



**University of
Zurich^{UZH}**

Dimensionality Reduction for Distributed Machine Learning

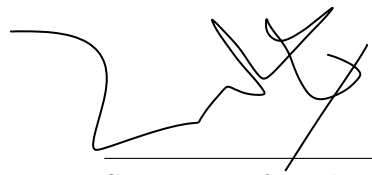
*Nazek Olabi, Michael Vuong
Zurich, Switzerland
Student ID: 19-936-582, 16-925-950*

Supervisor: Chao Feng, Dr. Alberto Huertas Celdran
Date of Submission: July 8, 2025

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, 27.3.2025



Signature of student

Abstract

Federated Learning (FL) is an emerging approach in machine learning, facilitating collaborative training of a model across many devices while keeping raw data local to support privacy concerns. However, FL has many challenges especially when considering high dimensional data which can exacerbate computational costs, communication costs, and have impact on the performance of the model. This project examines the use of dimensionality reduction methods and its impact on the overall performance of federated learning systems. Two methods for dimensionality reduction are considered, Principal Component Analysis (PCA) and Autoencoders. Both PCA and Autoencoders are implemented across three datasets in a comparison study in order to investigate, how each method impacts model accuracy, train time, and other metrics on three datasets: MNIST, Fashion-MNIST, Kuzushiji-MNIST. The results show that dimensionality reduction methods substantially reduce communication costs (training time) while preserving model performance; PCA generates the most computationally efficient solution for linearly structured datasets and Autoencoders produce the most efficient solution for complex non-linear datasets. The implications of Non-IID data and clustering techniques are examined. This project offers insights into the trade-offs between computational efficiency and model performance in FL contexts, as well as guidance for selecting dimensionality reduction approaches according to data features and system limitations. This work will contribute to more scalable and privacy-preserving distributed machine-learning algorithms.

Zusammenfassung

Federated Learning (FL) ist ein neuer Ansatz im Bereich des maschinellen Lernens, der das gemeinsame Training eines Modells über viele Geräte hinweg erleichtert und gleichzeitig die Rohdaten lokal hält, um den Datenschutz zu gewährleisten. Stattdessen werden nur die Modelparameter geteilt. FL birgt jedoch viele Herausforderungen, insbesondere bei hochdimensionalen Daten, die den Rechenaufwand und die Kommunikationskosten erhöhen und die Leistung des Modells somit beeinträchtigen können. Dieses Projekt untersucht den Einsatz von Methoden zur Dimensionalitätsreduktion und deren Auswirkungen auf die Gesamtleistung von föderierten Lernsystemen. Es werden zwei Methoden zur Dimensionalitätsreduktion betrachtet, die Hauptkomponentenanalyse (PCA) und Autoencoder. Sowohl PCA als auch Autoencoder werden in einer Vergleichsstudie für drei Datensätze implementiert, um zu untersuchen, wie jede Methode die Modellgenauigkeit, die Trainingszeit und andere Metriken (Konfusionsmatrix) auf drei Datensätzen beeinflusst: MNIST, Fashion-MNIST, Kuzushiji-MNIST. Die Ergebnisse zeigen, dass Methoden zur Dimensionalitätsreduzierung die Kommunikationskosten bei gleichbleibender Modellleistung erheblich reduzieren; PCA erzeugt die rechnerisch effizienteste Lösung für linear strukturierte Datensätze und Autoencoder produzieren die effizienteste Lösung für komplexe nicht-lineare Datensätze. In diesem Fall entspricht das der Komplexität der Datensätze.

Die Auswirkungen von NICHT-IID Daten und Clustering-Techniken werden untersucht. Diese Forschung bietet Einblicke in die Kompromisse zwischen Recheneffizienz und Modellleistung in FL-Kontexten sowie Anleitungen für die Auswahl von Dimensionalitätsreduktionsansätzen je nach Datenmerkmalen und Systembeschränkungen. Diese Arbeit wird zu skalierbaren und datenschutzfreundlichen verteilten Algorithmen des maschinellen Lernens beitragen.

Acknowledgments

We would like to take this opportunity to thank Professor Stiller and Chao for supervising our work. Their regular support enabled us to complete our work to a good standard.

In particular, we would like to thank Chao for his valuable technical support and for the informative and insightful meetings throughout the course of our work. Through the weekly meetings with him, we were able to build a deep knowledge of Federated Learning. Not only did we gain a deep understanding of the theoretical concepts, but we also learned about exciting applications and gained practical experience.

His technical expertise, helpful explanations and constructive suggestions have made a significant contribution to our progress. We are particularly grateful to him for his patience and his willingness to answer our questions in detail and always support us when we faced challenges.

Finally, we would like to thank Sven Küderli, a good friend of us for his help in providing us with technical support access to necessary infrastructure. We are especially thankful for his continuous technical support and guidance throughout the process, which greatly contributed to the successful completion of the experiments.

Contents

Declaration of Independence	i
Abstract	iii
Zusammenfassung	v
Acknowledgments	vii
1 Introduction	1
2 Background	3
2.1 Federated Learning	3
2.1.1 Centralized and decentralized Federated Learning	4
2.1.2 Centralized Federated Learning	4
2.1.3 Decentralized Federated Learning	6
2.1.4 Aggregation	7
2.1.5 Non-IID Data	8
2.2 Dimensionality reduction	9
2.2.1 Singular Value Decomposition (SVD)	9
2.2.2 Principal Component Analysis	11
2.2.3 Autoencoder	13

3	Related Work	15
3.1	SVD applications in Federated Systems	15
3.2	Principal Component Analysis (PCA) for Dimensionality Reduction	16
3.3	PCA applications in Federated Learning	17
3.4	Autoencoder and Dimensionality Reduction	18
3.5	Dimensionality Reduction in Federated Learning	19
4	Implementation	21
4.1	Motivation	21
4.2	Experimental Setup	21
4.2.1	Datasets	21
4.2.2	Hyperparameters	25
4.2.3	Functions	27
4.3	Pipeline	34
5	Evaluation and Results	35
5.1	Results for different Alphas without clustering	36
5.2	Results for different Alphas with clustering	43
5.3	The effect of clustering on balanced and imbalanced data	50
6	Summary and Conclusions	55
6.1	Future Work	56
	Bibliography	57
	List of Figures	58
	List of Tables	59

A	Installation Guidelines	63
A.1	Introduction	63
A.2	Prerequisites	63
A.3	Setup the project	63
A.4	Folders	63

Chapter 1

Introduction

In recent years, data processing has witnessed a discontinuous change. Due to the increased volume of data with interconnected devices, it became necessary to revisit some methods to improvise their efficiency while taking privacy into consideration. One of these methods is called Federated Learning (FL). FL is a more decentralized approach to machine learning, in which models are trained directly on local devices without sending sensitive data to a central server. This method has the prospect of resolving important challenges of traditional machine learning, like data privacy coupled with data security. On the other hand, FL distributes computation among different devices, where it is totally different from classic methods because the data never leaves local devices which help in casement from external threats or from unauthorized access. On the side of privacy, this would give great safeguards, i.e., the data is never allowed to leave the local devices and hence is safeguarded from any threat or unauthorized access. On the bandwidth side, the modeling updates took place with minimal resource utilization compared with certain heaps of data. Lastly, on scale, Federated Learning is very scalable. The decentralized nature makes it move to a huge range of devices that engage in model development at the same time.

The most lucid proof of the importance of Federated Learning comes in the scope of sensitive data, like healthcare, finance, and personal mobility services. The need has arisen for a significant development of machine learning models that stay super powerful while not compromising privacy. However, the great challenge comes with how we can process large and high-dimensional data pervasive in today's digital world. Such data is huge and very much high dimensional. In many real-life examples, datasets have hundreds, thousands, or even millions of features. These model complexities and vast feature domains add constraints to machine learning systems by increasing computational time and space, inhibiting training progress, and increasing overfitting risks. The learning capability of models tends to diminish as dimensionality increases beyond the number of available data points. Not all features, hence, offer value for the modeling process. Some of the features can be deleted or ignored without any considerable implication for model performance.

An efficient way to curb these limitations is dimensionality reduction. By projecting the original high-dimensionality input space onto a low-dimensional space but with most information from the original example, a learning process runs more efficiently and is

broader in terms of generalization power. The study investigated the role of two main dimensionality reduction methods in the federated learning context: Principal Component Analysis (PCA) and Autoencoders. While building upon diverse backgrounds, the two methods have their specific characteristics. Principal Component Analysis is a classical linear transformation method for dimensionality reduction. It relies on the selection of principal components, onto which the data are projected with the aim of maintaining maximum variance possible. PCA is usually less complicated and clearer to follow. However, it is broadly suitable only for linear data. Autoencoders are neural networks designed to push data through a bottleneck representation of low dimensionality. Autoencoders can model complex and nonlinear structures, in contrast to PCA, where some of their flexibility comes along with higher computational costs and greater model complexity.

This project studies the impact of various methods in enhancing the efficiency and precision of Federated Learning systems. In particular, the influence of the choice of dimensionality reduction on the computational performance, training time, and model accuracy will be discussed. The next sections will first address the theoretical background on PCA and autoencoders, and will state their implementation in Federated Learning. Then, empirical results from the simulation will be given, aiming at describing the relative performance of both methods alongside providing recommendations on their application in real-life settings. This work thus provides insights about dimensionality reduction generally in Federated Learning while also laying grounds for further research in this exciting and futuristic field.

Chapter 2

Background

2.1 Federated Learning

Federated Learning (FL) is a decentralized machine learning approach that allows multiple devices or institutions to collaboratively train a shared model without directly exchanging raw data. The concept was first proposed in 2016 by Google as a solution to protect study participants' privacy in machine learning applications, primarily in mobile environments, where user data must remain in local devices [1], [2]. Unlike traditional centralized learning, where data is collected and stored in a single location for model training, federated learning keeps authentic sensitive data at the edge devices while transmitting only model updates to the central coordinator [2].

A typical implementation of FL comprises many round executions in instantiating distributed coordinated training with aggregation and global model updating. In the beginning, the central server or the coordinating entity initializes the global model and sends it to the participating devices or institutions. Each participant should carry out local training on their individual data, independently impacting the model parameters. Instead of transmitting the train data, updates derived on the model, such as gradients or model weights, are sent to the central server by the individual participant devices participating in the training process [1]. The server aggregates these updates and improves the global model, typically by averaging algorithms such as Federated Averaging (FedAvg). This iterative process is repeated until a satisfactory degree of convergence is reached or a maximum number of training rounds are carried out [2].

One of the greatest advantages of FL is the aspect that it facilitates collaborative learning in such a way that privacy concerns are somehow resolved. As raw data never leaves the local devices, FL is entirely applicable to sensitive domains such as healthcare, finance, and personalized mobile services [1]. For example, hospitals can jointly train a disease prediction model without directly sharing patient records with each other, hence complying with data protection laws such as the GDPR.

FL has enabled several mobile applications to enhance their performance through a keyboard prediction application from Google on millions of devices without having direct access to their users' personal text inputs [2].

Moreover, FL opens up numerous technical challenges. A tremendous challenge is communication efficiency, whereby updates over a distributed network of clients initiate considerable overhead on the network. The overhead can be regulated, for example, by model compression, reducing the number of communication rounds, adaptive learning schedules, etc. Another challenge is heterogeneity of the data, in that stiff differences among the data of different participants may lead to Non-IID cases, which could bring about biased models if the biases aren't compensated for. Advanced aggregation techniques and personalization techniques would thus be necessary to ensure fairness and effectiveness of learning on varied datasets [1].

Security and robustness are also major concerns in federated learning. Since only model updates are exchanged instead of raw data, FL remains vulnerable to model-poisoning attacks, where malicious participants intentionally manipulate their local updates to introduce biases or backdoors into the global model. Additionally, membership inference attacks pose another threat, as adversaries may attempt to determine whether a specific data point was included in the training process. Addressing these and other threats requires privacy-preserving techniques such as differential privacy, secure multiparty computation (SMC), and homomorphic encryption, which aim to protect model updates from malicious actors [2].

Despite such challenges, however, FL marks an incredible step forward towards privacy-preserving machine learning and distributed AI. The ongoing research continues to refine FL techniques in order to make them scalable, effective, and secure. As the need for decentralized AI solutions grows, FL is poised to play a crucial role in such applications as smart devices, healthcare, and financial analytics, up to autonomous systems [1].

2.1.1 Centralized and decentralized Federated Learning

The main idea in federated learning is sharing update parameters of the participants without the need to share data directly. In federated learning multiple ways enable this. In this project the main focus is on Centralized Federated Learning and Decentralized Federated Learning, which basically differ from each other on structural level. Nevertheless, both settings are federated learning and in both settings no data is shared in addition to other federated learning features.

2.1.2 Centralized Federated Learning

In Centralized Federated Learning (CFL), the model is more similar to traditional machine learning, where a central server manages the training overall. In CFL, however, only the updated parameters of the model, not the raw data, are shared with the server at each aggregation step. Initially, a server and collaborators are initiated [3]. These

collaborators are distributed the initial global model to carry out their training. Collaborators then carry out the training in a fully local manner on each participant's respective device, utilizing their self-collected data. For example, an iPhone could fulfill this role by collecting data on app usage, time asleep, and the participant's distance walked, provided it had access to do so. Data collection is complete when the collection protocol ends, then training begins. After their local training is finished, each collaborator sends their updated weights to the server as shown in figure 2.1. The server receives this information and aggregates the updates together using an appropriate function (e.g., FedAvg) to calculate a new global model, and begins again by choosing collaborators for the next round and redistributes the updated global model. The above process repeats itself.

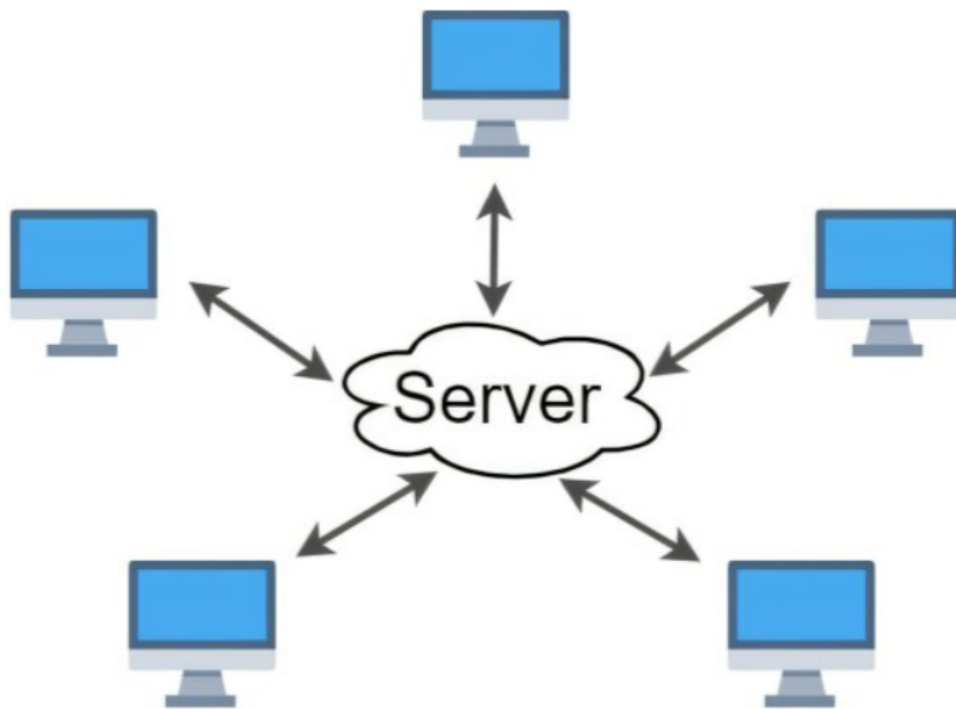


Figure 2.1: A central server with 5 clients

As a central entity is used to select participants and aggregate them, the coordination and synchronization of the whole system is thus improved, as the entire process can be controlled from a single point, the complexity is as well manageable. However, this is associated with disadvantages, since everything is located in one point. For instance, if someone attempts to attack the system, they only need to focus on a single target, as the central server is the most critical component of the entire structure. Additionally, in this case the server can also be considered as a bottleneck for the whole system, as all participants selected have to send their local updates to a single system. This leads in case of a serious error of the the server to the so called single point of failure, in which the whole system can even be shut down.

