavy request burden of the server process and to not affect its other functions.

2.5.8 Hyperparameter Tuning

To train an ML model on a dataset, there are often parameters to be set before training. These parameters are called hyperparameters [Probst et al., 2019; Weerts et al., 2020] and in many cases, the performance of an ML model depends highly on its hyperparameters [Weerts et al., 2020]. Hyperparameters can be set to default values from a software package. But for a model to fit to a specific dataset better, its hyperparameters are usually adjusted, or tuned.

Hyperparameter tuning can be done manually or automated by softwares because the process can be mechanical and computationally intense. It usually involves setting a range for a set of hyperparameters and letting the model run under all the combinations. After each round, either a heuristic would pick the next candidate parameters to run with, or the next combination is tested in an enumerative way. Currently, the most efficient tuning method is random grid search [Bhat et al., 2018; Amos et al., 1996], which essentially sets boundaries for tunable hyperparameters and draw values from the hyper space following a random distribution.

2.6 Data Preprocessing

Data preprocessing actions are taken on the basis of existing datasets, where issues such as missing value, duplicated instances, inconsistent format, and high dimensionality may exist. After applying appropriate algorithms (e.g., imputation, selection), the dataset should be in a consistent format and ready to be used by further DM techniques. Our main focus of data preprocessing step in this work, data reduction, involves removing duplicated instances as well as dimension reduction actions such as principal component analysis (PCA) [Wold et al., 1987] or feature selection (FS) methods [Chandrashekar and Sahin, 2014]. A simple data preprocessing pipeline is shown in Figure 2.6.

Due to the high quality of our dataset (see Section 2.7), data cleansing and data integration actions are not part our scope. We applied, however, normalisation techniques to cope with the value magnitude differences between datasets. The choice of high quality dataset is a reflection of consideration of keeping the training task simple when we focus on exploring the federated learning framework. We also innovatively incorporated the dimension reduction technique, PCA, into the federated pipeline to explore the feasibility of having data-preprocessing units in it.

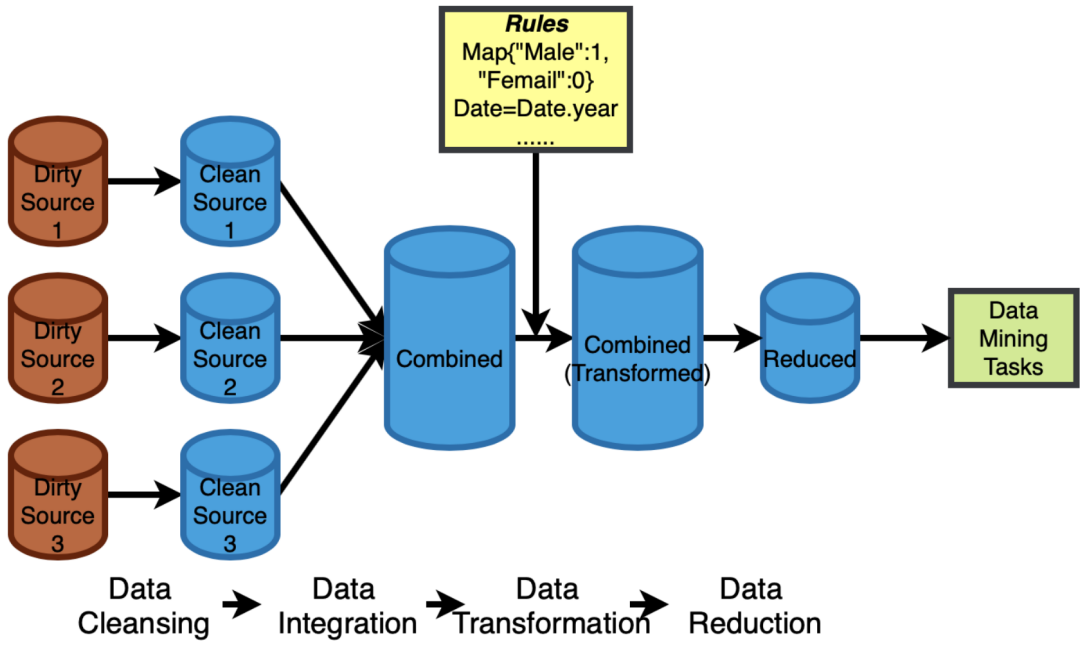


Figure 2.6: A Data Preprocessing Pipeline

2.6.1 Principle Component Analysis

PCA is a widely-used, non-interpretable dimension reduction technique that extracts a smaller number of new features from the original dataset to create a lower dimension (compared to its original dimensionality) representation of the original dataset. Different from feature selection methods where existing, important features are selected according to some importance metrics, PCA decides which new features, or principle components (PCs), to extract based on their ability to explain variances in the original dataset. The ratio of the variance explained by the PCs within the total variance is called the *explain ratio* of the projection. Being a non-interpretable method means that it is hard and non-intuitive for humans to understand why a PC is generated, other than the fact that it explains more variance than the others. It can be a drawback when dealing with tasks where interpretability is a requirement (e.g., in medical science [Katuwal and Chen, 2016; Vellido, 2019] and studies involves ethics [Liu et al., 2018]). However, most of the neural network (NN) based models are not interpretable themselves, so there is no point in restricting the dimension reduction methods to be interpretable.

2.6.2 Federated Data Preprocessing

Our dataset (see Section 2.7) is a high quality dataset with little noise or inconsistency. However, since it is never the case for real-world datasets [Hernández and Stolfo, 1998], we should always consider applying data preprocessing techniques to ML tasks. In an FL context, one approach is to ask the clients to undertake the steps themselves and provide the FL framework a clean dataset. There are two problems with this approach. Firstly, there is no universal standards for data preprocessing techniques. Actions undertaken by different clients might be diversed. Also, due to privacy concerns, individual actions might not be shared with others in detail, making it hard to have consistent data from different clients. Secondly, each client only considers their own dataset when performing the preprocessing steps. PCA, for example, reduces the dimensionality and specifically maximises the explained variance ratio for the applied dataset. This might cause issues in a federated environment as our goal is to build a shared model that captures patterns beyond single datasets. In other words, a more global vision is required. Therefore, we decide to incoperate the data preprocessing step, specifically PCA in our case, to the FL pipeline so that the data preprocessing steps can have a global standard and a more general view over the entire field. Details of the federated data preprocessing implementation will be covered in Section 3.2.4.

Note that we are using feature extraction (i.e., PCA) over feature selection methods in our study. Details of the reasoning will be covered in Section 3.2.4 and we will also provide a possible alternative implementation of using feature selection methods for the same process.

2.6.3 Feature selection

Feature selection (FS) refers to a broad category of dimension reduction techniques, which, in contrast of feature extraction methods like PCA, reduces data complexity and dimensionality by retaining part of existing features and ignoring all others completely [Miao and Niu, 2016; Khalid et al., 2014]. FS methods have been widely used in CV, bio-informatics and many other ML tasks for its good dimension reduction performance and its ability to enhance interpretablity and explainability of the result (i.e., how well human researchers can explain why a result is generated by applying domain knowledge) [Miao and Niu, 2016; Pirovano et al., 2021; Haury et al., 2011]. FS methods require some ranking metrics among the features to be applied to decide which features to retain or ignore. Such metrics are based on statistics [Chandra and Gupta, 2011; Bejan et al., 2012], information theory [Sebban and Nock, 2002; Hancer et al., 2018], or heuristics [Diao and Shen, 2015] and can sometimes come with high calculation complexity. In some cases [Bolón-Canedo et al., 2015; Zhao et al., 2016], researchers use FS methods in a distributed environment to cope with the calculation complexity that comes with big data and to increase the efficiency for such methods. These distributed variations often requires a voting mechanism to unite a

global set of features to select. A similar mechanism will be covered in Section 3.2.4.2.

2.7 Nanotechnology Case Study

In recent years, scientists have been exploring possibilities of applying machine learning algorithms to nanoparticles and material science extensively [Stocks and Barnard, 2021; Yin et al., 2021; Parker and Barnard, 2021] to acquire data-driven insights of the field. However, as identified by Barnard et al., one of the challenges of applying ML methods to nanoparticles is the small size of nanoparticles datasets. Reasons for such challenge includes high cost of raw materials as well as the high cost of generating experiment results (e.g., highly precise instruments requirement). Creating synthetic data for specific materials could mitigate this issue but it still requires high-performance computers to simulate the results with extensive domain knowledge.

One possible solution is to use several different datasets measuring the same features and for the same purpose (e.g., predicting material catalyst activity) from different sources. This may help scientists discover patterns beyond a specific material. However, difficulty comes from the lack of comprehensive source data since many scientists tend to dedicate to a limited set of research areas. Additionally, collaboration with other scientists may not be feasible due to privacy or confidentiality concerns. Therefore, our work proposes solving the collaboration privacy problem with a federated learning framework.

Our work uses eight synthetic nanoparticle datasets, each of which produced by high-performance computer simulations. They are designed to have the same set of 105 features and one goal (i.e., catalyst activity of that material) while all being numeric values. Each of these datasets is measured either on different materials (silver, gold, palladium, and platinum), using different method (i.e., MD and DFTB), under different conditions (thermodynamically or kinetically limited), or with different sizes (i.e., number of atoms) to reflect data diversity between collaborators. This is a suitable case study for our purpose of research because nanoparticles data is usually measured and collected in formats of tabular data and small in volume and overlaps with the context of our research question. It also adds challenges to the task because conventionally, FL techniques are mostly used in classification tasks with non-tabular data while our tabular datasets require a regression solution.

2.8 Evaluation Metrics

The goal of our work is to build a high-performance model for all the clients. Therefore, we need to verify the performance of a model using some evaluation metrics. We measure our training results using two metrics: the Mean Squared Error (MSE)

and Mean Absolute Error (MAE). The reason of using two different metrics is that MSE tend to put high penalties on larger errors while MAE treats all the prediction errors equally [Chai and Draxler, 2014]. This can be important for identifying outliers in the dataset and better perceive the result. If the two metrics both show low losses, it indicates our result model has a relatively consistent and good performance. Mathematical details of the two metrics will be covered in the Methodology Chapter.

2.9 Summary

In this chapter, we provided a detailed breakdown of our project's background, the FL technology and related topics with extensive literature reviews. We would like to proceed to the next chapter where some important mathematical expressions and derivations are provided as supporting evidences of our project methodology.

Chapter 3

Methodology

3.1 Outline

In this chapter, we will provide some important mathematical expressions and derivations for the techniques that we introduced in the Background Chapter. The purpose of this is to support our understandings of these base techniques and provide theoretical background for our innovative modifications. We will follow the order of the pipeline execution, that is data preprocessing, model training, and evaluation, to provide our derivations in this chapter.

Section 3.2 will be focusing on PCA and our federated adaption of it. Section 3.3 talks about stochastic gradient descent, which is the basis for the FedSGD and FedAvg algorithms (see Section 2.5.3). Section 3.4 talks about the original FedAvg algorithm and our implementation that uses weights in the process. Finally, Section 3.5 focuses on the evaluation metrics we use to measure our results.

3.2 Principle Component Analysis

PCA is built for minimising the MSE caused by projecting the original data $\boldsymbol{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_m}\} \in \mathbb{M} \times \mathbb{N}$ onto a hyper-plane in dimension K, where $K << N$, by maximising the variance captured by the PCs, denoted by $\boldsymbol{u_1}, \boldsymbol{u_2}, \ldots, \boldsymbol{u_k}$. We find the PCs in an iterative approach, with the first PC, $\boldsymbol{u_1}$, calculated by maximising the Lagrangian given by:

$$\boldsymbol{u_1}^T \boldsymbol{S} \boldsymbol{u_1} + \lambda_1 (1 - \boldsymbol{u_1}^T \boldsymbol{u_1}) \tag{3.1}$$

where λ_1 is a factor and S is the covariance matrix defined as:

$$\boldsymbol{S} = \frac{1}{K} \sum_{k=1}^{K} (\boldsymbol{x_k} - \overline{\boldsymbol{x}})(\boldsymbol{x_k} - \overline{\boldsymbol{x}})^T \tag{3.2}$$

$\boldsymbol{u_1}$ is calculated to be an eigenvector of $\boldsymbol{S}$ while λ_1 is the largest eigenvalue. Similarly,

other PCs can be calculated, and the final projection matrix we use to reduce the dimensionality of $\boldsymbol{X}$ can be denoted as $\boldsymbol{U} = \{\boldsymbol{u_1}, \boldsymbol{u_2}, \ldots, \boldsymbol{u_k}\} \in \mathbb{K} \times \mathbb{N}$. The ratio of variance captured can be calculated by divide the sum of eigenvalues of the PCs by the sum of all eigenvalues of the covariance matrix, which can be represented as:

$$R = \frac{\sum_{i=1}^{K} \lambda_i}{\sum_{j=1}^{N} \lambda_j} \tag{3.3}$$

We can see that for a dataset $\boldsymbol{X} \in \mathbb{M} \times \mathbb{N}$, the denominator of R stays constant while R grows as K increases. Therefore, the closer R is to 1, the closer the projected hyperplane is to the original dataset, and the better it captures all patterns.

3.2.1 Federated PCA

The federated adaptation of PCA assumes each dataset to have their unique distribution and wants the final projection to reflect such uniqueness to some degree. To have even contribution from each client, we sum all the projection matrices $\{\boldsymbol{U_1}, \boldsymbol{U_2}, \ldots, \boldsymbol{U_n}\}$ and take the average as:

$$\overline{\boldsymbol{U}} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{U_i} \tag{3.4}$$

and use $\overline{\boldsymbol{U}}$ as the final projection matrix for all clients.

In order to get different contributions from different client data distributions, we put weights $w_1, w_1, \ldots, w_n \in \mathbb{R}$ on each projection matrix. $\overline{\boldsymbol{U}}$ is generally represented as:

$$\overline{\boldsymbol{U}} = \frac{1}{n} \sum_{i=1}^{n} w_i \boldsymbol{U_i} \tag{3.5}$$

When using datasets sizes as the metric, $w_i = \frac{|\boldsymbol{X_i}|}{\sum_{j=1}^{n} |\boldsymbol{X_j}|}$; and when using explained variance ratio, $w_i = \frac{R_i}{\sum_{j=1}^{n} R_j}$, where R_i is the explain ratio defined in equation 3.3.

3.3 Stochastic Gradient Descent

SGD is a common method of iteratively optimising a loss function L with regards to a dataset $\overline{\boldsymbol{X}} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_m}\} \in \mathbb{M} \times \mathbb{N}$ given the assumption that L is a convex function. The average loss of a dataset can be represented as:

$$L(w) = \frac{1}{m}\sum_{i=1}^{m} L_i(w) \tag{3.6}$$

where L_i is specifically related to the i^{th} observation in the dataset, and w is the parameters we wish to optimise L against. Upon every observation of the dataset, w can be updated as:

$$w = w - \eta \Delta L(w) \tag{3.7}$$

$$= w - \frac{\eta}{m}\sum_{i=1}^{m} \Delta L_i(w) \tag{3.8}$$

where η is the learning rate of the model.

SGD differentiates from standard gradient descent in the way that it chooses a random subset of data to start the optimisation process and iteratively use all data provided. This helps avoiding local minimums and reduces overhead from training on large datasets when L is complex.

We also used momentum in the SGD process, which adds part of the previous result to the current optimisation process. Denoting the k^{th} update as w_k, using momentum, we have:

$$w_k = w_{k-1} - (1-\beta)\frac{\eta}{m}\sum_{i=1}^{m} \Delta L_i(w) + \beta w_{k-1} \tag{3.9}$$

where $\beta \in [0, 1]$ is the ratio we want previous updates to affect our current update. The effect of momentum in SGD can be illustrated from the comparison in Figure 3.1.