2.1.3 Decentralized Federated Learning

Decentralized Federated Learning (DFL) does not run in a centralized manner as it is revealed by the mane. In terms of decentralization, it refers to the data, the nodes, and the aspect of communication. In federated learning, there is typically a central server that will manage most of the operation[4]. In DFL, the federated participants will take the position of the server. Any participant can be used as a server. Here, it becomes a collaborative learning process and is distributed. DFL networks will consist of participants, where participants will be individuals who own mobile devices. The devices will be able to collect data and perform analysis on their devices. The participants will also have the capability of training models on their dataset. After training the model, the participant would generate the update parameters. The next question is where the participant can send the update parameters.

The DFL system does not utilize a central server, and so when a participant wants to send an update out, they send it to neighbors as shown in figure 2.2. A neighbor can be considered any node that is connected to the current participant. For example, it could be the nearest participant, etc. Participants send their update parameters to their neighbors. DFL makes use of 2 types of participants. They are referred to as aggregators and trainers. Trainers send their update parameters to their neighbors and update neighbors to send their update parameters to each other. Trainers do not aggregate updates meaning they do not create a new model following each round of training. The job of the trainers is to train the model on the train data and generate update parameters.

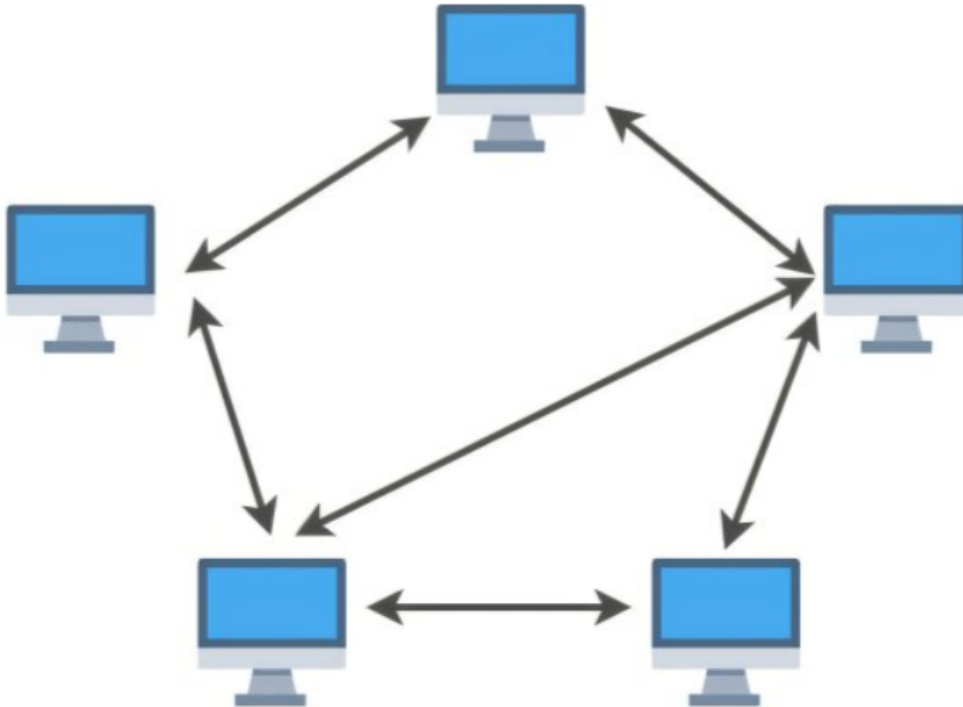


Figure 2.2: The clients exchange parameters with their neighbors

This can be further explained in a similar manner to how a mobile device produces weekly report. For this situation, each participant trains on their local data set. However, instead

of sending updates to a central server, the updates are sent to neighboring participants. The aggregators then act in important role by collecting and aggregating the updated from their neighbors, creating an updated model in each round. This is similar to gathering insights from multiple weekly reports ensuring future reports will contain more relevant information

The training process is marked as completed when all aggregators reach a consensus after a predefined number of rounds. Differing from CFL, where there is typically a single entity managing most of the training process, the responsibility is shared among many participants in DFL process. DFL as mentioned is decentralized, which relieves the single point of failure appearing in the centralized federated learning settings.

However, DFL comes with challenges. Since every node in the DFL setup has to share its updates with each of its neighbors, the overhead in communications increase thus increasing the time taken by the whole process. Having information distributed also creates difficulties in being in control, especially since critical information will not be stored in a single location. In contrast to CFL, where data is stored in a single location at a central server, being able to access and search becomes simpler. In DFL, data is distributed among many nodes, which can make retrieval difficult.

2.1.4 Aggregation

Aggregation is the act of bringing together a large set of data into a more palatable structure. In the context of machine learning, this means the central server in centralized federated learning (CFL) or the aggregating nodes in decentralized federated learning (DFL) will each receive many update parameters from the model participants. Their task is to combine all of the update parameters they receive and produce a new output, a new global model, which will be used in future rounds of training. A key aspect of aggregation, however, is to ensure any updates are legitimate. Not all contributions from model participants are guided with the same intention of providing updates that are in the best interest of the model; adversarial or malicious clients may take strides to influence the training process for their ends, which could limit the effectiveness of the machine learning model. Given these adversarial and dysfunctional threats, having a Byzantine fault-tolerant aggregation protocol is fundamentally necessary. These rules provide a level of filtration of contributed updates. The hope is, when an update comes from a compromised or erratic client, it should be able to maintain the robustness and not adversely affect the overall update to the model.

Federated Averaging (FedAvg) is one example of an aggregation mechanism. Prior to the initiation of training in federated learning (FL), a subset of possible participants must be chosen in order to mitigate the downstream risks of untrustworthy or malicious participants. Each participant chosen to participate receives the current global model and then commences training independently on data they have accumulated locally for the period of time agreed upon. Rather than to return their local data, when the local training has been stopped, the participating participants return the parameters for model updates to the central server after faithfully relocating those parameters. The contents of the returned data is only those learned parameters. When an adequate number of updates

have been received at the server, which could mean the server may receive more than the adequate number of updates, the server will begin the aggregation process using the model parameters returned by each of the participating models (the parameters represent updates to their models). The server will derive a new global model from the aggregated parameters - aggregating the model update parameters received from the participants into a model update configuration for a global model which the server refers to as the new global model. The server will then typically allocate the new global model back to the same group of participants to undergo the localized training process for a new round of updates. This protocol continues until a fixed or predetermined threshold has been reached such as a specified accuracy level.

The Federated Averaging algorithm includes two main components: the server, and the clients (participants). At the beginning of each round, the server will select the participants according to some predetermined criteria.[5] As an example, if the local model lives on an iPhone, the selection criteria may include the iPhone being fully charged and running a particular version of iOS. The selected participants will all receive the most up-to-date global model, and these participants will begin some training of their own model, perhaps via a gradient descent process, wherein the model has its weights updated. When the training for that round has finished, the participants will send back their updated model parameters to the central server. The server will aggregate all of the updates received by averaging, hence the name "Federated Averaging," to create a new global model. The process begins again starting with the next round of training using the updated model.

2.1.5 Non-IID Data

A substantial challenge in Federated Learning is the fact that every client has different data and because Federated Learning does not permit exchange of data. A realistic scenario where this happens is if data from smartphones are being used. Each smartphone reflects the preferences of a user which differ from each user. Also each person uses a smartphone for certain purposes and some use it more often than other. This leads to heterogeneity in data distribution across devices. Even though there are aggregation methods specifically for Federated Learning as explained above Non-IID data still slows down the convergence process of training. [6]

There are several issues with Non-IID data, the first being different data distribution. A person who likes cooking will have more cooking related data than a person who likes reading. The problem is that one model which uses the data from the cooking person will have little to none data about reading which makes it difficult for the model to produce some useful results. [7]

Another issue is that the data volume can vary a lot between clients. Not every user uses the smartphone equally often. This means that it has to be taken into account that some user data are bigger than others. But due to the aggregation even models which use smaller amount of data, which are not representative of the general population of the data, can have an impact on the global model.

Even though Non-IID data are inherently hard to use there exists method to deal with them.

There are more ways to utilize multiple models than just averaging over them. More often than not it is computationally expensive to have multiple models training on data. This is where knowledge distillation comes into play. Knowledge distillation makes use of a student teacher system. The teacher model is trained on a large amount of data which in return will have good generalization capabilities and knowledge. Such model is generally not used for deployment as it is cumbersome and requires huge computation. Once this model has been trained, a different kind of training can be used for smaller model more suitable for practical use called distillation. The goal is to transfer the knowledge from the "big model" to the smaller ones. Knowledge is transferred by creating "soft targets". These "soft targets" are probabilities produced by the teacher model. They serve as guidepost or hints for the student models. By mimicking the teacher model's output distribution, the participant models can learn from its knowledge.

Another method is to weigh the aggregation parameters of each model to the amount of data as well as the number of label, classes and distribution. The method divides the local models into two groups based on the heterogeneity level of the data the model has trained on. The heterogeneity level depends on volume and variance of the training data of each client. Depending on the threshold, the groups are being created. Each group has a different aggregation method (weighing).

2.2 Dimensionality reduction

High-dimensional data is frequently encountered in many areas, particularly in data analysis and machine learning. Such data often contains a significant amount of irrelevant information, which not only increases computational complexity but also negatively impacts the performance of algorithms. To address this, various algorithms are used to reduce the dimensionality of the data. Linear techniques such as SVD and PCA play a crucial role and are widely used in research. In addition, non-linear methods like autoencoders provide alternative approaches for effective dimensionality reduction.

2.2.1 Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a widely used technique in linear algebra that decomposes a matrix into three fundamental components. Given a matrix A , the SVD of that particular matrix is given by:

$$A = U\Sigma V^T$$

where U and V are orthonormal matrices, and Σ is a diagonal matrix that contains the singular values of A . The columns of U represent left singular vectors, and together they

form an orthonormal basis of the column space of A , while the columns of V represent right singular vectors, forming an orthonormal basis of the row space of A . The entries in the diagonal matrix Σ are called singular values, and they determine the significance of each corresponding singular vector in capturing the structure of A . The singular values are equal to the root of the eigenvalues $A \cdot A^T$ ordered in descending order. These eigenvalues are equal to the eigenvalues of $A^T \cdot A$. By Computing the eigenvectors of the outer product of both $\text{opd}(A^T, A^T) = A^T \cdot A$ and $\text{opd}(A, A) = A \cdot A^T$ the matrices V and U are formed respectively [8], [9].

This kind of decomposition reveals significant structures in the data and has important applications in dimensionality reduction, where high-dimensional data is represented in a more compact form while still preserving its most relevant structures. The singular values in Σ govern this process. Generally, they are arranged in descending order, with the largest values corresponding to the most important characteristics of the data, while the smallest values contribute the least.

To approximate the original matrix, small singular values can be discarded:

$$A' = U'\Sigma'V'^T$$

where Σ' is a reduced diagonal matrix containing only the largest k singular values, while U' and V' contain the corresponding singular vectors. This ensures that the most critical information is preserved while reducing the complexity of the data [8]

Given a sample data matrix A :

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Singular Value Decomposition (SVD) using python is performed on matrix A , which decomposes A into three matrices U , Σ , and V^T as follows:

$$A = U\Sigma V^T$$

The resulting matrices are:

$$U = \begin{bmatrix} -0.2148 & 0.8872 & 0.4082 \\ -0.5206 & 0.2496 & -0.8165 \\ -0.8263 & -0.3879 & 0.4082 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 16.8481 & 0 & 0 \\ 0 & 1.0684 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$V^T = \begin{bmatrix} -0.4797 & -0.5724 & -0.6651 \\ -0.7767 & -0.0757 & 0.6253 \\ -0.4082 & 0.8165 & -0.4082 \end{bmatrix}$$

By selecting only the top $k = 1$ singular value, the dimensionality is reduced resulting in the following matrices:

$$U_k = \begin{bmatrix} -0.2148 \\ -0.5206 \\ -0.8263 \end{bmatrix}$$

$$\Sigma_k = [16.8481]$$

$$V_k^T = [-0.4797 \quad -0.5724 \quad -0.6651]$$

The reduced data matrix A_k is then approximated by the product of these reduced matrices:

$$A_k = U_k \Sigma_k V_k^T = \begin{bmatrix} 1.7362 & 2.0717 & 2.4073 \\ 4.2072 & 5.0202 & 5.8332 \\ 6.6781 & 7.9686 & 9.2592 \end{bmatrix}$$

2.2.2 Principal Component Analysis

Another popular method for reducing dimensionality is Principal Component Analysis (PCA), typically calculated through Singular Value Decomposition (SVD). While SVD offers a broad matrix factorization framework, PCA specifically determines the directions of maximum variance within the data and projects it into a lower-dimensional space. PCA operates by transforming a dataset represented as a matrix X into this reduced representation while aiming to maintain as much of the original variance as possible. This transformation involves calculating the principal components—orthonormal directions that encapsulate maximal variance. These components are derived from solving the eigenvalue problem associated with the covariance matrix:

$$S = \frac{1}{N} \sum_{n=1}^N (x_n - \bar{x})(x_n - \bar{x})^T,$$

where $\bar{x}$ denotes the mean of the dataset. The eigenvectors u_i obtained from S correspond to its largest eigenvalues λ_i and are considered principal components [10].

The reduction in dimensionality occurs via projection onto the first k principal components:

$$X' = U_k^T X,$$

with U_k containing these top k principal components. The remaining components relate to smaller eigenvalues, signifying directions with minimal variance, hence regarded as less significant.

Additionally, PCA can be examined from a perspective of reconstruction. The first principal component (PC1) represents the direction that minimizes total squared reconstruction error when projecting data onto a lower-dimensional subspace. Successive components (PC2, PC3, etc.) are identified in succession and remain orthogonal to their predecessors.