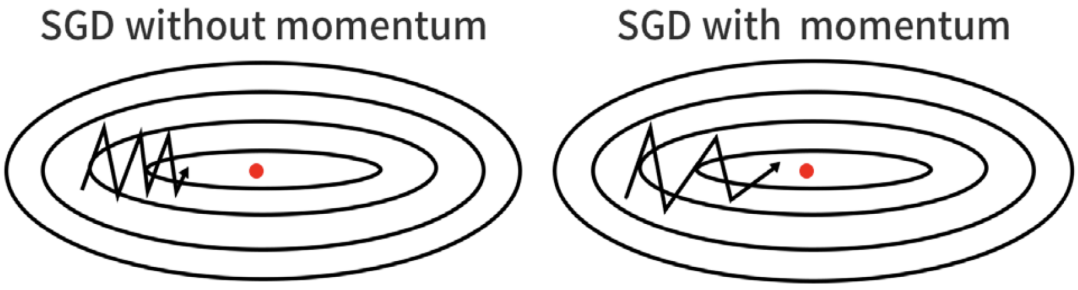


Figure 3.1: Comparison between SGD with and without Momentum [Du, 2019]

where previous result not only cancels out part of the current gradient that does not

contribute towards the optimal direction but also promotes the result towards its optimal direction.

3.4 Model Aggregation

In a federated context, we assume that clients perform SGD locally and send the updated w to the federator. In our specific models, w includes the weights and biases for each model layer.

We represent updates as $w_1, w_2, \ldots, w_k$ for the k clients in the training session and represent the aggregated update $\overline{w}$ as:

$$\overline{w} = \frac{1}{k} \sum_{i=1}^{k} \lambda_i w_i \tag{3.10}$$

where λ_i is the weight for the i^{th} update. In FedAvg or FedSGD, $\forall i \in [1, k], \lambda_i = 1$. For our weighted modification of FedAvg, λ_i is the ratio of the metric value for that specific client in the sum of all clients, that is:

$$\lambda_i = \frac{E_i}{\sum_{j=1}^{k} E_j} \tag{3.11}$$

where E is the evaluation metric value. This could be dataset size and cross-validation score in our implementation.

Since data heterogeneity is a major challenge in the FL context [Li et al., 2020b], it is vital to analyse a general convergence behaviour of the system given non-i.i.d. data. We followed the deduction from Li et al., Haddadpour and Mahdavi, and Khaled et al. and got the following expression for a relationship between the number of local updates in one round of communication E, and the learning rate η:

$$\|\tilde{w}^* - w^*\|_2 = \Omega((E-1)\eta) \cdot \|w^*\|_2 \tag{3.12}$$

where Ω hides some constants from the equation, $\tilde{w}^*$ is is the solution produced by FedAvg with a small enough constant η while w^* is the optimal solution for w. The equation shows that with a constant learning rate, the model will eventually converge to a sub-optimal solution that is at least $\Omega((E-1)\eta)$ away from the optimal. On the other hand, another theorem represented as:

$$\mathbb{E}[F(w_T)] - F^* \leq \frac{\kappa}{\gamma + T - 1}\left(\frac{2B}{\mu} + \frac{\mu\gamma}{2}\mathbb{E}\|w_1 - w^*\|^2\right) \tag{3.13}$$

where

$$B = \sum_{k=1}^{N} p_k^2 \sigma_k^2 + 6L\Gamma + 8(E-1)^2 G^2 \tag{3.14}$$

with $L, \mu, \sigma_k, G, \kappa, \gamma$ defined as problem specific constants and Γ being the degree of heterogeneity measured on the dataset, indicates that with a decaying learning rate η and $E > 1$, FedAvg can converge to the optimal solution. Therefore, in our implementation, specifically the hyperparameter optimisation part (i.e., grid search), we provide only decaying learning rate options for the algorithm.

3.5 Evaluation Metrics

In this section, we provide mathematical representations for the evaluation metrics. Let $\boldsymbol{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_m}\} \in \mathbb{M} \times \mathbb{N}$ be the dataset, and $\hat{\boldsymbol{Y}} = \{\hat{y_1}, \hat{y_2}, \ldots, \hat{y_m}\} \in \mathbb{M}$ be the regression targets. $\boldsymbol{Y} = \{y_1, y_2, \ldots, y_m\} \in \mathbb{M}$ are the predictions made with regards to $\boldsymbol{X}$, then MSE is defined as:

$$MSE = \frac{1}{M}\sum_{i=1}^{M}(y_i - \hat{y}_i)^2 \tag{3.15}$$

such that the closer the predictions are to the actual values, the less the loss. This, however, puts more penalty on larger prediction errors than smaller ones due to the square of the difference. MAE deals with this by the definition:

$$MAE = \frac{1}{M}\sum_{i=1}^{M}|y_i - \hat{y}_i| \tag{3.16}$$

This way, both large and small errors are equally penaltised by their difference with the expected value.

Chapter 4

Design and Implementation

4.1 Outline

In this section, we cover details of our FL implementation, which is a critical and majority part of this project. The source code we developed is publically available on GitHub [Huang]. We spent most of the time of the project building the framework from scratch mainly for three purposes.

Firstly, we want to have an in-depth understanding of the infrastructure. This allows us to better spot difficulties and design alternatives in the implementation process to suit our research needs. Secondly, the implementation process allows us to have full control of the complex system and test whatever new feature we want in our system much easier compared to possibly ending up with manipulating the existing packages such as TensorFlow Federated.

Lastly, since federated learning is a framework designed to be used in real industrial scenarios, a large proportion of its technologies have already been well established. The core of such frameworks is how to properly assemble its components such that it works as expected. Therefore, by manually implementing the pipeline, we get to understand not only the basic components, but also build up a systematic view of how these components work towards our goal.

Among the three models we introduced in the Section 2.5.2 (i.e., horizontal, vertical, and FTL), we focus on the horizontal model applied in a cross-silo scenario. This is because we set our specific scenario in this project to be with data centres with similar research goals. In this case, we can have a smaller number of data owners compared to the number of mobile device owners, which can be millions. It is also more realistic that we emulate the cross-silo scenario and get more accurate results in our local machine.

Also, cross-silo clients are more likely to have well-formatted and clean data related to the learning goal, which our nanoparticles datasets satisfy. A better network connection is as well assumed for the clients. So, we do not have to consider issues like

dropping-out in cross-device scenarios.

We separate the section into three sub-sections. In Section 4.2 we talk about the implementation of the general pipeline. This includes communication between federator and clients, encryption, data preprocessing, and model update methods. Specifically, we describe our innovative way of performing PCA in a federated context and model aggregation in the model update step. In Section 4.3 we talk about the ML training process in our pipeline, especially model design, training functions, and the hyperparameter tuning mechanism. In Section 4.4 we compare out implementation with the existing TFF package.

4.2 General Pipeline

4.2.1 Outline

We split our FL pipeline into four stages, namely connection establishment, federated data preprocessing, training, and ending. A pipeline is shown in Figure 4.1, with process number labelled on the lines. It is an adaptation of conventional FL pipelines in the literature, where we put an extra federated data preprocessing step between connection establishment and the actural training processes.

In the next two paragraphs, we provide a one-round walk-through example of the whole federated learning pipeline to help readers better understand the process. Pseudo-code of the walk-through is shown in Algorithm 2 and Algorithm 3.

At initialisation, a federator is provided with information including the designated model, hyperparameters, number of clients, and its public and private keys for encryption. The federator then starts listening on a port of a hosting server and waits for clients to connect. Clients are initialised with their public and private key, too. Clients will try to establish connection with the federator. After connecting, a client would send out its public key (process 1) and wait for the response from the federator. The federator would save information about that client, send out training related parameters to the client and keep waiting until it has got the expected number of clients. After establishing connections with all the clients, the federator will send its public key to all the clients (process 2) and both the federator and the clients would go into the federated data preprocessing stage.

Within the stage, a set of standard dimension reduction operations will be performed, whose details will be covered in Section 4.2.4, and the clients would obtain a reduced version of their local dataset through the federated process (process 3), and both the clients and the federator would enter the training and reporting stage. In this stage, clients would perform their local trainings following the parameters given by the federator in the first stage. After training, it will report back the latest information

of the model in an encrypted message (process 4). The federator, after receiving updates from all the clients, will perform a model aggregation that combines all the information into a new global model. This new model is then distributed to all the clients (process 5) as a starting point for a new round of training. After several communication rounds of model information, the training will come to an end and both clients and the federator would go into the final end stage and terminate the process.

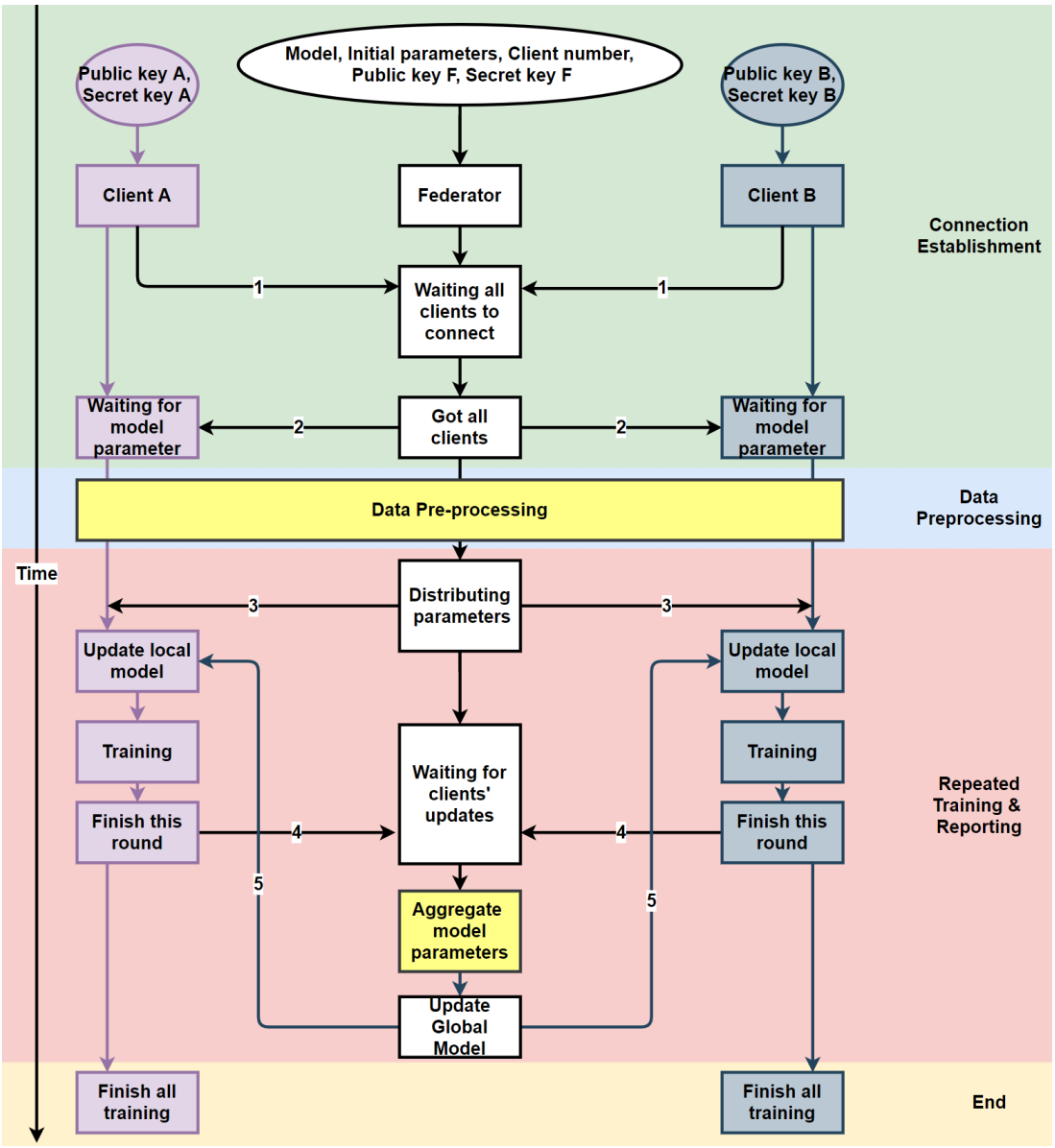


Figure 4.1: Our pipeline for federated learning.

According to the pipeline we described, we identify that client-federator communi-

cation is the basis of all other operations while some other dependencies also exist. A dependency relationship is shown in Fig 4.2. We follow the order of the dependencies from left to right to start our detailed explanation of the components in this section.

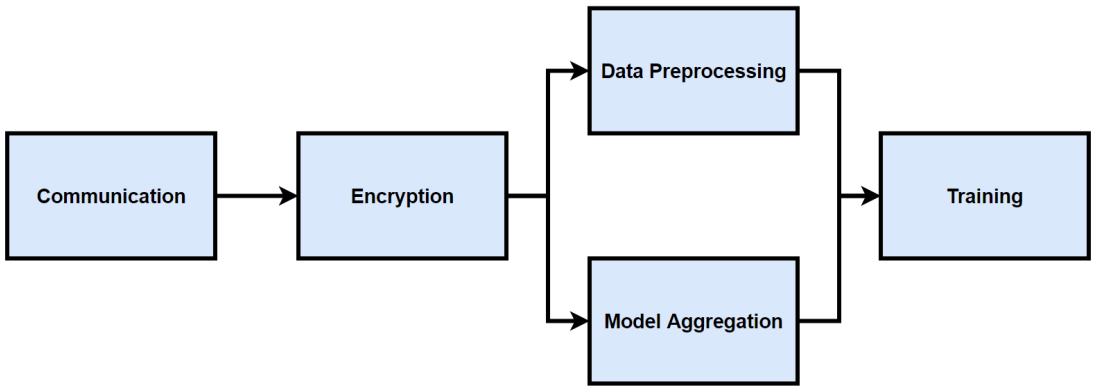


Figure 4.2: Dependencies between components. Arrow "pointing to" means "is basis of".

4.2.2 Communication

The basic communication functionality was built using the Python socket module that provides low-level interfaces to Transmission Control Protocol (TCP) communication between multiple network endpoints.

In a server-client setup, either real-world or emulated on a local host, a client should be able to communicate with the server at any moment without a long waiting time. This is also necessary in the FL pipeline. When establishing connections or reporting local updates, clients might arrive at the federator with local information in any order at any time and it is impossible to predict it. Reasons for such differences include client hardware differences, dataset statistical heterogeneity (e.g., sizes, feature quality, etc) and many more. The client information that gets passed is either one's basic information, such as the public key or identity, or training-related information, such as gradients and biases. Such information varies in size and requires specific actions from the federator. It can take a relatively long time for the federator to process such information. If the federator works in a single-threaded way, a long processing time could cause blocking or even timeout errors on clients trying to communicate with it. It is also not computationally efficient to process requests one at a time when multiple cores are available. Therefore, we design our federator to be a multi-threaded server that can deal with asynchronous requests, such that communication establishment requests can be processed interleaved with the client information processing process. Such design, compared to a single-threaded version,

provides the federator higher availability and reduces turn-around time on the client side. Algorithm 3 provides a pseudo-code for the multi-thread federator implementation, where the `SpecificFederatorProcessFunction` represents the client data processing function.

We could abstract different stages in our pipeline into a same set of actions that clients and the federator carry out repeatedly. That is, clients train locally for some time, and synchronise to wait for actions on the federator to finish. Then, clients are released back to local work again and such process repeats. On observing this pattern, we designed the entire FL process in the way similar to Algorithm 3, only changing the `SpecificFederatorProcessFunction` to different functions for different stages. In addition, we let the federator transfer to the next stage when one stage finishes so that it knows what actions to take for the incoming client information.

4.2.3 Encryption

Encryption guarantees information integrity during the communication processes. Two stages that we intensively used the encryption techniques are data preprocessing and training stages, where in both cases, our focus are messages that carry client data information (i.e., principle components or model parameter matrices).

As introduced in Section 2.5.6, it is good practice to use HE such that even the federator would not possess information about client data. Also, some researchers choose to implement BatchCrypt to reduce the large overhead of encryption and decryption calculations. However, one of the requirements to perform HE is to have all the clients encrypt and decrypt using the same public-private key pair. This makes the HE process different from normal public-key encryption algorithms like RSA. In HE, both the public and private keys are kept on a client. The server would receive the encrypted values, aggregate them and return the updated values to the clients. A client can then use its private key to decrypt the updated value. No further encryption from the server is needed as the updated value is itself encrypted. An illustration of the implementation is shown in Figure 4.3.