Given that determining eigenvectors in covariance matrix calculations can be computationally intensive, employing SVD provides an efficient alternative for obtaining principal components. In terms of decomposition:

$$X = U\Sigma V^T,$$

the right singular vectors found in columns of V align with the principal components; meanwhile, singular values located in Σ signify their relative importance. Thus, SVD stands out as both stable and widely embraced for practical implementations of PCA. By selecting the first k principal components, PCA provides a more efficient representation of the data while minimizing information loss. This method is often used in noise reduction, feature extraction, and visualization of high-dimensional data [10].

Given:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

The covariance matrix S is calculated using python as:

$$S = \begin{bmatrix} 9 & 9 & 9 \\ 9 & 9 & 9 \\ 9 & 9 & 9 \end{bmatrix}$$

The eigenvectors and eigenvalues are computed, with the top eigenvector corresponding to the largest eigenvalue:

$$U_k = \begin{bmatrix} -0.5774 \\ -0.5774 \\ -0.5774 \end{bmatrix}$$

The data is projected onto the top $k = 1$ principal component resulting in the reduced data matrix:

$$A' = \begin{bmatrix} 5.1962 \\ 0 \\ -5.1962 \end{bmatrix}$$

This transformation captures the maximum variance in the data along a single principal direction, effectively reducing dimensionality.

2.2.3 Autoencoder

Autoencoders are a type of neural network used for unsupervised training. They consist of an encoder and a decoder part. Between the encoder and decoder exist a latent space which consist of the embedding or encoding of the encoder. Autoencoders can be used for various task such as learning representation of classification task (e.g. dimensionality reduction), various detection task and even generative tasks. The encoder is usually used for classification tasks where as the decoder is used for generation of text and image. [11]

The architecture of an autoencoder is basically a concatenation of 2 neural networks. The input data goes to the first part, the encoder where it gets transformed into a compressed representation. This representation usually has a lot less dimension than the original data, therefore it is easier to be used. The decoder, recreates the input data from the compressed representation. The goal here is to minimize the reconstruction error.

Mathematically speaking, the autoencoder can we seen as follow:

Let:

$$\begin{aligned} \mathbf{x} \in \mathbb{R}^d & \text{ be the input data of dimension } d, \\ \mathbf{z} \in \mathbb{R}^m & \text{ be the latent representation (where } m < d), \\ f_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^m & \text{ be the encoder function with parameters } \theta, \\ g_\phi : \mathbb{R}^m \rightarrow \mathbb{R}^d & \text{ be the decoder function with parameters } \phi, \\ \hat{\mathbf{x}} \in \mathbb{R}^d & \text{ be the reconstructed output.} \end{aligned}$$

The autoencoder is represented as:

1. Encoding Step (Dimensionality Reduction):

$$\mathbf{z} = f_\theta(\mathbf{x}) = \sigma(W_e \mathbf{x} + \mathbf{b}_e)$$

where:

- $W_e \in \mathbb{R}^{m \times d}$ is the encoder weight matrix,
- $\mathbf{b}_e \in \mathbb{R}^m$ is the encoder bias,
- $\sigma(\cdot)$ is a non-linear activation function (e.g., ReLU, sigmoid).

2. Decoding Step (Reconstruction):

$$\hat{\mathbf{x}} = g_\phi(\mathbf{z}) = \sigma(W_d \mathbf{z} + \mathbf{b}_d)$$

where:

- $W_d \in \mathbb{R}^{d \times m}$ is the decoder weight matrix,
- $\mathbf{b}_d \in \mathbb{R}^d$ is the decoder bias.

Autoencoders have numerous application. The first and most obvious one is dimensionality reduction. Hinton [11] created an experiment where he would use a deep autoencoder for dimensionality reduction. He started with a set number of hidden neurons, which he gradually reduces until he reached a bottleneck of 30 neurons. Then he compared it with the first 30 components of PCA and learned a representation that was qualitatively easier to interpret, clearly separating data clusters. It is worthy to note that if only linear activation functions and a single hidden layer is used for the autoencoder, then it is very similar to PCA. [12]

Directly using dimensionality reduction, Information retrieval and Search engine optimization is another application. Having a compressed representation means that it is easier to search through for example database which can improve search time.

Because the autoencoder aims to minimize the reconstruction error, it can be taken advantage of and used for Anomaly detection. By forcing the autoencoder to reconstruct the most needed features, it learns to reconstruct precisely. If now there were anomalies in the data, the reconstruction error would go up by a set amount which indicates that there is an anomaly in the data. [13]

Chapter 3

Related Work

3.1 SVD applications in Federated Systems

Singular Value Decomposition (SVD) has several applications in machine learning as dimensionality reduction, data compression, and noise reduction. In the context of Federated Learning, SVD is being used as an efficient way of dealing with the high communication costs now defined as a cost problem in most distributed systems. According to Kwon and Park, SVD could make communication more efficient in FL without sacrificing model performance [14].

In their work, the global model parameters were factorized into low-rank matrices using SVD. Instead of transmitting the entire parameter matrices between the global server and clients, only compressed representations—top singular values and their corresponding singular vectors—were exchanged. This severely restricted the communication overhead while keeping relevant details for model updates [14].

The experiment employed the handwritten digits MNIST dataset with the CNN and DNN models. Interestingly, the SVD-based method showed results quite similar to traditional FL algorithms, such as FedAvg, in terms of accuracy, even with a halving of the amount of data transmitted. This study underlines the importance of SVD as a communication-efficient manner to use in FL-based systems, and it could prove to be useful in situations with limited bandwidth or enhanced costs of communication [14].

Another work by Liu, Pejo et al. investigated the role of singular value decomposition (SVD) in improving data privacy in federated learning applications. In their study, they focused on the use of SVD in data preprocessing. Singular value decomposition is a powerful and effective technique that helps to understand the structure of the underlying data without revealing the sensitive information of individual participants. It enables the federated system to process data and draw insightful conclusions without violating the privacy of individual participants and is therefore suitable for privacy-sensitive applications of federated learning [15].

One of the key concepts which permeate their approach is secure aggregation. In this technique, it is guaranteed that the contribution of every single node in the federated

network remains hidden during the aggregation phase. The results are then combined in such a way that it is impossible to trace the individual inputs of the participants, which is very important when considering the privacy perspective of a distributed learning environment [15].

In addition to the above, the authors also introduced noise to further improve privacy. The system could obfuscate individual contributions to larger aggregate results even more effectively by adding controlled random noise into the aggregated results. The noise is designed in such a way that it will mask the local data from discovering any meaningful or sensitive information, meanwhile, the system could still adequately perform and remain useful [15].

Together with SVD for preprocessing and noise addition, a well-fortified privacy framework is provided to the federated learning. Using these approaches in parallel provided a strong guarantee of privacy with no degradation in terms of the performance and utility of the machine-learning model. The implementation of advanced techniques of data processing and strategies on privacy protection provides a holistic approach to secure and privacy-enhanced federated learning [15].

3.2 Principal Component Analysis (PCA) for Dimensionality Reduction

A number of studies have delved into the use of Principal Component Analysis (PCA) as a technique for dimensionality reduction, each based within a different area of application. Image processing, neuroimaging, and many other high-dimensional data applications deal with PCA in different scopes of success and limitations. In 2017, a study is conducted into how efficient PCA was at reducing the dimensionality of high-resolution images. The study showed that PCA could significantly reduce sizes while keeping relevant image features intact; hence, it is an important technique for image transmission and storage [16]. In that regard, using eigenvectors of the covariance matrix to transform pixel values into a new basis makes PCA capture the highest variance of the dataset, leading to redundancy reduction and improvement of computational efficiency. Further, the study outlined that PCA-based compression methods can do as high as a 35.3% reduction in image size while retaining visual quality. Such a reduction is beneficial to mobile devices and in bandwidth-limited environments, where data transfer speeds are increased, and storage requirements are optimized.

In the neuroimaging field, Artoni et al. focused on PCA as a preprocessing technique for Independent Component Analysis (ICA) in EEG. Their results indicated that the PCA-based rank reduction that kept 95% of the variance of the original data had a very high capability of losing physiologically meaningful independent components [17]. The rank reduction affected both the quality and stability of ICA-derived components and, in turn, diminished the reliability of source localization and interpretation of signals. The paper advised applying PCA with caution in situations where preserving the integrity of the signal is of utmost priority, despite the fact that the PCA methods might be effectively

utilized to reduce computational complexity. Likewise, the research emphasized that PCA was not the ideal singular blind source separation method for EEG data due to its emphasis on variance rather than statistical independence [17].

While PCA has clear advantages in terms of dimensionality reduction, different areas impose various limitations on its applicability. Such is the case with image processing, where it performs very well in feature compression and reduction. Using PCA in high-dimension datasets would therefore introduce serious risks of unintentional information loss when used in neuroimaging and biosignal processing, wherein such high-dimension data tend to express complex physiological interactions. Recent studies suggest the formulation of hybrid approaches, incorporating PCA along with deep learning or other dimensionality reductions, to reduce such limitations through balancing computational efficiency with the fidelity of the original data. These studies portray the ambivalent nature of PCA in dimension reduction, where, on the one hand, it reduces redundancies and is capable of improving computational efficiency, yet, if performed too aggressively, could result in the loss of critical features from the data. Therefore, into the domain of application-specific needs, the very application of PCA should seek to retain essential information while improving its efficiency in processing time. The future could pursue the adaptability of PCA or its use aggressively with neural networks to randomize performance differences [16], [17].

3.3 PCA applications in Federated Learning

Principal Component Analysis (PCA) is a popular dimensionality reduction technique that has recently been rescaled within federated learning (FL) to cope with new challenges arising from decentralized data processing. Since the use of FL is becoming more prevalent in privacy-sensitive applications, research interest in incorporating PCA into federated frameworks is rising significantly. This section surveys related studies that have explored PCA applications in FL, particularly in relation to privacy preservation, scalability, and computational efficiency.

Recent works have proposed Federated PCA (FPCA) algorithms for the distributed yet private computation of principal components more efficiently. Notable in this area is a federated, asynchronous, and differentially private PCA algorithm that incrementally computes local model updates using a streaming procedure while adaptively estimating the leading principal components under memory constraints [18]. This approach makes the system resistant to straggler nodes and failed devices through an input-perturbation mechanism that guarantees differential privacy (DP) and performs comparably to PCA done in a centralized manner.

The blend of PCA with FL tackles many issues at once. The computation of PCA in its classical form is burdened by massive storage and computation requirements due to quadratic memory dependencies on data dimensions. In line with this, federated algorithms have been designed to operate under low-memory constraints. This, in turn, makes leveraging FL more efficient on edge devices. A federated subspace tracking algorithm for missing information also guarantees approximation accuracy while maintaining

communication efficiency [18]. However, not implementing differential privacy is present in many existing works, thus leaving the recent FPCA techniques to fill an important gap.

Moreover, studies have proposed federated PCA approaches in the context of security applications. In Federated Learning (FL) frameworks, Kernel Principal Component Analysis (KPCA) has been used to mitigate adversarial attacks such as label-flipping attacks. By introducing KPCA with multi-path service routing, FL systems were able to increase their efficiency and robustness against harmful perturbations [19]. This technique has shown improvements in terms of accuracy and execution time over conventional FL implementations. Federated PCA methods usually use hierarchical aggregation architectures like the Distributed Agglomerative Summary Model (DASM), which aggregates updates in a bottom-up manner according to the tree structure [18]. This architecture balances scalable computation with efficient merging of subspaces, which is much needed for large-scale decentralized datasets.

Thus, PCA applications in FL have iteratively transitioned to mitigate major flaws present in their centralized counterparts, offering solutions for privacy preservation, computational efficiency, and robustness in adversarial environments. The integration of differentially private mechanisms and hierarchical data processing architectures renders federated PCA amenable to effective real-world deployment in decentralized settings. However, challenges in maintaining high-quality principal component estimations under strict privacy restrictions still exist as an ongoing research area [18], [19].

3.4 Autoencoder and Dimensionality Reduction

Thanks to the architecture and design of the autoencoder it is well suited for dimensionality reduction.[11] The encoder part of the autoencoder compresses the input data into an efficient representation and the decoder reconstructs the input data. During the whole process the autoencoder aims to learn important features of the data while minimizing the reconstruction error. So if only the encoder part is used, then the output would be a compressed representation of the original data which exactly is dimensionality reduction.

The question may arise, why use neural networks for dimensionality reduction, when other dimensionality reduction methods such as PCA already exists. The answer to this question is that neural networks can capture non linear structures in data. PCA is mainly limited to linear transformation. Neural network can use non linear activation functions such as sigmoid, RELU or tanh which all are commonly used. By also adding a bias to the weights all types of data can be captured by neural networks. [20]

Even though autoencoders have strong power, they come with some downsides. Because autoencoders are neural networks, they also share their weaknesses To be able to use the autoencoder for dimensionality reduction, one has to train one first. Training an autoencoder can have a lot of computation cost. In order to be able to capture nonlinearity and having deeper layers, the autoencoder takes some time and resources to train.

3.5 Dimensionality Reduction in Federated Learning

Even though FL has gotten a lot of attention, little to no research exists where dimensionality reduction methods are used in FL, especially in context of training time induced by the complexity of datasets. Most existing studies in FL focus on communication efficiency, privacy concerns, or model performance. However, using dimensionality reduction methods, such as PCA or Autoencoders in reducing the computational resources and training time in FL settings remains underexplored. This project aims to investigate how and if DR methods can be used in federated learning to save computational resources and training time without affecting model performance.

Chapter 4

Implementation

4.1 Motivation

The goal of this series of experiments is to set up a proper environment for testing different dimensionality reduction techniques on several datasets in order to compare the performances between them. It is especially important to measure the impact of the dimensionality reduction methods, this includes positive and negative impact.

In order to achieve this, several datasets, hyperparameters and functions are explained and how the values are chosen.

4.2 Experimental Setup

4.2.1 Datasets

The experiment covers several different datasets, the first being the MNIST dataset. This is one of the most known dataset and widely used in machine learning. It consists of 70 000 grayscale images of handwritten numbers ranging from 0 to 9 in form of 28x28 pixels. They were collected from a wide range of people such as students and United States Census Bureau. This dataset is easy to use which means that it has been used a lot making it easier to benchmark new algorithms. It is easy to use because it is a complete dataset. It has well-labeled data, simple structure and is relatively small. It is also not necessary to preprocess intensively and it is available in most machine learning libraries making it easier to import. 60 000 out of the 70 000 images are training images where as the remaining ones are part of the test set.

Nowadays modern models can achieve extremely high accuracy making it too simple and less challenging. This is due to made such that it suits machine learning well. For example all digits are centered and normalized, unlike real-world handwritten digits. Therefore other datasets which are more challenging should be considered, for example the Fashion-MNIST dataset.

The Fashion-MNIST dataset has similar structure as the MNIST dataset, where it contains also 70 000 grayscale images of 28x28 pixels. It also has the same amount of training as well as test images. The Fashion-MNIST dataset consists however images of clothing items, thus its labels are different, but it still has ten classes.

Label	Clothing Item
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Table 4.1: Labels and categories in the Fashion-MNIST dataset

The fact that it has clothing items instead of handwritten digits makes it harder to classify therefore a bigger challenge for modern models. This is particularly useful because this dataset can be used to further improve the models where as it is nearly impossible on the MNIST dataset in terms of accuracy. Also clothing items resemble more of the real world than handwritten digits.

Criteria	MNIST	Fashion-MNIST
Dataset Creator	Yann LeCun et al.	Zalando Research
Purpose	Digit classification	Real-world image classification
Number of Classes	10 (Digits 0-9)	10 (Clothing categories)
Complexity	Simple, high accuracy	More complex, harder to classify
Visual Structure	High contrast, clear digits	Detailed textures, various shapes
Recognition Difficulty	Easy (>99% accuracy)	Harder (~94% max accuracy)
Real-World Application	Limited to digits	Fashion, retail, e-commerce
Diversity of Data	Uniform, centered digits	Different styles and overlapping pixels

Table 4.2: Comparison between MNIST and Fashion-MNIST datasets

Another dataset that has similar structure to the preceding two is the Kuzushiji-MNIST dataset. This dataset is created from the past of Japan. Historically Japan has a unique story. Japan and its culture have been isolated from the world for a long time until the Meiji restoration in 1868. This caused drastic changes in the Japanese language, writing and printing systems. At this point due to the modernization of Japan, the old way of writing, the cursive Kuzushiji, therefore most Japanese today cannot read books published over 150 years ago. Even though these books have been digitized they have never been transcribed. [21] This dataset is built out of the handwritten characters from that time and processed into a MNIST-like structure such as containing 60 000 training samples and 10 000 test samples and ten classes. The figure 4.1 shows an example with ten classes.

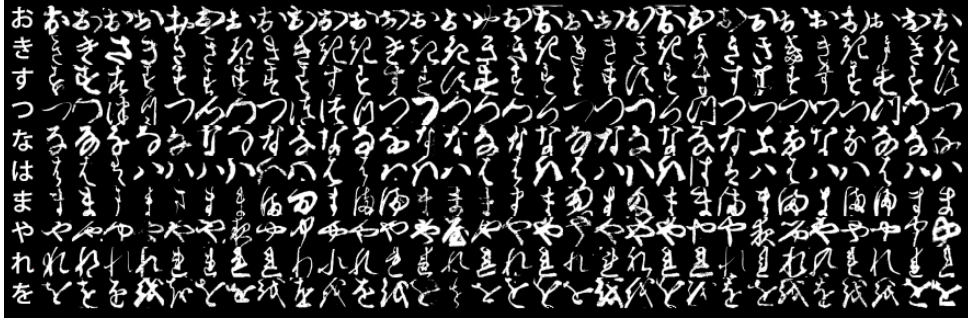


Figure 4.1: The first column represents the modern hiragana characters

Coming to the highest difficulty of considered datasets are the Cifar datasets which have two variants one with ten classes and one with 100 classes giving them their name Cifar 10 and Cifar 100. This dataset also has similar structure than the preceding ones among other things like 60 000 training samples and 10 000 test samples. The biggest difference however is that while the preceding datasets only have grayscale images, this dataset contains three color channels (RGB) and the size of the images is 32x32 pixels. Cifar 10 dataset contains ten real world objects.

Label	Category
0	Airplane
1	Automobile
2	Bird
3	Cat
4	Deer
5	Dog
6	Frog
7	Horse
8	Ship
9	Truck

Table 4.3: CIFAR-10 Categories

This makes it a lot harder for the models to achieve high accuracy. Another challenge is the running time. Because of its more complex structure some models will have a longer running time which makes this dataset perfect for benchmarking Predictive Performance as well as Computational Performance together.

The last dataset included in the experiments are tabular datasets. These datasets are organized in rows and columns. They are similar to spreadsheets and databases. Usually a row represents an entity for example a person and the column represents a feature (dimension). Tabular dataset usually require some preprocessing by dropping some not needed columns and dealing with missing values.

In the end the MNIST, the Fashion-MNIST and the Kuzushiji-MNIST were chosen as the most suitable datasets, because they all have similar structure. This makes it easier to compare the impact of the complexity of the datasets, as only one model is needed for all three datasets. The Cifar datasets would need a separate and more complex model which

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Figure 4.2: Example of the titanic dataset

is not the focus of the project. The complex model have an impact on the training time which reduces the effect dimensionality reduction would have on the training time. On average the tabular datasets have less dimensions than image datasets, which also reduces the impact of dimensionality reduction in terms of computational performance. This is why tabular datasets were also dropped.

4.2.2 Hyperparameters

Hyperparameters are crucial for machine learning. Unlike model parameters which are learned during training, they have to be set before training and influence the model. For example setting the number of epochs at the beginning is a hyperparameter. Some model converge after higher number of epochs other model converge faster. Therefore setting the number of epochs has a direct impact on the models performance. Hyperparameters can be categorized into different groups.

The first one is model related parameters. These define the structure of the model and determine its ability to learn from data. By creating a model the number of layers has to be taken into account first. Deeper models are able to learn more complex patterns but for example also run risk of overfitting because the models tries to fit to the data really well where it then can have problems when it gets data that it has never seen before.

The same goes for the number of neurons of a layer. The question how many neurons one should use for a hidden layer is a difficult one as there is no best solution. There are guidelines but usually this hyperparameter is set through experimentation.

Activation functions define how the input and how information is passed to the next layer. The innovative aspect of the activation functions is that they can model non-linear data. For example if the task is to classify, a Softmax function can help converting the data into probability. Then the model chooses the category with the highest probability. Sigmoid produces values of between 0 and 1 which makes it suitable for binary classification. Because relevant datasets are all multiclass classification tasks, the activation function of the last layer in the model of this project is always softmax or a variation of it (e.g. log softmax

Task	Common Activation Function
Binary Classification	Sigmoid
Multi-class Classification	Softmax
Regression	Linear (No Activation)
Image Processing (CNNs)	ReLU

Table 4.4: Activation Functions for Different Output Layers

The next type of parameters are training related parameters. These parameters define how the model should learn from the data. Learning rate defines how the weights are updated per training step. Setting them too high may lead to the model never converging, and setting it too low may take too long or the model could be stuck in a local extrema. For example let 0 be the target point, the starting point be 3 and the range between negative 9 and positive 9. The goal is getting closer to the target point each step. By choosing a large learning rate it could lead to a scenario where the model oscillate between 3 and negative 3. This would mean that the model never converges. In case of this experiment, the starting point was 0.001 then adjusted over time. This was considered separately for each experiment.

The batch size defines the number of samples that will be propagated through the network. The key concept here is to split the dataset into batches and not processing the whole

dataset. For instance let's say there are 1020 training samples and the batch size is set to equal to 100. Then the algorithm takes the first 100 samples and trains the network. Then it takes the next 100 samples (101st to 200th) and so on. This procedure will keep going until all samples have been propagated through the network. The only problem here is that the last 20 samples are not divisible by 100. The simplest solution would be ignoring them. Using a smaller batch size also requires less memory as the training is done with less samples. Typically the model converges faster with smaller batch sizes, because the weights are being updated after each propagation. Smaller batch size does also mean that there are less samples for each propagation so the gradient can be less accurate depending on the number. It has been observed in practice that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize. In the experiment the batch size is 64.

Epochs is the number of times the entire dataset is passed through the model. One of the most important issues in machine learning is overfitting. When the number of epochs is greater than the epochs necessary, then the model starts to learn patterns that are specific to this data. This means that it will perform well on the training data but fails to perform on unseen data such as the test data. By choosing a smaller number necessary will lead to the model not converging yet. Usually early stopping is used to prevent overfitting. In order to find the optimal number of training epochs, several tests were run, where the number of epochs was set to a high number, to see the convergence behavior and thus determine the optimal number of epochs.

4.2.3 Functions

Training Function

The experiments are being accompanied with variety of functions which help the training loop. The first function used is the training function. Algorithm one shows the training function used in the experiments which also computes the loss each batch.

Algorithm 1 Training Function for Neural Network

```

1: procedure TRAIN(...)
2:   Set network to training mode
3:   for each (batch_idx, (data, target)) in train_loader do
4:     optimizer.zero_grad()
5:     data  $\leftarrow$  data.detach()
6:     output  $\leftarrow$  network(data)
7:     loss  $\leftarrow$  NLL_Loss(output, target)
8:     loss.backward()
9:     optimizer.step()
10:    if batch_idx mod log_interval = 0 then
11:      Append loss to train_losses
12:      Append (batch_idx  $\times$  64) + ((epoch - 1)  $\times$  len(train_loader.dataset)) to
        train_counter
13:    end if
14:  end for
15: end procedure

```

Testing Function

Algorithm two shows the test function used that operates on the test data. These two functions are used through the whole series of experiments on every dataset except the tabular datasets.

Algorithm 2 Testing Function for Neural Network

```

1: procedure TEST(network, test_loader, test_losses)
2:   Set network to evaluation mode
3:   test_loss  $\leftarrow$  0
4:   correct  $\leftarrow$  0
5:   Disable gradient calculation
6:   for each (data, target) in test_loader do
7:     output  $\leftarrow$  network(data)
8:     test_loss  $\leftarrow$  test_loss + NLL_Loss(output, target, size_average=False)
9:     pred  $\leftarrow$  argmax(output, dim=1)
10:    correct  $\leftarrow$  correct + count(pred == target)
11:   end for
12:   test_loss  $\leftarrow$  test_loss / length(test_loader.dataset)
13:   Append test_loss to test_losses
14: end procedure

```

PCA Algorithm

The PCA algorithm uses PCA from the sklearn library in order to perform PCA. The algorithm first takes the images and flattens them in order to perform PCA. After the dimensionality reduction, it reconstructs the images into their original shape (28x28) so it will be comparable to the images that were not reduced. Algorithm three shows how the PCA pipeline looks like.

Algorithm 3 Dimensionality Reduction PCA

- 1: Initialize empty lists: *train_data* and *train_labels*
 - 2: **for** each (*data*, *labels*) in *train_loader_pca* **do**
 - 3: Append *data* reshaped into vectors to *train_data*
 - 4: Append *labels* to *train_labels*
 - 5: **end for**
 - 6: Concatenate *train_data* along dimension 0
 - 7: Concatenate *train_labels* along dimension 0
 - 8: Convert *train_data* to NumPy array: *train_data_np*
 - 9: Initialize PCA reducer with *n_components* = 100
 - 10: Apply PCA to reduce dimensionality: *train_data_reduced* $\leftarrow$
fit_transform(*train_data_np*)
 - 11: Reconstruct data from PCA components: *train_data_reconstructed_np* $\leftarrow$ **inverse_transform**(*train_data_reduced*)
 - 12: Convert reconstructed data to PyTorch tensor: *train_data_reconstructed* $\leftarrow$ **tensor**(*train_data_reconstructed_np*)
 - 13: Reshape reconstructed data to (*N*, 1, 28, 28)
 - 14: Normalize reconstructed data
-

Autoencoder Algorithm

The dimensionality reduction with the autoencoder is similar to the pca one. It uses a pretrained autoencoder to reduce dimensions as well as reconstructing the images into their original shape. Algorithm four shows the dimensionality reduction via autoencoder.

Algorithm 4 Dimensionality Reduction Autoencoder

```

1: Initialize autoencoder model
2: for each epoch in 1 to num_epochs do
3:   Set autoencoder to training mode
4:   Initialize train_loss  $\leftarrow 0$ 
5:   for each (images, _) in train_loader_auto do
6:     Move images to device
7:     Zero gradients of optimizer
8:     Compute autoencoder output: outputs  $\leftarrow$  autoencoder2(images)
9:     Compute reconstruction loss: loss  $\leftarrow$  BCELoss(outputs, images)
10:    Backpropagate loss
11:    Update model parameters using optimizer
12:    Accumulate training loss
13:  end for
14:  Compute average training loss
15:  Reduce Dimensions:
16:  (latent_features, labels)  $\leftarrow$  reduce_dimensions(train_loader_auto, autoen-
    coder.encoder, device)
17:  Detach latent_features
18:  Reconstruct Images:
19:  reconstructed_images  $\leftarrow$  autoencoder.decoder(latent_features)
20:  Reshape reconstructed_images to (N, 1, 28, 28)
21: end for

```

FedAvg Algorithm

The series of experiments uses the FedAvg algorithm to simulate the federated learning environment. Algorithm 5 shows Federated Averaging algorithm. This algorithm takes all the local weights, sums them up and returns the average in the end. The average weights then are being redistributed back to the clients.

Algorithm 5 Federated Averaging Algorithm

```

1: procedure FEDERATEDAVERAGING(local_weights)
2:   Initialize avg_weights as a list of zero arrays, same shape as the first model's
   weights
3:   for each weights in local_weights do
4:     for  $i = 1$  to  $\text{length}(\text{avg\_weights})$  do
5:        $\text{avg\_weights}[i] \leftarrow \text{avg\_weights}[i] + \text{weights}[i]$ 
6:     end for
7:   end for
8:    $\text{num\_models} \leftarrow \text{length}(\text{local\_weights})$ 
9:   for  $i = 1$  to  $\text{length}(\text{avg\_weights})$  do
10:     $\text{avg\_weights}[i] \leftarrow \text{avg\_weights}[i] / \text{num\_models}$ 
11:   end for
12:   Return avg_weights
13: end procedure

```

Partitioning Function

In order to simulate Non-IID data environment, the data has to be partitioned in to subsets equal to the number of clients participating in the training process. The parameter alpha determines how the data is split. The higher the alpha the more balanced the split will be and the lower alpha (<1) the more unbalanced the split will be.

Algorithm 6 Dirichlet Partitioning

```

1: procedure PARTITION(dataset, partitions_number, alpha, seed)
2:   Set random seed to seed
3:   Extract labels y_train from dataset.targets
4:   Compute num_classes  $\leftarrow$  unique classes in y_train
5:   Initialize empty dictionary net_dataidx_map
6:   Create class_indices mapping each class to its sample indices
7:   for each class k in class_indices do
8:     Shuffle indices in class_indices[k]
9:   end for
10:  Initialize idx_batch as an empty list for each partition
11:  Set min_size  $\leftarrow$  10
12:  for each class k in class_indices do
13:    Split class_indices[k] into partitions_number equal parts
14:    for i = 1 to partitions_number do
15:      Assign at least min_size samples from class k to partition i
16:    end for
17:    Remove assigned samples from class_indices[k]
18:  end for
19:  for each class k in class_indices do
20:    Compute remaining samples remaining_indices
21:    Generate Dirichlet proportions for distributing samples
22:    Compute cumulative sample splits
23:    Split remaining_indices based on computed proportions
24:    for i = 1 to partitions_number do
25:      Assign remaining samples to partition i
26:    end for
27:  end for
28:  for i = 1 to partitions_number do
29:    Shuffle idx_batch[i]
30:    Assign idx_batch[i] to net_dataidx_map[i]
31:  end for
32:  Return net_dataidx_map
33: end procedure

```

Clustering

The clustering algorithm utilizes a greedy approach to group clients into clusters based on the similarities of the distributions of their labels. The algorithm begins with establishing the labels distributions for each client, which represents the proportion of each class label in its local dataset. This is done without sharing any raw data and therefore preserves privacy for each client. Once the label distributions are determined, the algorithm assigns each client to some cluster according to the similarity of the client's label distribution and the label distributions for each cluster. The first couple of clients, everyone starts out in different clusters, while for subsequent clients the algorithm determines to which cluster the client has the closest label distribution in comparison to the existing clusters. The sum of the absolute differences of the clients label distribution and the existing clusters' label distribution is used to determine which cluster to assign the next incoming client. As the number of clients increases, the clusters naturally will take shape and accumulate label distributions that are most similar to the clients assigned to them.