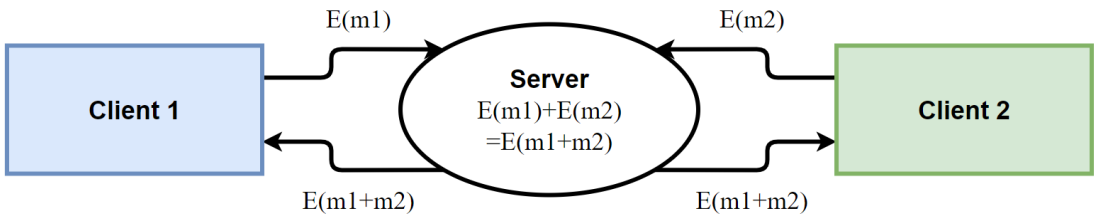


Figure 4.3: Homomorphic encryption process between a server and two clients. $m1$ and $m2$ are client values and E is the same HE algorithm.

We implemented an initial version of the FL pipeline using the Python `paillier` module, an additive HE package, following this procedure. After several simple and naive tests, as expected, we observed a huge overhead of encryption and decryption, which took up more than 90% of time in training rounds. We then made a decision on whether to implement the BatchCrypt algorithm in our pipeline. By implementing it, the FL framework is more complete and closer to what we read about in the literature, but it will potentially take a long time due to its heavy mathematical background that we are not familiar with. On the other hand, not implementing it means we have to find a suitable alternative in this case. We eventually chose to mitigate the risk of BatchCrypt implementation taking long time and use RSA encryption with a relaxation assumption.

We first compared the time consumption of the RSA and HE to establish the relative time difference that is caused by this change. As it turned out, there is little difference between performing HE and RSA same matrices. Therefore, we take the fact as these two algorithms would perform similarly from a time perspective. This is important because we do not want significant differences caused by this change. Based on this similarity, our relaxed assumption is that the federator is an honest participant and does not care about client data for its own benefit. In other words, we choose to trust the federator. Our alternative method is to convert the entire matrix into a formatted string and use RSA on that string to reflect the batch processing technique in BatchCrypt. We avoided using the actual BatchCrypt technique at the cost of the federator knowing the client updates.

To ensure robustness of the encryption, we chose to use a 2048-bit long RSA key. This key length, however, is proven to be not post-quantum safe in 2019. That means a quantum computer can decrypt the encrypted message without the key in a relatively short period. But that is one extreme case we do not consider in our study. The 2048-bit key is safe in all traditional scenarios.

4.2.4 Federated Dimension Reduction

4.2.4.1 Federated PCA

We choose to use PCA to reduce the dimensionality of our dataset in the data preprocessing stage. The discussion over adopting either feature extraction or FS method lies in our justification on whether we need interpretability for our FL process. From the original McMahan et al. paper we can see that they used a CNN and an LSTM for two machine learning tasks respectively. Also, in many later literature [Li et al., 2020a; Rieke et al., 2020], neural networks (NNs) have been more extensively used and studied in the FL context compared to less complex but explainable models such as linear regression or decision trees. According to our assumption in the study, FL has the ultimate goal of helping clients to gain data-driven insights in their field of interest. Under this assumption, NNs are generally better models compared to lin-

ear models in discovering higher degrees of complexities in many areas. Therefore, we choose NNs as our main models for this project. Due to the complicated inner structure and the forward and backward pass mechanism of NNs, they are often considered to be non-interpretable. Therefore, we chose PCA as our base method for the dimension reduction functionality.

For PCA to work in a federated environment, there are two approaches. Naively, each client would perform PCA completely locally, without any information from other clients, with a target number of PCs. The number could be a hyperparameter to the pipeline, or communicated and agreed among the clients. The drawback for this approach is that the clients would only project its local data onto a space with consideration of its own data only, which does not capture a global insight. Our modified approach is built upon this basis. After the number of PCs is shared among the clients, each client would send their PCs to the federator for aggregation. A weighted-averaged PC is shared to all the clients for the projection. In this way, all the clients are reducing their dimensionality based on a global distribution of the data from all clients and would contribute more to the potentially unaware patterns beyond their local scope. The detailed communication process is shown in Figure 4.4.

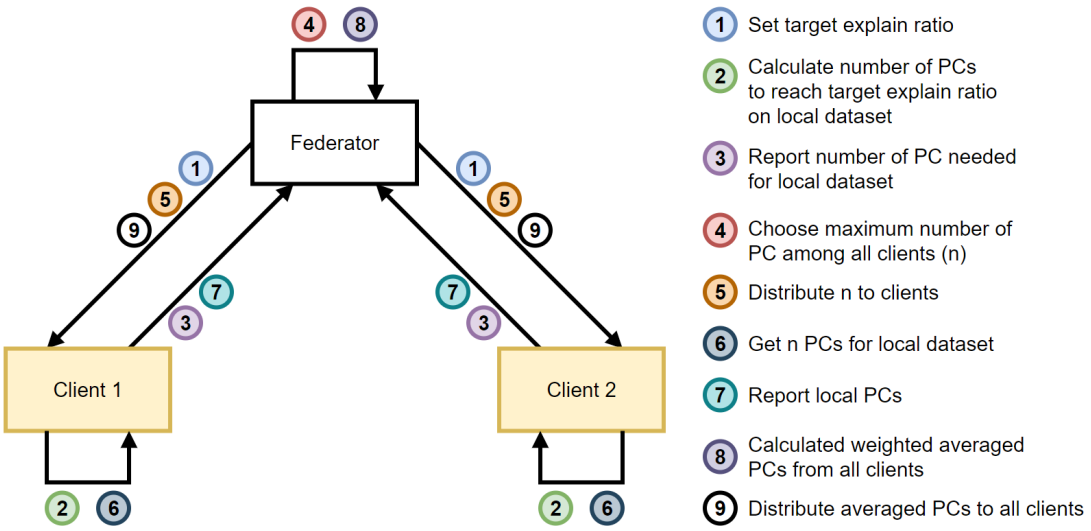


Figure 4.4: Multi-round communication for the federated PCA process with steps explained.

To decide the weights of each PC, which is step 8 in Figure 4.4, we have come up with three strategies, which will be performed globally. To start with, we use even weights. This would not take into account any dataset specific biases and allow even contributions by all clients to the projection matrix. Alternatively, we could weigh the PCs based on the size of the original dataset of that client. In this way, we are favouring clients with larger datasets as they might capture a large portion of the trends in

the field with their dataset. Clients with smaller datasets could also benefit because they will essentially have a much broader vision that is only captured through larger amount of data. However, it also means trends in these smaller datasets can potentially be hidden by those in the larger ones and it might even result in a worse testing outcome for them.

The third approach is to use the explained ratio of a projection as the weight. When deciding the number of PCs we required, we set a global minimum for explain ratio among all the clients and ask them to calculate how many PCs is required specifically for their local dataset to reach the explain ratio (step 2). Generally, it requires more PCs to capture the same amount of variance on larger datasets compared to smaller ones due to them having higher order of complexity. Therefore, the larger the dataset, the more PC is required. Since the number of PCs is determined globally (step 4), clients with smaller datasets can have higher explained ratio with more PCs. In this way, using the explained ratio as the weight for the PC aggregation process would favour clients with smaller datasets since their higher explained ratio would make the weighted PCs closer to theirs, leading to a focus on their visions and potentially better testing result.

4.2.4.2 Feature selection alternatives

Despite our choice of using PCA, we have discussed an alternative approach of using an FS approach in our dimension reduction step. The overall process is similar to what we have seen in Figure 4.4. One alternative is that we substitute the target explained ratio with an evaluation metrics threshold against the specific FS methods we use and filter the set of features. Features passing the threshold will be passed to the federater. Based on some voting mechanism, such as metric values carried with each of the features, a set of features will be chosen from all the clients' results to be the global feature set we distribute back to the clients. Another alternative is that we set the number of features required and generate a fixed number of features from each of the clients. The first method could lead to a relatively large but also more comprehensive feature set since it guarantees the metric value threshold we set. The second method, due to having a fixed number of features to select, will be more likely to have better dimensionality reduction guarantee.

4.2.5 Model Update

When a client finishes all its local training, it will report the latest model information back to the federator. The information includes model weights and biases that is required to represent a model's status given a fixed structure. The federator, after receiving updates from all the expected clients, will start to aggregate the updates and produce a new global model. How the new model performs on the entire field

of study depends heavily on how the model information is aggregated.

We have come up with three aggregation metrics for the process. Similar to the PC aggregation metrics, we used both even contribution and size of the dataset as our metrics. The advantages of these two metrics are also similar to those in the PC aggregation case. We focus on introducing our thrid metric, the cross-validation score of the local models.

The cross-validation score comes from the K-Fold cross-validation technique in the local training process (covered in Section 4.3). It represents a model's ability of predicting unseen data. We use an accumulated MSE value in the validation process to represent how much loss is generated by the validation data. The higher the loss, the worse the model performs. We decided to use the averaged loss because as a dataset gets larger, there are also more data instances to calculate losses on. It could thus result in a higher accumulated loss even individual loss might be smaller than other models. Ratio of this individual loss value among all clients is used as the weight when performing aggregation of the models to focus on a model's prediction ability.

4.3 Training Process

As mentioned in Section 2.5.4, one could use an existing ML algorithm easily in an FL environment. For our learning task, a standard yet highly customisable training process is implemented to provide the flexibility on testing from a model building perspective (i.e., training a well-performing model). The entire training process is encapsulated into one interface in our implementation and theoretically it could be replaced by another training function with correct input and output. The structure of our training function is shown in Algorithm 1.

Algorithm 1: Training structure

```
for i := 0 ... numOfFolds do
    validationFold, trainingFolds ← SplitDataset(localData, numOfFolds);
    for tFold ∈ trainingFolds do
        for n := 0 ... numOfEpochs do
            for batch ∈ miniBatches do
                model ← Train(model, batch);
            end for
        end for
        cvScore ← Validate(validationFold);
    end for
end for
End();
```

In the pseudo-code, *model* refers to the base model we choose when initialising the system from the federator side. We implemented several simple models in the system to cover the naive ML task in our case study. We used a Linear Regression model as the base of the linearality models and added both LASSO and Ridge regression regularisation options on that. For non-linear models, we have a simple Multi-Layer-Perceptron regressor (MLP Regressor) as well as a three-layer Deep Neural Network (DNN). Within the `Train` function, we iteratively optimise the model using SGD with momentum to help with faster convergence. All models are based on the PyTorch package with some modifications to suit our system.

We have also introduced hyperparameter tuning mechanism in our training process as a necessary step in many ML tasks. Besides the decision of using FedAvg or FedSGD (see Section 2.5.3) and the hyperparameter changes from the decision, we have identified learning rate and momentum as two key factors for the SGD process in general. Thus, for each of the folds in our K-Fold implementation, we tune the hyperparameters using a manually implemented grid search strategy with the MSE cross-validation score as our metric for judging performance of a set of hyperparameters.

The hyperparameter tuning process, unlike federated PCA, was implemented on a client level. That means in each communication round with the federator, the locally optimised hyperparameters, in this case learning rate and momentum, are not shared with the federator or other clients. This is due to consideration that what is locally optimal might not be global. We enable the system to generate a final shared model but also allow each client to obtain a set of hyperparameters suitable for its local dataset. This also adds extra security of clients' local datasets because without knowing the local hyperparameters, it is impossible to reconstruct the exact model with only model weights and biases. On the other hand, this approach could potentially result in issues such as updated model being inconsistent due to hyperparameter difference between clients. But we would like to test and see how it will perform.

4.4 Comparison with TFF

We would like to compare our implementation of FL (referred to as OFL, Our Federated Learning, in this section, to differentiate with the TFF APIs) with that of TFF so that we understand our differences with the state-of-the-art implementation. Since the TFF project is still in development, we take the latest stable version documented (TFF 0.19.0 based on tensorflow 2.5.0) as our subject of study in this work.

TFF provides the Python language with APIs to access its lower-level implementations in `C++`. Two sets of APIs are available for different purposes: Federated Learning (`FL`), which provides high-level interfaces for applying existing ML models

to a task, and Federated Core (FC), which provides low-level interfaces for building customised federated algorithms such as using dataset size as metrics for model aggregation. What FL provides is similar to the what we discussed in Section 4.3 where we both abstract the local training process as a single function. The only difference is that TFF allows a general ML model to be plugged in, that includes standard ones built on Keras or PyTorch, while OFL requires a customised model class we created to suit our special purposes. Although the process of converting a PyTorch model to the model class in OFL is simple and intuitive, TFF is a more generic tool and is more convenient in this process.

From a low-level view, FC provides the flexibility of implementing custom federated computation processes such as the FedAvg algorithm or different aggregation metrics in model aggregation stage. The FC APIs are strongly typed and required extensive background knowledge in federated learning. Many of the implementation details are hidden under the heavy use of Python decorators and it can be less straightforward to work around the relationships between different computational components. This implementation, however, is designed for a broader federated computation environment rather than merely the server-clients setup we usually see in a federated learning context. Therefore, we believe OFL is an easier and clearer implementation designed specifically for federated learning tasks. On the other hand, through implementing the federated data preprocessing functionality in OFL, we feel the focus of TFF and OFL is quite different since we do not have access to the most basic communications (i.e., send and receive functionalities) in TFF while they are very important parts of the OFL implementation for customisation.

Finally, from a real-world application point of view, TFF supports being deployed using docker containers, a virtual environment for convenient and platform independent deployments, while our implementation is more exploratory and experimental. Although we designed our emulated server and clients to communicate through a real and generic host-port setup, there is no extended optimisation or security guarantee on that so far. Therefore, we still consider TFF to be a more complete development environment than OFL.

4.5 Summary

In this chapter, we provided implementation details of our FL pipeline as well as comparisons from three aspects. We have covered the most important design decisions and justifications on them. Although some might seen intuitive after reading, they were results of a careful and systematic design and reflects our expectation of focusing on and understanding the federated context and process. The key takeaway from this chapter is that even though our implementation is different and not as flexible as the Google TFF implementation, it is reliable enough for our goal of

understanding the federated learning context and have our own customisation on its components. As there is no fundamental difference between our understanding of the role of an ML training process in this pipeline with that in the TFF package, given the same client datasets and training process, we would expect very similar results from both implementations.

Also, aside from the pipeline itself, highlights of our implementation are our innovative federated data preprocessing stage as well as the aggregation methods for both the PCs and the model information. We would like to share and discuss our results of introducing these components into our FL system and also ML tasks related statistics in the next chapter.

Chapter 5

Results

5.1 Outline

In this chapter, we share the testing results and analysis on the case study of nanoparticles using FL technology. Testing in this project is focused on two aspects. First, we need to validate the FL pipeline is properly implemented by seeing how well an untuned model performs compared to a traditional centralised training scenario. Second, we need to test the innovative components in the pipeline and see if they provide better results. We will also try to analyse how the change in testing results relate to differences of the client datasets from both the nanoparticles domain perspective and data science perspective.

In section 5.2, we will discuss our testing plan for the system, which includes identifying the testing items and then set corresponding benchmarks. We would provide our definition of results being positive or successful and our justifications of these definitions. These benchmarks will be split into individual aspects of the system for us to focus on one at a time. And in section 5.3, we examine the detailed display and analysis of the experiment results for these items individually. When all components are individually tested, we demonstrate the result of the system using the best settings we found in the previous tests.

5.2 Testing Plan

We divided tests into the general pipeline working ability tests and ML model building ability tests. We did not test the privacy-preserving feature for two reasons. Firstly, we know that our implementation is relaxed using the RSA encryption (see section 4.2.3) so the model information is known to the federator. Second, it is a known fact that given model gradients and biases, it is possible for a malicious third-party to approximate client datasets using Generative Adversarial Networks (GANs) [Wang et al., 2019; Zhang et al., 2019] and it remains an open question to solve which we do not cover in our study. Other than these two factors, we also acknowledge our encryption method is robust enough against most attacks (see section 4.2.3) so we do

not consider information leaks during communication.

For the first tests, we verify the pipeline by comparing its behaviour on simple tasks with that of a centralised training task with all data aggregated. According to Yang et al., due to statistical heterogeneity of client data, an accuracy loss is expected from the result of FL generated models compared to that trained in a centralised way. In our case, since the dataset is purely synthetic, meaning a rather clear relationship between the prediction goal and the features can be discovered more easily than a completely natural dataset with much higher levels of random noises, we expect a rather small accuracy loss when comparing these two results. Secondly, we check whether our system will perform better on i.i.d. data than non-i.i.d. data. We aggregate and shuffle all client data and re-assign it to the clients with their dataset sizes unchanged. After training, we expect a higher accuracy (or lower loss in our case) than the previous non-i.i.d. data experiments but still a lower accuracy than the centralised benchmark. Note that in both cases, we would use PCA for dimension reduction. And we use an averaging metric for the federated case.

We then explore how our innovative PC and model aggregation methods make a difference to the original pipeline results. We test these methods independently and then combine the methods. The benchmark here is the model trained using average for both PC and model aggregation methods because these are all tests in the federated environment. All other parameters should be controlled during these experiments.

After untuned experiments, we select the aggregation method combination with the best training results to test our hyperparameter tuning mechanism. We will tune the learning rate and momentum (if it exists) of the model in our study automatically and manually optimise the number of epochs and batch size. According to Li et al. and many others who studied the FL convergence behaviours, in each tuning round, we only set the learning rate to a smaller number by a multiply coefficient of a number slightly smaller than 1 (e.g., 0.99, 0.98, etc).

Lastly, we would like to see how far the FL pipeline can help reducing the amount of required data on the clients' side. This can be important when each of the clients wants to reduce the cost of data collection while also building a reliable and robust model. We could give an estimation of the reduction in model performance with less data. We aggregate, shuffle, and divide all data from clients into several datasets and train on these new clients using the same FL pipeline. We identify how much data is required on each client for the final result to have an acceptable accuracy loss. Note that we do not have a clear definition for acceptable so we would try to observe the pattern of reduced data, more clients, and the learning accuracy. The benchmark in this case would also be training on a centralised set as it maximises model performance and trade-off a client has to accept by providing less data.

5.3 Result Analysis

5.3.1 Federated Learning

We first set our benchmark on an untuned FL task with an 85% expected explain ratio, one epoch each training round for 1000 rounds on a linear regression model. We have the MSE loss of the test set over model iterations plotted as Figure 5.1.

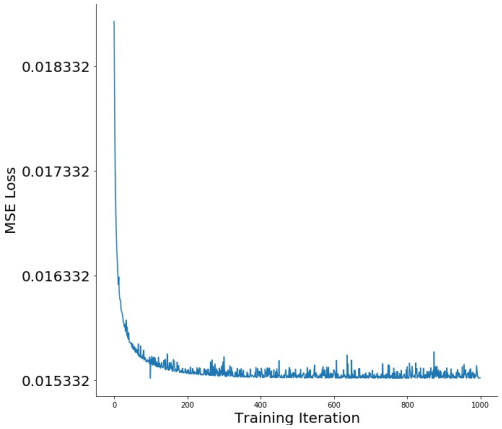


Figure 5.1: MSE losses for centralised linear regression training.

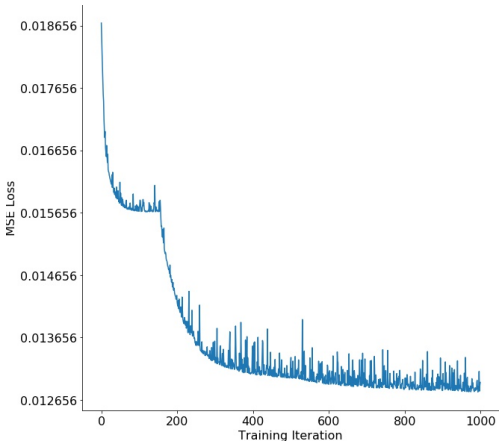


Figure 5.2: MSE losses for centralised MLP-Regression training on the first 1000 rounds.

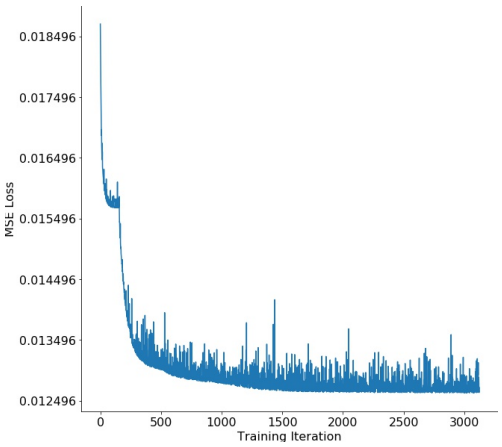


Figure 5.3: Continue training model in Figure 5.2 for more rounds.

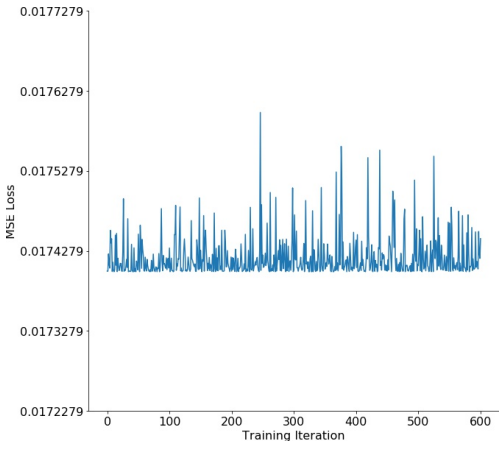


Figure 5.4: First 600 rounds with 2 epochs each round on a MLP-Regression model.

We found that after about 300 rounds of training, the model reaches convergence under this setting. The minimum loss is slightly greater than 0.015 and fluctuates at

a small scale. We then used the same set of parameters on a MLP-Regression model with one hidden layer with four neurons. The first 1000 training rounds give us Figure 5.2, which has not yet converged. Therefore, we trained it for more rounds. As seen in Figure 5.3, the model converges to a loss of 0.0126 after 2000 rounds.

We have also tested using two epochs on the MLP-Regression model. For the first 600 rounds of training, we see the result as shown in Figure 5.4, which does not converge very well compared to previous results and fluctuates a lot (due to overfitting). Therefore, we use the result from Figure 5.3 as our centralised benchmark since it considers potential non-linear relationships in the dataset using a simple NN and also converges to a lower loss.

We then train the model using our FL pipeline on the eight nanoparticle datasets. For the FedSGD case, we have per-client and total losses shown as Figure 5.5 and Figure 5.6 respectively. Notice that the models have mostly converged at about 25 rounds of communication and a total loss of about 0.18 is achieved. Compared to results from centralised training, the federated case is worse in prediction accuracy. However, the total loss is accumulated over more than 10,000 data instances among all client datasets. It is already a very good result.

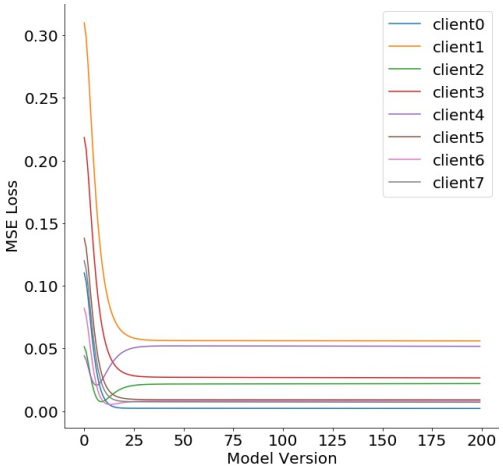


Figure 5.5: Per-client losses of an MLP-Regression model.

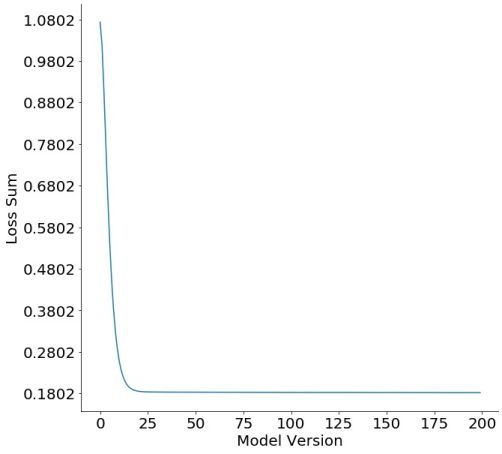


Figure 5.6: Sum of all model losses.

We have, however, observed unusual behaviours in the per-client losses shown in Figure 5.5. Namely, client4 (Ag-quantum), client2 (Ag-classical), and client6 (Au-kinetic) has show losses growing after the first few communication rounds of decreasing. An investigation into properties of those datasets (e.g., dataset size) on those clients did not find direct causes of such behaviour. Therefore, we switched to a linear regression model to perform the same experiment again to see if there is any difference. The result of it is shown in Figure 5.7 and Figure 5.8. We can see that although the

total loss is greater than that in Figure 5.6 and the decreasing tendency is less steep in Figure 5.7, all three clients still show increasing in loss after the first few communication rounds. Therefore, we believe that such increase is caused by the central model learning a general pattern from all the datasets that are more suitable on the other datasets than these three. In other words, there might be some differences in physical properties or experimental differences from these three datasets compared to the others. We provide a possible explanation below.

Client2 and client6 are measured on the most disordered and anisotropic nanoparticles datasets among all. They are expected to be very physically messy and thus having similar properties. According to the dataset profile, these two datasets were generated under very similar conditions (Kinetically limited, with temperature between 273K to 973K), using the same method (MD), and having very similar nanoparticle sizes. Therefore, it is possible that their behaviours on the models are different from the other datasets. Client4, on the other hand, has very physically neat and tidily ordered particles which are also small in sizes (number of atoms). However, it is measured using a totally different method (DFTB) under different conditions (thermodynaically limited with temperature being 0K). Upon observing their behaviours on the model, we could conduct further domain specific experiments from this perspective to validate the reason for this behaviour and have a good explanation on the model prediction behaviour.

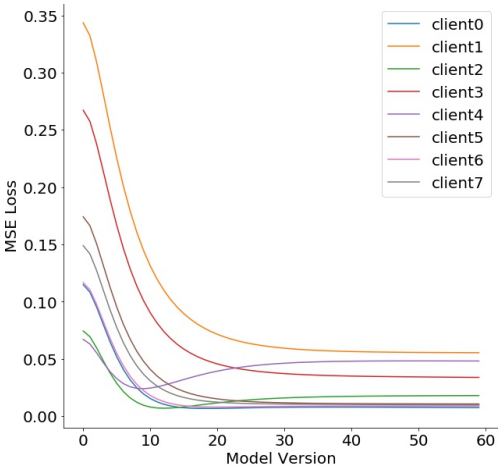


Figure 5.7: Per-client losses of a linear regression model.

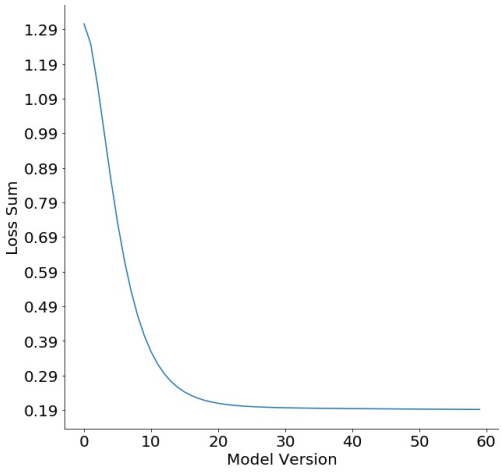


Figure 5.8: Sum of all model losses.

In addition to setting the benchmark using centralised training on all data combined, we also carried out experiments where individual models were trained using local data only. The models performed very well and surpassed the individual performance of that we observed in Figure 5.7. This was not expected but we have several hypothesis on this behaviour. Firstly, our synthetic datasets have very trivial relation-

ships between features and the regression goal. The benefit of collaboration might not be obvious when individual datasets are capable of building good models already. Secondly, the shared model only captures the general trend in the domain, which should be further fine-tuned to fit each client's specific need. While both could be valid explanations, we believe the latter is closer to the essence and goal of the FL technology. Therefore, we consider this performance degradation to be acceptable.

Through the experiments above, we can see our FL pipeline would produce a relatively good shared model for all decentralised clients with reduced accuracy compared to a centralised model. This shows our FL implementation is working correctly. We further address this point in the next section by comparing the results of an i.i.d. dataset following the convention we set in Section 5.2.

5.3.2 I.I.D. Assumption

To avoid coincidences in the training behaviour, we have shuffled and split the original datasets three times and trained using the same set of hyperparameters to observe the behaviour of FL on an i.i.d. dataset. Note that here i.i.d. only refers to the feature distribution of the datasets, that is, instances measured on different items being put in the same dataset. We did not change the number of data instances for a client in order to compare the result against the original datasets.

We measured both MSE and MAE on all three experiments. Results from the first group are shown in Figure 5.9 - 5.12 and the other two groups are shown in the appendices. From all three results we can see both MSE and MAE losses are very small and specifically, MSE has reached lower values compared to that in Figure 5.6 but greater than the centralised training results from Figure 5.1 and Figure 5.3. This means the iid-ness of the client datasets would indeed affect the result of the FL pipeline (see section 2.5.5). It also means our implementation is correct and is able to reproduce results from previous literature.

5.3.3 Federated PCA

In this section, we analyse test results on the three PC aggregation metrics: average, dataset size, and explain ratio. For faster convergence, we follow results by McMahan et al. and adopt large epoch and mini-batch strategy for our experiments. We compared the three methods under 5, 10, and 20 local epochs each communication round and got results as Figure 5.13.

We can see in both 5 and 10 epochs cases that the "size" and "explain ratio" metrics converge to smaller losses than the "average" metric, while the 20 epochs case shows different results. Also, we discover the smallest MSE and MAE values do not differ significantly between the 10 and 20 epochs groups. Therefore, we believe the

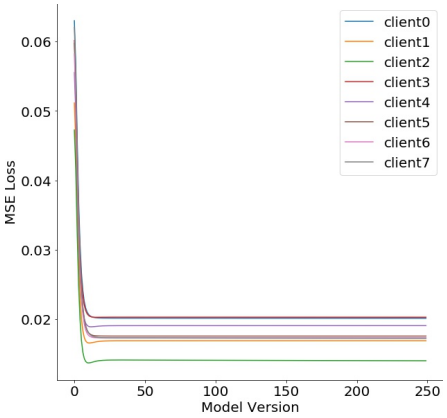


Figure 5.9: Per-client MSE losses for i.i.d. dataset1

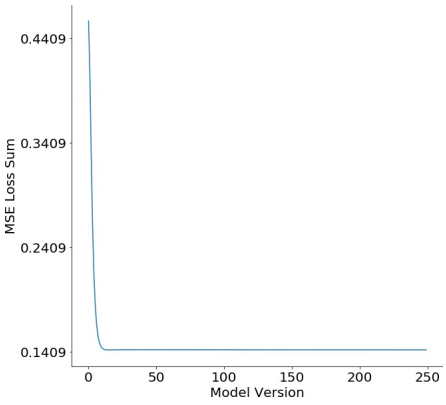


Figure 5.10: Sum of MSE losses for i.i.d. dataset1

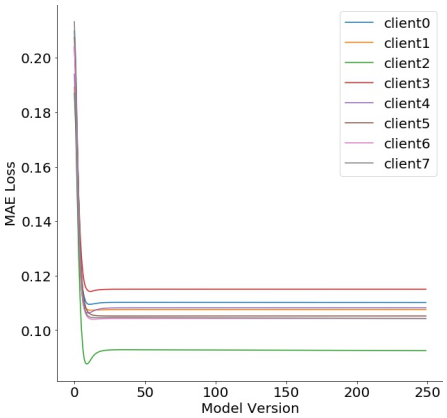


Figure 5.11: Per-client MAE losses for i.i.d. dataset1

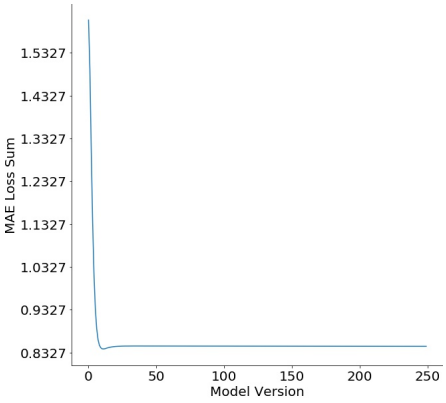


Figure 5.12: Sum of MAE losses for i.i.d. dataset1

two innovative methods used here (i.e., by size and by explain ratio) can accelerate the convergence of the model while achieving similar results of training for more epochs. If we do train for more epochs, the performance might degrade compared to the original "averaging" metric due to overfitting on the individual datasets rather than fitting all datasets well.