The greedy-based clustering approach is computationally efficient as it avoids the need for complex optimization processes, focusing on label distribution comparisons. The nature of the greedy-based clustering mitigates sharing or accessing raw data, only communicating the label distributions, which minimize communication overhead.

Algorithm 7 Clustering

```

1: procedure AGGREGATION(client_distributions)
2:   Return aggregated_data  $\leftarrow$  Convert client_distributions to NumPy array
3: end procedure
4: procedure CLUSTERING(aggregated_data, net_dataidx_map, num_clusters)
5:   num_clients  $\leftarrow$  Length of aggregated_data
6:   Initialize cluster_assignments as an array of  $-1$  for each client
7:   Initialize cluster_label_totals as a zero matrix of shape
   (num_clusters, dim of labels)
8:   Initialize empty dictionary clustered_net_dataidx_map with num_clusters keys
9:   for each client client_id and label distribution label_dist in aggregated_data do
10:    if client_id < num_clusters then
11:      Assign client_id to its own cluster
12:      Set cluster_label_totals[client_id]  $\leftarrow$  label_dist
13:    else
14:      Compute best cluster assignment using minimum label distribution differ-
      ence
15:      Assign client_id to best_cluster
16:      Update cluster_label_totals[best_cluster] with label_dist
17:    end if
18:    Add client indices from net_dataidx_map[client_id] to
    clustered_net_dataidx_map[cluster_assignments[client_id]]
19:  end for
20:  Return clustered_net_dataidx_map
21: end procedure

```

4.3 Pipeline

Each dataset has its own environment. In these environment the procedure is the same for all three datasets. At the start in phase one, the preprocess of data and initialization of models are being done, preparing them essentially for the coming training loop.

The second phase consists of the partition and clustering part. To be able to simulate federated learning, the dataset is split into subdatasets equal to the number of clients which is 10 in this series of experiments. Then there will be a scenario with and without clustering algorithm. If cluster is used, then after the partition. Algorithm 8 shows a short overview of the workflow of one experiment for one specific dataset.

Algorithm 8 Pipeline of the experiments

```

1: Initialize dataset
2: Normalize the dataset
3: Create 3 data loaders for PCA, autoencoder, and classic approach
4: Initialize all models, including autoencoder
5: Pre-train the autoencoder
6: Partition the dataset into subsets for federated learning
7: if using cluster then
8:   Apply clustering algorithm
9: end if
10: Start main training loop
11: for each round do
12:   Perform local training on each node
13:   Aggregate the models
14:   Update global model
15: end for

```

Chapter 5

Evaluation and Results

In this section, the results for the three provided data sets (MNIST, Fashion-MNIST and Kuzushiji-MNIST) under different alpha values are outlined and analyzed. The alpha parameter indicates the degree of unbalanced partitioning of the data in federated learning, where a lower alpha indicates greater unbalance and a higher alpha indicates more balance in partitioning of the data to clients simulating a federated learning systems with non-iid nature. In order to compare the results after applying dimensionality reduction techniques like PCA and Autoencoder with the classic case, where no dimensionality is not reduced, several evaluation metrics are used.

In order to compare the results after applying dimensionality reduction techniques like PCA and Autoencoder with the classic case, where dimensionality is not reduced, several evaluation metrics are used. These include Accuracy, Loss, Precision, Recall, F1-Score, and Training Time, which are defined as follows:

- **Accuracy:** The proportion of instances, that have been correctly classified, to the total number of instances. It provides a general measure of the model's overall performance [22].
- **Loss:** A measure of the model's error, that is captured based on the difference between predicted and actual labels [22].
- **Precision:** The proportion of correctly predicted positive instances among all instances predicted as positive. It is defined as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

where TP (True Positives) are correctly classified positive samples and FP (False Positives) are incorrectly classified negative samples [22].

- **Recall:** The proportion of actual positive instances that are correctly classified. It is defined as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

where FN (False Negatives) are positive samples that were incorrectly classified as negative [22].

- **F1-Score:** The harmonic mean of Precision and Recall, providing a balanced measure when the dataset is imbalanced [22]. It is calculated as:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Training Time:** The total time required to train the model across all federated learning rounds. A lower training time indicates a more efficient approach, which is particularly important in distributed environments.

5.1 Results for different Alphas without clustering

Among all the datasets evaluated, the MNIST dataset of handwritten digits is the simplest. As shown in figure 5.1, the models achieve high accuracy and low loss values in all approaches. The values of almost all evaluation metrics of all three models are so close to each other that almost no differences are noticeable. PCA has the highest loss consistently along all alphas, which could indicate loss of important features. Precision, F1 score, and recall remain consistently high across all methods, with minimal differences. Both dimensionality reduction techniques (PCA and Autoencoder) significantly reduce the training time, with the reduction being more than half compared to the classic approach without dimensionality reduction.

Comparative evaluation of the Kuzushiji-MNIST dataset with the classic approach, PCA, and autoencoder as shown in figure 5.2 demonstrates performing differences in terms of accuracy, loss, precision, recall, F1-score, and training time. The classic approach consistently performs higher in accuracy compared to other approaches. For instance, at $\alpha = 20$, the classic approach achieves the highest accuracy of 87.63% while PCA improves an stable accuracy of 87.27% with all alpha values tested. Comparatively, the accuracy for autoencoder is slightly lower, peaking at 86.73% at $\alpha = 5$ and then begins to decline slightly. While difference in accuracy is small (approximately 0.3%), PCA is performed at a consistent rate, but is unlikely to capture specific variation associated with the dataset like the classical approach or autoencoder. Although the accuracy for the autoencoder is slightly lower, the values contribute to an acceptable range for approximate inference. Evaluating the loss measures demonstrated at PCA obtains the highest loss consistently. PCA obtained loss values starting from 1.3674 all the way down to 0.8596. In relation to the classic approach, loss values are drastically less at farthest apart ($\alpha = 1$ $\alpha = 20$) from those of PCA with loss at a minimum of (0.5507) at $\alpha = 1$. Higher alphas also yielded somewhat lower loss at 0.5634 but yields a incurring loss value of greater loss beyond. These results presented demonstrate that PCA is inducing more errors beyond those of classic and autoencoder approaches.

The most significant difference arises in training time. The classic approach is the slowest, requiring between 1125s and 1839s. The increase in training time at $\alpha = 20$ is likely due to kernel-related inefficiencies, as training time is typically constant and independent of α under normal conditions.

while PCA achieves a drastic reduction, maintaining an average of 199s across all alpha values—over 82% faster than the classic method. The autoencoder, though slightly slower than PCA, still significantly reduces training time, averaging 210s, making it 80% faster than the classic method. This highlights the efficiency advantage of dimensionality reduction.

Among all approaches with the Fashion-MNIST and as shown in figure 5.3, the classic technique achieves the highest accuracy with a value of 88.83% at $\alpha = 10$. In general, as alpha gets higher, the classic method has a positive upward trend on the accuracy measures. Following closely is PCA, which maintains an accuracy of 88.52% at all alpha values. The autoencoder performance is noticeably lower, with its highest measure of accuracy achieving only 85.72% at $\alpha = 1$, which indicates a slight gap between the performance of the autoencoder and the accuracy performance from the classic approach. Similarly, examining the loss values at the different alpha values, PCA displays the highest loss at all alpha values, beginning with an initial high of 1.3674 ($\alpha = 0.1$), with loss progressively decreasing before leveling off around 0.8805 ($\alpha = 20$). The classic technique demonstrates the lowest loss on each alpha for the best result of 0.3567 ($\alpha = 5$) which demonstrated to be the lowest loss, so overall for the accuracy performance measures and with respect to the loss measures, we can explain that classic performance is taking advantage of maintaining the full-dimensional data, as well as lower classification errors. The autoencoder performs better than PCA, although it is still higher in loss compared to the classic method, with the autoencoder achieving the lowest loss of 0.4219 ($\alpha = 1$), overall we can see that the loss progresses to increase as alpha increases.

For the precision, recall, and F1-score metrics, the classic approach clearly outperforms the other two. The classic approach achieves precision: 0.8885, recall: 0.8883, and F1-score: 0.8883 at $\alpha = 10$. All metrics indicate that classic is the most reliable in terms of classification. PCA experiences a significant drop in the metrics, particularly recall, dropping to recall = 0.6927 at $\alpha = 20$ and leading to a lower F1-score of 0.6859. The autoencoder performs between the two methods, with high performance, demonstrating its best precision at 0.8571 ($\alpha = 1$) and recall at 0.8572 ($\alpha = 1$), although it does not reach the performance of the classic method. There is also a substantial difference in training time. The classic method takes the longest compared to the other two methods, taking a range from 1524s ($\alpha = 10$) to 1742s ($\alpha = 20$). PCA trains much quicker and takes an average of time of 220s per alpha, which is approximately 85% to 87% time saving compared to the classic method. The autoencoder was slightly slower than PCA at an average of 230s but was still much quicker than the classic method and provides a reasonable time savings option.

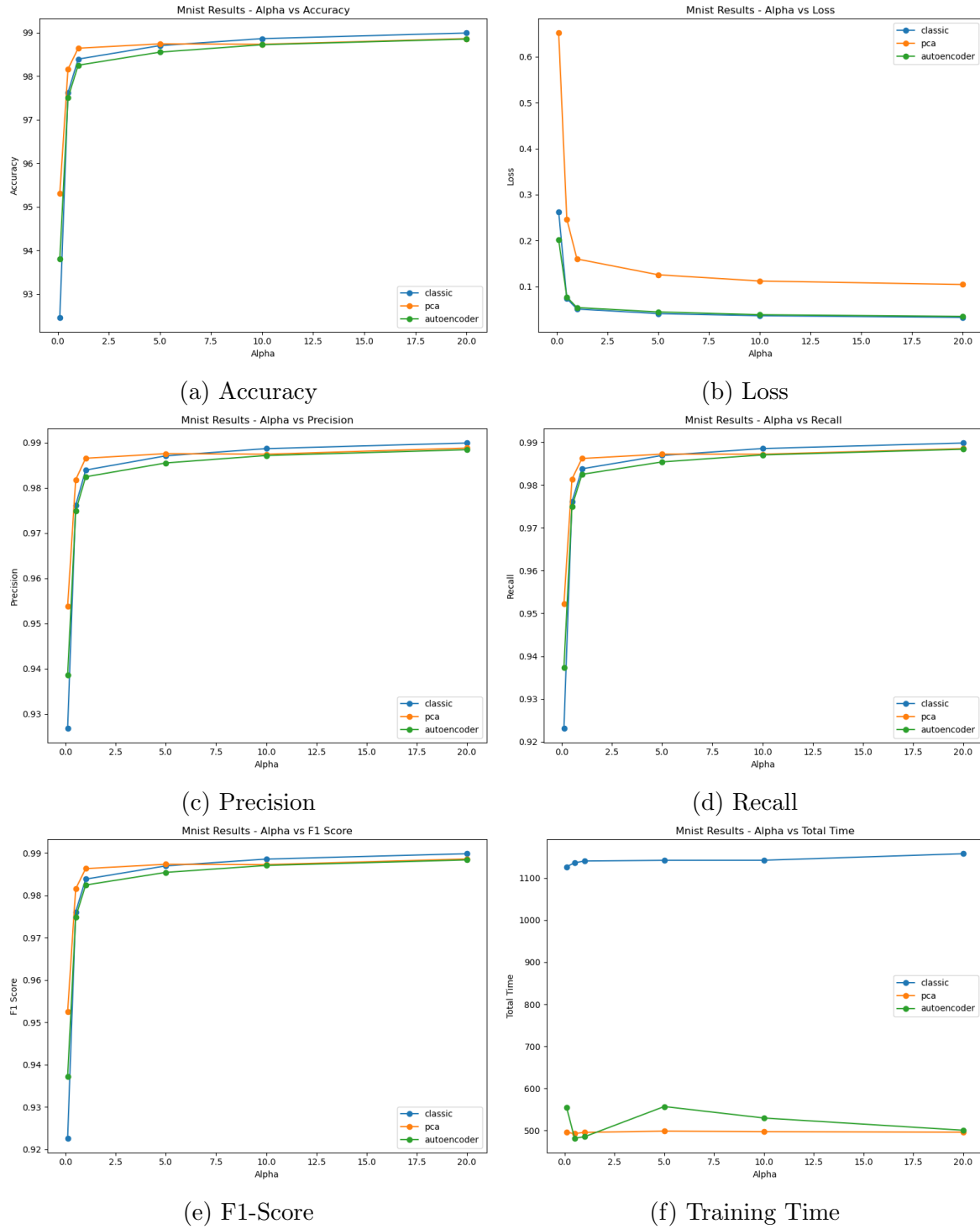


Figure 5.1: Evaluation metrics for the MNIST dataset across different techniques and for different alphas