To check if our assumption of different PC aggregation method favouring clients of different sizes, Figure 5.13 shows that aggregating by "size" always has the minimum loss variance over model versions (especially in the 20 epochs cases); that is, its loss values vary in a smaller range compared to the other two methods. We believe it is because the "size" metric would favour large datasets such that the final model is better at predicting those datasets thereby minimising the overall loss. However, this is only considering the sum. We would expect the individual loss of the dataset to be

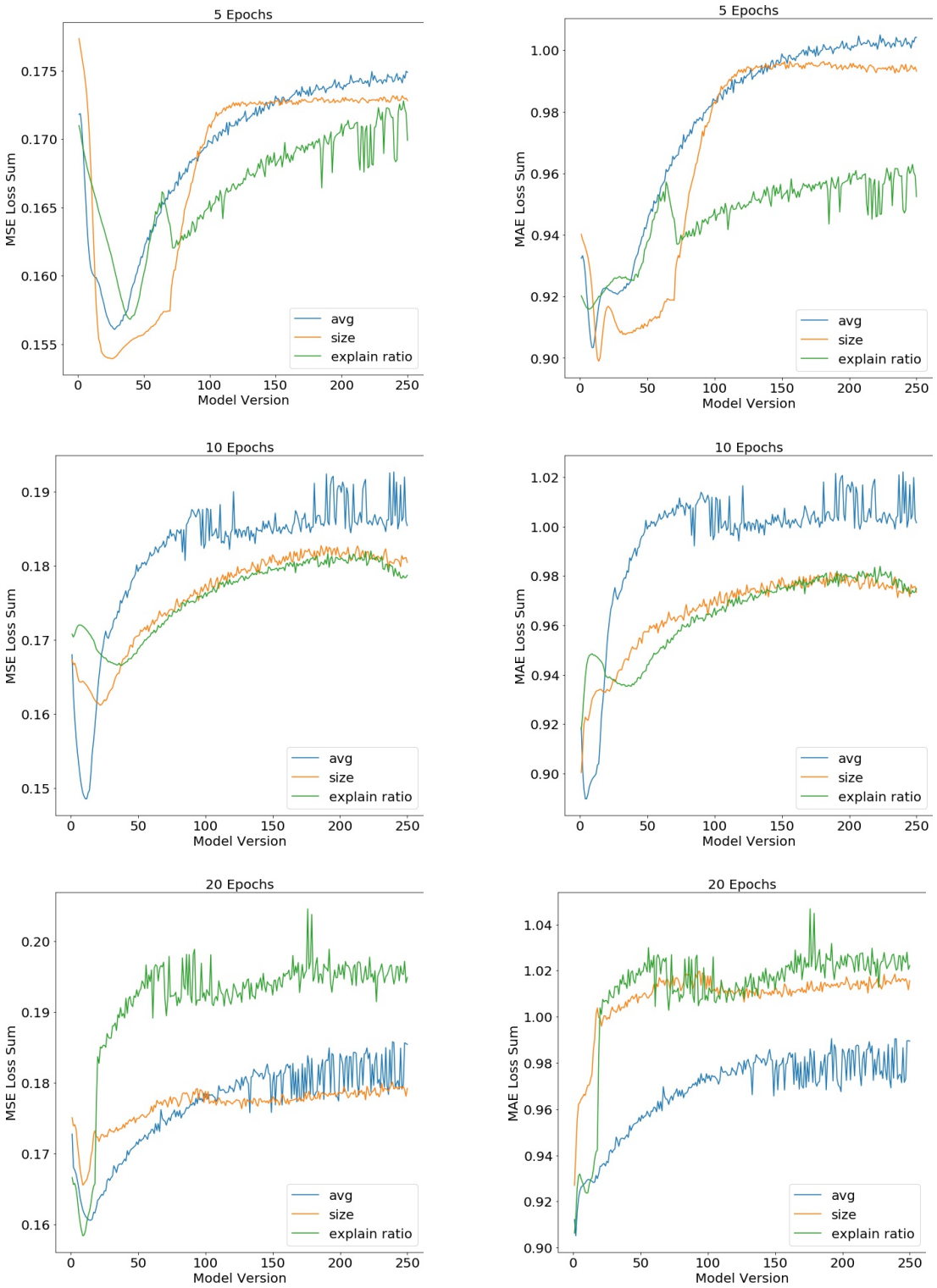


Figure 5.13: Loss sum comparisons between different aggregation methods and epoch numbers. Left is MSE loss and right is MAE loss. All graphs ignored the initial random model result.

lower. To investigate this assumption, the per-client, per-sample losses of the three methods was measured under 10 epochs condition with "averaging" as the benchmark to get the average loss values for each training instance of a client. The results are shown in Figure 5.14.

We can observe for client0 (blue line), which has the smallest amount of data (only 3% of the total amount), its per-sample MSE loss is about 0.0001 for the first two metrics but reduced significantly to 0.00005 under the "explain ratio" metric. Similar observation can be made on its MAE losses. We believe our theory of the "explain ratio" metric favouring smaller dataset is correct in this case. However, we see the exact opposite behaviour on client4 (purple line), which has the second smallest dataset (3.5%). Its MSE and MAE losses are smaller using the "size" metric but larger using the "explain ratio" metric. Also, this effect is not significant on the third smallest dataset (client1 with the orange line). One explanation of this inconsistent behaviour is that the influence of client0's explain ratio is more significant on its own dataset but not general enough on other clients' datasets. Also, although we see better performance of the model using the "size" metric in Figure 5.13 with a sensible explanation, on a per-sample scale, it is not very significant.

From the analysis above, we believe our PC aggregation has limited but positive effects under certain conditions. These conditions include how significant one metric is among all other clients, how many training epochs one uses, and how related each of the datasets are. Although we can see from Figure 5.13 that given a big enough epoch and training rounds, the "averaging" metric would out-perform the other two metrics and achieve low losses, further testing will be undertaken using the 10 epochs setting where both high training efficiency and low losses are achieved by the "size" metric.

5.3.4 Model Aggregation Metrics

We follow the best PC aggregation metric ("size") and carry out experiments on the model aggregation metrics with the same set of hyperparameters. For "averaging", "size", and "cross validation score", we have the sum of MSE and MAE for the test results as Figure 5.15.

The results from the "cross validation score" metric are very unstable and do not converge to a better model. The reason for such behaviour could be that while the cross validation score is calculated every round based on the new model's performance on the a client's dataset, there is no guarantee newly updated model from the federator perform well universally for all clients. This could lead to varied values for the metric, and thus poorly performing models.

Alternatively, the "size" metric improves the prediction ability of models but has

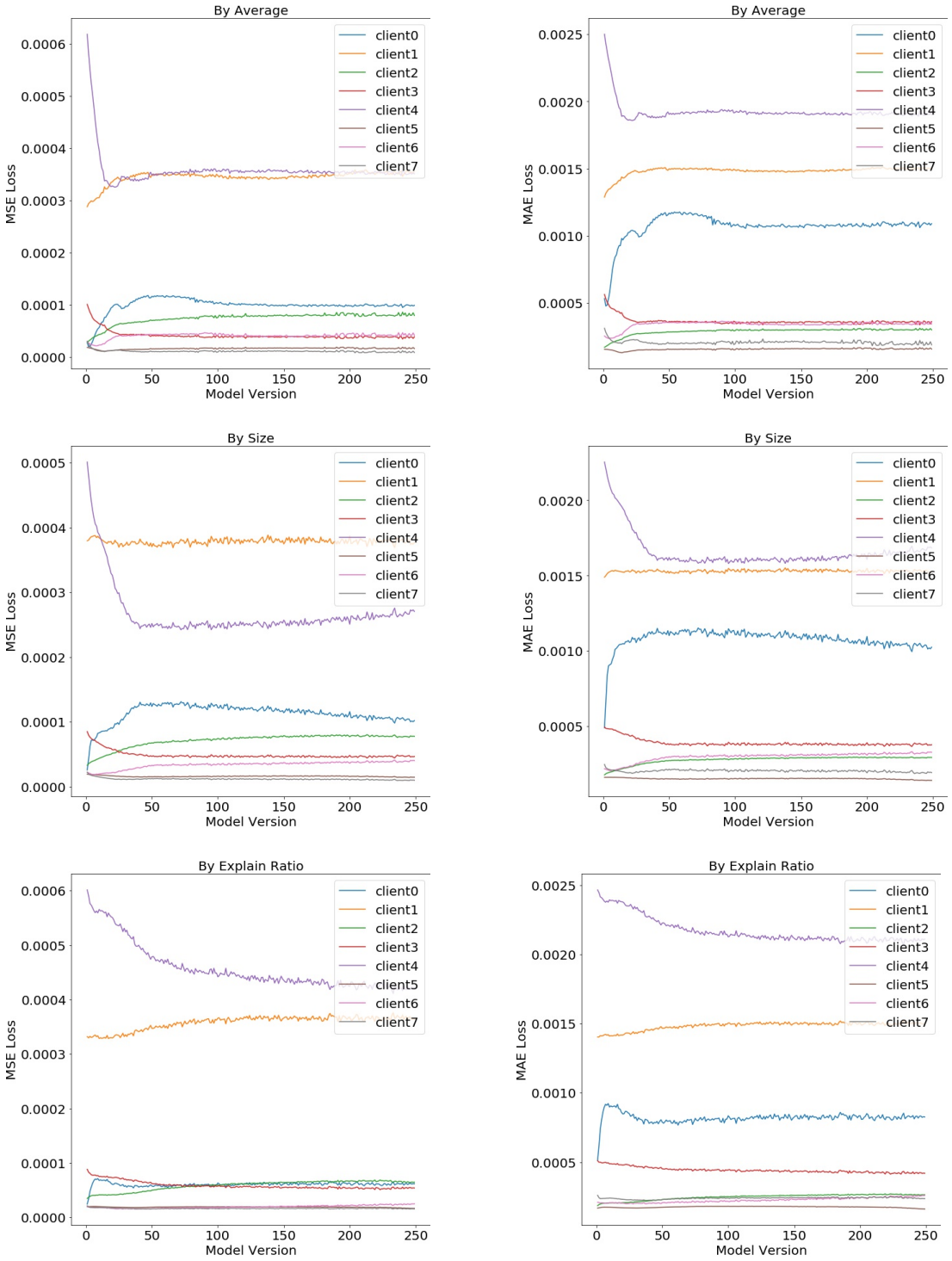


Figure 5.14: Per-client, per-sample losses using different PC aggregation metrics. All graphs ignored the initial random model result.

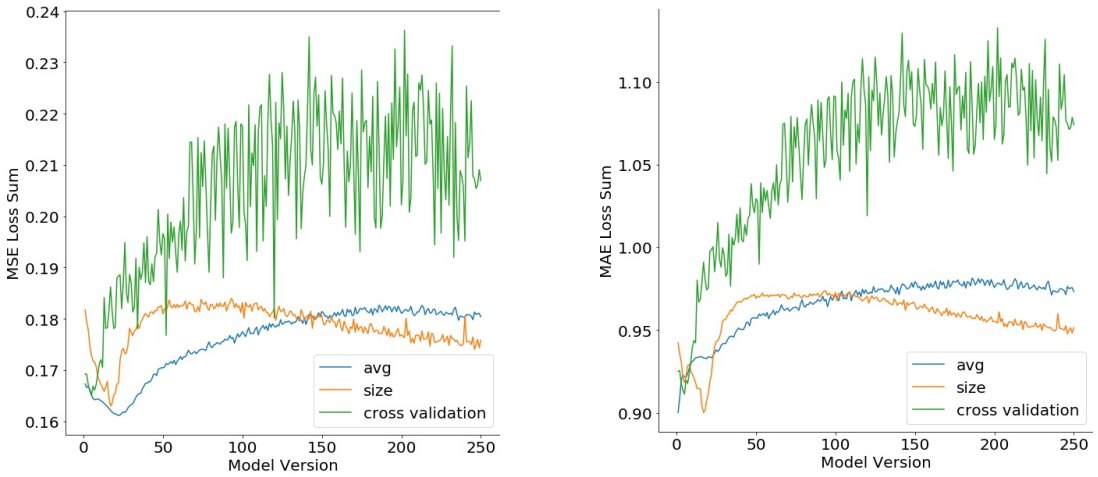


Figure 5.15: MSE and MAE sums for the three metrics.

slower rate of convergence. It also results in higher prediction error at the beginning compared to the "averaging" metric.

From a per-client, per-sample perspective, we show results in Figure 5.16. Firstly, we can see the same unstable behaviour of the "cross validation score" metrics. This is true for all the clients and we therefore consider this a bad metric to use in this case. Secondly, in both "size" graphs, as the model is updated, the only client that has not converged is client0, the smallest dataset among all clients. This behaviour matches our expectation for this metric, that is, it would favour those with larger datasets more than those with smaller ones.

Aside from comparing prediction errors alone, we can also draw conclusions from a domain knowledge perspective. For all tests, we can see very similar behaviours between client2 (green line) and client6 (pink line), especially when we closely examine the MAE scores. Not only are their prediction losses are close in value, their reactions to different model versions are very similar. This matches our domain knowledge expectation of those two datasets, that they are contain nanoparticles that are more structurally disordered than the other datasets and should be similar to each other. This result shows we are able to use the FL pipeline to gain insights into our field of study by observing model behaviours on the datasets. However, limitations also exist in this regard. For example, we also expected client4 (brown line) and client5 (purple line) to have similar behaviours due to both datasets are populated by more ideal zonohedrons (nanoparticles with less structural disorder). The result actually shows quite big difference in those two datasets. Although chances are those two datasets do not have a high physical similarity indeed, we would still recommond taking the results as an indicator rather than the definitive factor when relating with domain knowledge.

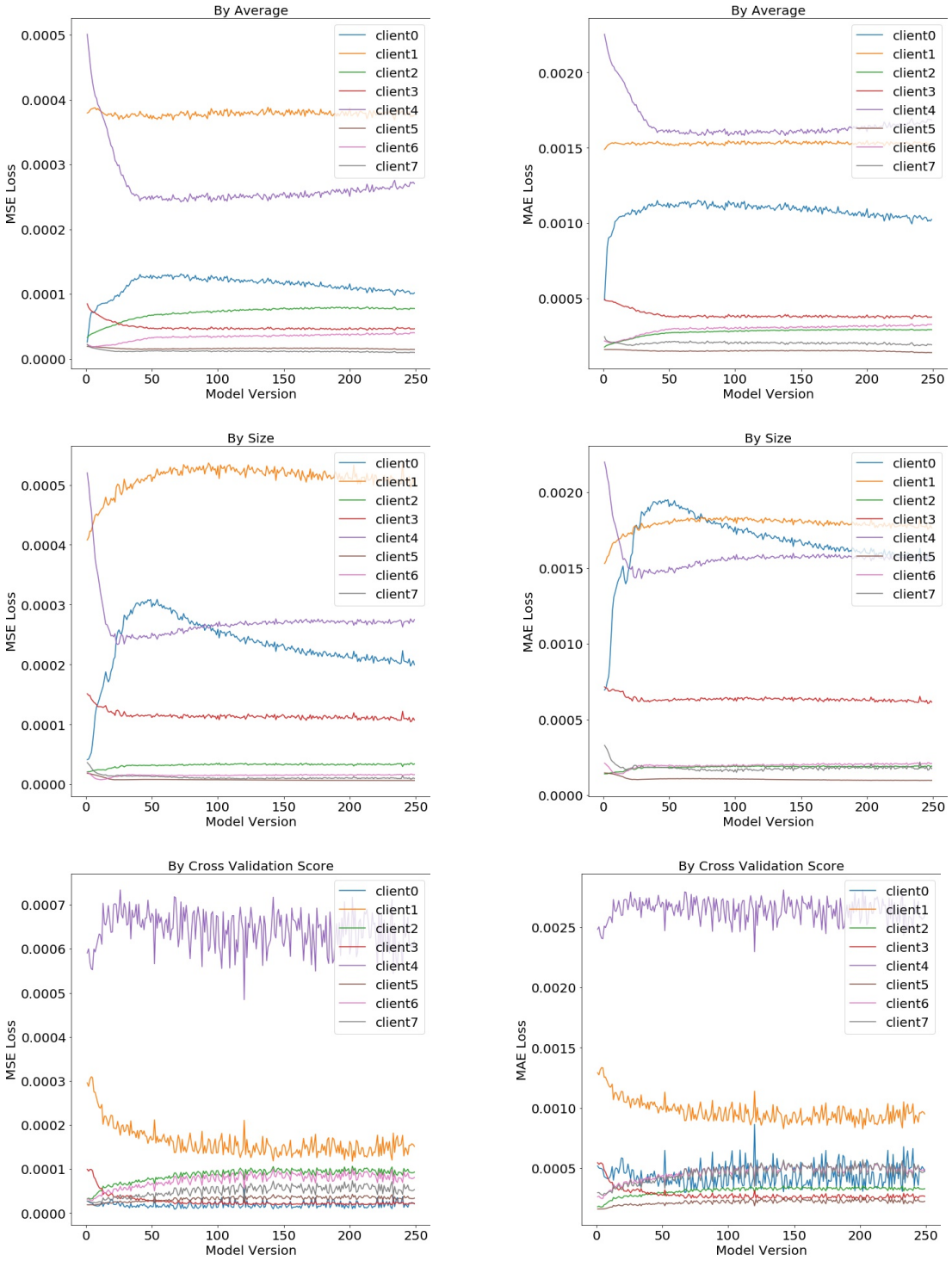


Figure 5.16: Per-client, per-sample losses using different model aggregation methods. All graphs ignored the initial random model result.