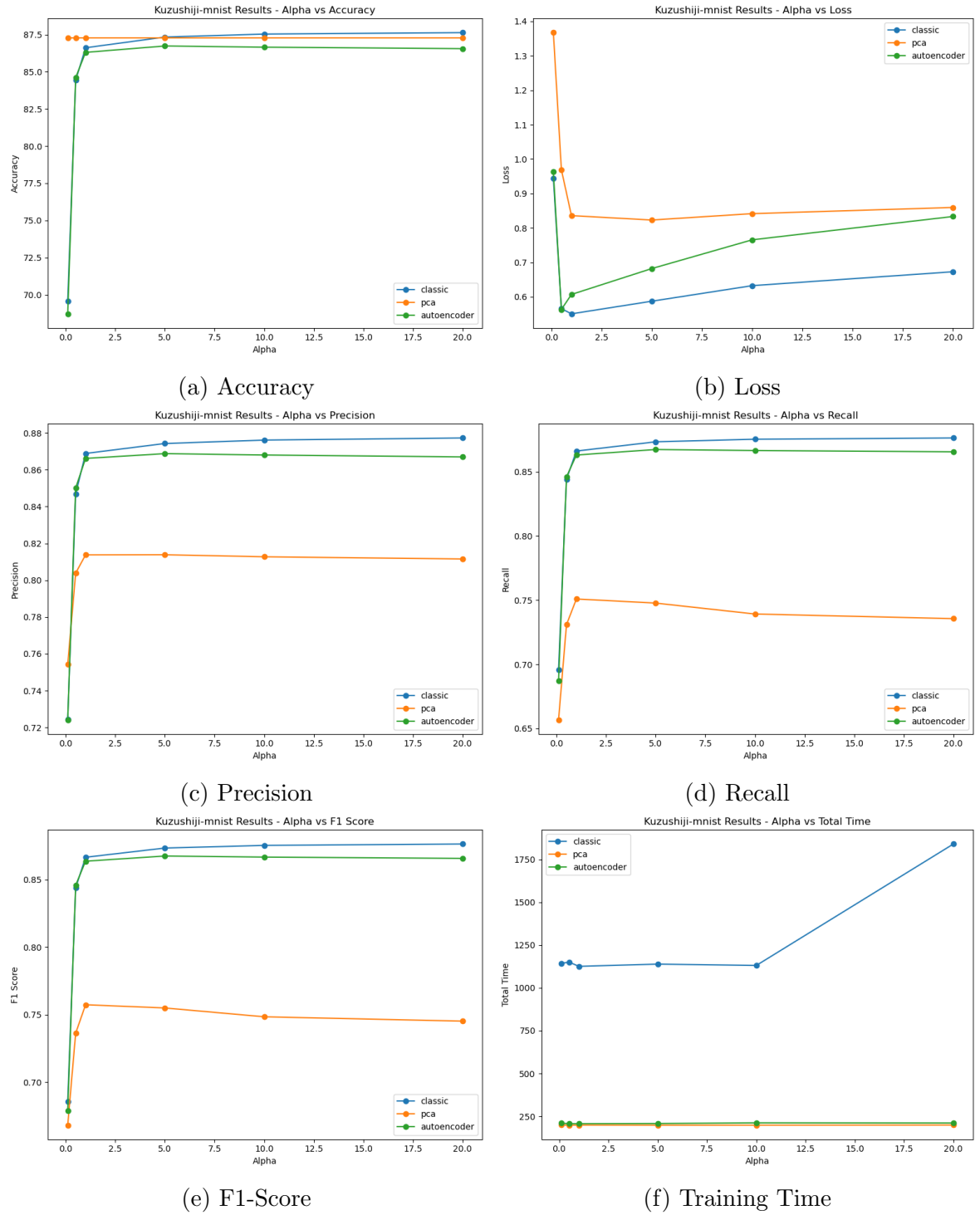


Figure 5.2: Evaluation metrics for the Kuzushiji-MNIST dataset across different techniques and for different alphas

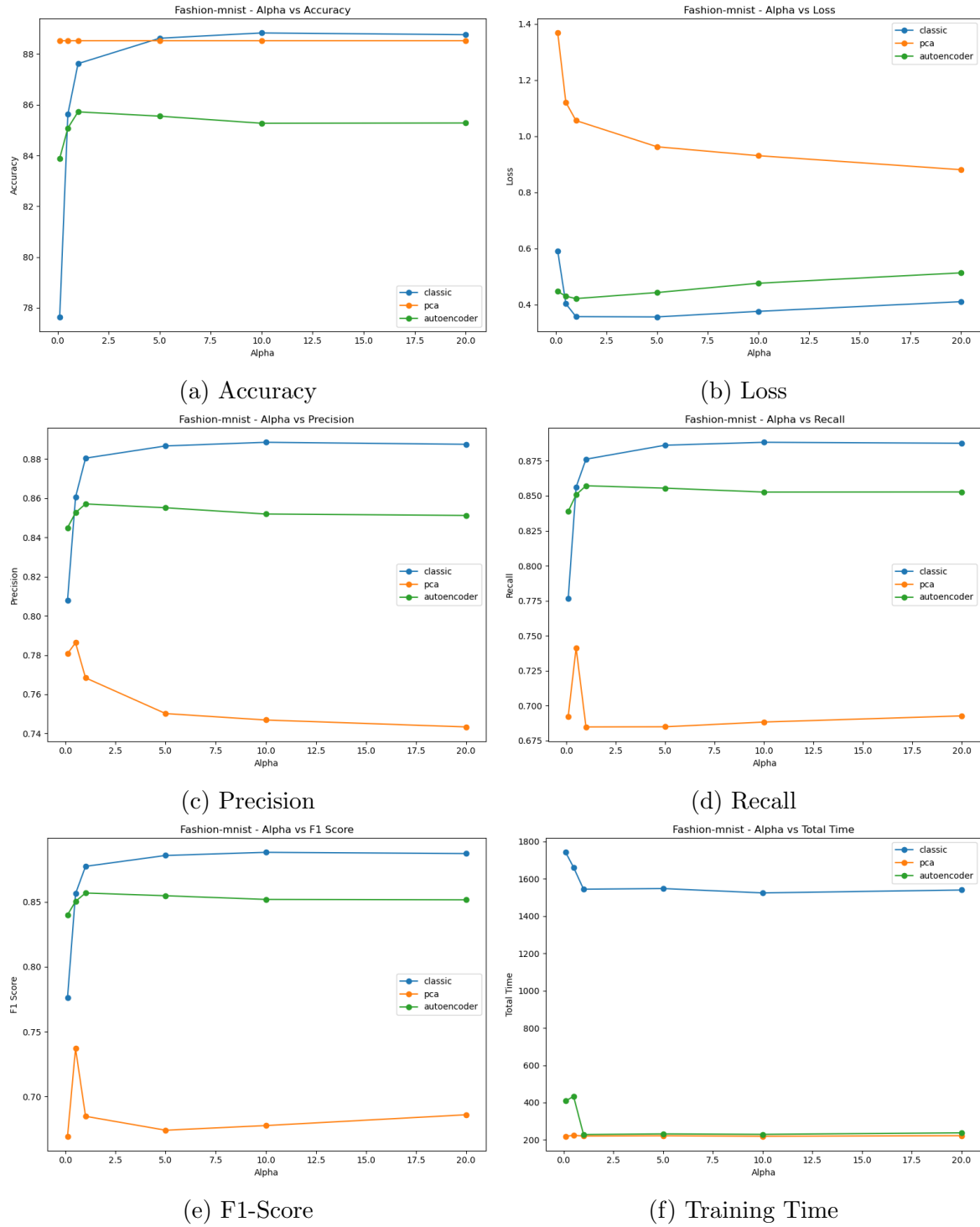


Figure 5.3: Evaluation metrics for the Fashion-MNIST dataset across different techniques and for different alphas

In general in every dataset and as shown in figure 5.4, the classic method achieves the highest accuracy, while PCA and the autoencoder consistently marginally underperformed, especially as increase in the complexity of the dataset increases. In the MNIST dataset, there was little variation, but then the Kuzushiji-MNIST and Fashion-MNIST datasets saw the classic method outperform the other methods, and the autoencoder showed slightly poorer performance. PCA consistently has the largest loss of all of the methods across all of the datasets, which can be described as loss of more important features. The classic method shows the lowest loss, demonstrating a method that retains more discriminative information. The autoencoder outperforms PCA, but does produce more loss than the classic approach even with some improvement, indicating that the autoencoder retains some more information than the PCA, but ultimately loses some of the full dimensional representation. In both precision, recall, and F1-score, the classic method performed the best, which ensured a more stable balance across all evaluation metrics. PCA has the lowest recall and F1-score, as the reduction in dimensionality contributes to misclassifications. The autoencoder outperforms PCA, but does not match the classification performance of the classic approach.

The greatest benefit of PCA and autoencoders compared to the classic approach is the reduced time to train the data. The classic approach is slowest, the PCA approach is the fastest and saves considerable training time, and the autoencoder to train the data takes longer than PCA but is still much faster than the classic approach.

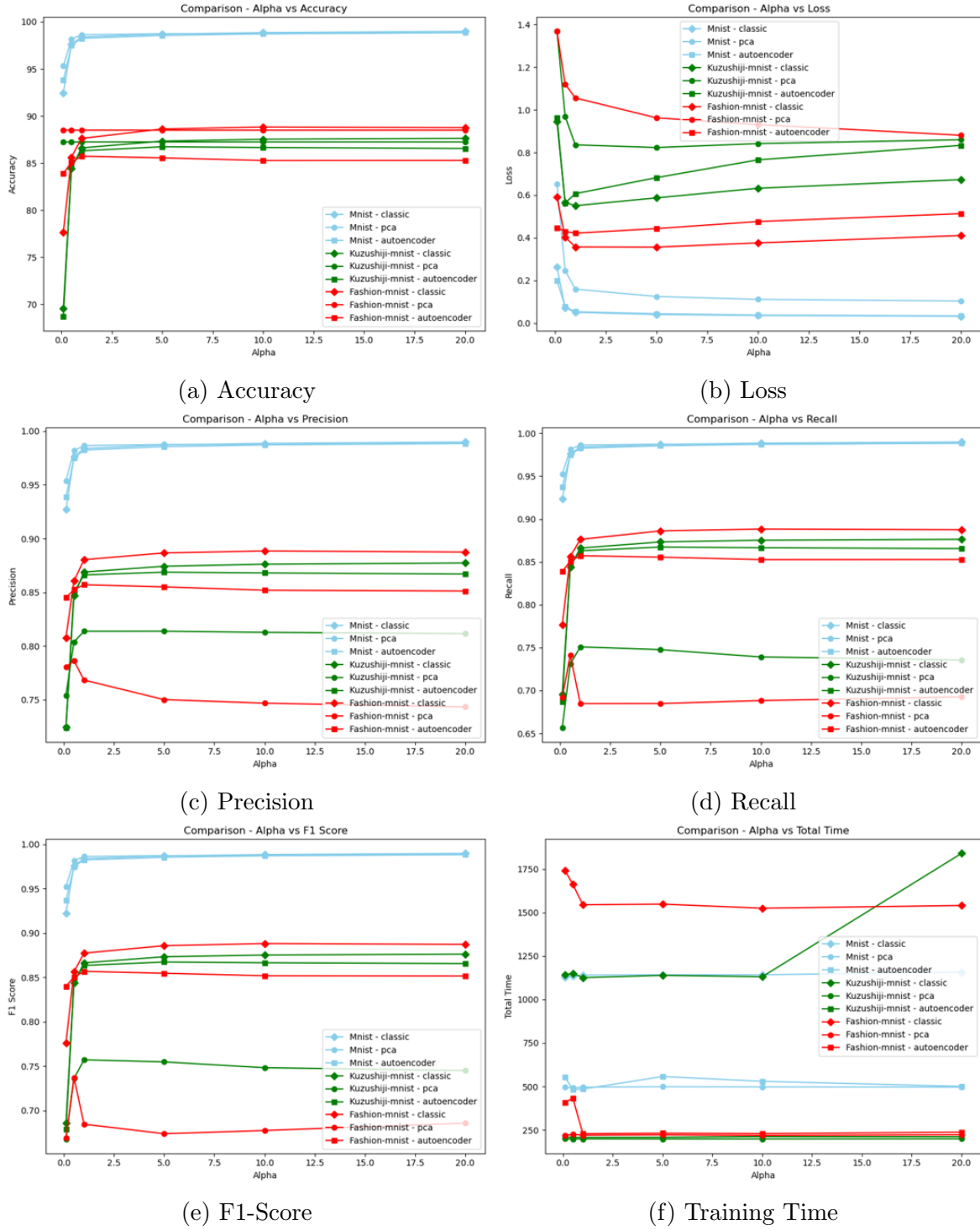


Figure 5.4: Evaluation metrics for all datasets across different techniques and for different alphas

5.2 Results for different Alphas with clustering

To investigate the effect of cluster, the same experiment is repeated after clustering the clients into six clusters and evaluating the matrices over different alphas with different datasets.

As shown in figure 5.5 utilizing clustering enhances the models' effectiveness on the MNIST dataset as compared to the non-clustered setting, particularly in relation to loss and accuracy. Each of the classic (or baseline) model and the dimensionality reducing models provide improvements with clustering- with the classic model improving in accuracy and loss across each of the alpha values. For the classic model, accuracy improved from 98.99% (no clustering) to 99.09% (clustering) in the case of $\alpha = 20$, with loss improving considerably. The precision, recall and F1-score also improved for each model in cluster, further supporting the merits of clustering. Training time held steady when comparing models, it seems that every additional clustering step does not have associated inefficiencies in time. Use of PCA also shows distinct improvements compared with cluster. In cluster, accuracy held steady at 98.81 for $\alpha = 10$, only slightly improving on the non-clustered version at 98.73. PCA still shows the greatest overall loss suggesting that clustering still is not improved on PCA's classifications.

The autoencoder also presents enhancements, and it reached a performance of 98.94% accuracy at $\alpha = 20$, compared to a low 98.85% of the non-clustered autoencoder. There is an overall loss reduction, along with enhancements in precision, recall, and F1-score, which confirms the autoencoder is a solid contender to the classic approach. Despite performance improvements, training time fluctuates more than in the non-clustered setting, which can be attributed to computational overhead or kernel-related inefficiencies. Since training time is primarily compared between models in this study, these fluctuations do not impact the overall conclusions regarding model efficiency.

The models' performance on the Kuzushiji-MNIST dataset after applying clustering and as shown in figure 5.6 tends to be roughly the same at clustered versus non-clustered condition, but with some incremental improvements in accuracy and loss under different approaches. The classic approach continues to achieve the highest accuracy, peaking at 87.33% at $\alpha = 10$ compared to the non-clustered version which peaked at 87.63% at $\alpha = 20$. Loss values were similar, indicating that clustering does not appear to have a strong effect on retention of features. The precision, recall, and F1-score remained stable within approaches throughout clustering, also maintaining the classic approach's reliability. PCA held a fixed accuracy of 87.27 across alphas, just as seen in the non-clustered setting. There were some reductions in loss due to clustering at lower alphas which have potential implications for improved feature retention. However, PCA retains the highest overall loss, suggesting continued limitations in preserving rich features after dimensionality reduction. Precision, recall, and F1-score were similar with some variances exhibited, but ultimately lower relative to the classic approach.

The autoencoder shows slight advancements in accuracy and loss, achieving an accuracy of 86.68% at $\alpha = 5$, a slight improvement over 86.55% for a non-clustered scenario. Loss values are lower than for PCA but continue to rise as alpha increases, which reinforces that autoencoders do retain more meaningful representations than PCA, but do not entirely

reach the efficiency seen in the classic method. The precision, recall, and F1-score remain relatively the same, providing further evidence of its stable classification performance.

When applying clustering and as shown in figure 5.7, the performance across the various approaches on the Fashion-MNIST dataset remains mostly consistent with the non-clustered setting, with only slight modifications in accuracy and loss across the approaches. The efficiency ranking of the approaches remains unchanged, with the classic approach continuing to have the best overall classification performance, PCA still being the fastest and the autoencoder producing an acceptable balance between efficiency and accuracy. The classic approach continues to achieve the highest accuracy at 88.56% when $\alpha = 10$, which is a touch lower than the observed 88.83% accuracy in a non-clustered scenario. The loss values remain comparable with slight fluctuations at higher alpha values. The precision, recall, and F1-score remain virtually unchanged, reinforcing the conclusion that clustering does not affect the classic approaches overall classification ability.

PCA has consistent accuracy of about 88.52% across all alpha values, as in the non-clustered setup. Still, clustering leads to slightly lower loss at the higher alpha values, which may suggest a small improvement in how well PCA keeps important features. Even with this improvement, PCA has still the highest loss and lowest recall and lowest F1-score, confirming it struggles with fine-detail retention when dimensionality is reduced. The findings suggest that although clustering is beneficial, it does not offset the loss of information from PCA. The autoencoder improves slightly in accuracy from $\alpha = 1$ at 85.88%, compared to 85.72%, in the non-clustered setting. Loss, which is better than PCA's loss, but increases at the higher alpha values suggests that while the autoencoder is possibly capturing more meaningful representations than PCA, it continues to fall short of matching the classic approach in classification performance.

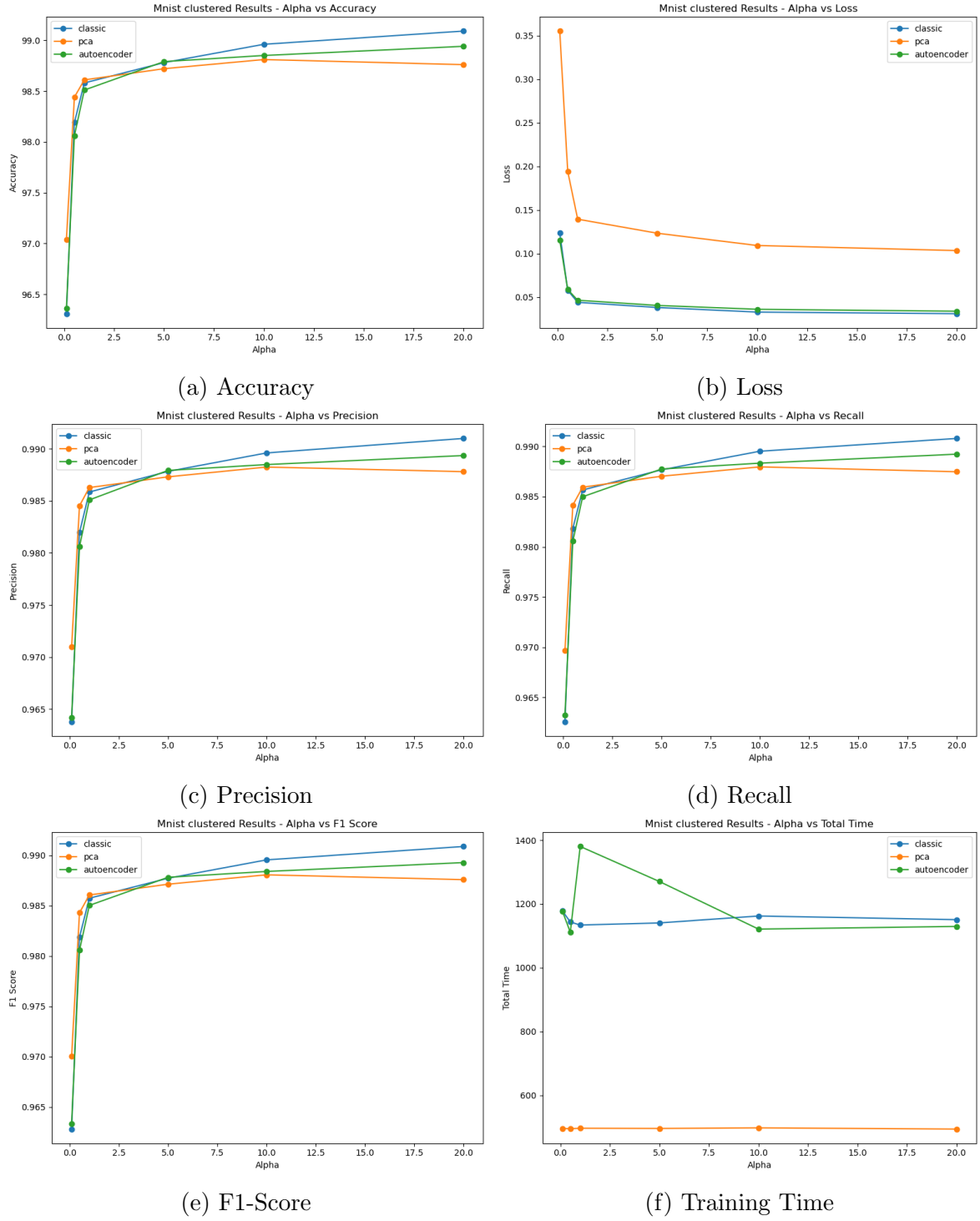


Figure 5.5: Evaluation metrics for the MNIST dataset clustered across different techniques and for different alphas

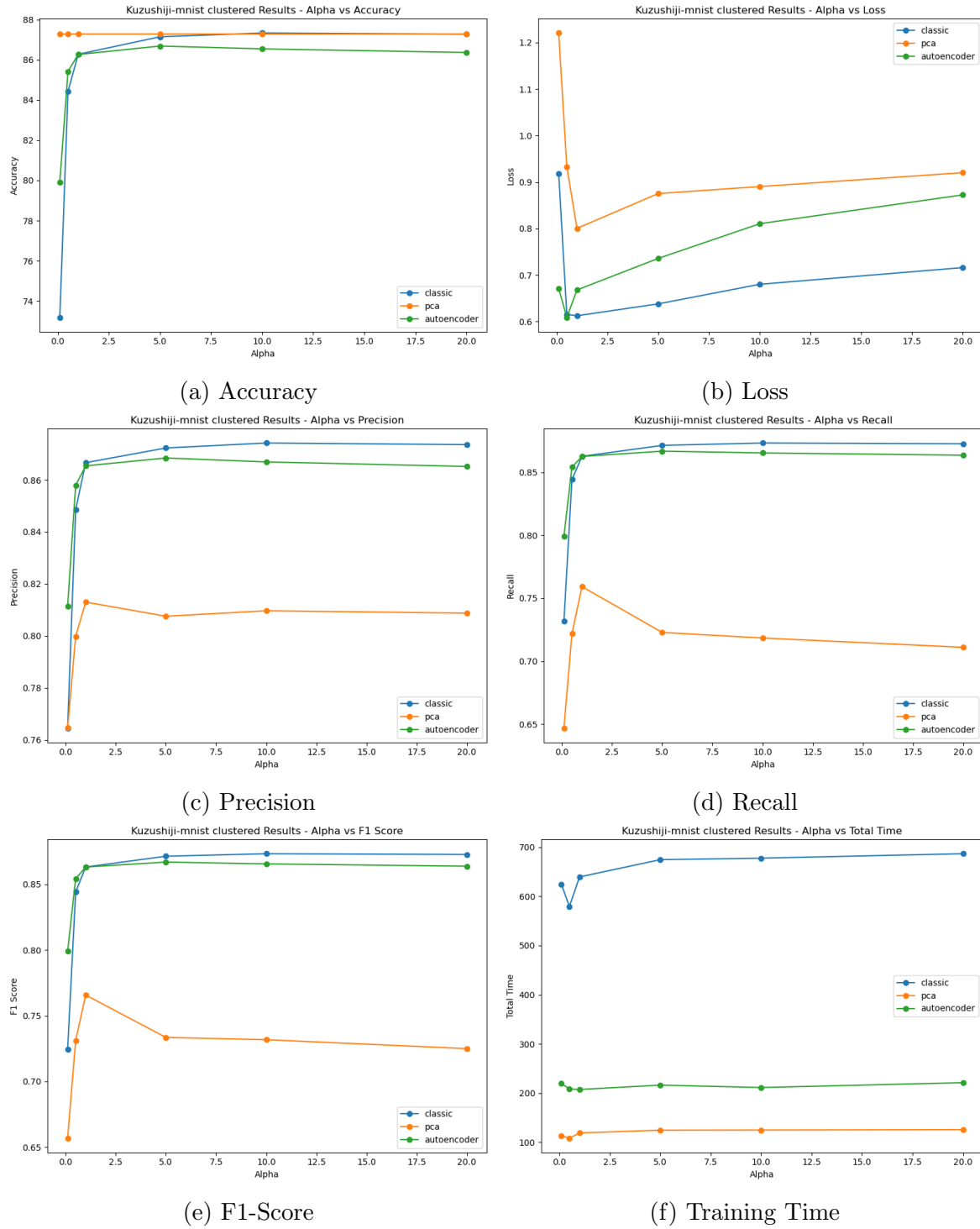


Figure 5.6: Evaluation metrics for the Kuzushiji-MNIST dataset clustered across different techniques and for different alphas

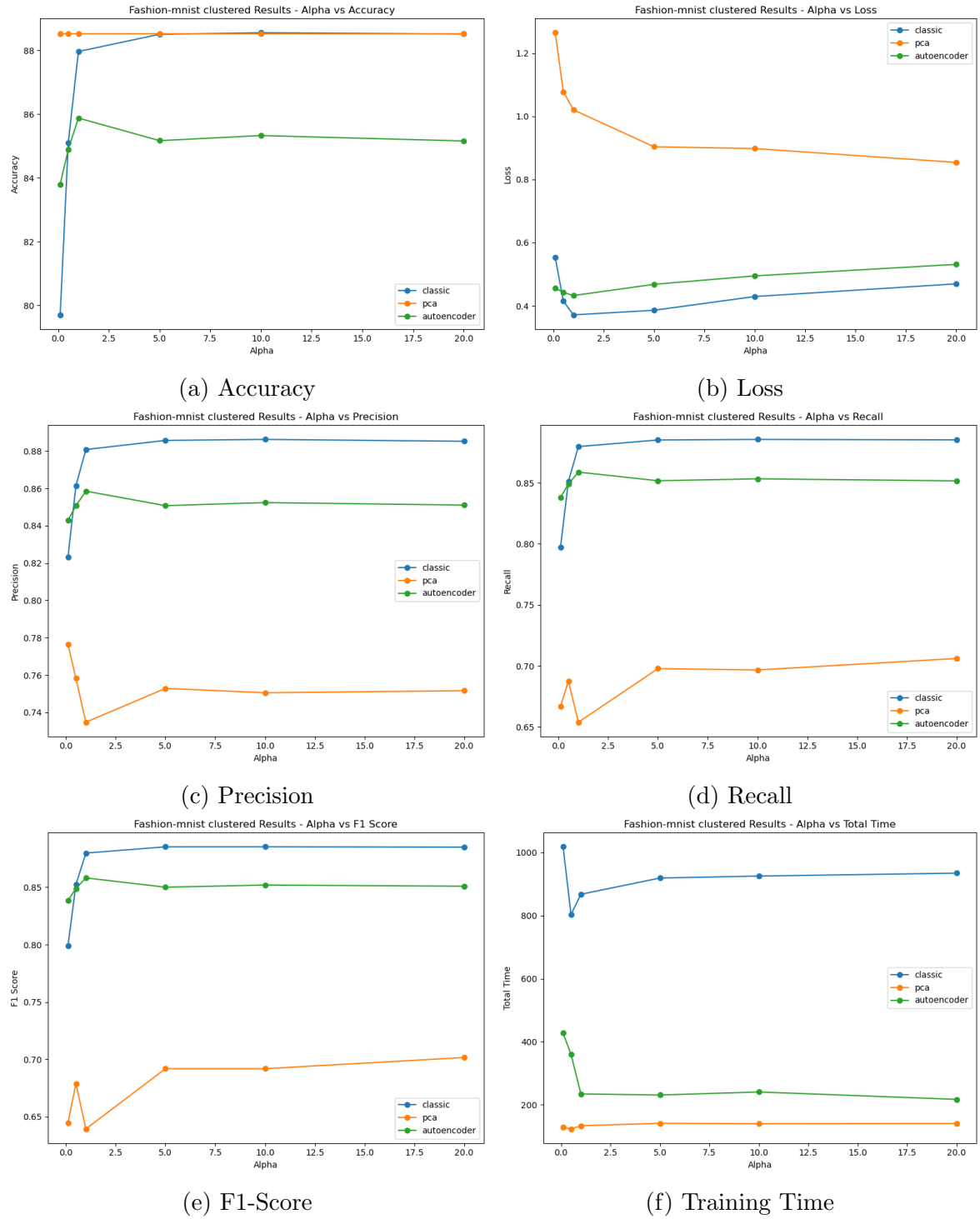


Figure 5.7: Evaluation metrics for the Fashion-MNIST dataset clustered across different techniques and for different alphas

Thus, even after applying clustering the following is observed:

The overall performance trends across MNIST, Kuzushiji-MNIST, and Fashion-MNIST remain consistent. Almost all evaluation matrices show improvement compared to the non-clustered results. The classic method has the highest accuracy and lowest loss, while the PCA and autoencoder methods have improved computational efficiency at some cost to classification performance. Dimensionality reduction has different effects depending on dataset complexity, but there is a consistent trade-off between accuracy and time across all dimensionality reduction methods. The classic method maintains the highest accuracy and lowest loss, a sign that it has not lost features that are important, in comparison with the dimensionality reduction methods, but it does this at a cost, as it is the least time-efficient. PCA is the quickest method, which would support it being an appealing option for efficiency, but has also the highest loss and lowest recall, which indicates it has lost some important information. The autoencoder, although slower than PCA, retains important relevant features that result in better classification performance, thus providing a good balance of efficiency and accuracy over the other approaches as shown in figure 5.8.

Across all datasets, we observe a general trend where dimensionality reduction (PCA & autoencoder) is considerably faster in terms of training time compared to the classical method, which benefits from access to higher-dimensional data. In general, PCA observed the least training time, followed by the autoencoder method. Overall, the classical method was consistently at least twice as slow, but it also produced the least classification error; at least more than twice the error as dimensionality reduction methods. These results demonstrate that while the classical method was suitable for tasks where one was concerned with achieving the most accuracy and little concern for how long it took to train the task, both dimensionality reduction methods offered a suitable alternative that will continue to yield relatively high accuracy and save time as shown in figure 5.8.