5.3.5 Hyperparameter Tuning

Hyperparameter tuning was the most time-consuming part of our experiments. Following the same workflow from previous non-tuning scenarios, we first set the benchmark using centralised training. The parameters we used are from the federated tests we carried out in previous sections (i.e., 10 epochs for each round of training). After approximately 60 rounds of training, the model has reached convergence as shown in Figure 5.17.

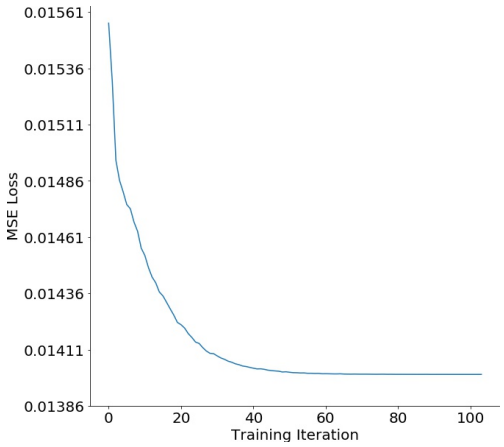


Figure 5.17: 100 rounds of 10-epochs centralised training with hyperparameter tuning.

Compared with the central benchmark without tuning in Figure 5.3, the MSE loss value has increased, which indicated our tuned model is worse than the original model. This can be a result of having too many epochs in one training round as we have seen in Figure 5.4. Therefore, we applied the same tuning scheme on the one-epoch MLP-Regression model to keep it consistent between centralised experiments. The result is shown in Figure 5.18.

Figure 5.18 shows that after 4,000 rounds of training and reaching convergence, the model still performs worse than that in Figure 5.3 even though they have the same epoch number setting. Although we do observe more smooth loss curve over training iterations, we would like to validate the reason why performances has not improved.

There are three possible reasons for the performance drop. Firstly, our original choice of hyperparameters are already suitable for the learning task already and any further tuning is not required. Secondly, the set of hyperparameters we tune is limited and the model behaviour depends on those not tuned. Thirdly, the initial result could be a product of luck, where specific records are chosen by the random algorithm to be included in the test set so that results appears to be good. To test these assumptions, we carried out controlled experiments using significantly different epoch number and mini-batch size and measured their performances using the same set of metrics.

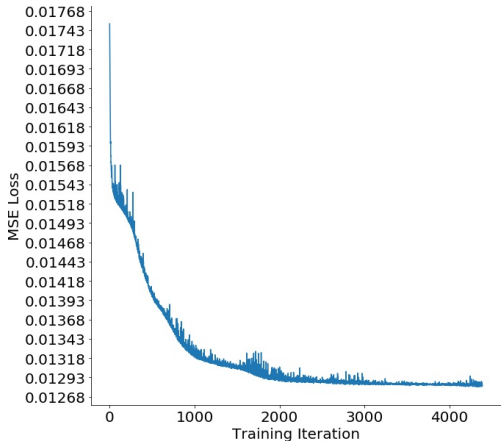


Figure 5.18: Losses from one-epochs centralised training with hyperparameter tuning.

As the results show (see Appendix 3), the models did not surpass the original one. At the same time, however, none of them have significant increases in the prediction loss, either. We carried out another experiment using the same hyperparameters as the original benchmark experiment. In this case (see Appendix 4) the loss was not as low as the original one and it took much longer to converge. Although, based on the trend of the graph, if we continue training, the model will eventually reach a lower loss, this must be traded-off against the low efficiency.

From these comparisons we conclude our initial result to be an optimal but lucky result with the ideal hyperparameters. A more general result, as we can tell from other experiments without hyperparameter tuning, is one with a slightly greater loss value of approximately 0.015, on average. According to this general outcome, we can observe improvements in both prediction losses and convergence time from Figure 5.17, proving that the tuning scheme is indeed effective and efficient.

One important conclusion is that in centralised trainings, we achieve very low loss values in both tuned and un-tuned results with differences at the scale of 0.001. This difference might not be as obvious in an FL scenario due to statistical heterogeneity.

In the first batch of tuning tests, we tune both the learning rate and momentum for each client locally. The result is shown in Figure 5.19.

Compared with the benchmark and previous un-tuned results, this result has a higher loss values and variances at both client level and in summation. We attribute to this controversial behaviour to different clients not sharing the tuned result with the federator, resulting in the updated model not being general enough for all clients. After receiving the latest aggregated model, one client might obtain better results than previous ones while another can have much worse results. That explains the

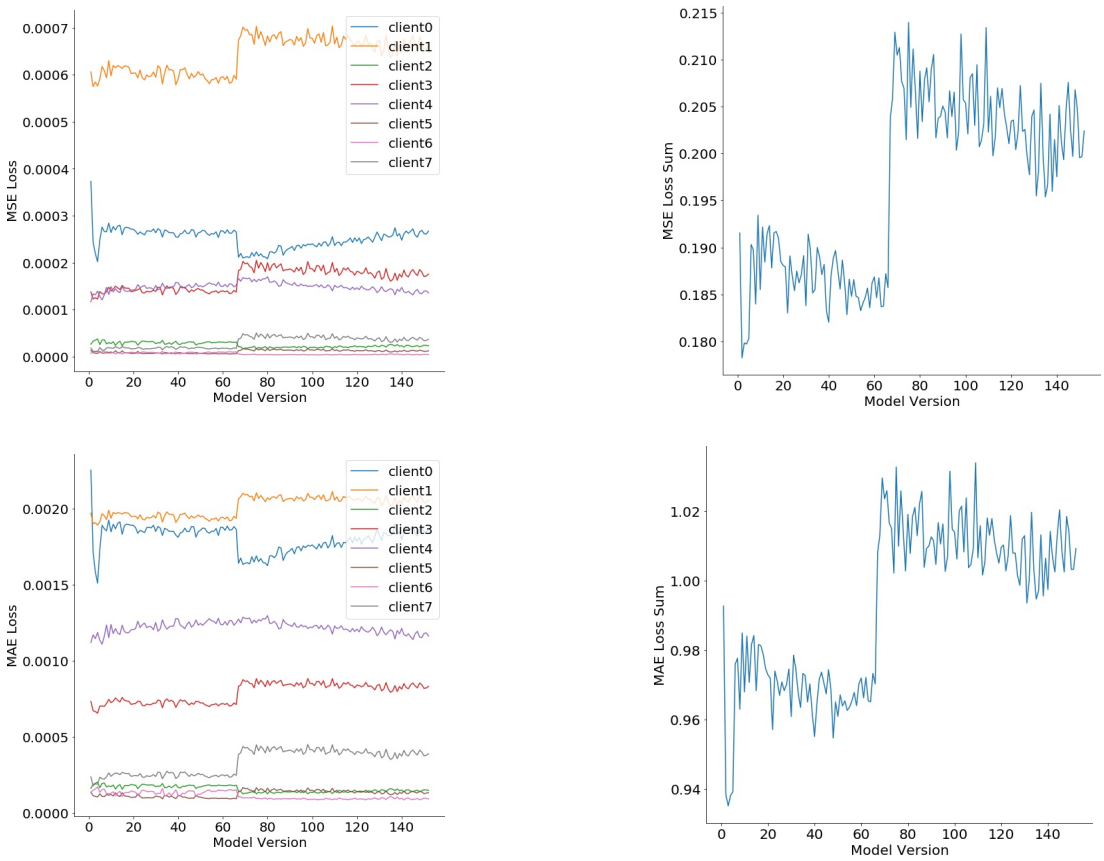


Figure 5.19: Results from tuning both learning rate and momentum locally.

high variance of losses between different iterations. Therefore, we believe our proposal of tuning at a local level without consideration of other clients is not a good scheme.

5.3.6 Less Data & More Clients

In previous sections, we have validated the effectiveness and efficiency of our FL pipeline in performing ML tasks. In this section, we would like to examine the relationship between the number of clients, the amount of data on each client and the learning efficiency. Since we only have eight individual datasets, in order to simulate more clients, we shuffled the datasets and evenly randomly distributed them into n bins as our new client datasets. We control the number of training iterations so as to examine the training behaviour change under the client number change in a resource-limited scenario. Also, pushing the limit of resource requirement implies that we care about the best behaviour possible. Therefore, we kept using the best combination of hyperparameters, including the PC and model aggregation methods. The comparison of MSE loss values after 250 rounds of training of each setup is shown in Figure 5.20. Individual training results can be found in Appendix 5.

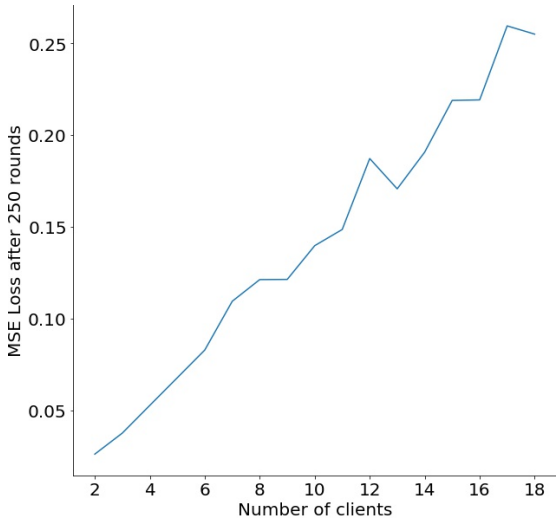


Figure 5.20: Comparison of loss values given different number of clients.

From the graph, we can tell that as the number of clients increases, the total accumulated loss grows approximately linearly. This behaviour can also be a result of having less data on each client. Therefore, in a real-world scenario, given an expected threshold of total loss, we could approximate how many clients and how much data is required to reach that acceptance threshold. However, this conclusion is not robust enough for two reasons. Firstly, the amount of data per client decreases as the number of client increases because we only have a limited amount of data available. The reason for the increasing loss value could be either or both of them but more data is required to validate the true cause. Secondly, we want to retain sufficient data on each client and therefore only split all of it into 18 parts in our experiments as a maximum. Although the trend seems linear from this figure, it might not be the case as the number of clients grows significantly. Again, it would require more data in order to test this hypothesis. Finally, this result might relate to the dataset itself. While this issue appears with these clean data sets, it might be worse with a more sparse or noisy dataset. Further work on this relationship should be carried out on datasets of different qualities to determine if this is the case.

5.4 Summary

In this chapter, we firstly verified our FL pipeline, demonstrating the correctness of our understanding and implementation of the technology. Then we compared our innovative methods for both PCA and model aggregation as well as the local hyperparameter tuning mechanism. Although some results are not as good as we expected, proper justifications along with possible alternatives are provided to assist our future work in this area. Due to the real-world implications of this approach, we further

explored performance drop problems when increasing the number of clients. The experiments still have limitations, but given more data in future studies, we believe it is feasible to use this method as a guideline for clients interested in this problem.

Chapter 6

Conclusion

6.1 Summary & Implication

In this work, we started our study with the intention of mitigating data scarcity issues using the novel FL technology, with a focus on tabular data, which is different from the original scope of FL. We carried out detailed literature reviews, discussing the underlying problem with data scarcity, analysing traditional methods, and introducing the FL technology. Based on the findings, we successfully implemented an FL pipeline from scratch, with an innovative federated data preprocessing mechanism and multiple aggregation metrics in different stages of the pipeline. We tested our system on the eight decentralised nanoparticles datasets and analysed the results. We summarise and conclude main contributions of our work as both the implementation of new components and the foundational study into the FL structure itself.

Our findings in this work have a few important implications to the research field of FL. First of all, our study has successfully validated the effectiveness and efficiency of training ML models in a data scarce context using the FL technology. Although some results (e.g., training loss differences between centralised training and FL) needs further investigation and verification with more data, the overall performance was satisfying within our datasets. Secondly, from a practical perspective, our pipeline is a straightforward implementation of the FL technology. Compared to the existing TFF package, we believe our work helps further researchers understand the FL process faster and more easily from reading and working with the code. Last but not least, upon observing drastic differences from using different aggregation metrics in the PC and model aggregating stages, we believe there might be optimal metrics statistically that helps training a better model. While the statement is worth verifying, it implies future FL practitioners to consider this as a contributing factor towards their training process.

6.2 Future Work

Generally for the FL technology, there are a lot of components that we did not study in detail in this work. Challenges mentioned in Chapter 2 require elaborative studies into each of the domains. This includes researches in cryptography, web-technology, statistics, and even electrocommunication (for more stable network access). In fact, a new framework called Swarm Learning [Warnat-Herresthal et al., 2021] has a structure that is better for privacy-preserving, collaborative machine learning tasks. It is worthwhile exploring new possibilities as well as limitations that come with such innovations.

As for the contents specifically from this work, there are several aspects that can either be further explored or validated. Firstly, we only have a very limited set of aggregation metrics for both PCs and model parameters. Further research into more mathematical and statistical aspects of the process is required to design more and better metrics. Secondly, we only considered PCA in the federated data preprocessing pipeline while there are usually much more actions in a common ML task. Further researches are required to implement a more standard and generic data preprocessing pipeline in a federated context. Thirdly, as mentioned in section 5.3.6, more data from different domains and of different quality is required to further validate some of our results.

Another natural progression of this work is to research the feasibility of applying the FL pipeline on algorithms outside the ML context, such as the widely used choice modelling methods in economics and business that is based on statistics and psychology. Successful application of the FL technique in these novel fields opens new possibilities of how these industries operate in a way that takes better advantage of the era of big data collaboratively.

References

Albrecht, J. P., 2016. How the gdpr will change the world. *Eur. Data Prot. L. Rev.*, 2 (2016), 287. (cited on page 21)

Alharthi, A.; Krotov, V.; and Bowman, M., 2017. Addressing barriers to big data. *Business Horizons*, 60, 3 (2017), 285–292. (cited on page 23)

Amos, N.; Stewart, C.; Bhat, P.; Cretsinger, C.; Won, E.; Dharmaratna, W.; and Prosper, H., 1996. The random grid search: A simple way to find optimal cuts. In *Computing in High Energy Physics' 95: CHEP'95*, 215–219. World Scientific. (cited on page 34)

Arpaci-Dusseau, R. H. and Arpaci-Dusseau, A. C., 2018. *Operating systems: Three easy pieces*. Arpaci-Dusseau Books LLC. (cited on page 33)

Barnard, A.; Motevalli, B.; Parker, A.; Fischer, J.; Feigl, C.; and Opletal, G., 2019. Nanoinformatics, and the big challenges for the science of small things. *Nanoscale*, 11, 41 (2019), 19190–19201. (cited on page 37)

Bejan, C. A.; Xia, F.; Vanderwende, L.; Wurfel, M. M.; and Yetisgen-Yildiz, M., 2012. Pneumonia identification using statistical feature selection. *Journal of the American Medical Informatics Association*, 19, 5 (2012), 817–823. (cited on page 36)

Bernus, P. and Noran, O., 2017. Data rich–but information poor. In *Working Conference on Virtual Enterprises*, 206–214. Springer. (cited on page 21)

Bhat, P. C.; Prosper, H. B.; Sekmen, S.; and Stewart, C., 2018. Optimizing event selection with the random grid search. *Computer Physics Communications*, 228 (2018), 245–257. (cited on page 34)

Bolón-Canedo, V.; Sánchez-Maroño, N.; and Alonso-Betanzos, A., 2015. Distributed feature selection: An application to microarray data classification. *Applied soft computing*, 30 (2015), 136–150. (cited on page 36)

Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedone, A.; McMahan, H. B.; Patel, S.; Ramage, D.; Segal, A.; and Seth, K., 2017. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 1175–1191. (cited on page 30)

Brankovic, L. and Estivill-Castro, V., 1999. Privacy issues in knowledge discovery and data mining. In *Australian institute of computer ethics conference*, 89–99. Citeseer. (cited on page 21)

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al., 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, (2020). (cited on page 20)

Chai, D.; Wang, L.; Chen, K.; and Yang, Q., 2020. Fedeval: A benchmark system with a comprehensive evaluation model for federated learning. *arXiv preprint arXiv:2011.09655*, (2020). (cited on page 28)

Chai, T. and Draxler, R. R., 2014. Root mean square error (rmse) or mean absolute error (mae). *Geoscientific Model Development Discussions*, 7, 1 (2014), 1525–1534. (cited on page 38)

Chandra, B. and Gupta, M., 2011. An efficient statistical feature selection approach for classification of gene expression data. *Journal of biomedical informatics*, 44, 4 (2011), 529–535. (cited on page 36)

Chandrashekar, G. and Sahin, F., 2014. A survey on feature selection methods. *Computers & Electrical Engineering*, 40, 1 (2014), 16–28. (cited on page 34)

Chen, J.; Pan, X.; Monga, R.; Bengio, S.; and Jozefowicz, R., 2016. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, (2016). (cited on page 28)

Diao, R. and Shen, Q., 2015. Nature inspired feature selection meta-heuristics. *Artificial Intelligence Review*, 44, 3 (2015), 311–340. (cited on page 36)

Domingo-Ferrer, J. and Torra, V., 2005. Privacy in data mining. *Data mining and knowledge discovery*, 11, 2 (2005), 117–119. (cited on page 21)

Du, J., 2019. The frontier of sgd and its variants in machine learning. In *Journal of Physics: Conference Series*, vol. 1229, 012046. IOP Publishing. (cited on pages 12 and 41)

Fontaine, C. and Galand, F., 2007. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*, 2007 (2007), 1–10. (cited on pages 31 and 32)

García, S.; Luengo, J.; and Herrera, F., 2015. *Data preprocessing in data mining*, vol. 72. Springer. (cited on page 20)

Haddadpour, F. and Mahdavi, M., 2019. On the convergence of local descent methods in federated learning. *arXiv preprint arXiv:1910.14425*, (2019). (cited on page 42)

Hancer, E.; Xue, B.; and Zhang, M., 2018. Differential evolution for filter feature selection based on information theory and feature ranking. *Knowledge-Based Systems*, 140 (2018), 103–119. (cited on page 36)

Haury, A.-C.; Gestraud, P.; and Vert, J.-P., 2011. The influence of feature selection methods on accuracy, stability and interpretability of molecular signatures. *PloS one*, 6, 12 (2011), e28210. (cited on page 36)

Hernández, M. A. and Stolfo, S. J., 1998. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data mining and knowledge discovery*, 2, 1 (1998), 9–37. (cited on pages 23 and 36)

Huang, W. FederatedLearning. https://github.com/jacobvons/FederatedLearning. (cited on pages 7 and 44)

Hutchinson, M. L.; Antono, E.; Gibbons, B. M.; Paradiso, S.; Ling, J.; and Meredig, B., 2017. Overcoming data scarcity with transfer learning. *arXiv preprint arXiv:1711.05099*, (2017). (cited on page 24)

Kairouz, P.; McMahan, H. B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A. N.; Bonawitz, K.; Charles, Z.; Cormode, G.; Cummings, R.; et al., 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, (2019). (cited on page 25)

Katuwal, G. J. and Chen, R., 2016. Machine learning model interpretability for precision medicine. *arXiv preprint arXiv:1610.09045*, (2016). (cited on page 35)

Ker, J.; Wang, L.; Rao, J.; and Lim, T., 2017. Deep learning applications in medical image analysis. *Ieee Access*, 6 (2017), 9375–9389. (cited on page 21)

Khaled, A.; Mishchenko, K.; and Richtárik, P., 2019. First analysis of local gd on heterogeneous data. *arXiv preprint arXiv:1909.04715*, (2019). (cited on page 42)

Khalid, S.; Khalil, T.; and Nasreen, S., 2014. A survey of feature selection and feature extraction techniques in machine learning. In *2014 science and information conference*, 372–378. IEEE. (cited on page 36)

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E., 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25 (2012), 1097–1105. (cited on page 23)

Laranjeiro, N.; Soydemir, S. N.; and Bernardino, J., 2015. A survey on data quality: classifying poor data. In *2015 IEEE 21st Pacific rim international symposium on dependable computing (PRDC)*, 179–188. IEEE. (cited on page 29)

Le, T. A.; Baydin, A. G.; Zinkov, R.; and Wood, F., 2017. Using synthetic data to train neural networks is model-based reasoning. In *2017 International Joint Conference on Neural Networks (IJCNN)*, 3514–3521. IEEE. (cited on page 23)

Li, L.; Fan, Y.; Tse, M.; and Lin, K.-Y., 2020a. A review of applications in federated learning. *Computers & Industrial Engineering*, (2020), 106854. (cited on page 49)

Li, T.; Sahu, A. K.; Talwalkar, A.; and Smith, V., 2020b. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37, 3 (2020), 50–60. (cited on pages 30 and 42)

Li, X.; Huang, K.; Yang, W.; Wang, S.; and Zhang, Z., 2019. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, (2019). (cited on pages 30, 42, and 57)

Liu, C.; Chakraborty, S.; and Verma, D., 2019. Secure model fusion for distributed learning using partial homomorphic encryption. In *Policy-Based Autonomic Data Governance*, 154–179. Springer. (cited on page 32)

Liu, X.; Wang, X.; and Matwin, S., 2018. Improving the interpretability of deep neural networks with knowledge distillation. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, 905–912. IEEE. (cited on page 35)

Liu, Y.; Yuan, X.; Zhao, R.; Zheng, Y.; and Zheng, Y., 2020. Rc-ssfl: Towards robust and communication-efficient semi-supervised federated learning system. *arXiv preprint arXiv:2012.04432*, (2020). (cited on page 28)

Lucas, P. J.; Robinson, R.; and Treacy, L., 2020. What is data poverty? *Nesta*, (2020). (cited on page 21)

Maletic, J. I. and Marcus, A., 2000. Data cleansing: Beyond integrity analysis. In *Iq*, 200–209. Citeseer. (cited on page 23)

McCool, M.; Reinders, J.; and Robison, A., 2012. *Structured parallel programming: patterns for efficient computation*. Elsevier. (cited on page 33)

McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; and y Arcas, B. A., 2017a. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, 1273–1282. PMLR. (cited on pages 24, 28, 49, and 61)

McMahan, H. B.; Ramage, D.; Talwar, K.; and Zhang, L., 2017b. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*, (2017). (cited on page 30)

Miao, J. and Niu, L., 2016. A survey on feature selection. *Procedia Computer Science*, 91 (2016), 919–926. (cited on page 36)

Mitchell, T. M. et al., 1997. Machine learning. (1997). (cited on page 20)

Pan, S. J.; Tsang, I. W.; Kwok, J. T.; and Yang, Q., 2010. Domain adaptation via transfer component analysis. *IEEE transactions on neural networks*, 22, 2 (2010), 199–210. (cited on page 24)

Pan, S. J. and Yang, Q., 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22, 10 (2009), 1345–1359. (cited on page 24)

Parker, A. J. and Barnard, A. S., 2021. Unsupervised structure classes vs. supervised property classes of silicon quantum dots using neural networks. *Nanoscale Horizons*, 6, 3 (2021), 277–282. (cited on page 37)

Perlich, C.; Dalessandro, B.; Raeder, T.; Stitelman, O.; and Provost, F., 2014. Machine learning for targeted display advertising: Transfer learning in action. *Machine learning*, 95, 1 (2014), 103–127. (cited on page 21)

Pirovano, A.; Heuberger, H.; Berlemont, S.; Ladjal, S.; and Bloch, I., 2021. Automatic feature selection for improved interpretability on whole slide imaging. *Machine Learning and Knowledge Extraction*, 3, 1 (2021), 243–262. (cited on page 36)

Probst, P.; Boulesteix, A.-L.; and Bischl, B., 2019. Tunability: importance of hyperparameters of machine learning algorithms. *The Journal of Machine Learning Research*, 20, 1 (2019), 1934–1965. (cited on page 34)

Qian, F. and Zhang, A., 2021. The value of federated learning during and post-covid-19. *International Journal for Quality in Health Care*, 33, 1 (2021), mzab010. (cited on page 29)

Ridzuan, F. and Zainon, W. M. N. W., 2019. A review on data cleansing methods for big data. *Procedia Computer Science*, 161 (2019), 731–738. (cited on page 23)

Rieke, N.; Hancox, J.; Li, W.; Milletari, F.; Roth, H. R.; Albarqouni, S.; Bakas, S.; Galtier, M. N.; Landman, B. A.; Maier-Hein, K.; et al., 2020. The future of digital health with federated learning. *NPJ digital medicine*, 3, 1 (2020), 1–7. (cited on page 49)

Rivest, R. L.; Adleman, L.; Dertouzos, M. L.; et al., 1978. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4, 11 (1978), 169–180. (cited on page 31)

Sagiroglu, S. and Sinanc, D., 2013. Big data: A review. In *2013 international conference on collaboration technologies and systems (CTS)*, 42–47. IEEE. (cited on page 20)

Sahu, A. K.; Li, T.; Sanjabi, M.; Zaheer, M.; Talwalkar, A.; and Smith, V., 2018. On the convergence of federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 3 (2018), 3. (cited on page 30)

Sarker, I. H., 2021. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2, 3 (2021), 1–21. (cited on page 20)

Sattler, F.; Wiedemann, S.; Müller, K.-R.; and Samek, W., 2019. Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*, 31, 9 (2019), 3400–3413. (cited on page 30)

Sebban, M. and Nock, R., 2002. A hybrid filter/wrapper approach of feature selection using information theory. *Pattern recognition*, 35, 4 (2002), 835–846. (cited on page 36)

Simard, P. Y.; Steinkraus, D.; Platt, J. C.; et al., 2003. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, vol. 3. (cited on page 23)

Souza, R. C.; de Brito, D. E.; Cardoso, R. L.; de Oliveira, D. M.; Meira, W.; and Pappa, G. L., 2014. An evolutionary methodology for handling data scarcity and noise in monitoring real events from social media data. In *Ibero-American Conference on Artificial Intelligence*, 295–306. Springer. (cited on page 23)

Stocks, R. and Barnard, A. S., 2021. Enhancing classical gold nanoparticle simulations with electronic corrections and machine learning. *Journal of Physics: Condensed Matter*, (2021). (cited on page 37)

Vellido, A., 2019. The importance of interpretability and visualization in machine learning for applications in medicine and health care. *Neural Computing and Applications*, (2019), 1–15. (cited on page 35)

Voigt, P. and Von dem Bussche, A., 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10 (2017), 3152676. (cited on page 21)

Wang, Z.; Song, M.; Zhang, Z.; Song, Y.; Wang, Q.; and Qi, H., 2019. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, 2512–2520. IEEE. (cited on page 56)

Warnat-Herresthal, S.; Schultze, H.; Shastry, K. L.; Manamohan, S.; Mukherjee, S.; Garg, V.; Sarveswara, R.; Händler, K.; Pickkers, P.; Aziz, N. A.; et al., 2021. Swarm learning for decentralized and confidential clinical machine learning. *Nature*, 594, 7862 (2021), 265–270. (cited on page 74)

Weerts, H. J.; Mueller, A. C.; and Vanschoren, J., 2020. Importance of tuning hyperparameters of machine learning algorithms. *arXiv preprint arXiv:2007.07588*, (2020). (cited on page 34)

Willemink, M. J.; Koszek, W. A.; Hardell, C.; Wu, J.; Fleischmann, D.; Harvey, H.; Folio, L. R.; Summers, R. M.; Rubin, D. L.; and Lungren, M. P., 2020. Preparing medical imaging data for machine learning. *Radiology*, 295, 1 (2020), 4–15. (cited on page 21)

Wold, S.; Esbensen, K.; and Geladi, P., 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2, 1-3 (1987), 37–52. (cited on page 34)

Xu, J.; Glicksberg, B. S.; Su, C.; Walker, P.; Bian, J.; and Wang, F., 2021. Federated learning for healthcare informatics. *Journal of Healthcare Informatics Research*, 5, 1 (2021), 1–19. (cited on page 29)

Yang, Q.; Liu, Y.; Chen, T.; and Tong, Y., 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10, 2 (2019), 1–19. (cited on pages 12, 26, 27, 28, 30, and 57)

Yin, H.; Sun, Z.; Wang, Z.; Tang, D.; Pang, C. H.; Yu, X.; Barnard, A. S.; Zhao, H.; and Yin, Z., 2021. The data-intensive scientific revolution occurring where two-dimensional materials meet machine learning. *Cell Reports Physical Science*, (2021), 100482. (cited on page 37)

Zhang, C.; Li, S.; Xia, J.; Wang, W.; Yan, F.; and Liu, Y., 2020. Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning. In *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, 493–506. (cited on page 32)

Zhang, J.; Chen, J.; Wu, D.; Chen, B.; and Yu, S., 2019. Poisoning attack in federated learning using generative adversarial nets. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 374–380. IEEE. (cited on page 56)

Zhao, L.; Chen, Z.; Hu, Y.; Min, G.; and Jiang, Z., 2016. Distributed feature selection for efficient economic big data analysis. *IEEE Transactions on Big Data*, 4, 2 (2016), 164–176. (cited on page 36)

Zhao, Y.; Li, M.; Lai, L.; Suda, N.; Civin, D.; and Chandra, V., 2018. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, (2018). (cited on page 30)

Appendices

Algorithm 2: Federator Pipeline

```
InitialiseFederator(model, hyperparameters);
ListenOnPort();
while WaitForAllClients do
    if NewClientConnections then
        SaveClientInfo();
        AskClientToWait();
    end if
end while
forall ConnectedClients do
    Send(model, hyperparameters);
    AskClientToProceed();
end forall
FederatedDataPreprocess();
for i := 0 . . . CommunicationRounds do
    while WaitForClientToReport do
        if ClientReports then
            SaveClientModelInfo();
            AskClientToWait();
        end if
    end while
    newGlobalModel ← AggregateClientsModelInfo();
    forall ConnectedClients do
        Send(newGlobalModel);
    end forall
end for
End();
```

Algorithm 3: Client Pipeline

```
InitialiseClient();
if ConnectedToHostPort then
    SendSelfInfo();
end if
localModel ← GetFederatorMessage();
WaitForFederatorToProceed();
FederatedDataPreprocess();
for i := 0 . . . CommunicationRounds do
    modelInfo ← Train(localModel, localData);
    SendToFederator(modelInfo);
    WaitForFederatorToProceed();
    localModel ← GetFederatorMessage();
end for
End();
```

Algorithm 4: Creating new Thread for message specific processes

```
while true do
    newConnection ← WaitToAcceptConnection();
    thread ←
     new Thread(newConnection, SpecificFederatorProcessFunction);
    thread.start();
end while
```

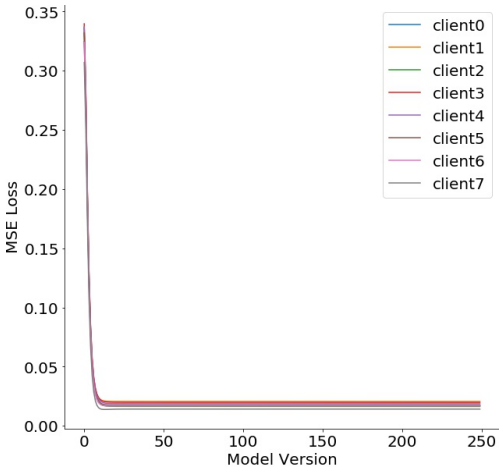


Figure 1: Per-client MSE losses for i.i.d. dataset2

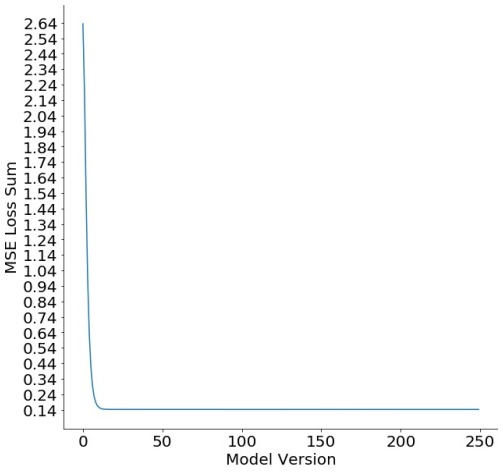


Figure 2: Sum of MSE losses for i.i.d. dataset2

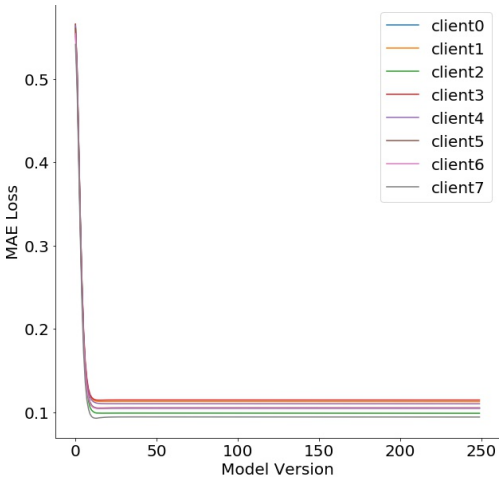


Figure 3: Per-client MAE losses for i.i.d. dataset2

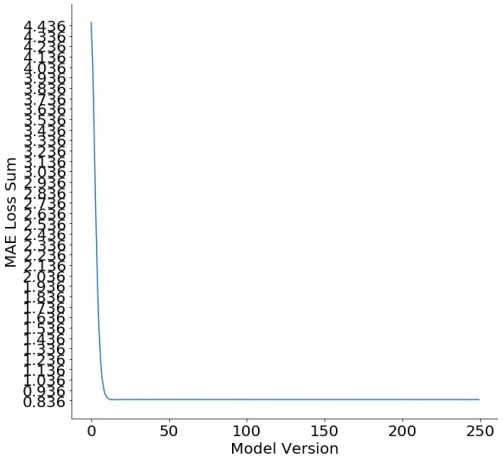


Figure 4: Sum of MAE losses for i.i.d. dataset2

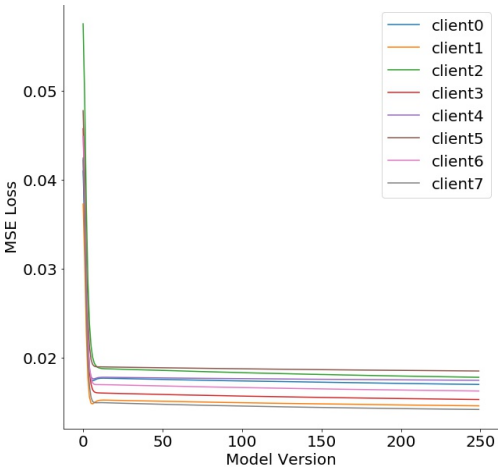


Figure 5: Per-client MSE losses for i.i.d. dataset3

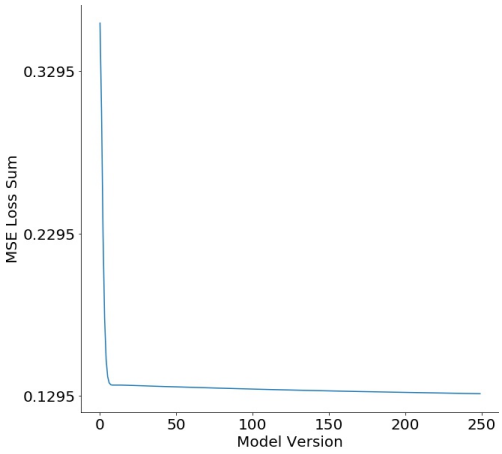


Figure 6: Sum of MSE losses for i.i.d. dataset3

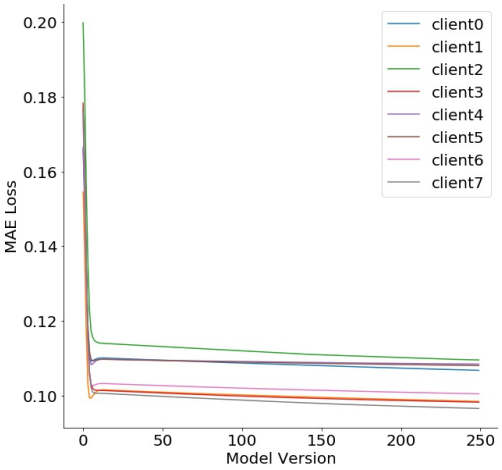


Figure 7: Per-client MAE losses for i.i.d. dataset3

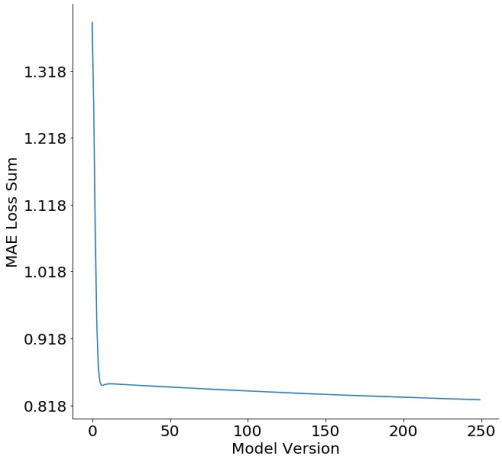


Figure 8: Sum of MAE losses for i.i.d. dataset3

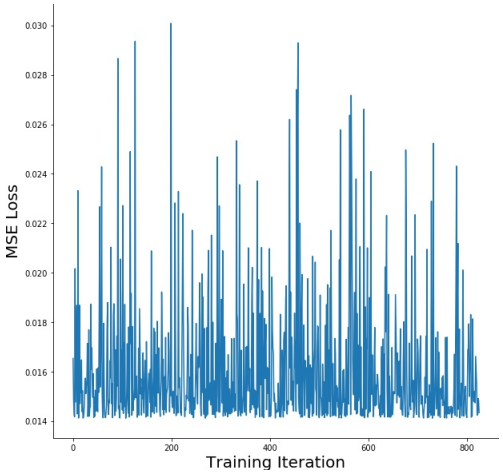


Figure 9: Results from 10 epochs per iteration with batch-size 10

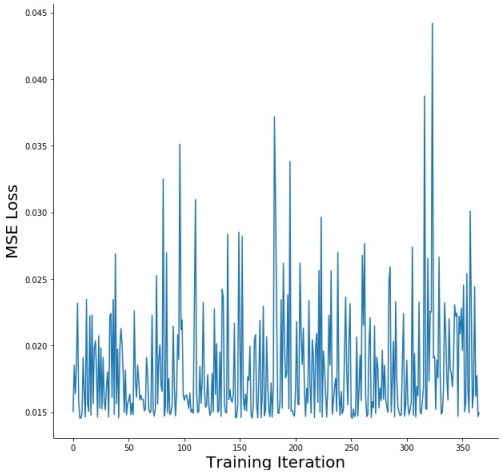


Figure 10: Results from 20 epochs per iteration with batch-size 5

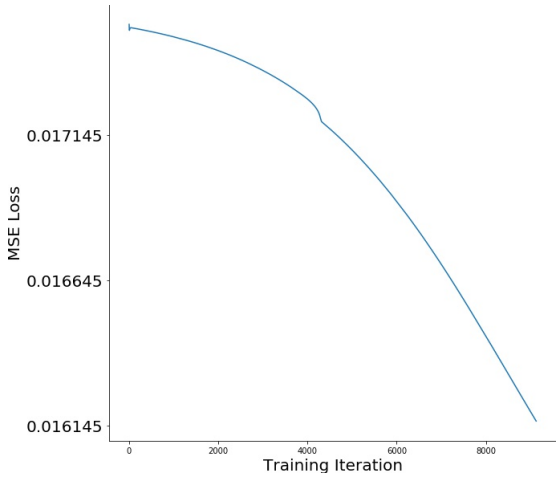


Figure 11: Re-training result using the same hyperparameter as benchmark set.

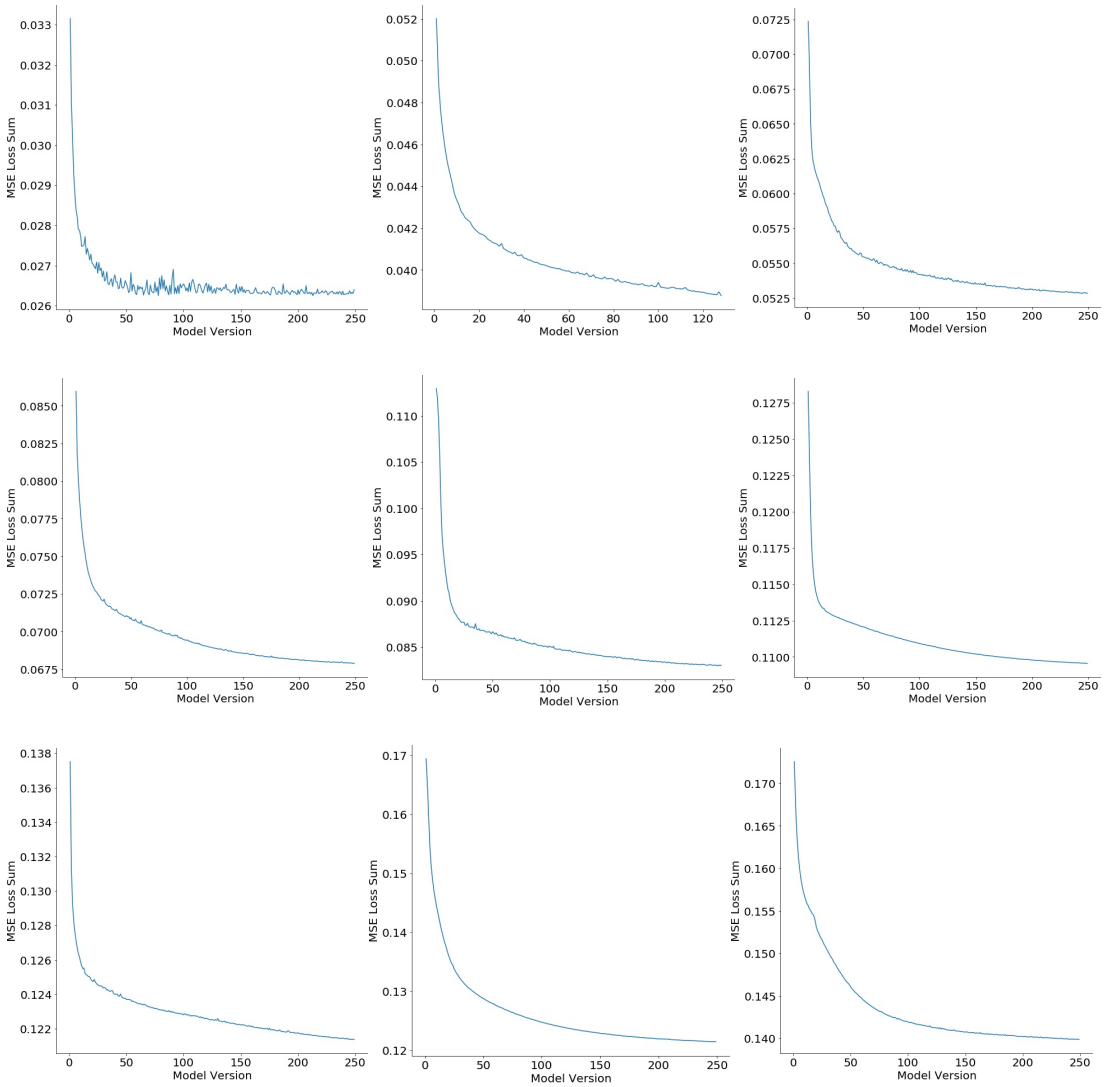


Figure 12: MSE loss of having 2 to 10 clients in training (left to right, top to bottom ordered for 2 to 10)

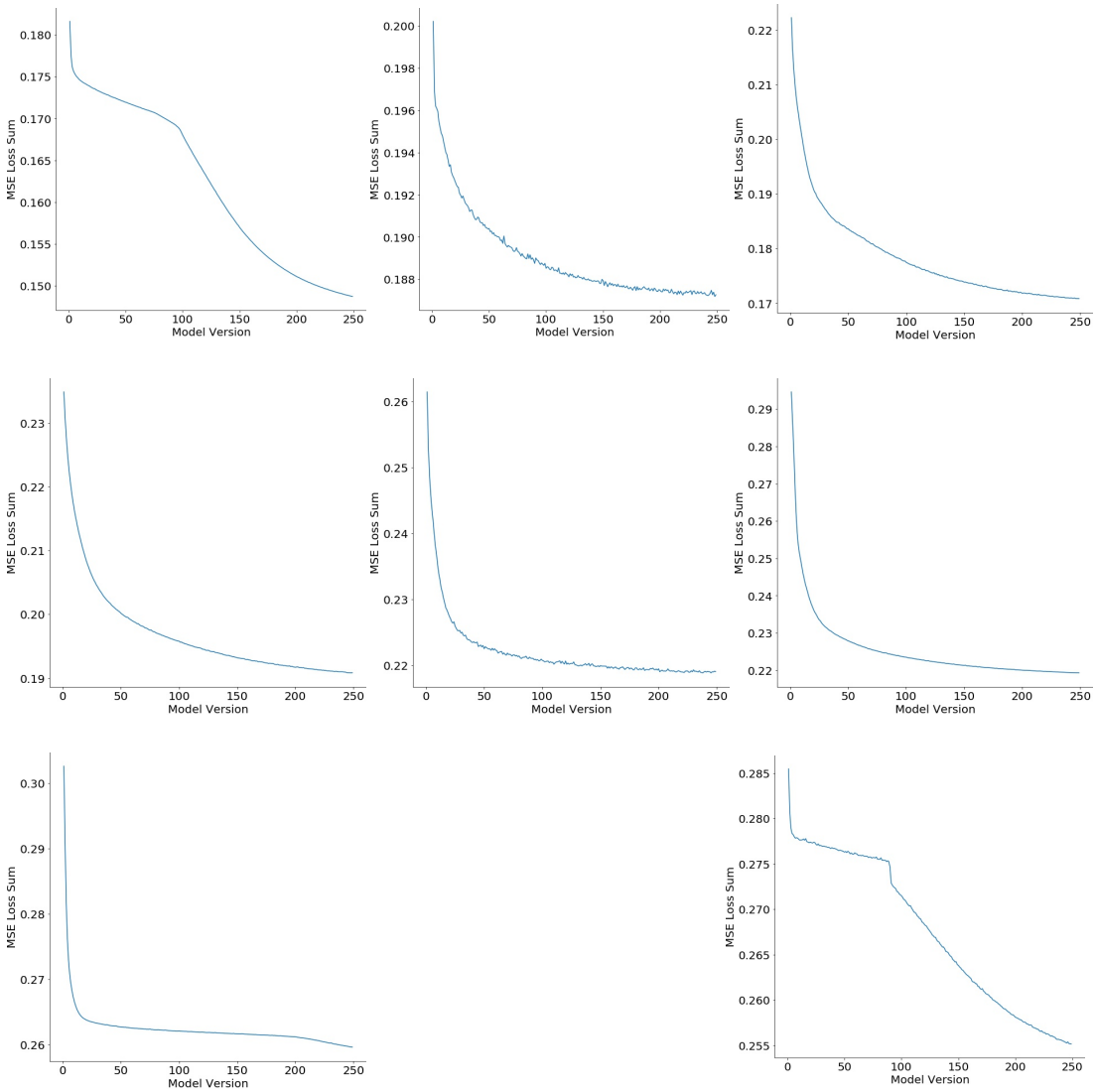


Figure 13: MSE loss of having 11 to 18 clients in training (left to right, top to bottom ordered for 11 to 18).