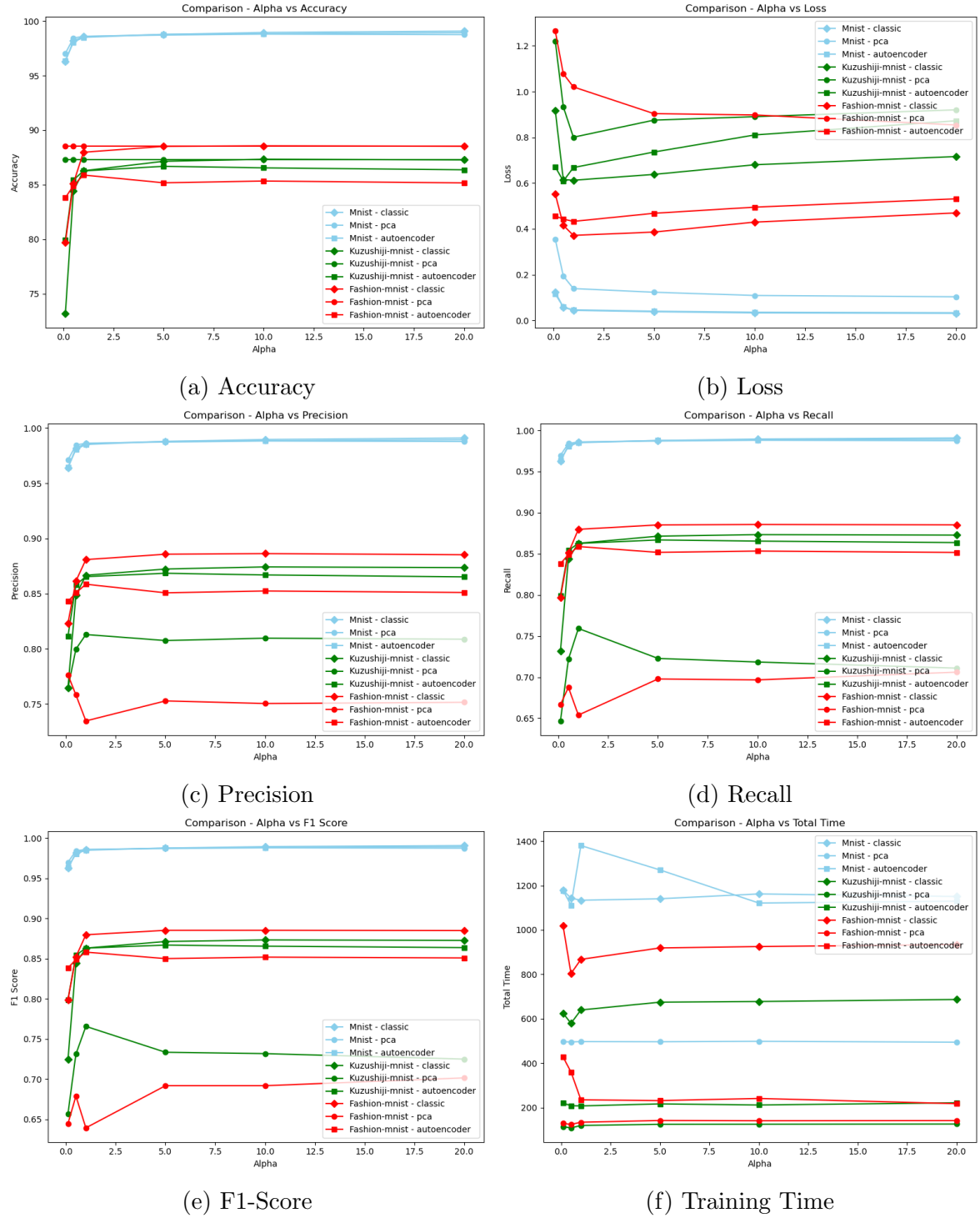


Figure 5.8: Evaluation metrics for all datasets clustered across different techniques and for different alphas

5.3 The effect of clustering on balanced and imbalanced data

In order to better understand the effect of applying clusters, the evaluation matrices are compared after applying cluster on highly unbalanced data ($\alpha = 0.1$) and highly balanced data ($\alpha = 10$).

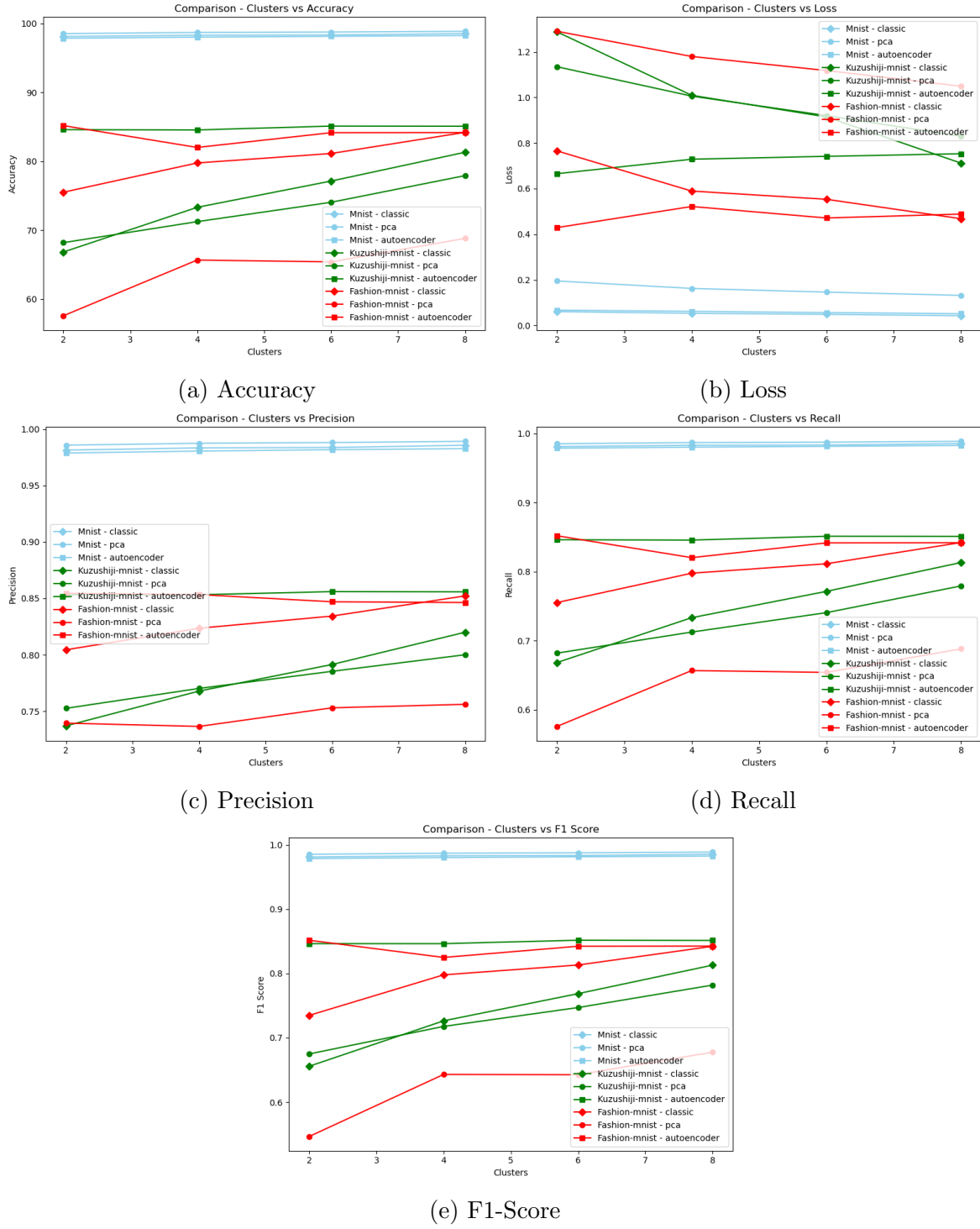
As shown in figures 5.9, 5.10 the evaluation of clustering effects on strongly imbalanced data ($\alpha = 0.1$) and strong balanced data ($\alpha=10$) shows marked differences in model performance. In general, models obtain higher accuracy, precision, recall and F1-score in the balanced data scenario compared to the imbalanced data scenarios, which has resulted in higher loss. This behavior may be explained by having difficulty learning from skewed data distributions, where the minority classes are neglected and classification becomes more difficult. In contrast, balanced data provides a more stable training environment making it easier for models to generalize across all classes.

An important finding from this analysis is that the model performance is quite sensitive to the number of clusters considered. At lower numbers of clusters (2 or 4), the models appear to be worse performing in terms of accuracy (lower) and loss (greater). This has to do with the clustering strategy utilized in this work where the first *num_clusters* clients are assigned as initial cluster representatives directly. This choice was made in order to limit computational costs and to avoid kernel crashes. Indeed, re-assigning the clusters as to maximize cluster allocations would likely have taken more resources than were available for this large-scale experiment. The first clients are chosen arbitrarily, and the global cluster structure is thereby dependent on the first assigned label distributions across clients. Thus, if there was great skew in the class distributions of the first clients, the cluster assignments may fail to balance any distributions identified later in the clustering process affecting classification performance. In contrast, as the number of clusters increases, the chances of finding representative initial clusters improves resulting in balanced distributions within clusters and thus better model performance.

In the unbalanced scenario ($\alpha = 0.1$), clustering has the most impact on improving evaluation metrics, particularly in the more complicated datasets, Kuzushiji-MNIST and Fashion-MNIST. Clustering has a greater impact on models with increasing number clusters, where the evaluation metrics, accuracy, recall, and F1-score are improved. On MNIST dataset, clustering still positively affects performance, but not to the same extent that is seen with Kuzushiji-MNIST and Fashion-MNIST, where the models' classification performance is already quite high even when fewer clusters are used or without clustering. In terms of the methods used, PCA has the most success from clustering, especially since clustering can mitigate the slow loss of important features through the process of dimensionality reduction. However, PCA struggles with the more complicated data sets, as the model fails to distinguish the finer differences in labeling after further reducing dimensions. The autoencoder also showed some positive benefit from clustering, but not as much as the classic (i.e., no dimensionality reduction). This can be attributed to the autoencoder approach utilizing its own feature extraction function within the evaluation.

In the balanced case ($\alpha = 10$), clustering still effects positively, although to a lesser extent—because the data already has an even distribution, the model performance is more

stable across metrics, and it becomes more of a small adjustment instead of a significant adjustment effect. The performance of MNIST dataset does not change much when increasing the number of clusters, which indicates that clustering is less of a factor when the dataset is sufficiently balanced. The performance of the autoencoder is also nearly the same, which reaffirms the adequacy of the internal representation. For more challenging datasets such as Kuzushiji-MNIST and Fashion-MNIST, there are still slight improvements from clustering; however, with the exception of providing performance gains and improvements, the overall improvement diminishes with a degree of clustering beyond six, suggesting that excessive clustering does not necessarily enhance classification performance further, which is in this experiment expected, since the total clients participating are ten clients.

Figure 5.9: Evaluation metrics for all datasets for different clusters and $\alpha = 0.1$

5.3. THE EFFECT OF CLUSTERING ON BALANCED AND IMBALANCED DATA 53

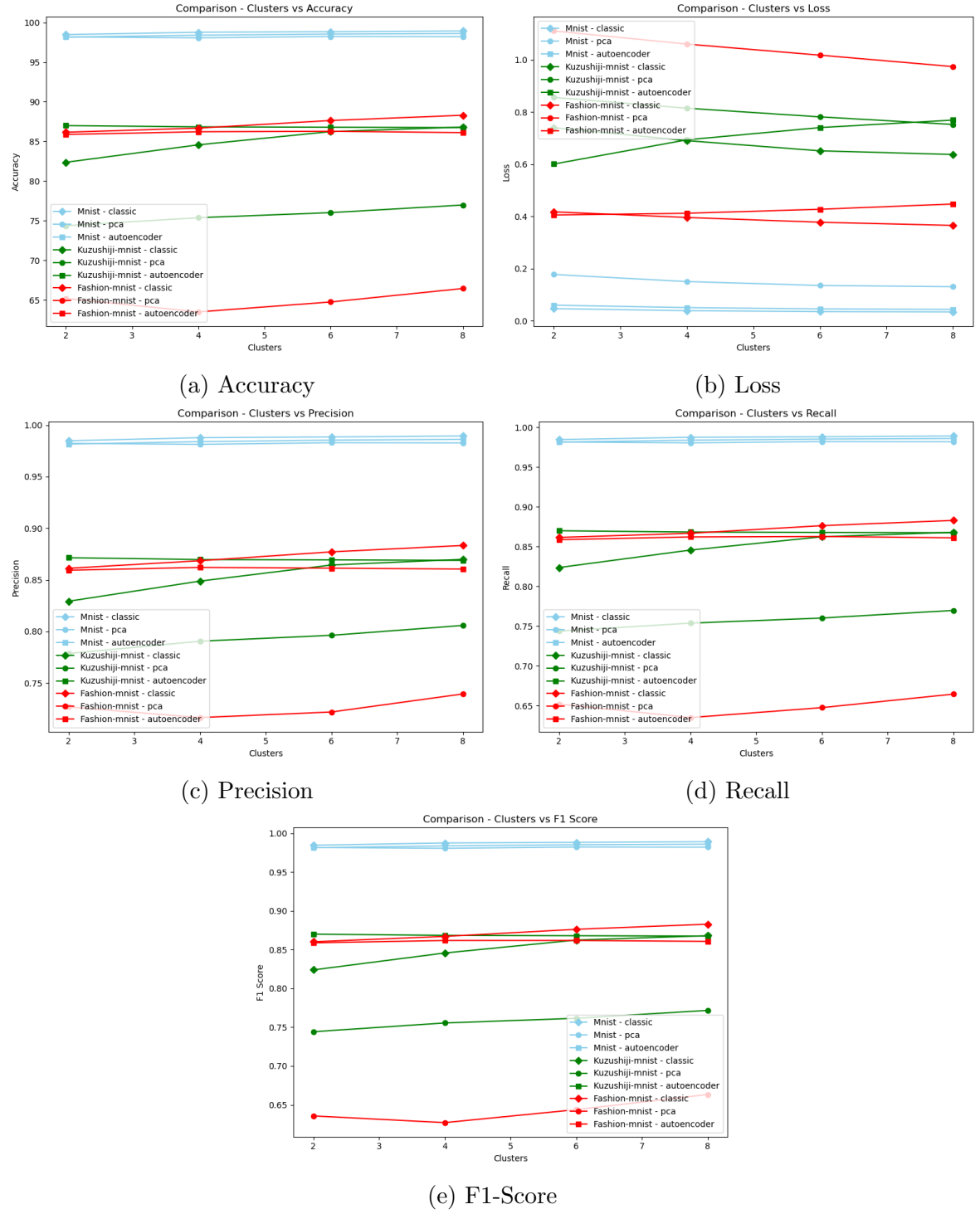


Figure 5.10: Evaluation metrics for all datasets for different clusters and $\alpha = 10$

Chapter 6

Summary and Conclusions

The goal of this project was to explore a field in federated learning which has not been thoroughly researched yet. The project investigated the impact of dimensionality reduction methods, particularly PCA and Autoencoder in federated learning setting.

The focus was on investigating whether the positive effect of dimensionality reduction can be used enable more efficient approach in federated learning. Another focus was on mitigating the negative effects of dimensionality reduction by applying a clustering algorithm, while mitigating its potential drawbacks, particularly in non-IID (non-independent and identically distributed) data scenarios, through clustering algorithms.

Even though the experiment environment does not cover all possible scenarios, the results were satisfying. The main discovery is that using dimensionality reduction massively reduces the training time on all three datasets used in this project. This comes with a prize which lowers the overall performance of the model, but the difference is not extreme. The gap got even closer when the clustering algorithm was applied, making the difference acceptable.

PCA and Autoencoder fulfill similar purposes, though in this project it was evident that for the more complex datasets (beyond the classic MNIST) PCA begins to show a problem capturing the underlying structure of the datasets due to it being limited to linear operations. The Autoencoder, however, has shown its advantages over PCA, because it has the ability to capture more complex structures thanks to its neural network structure which is able to introduce non-linearity via non-linear activation functions and deeper layers.

The clustering algorithm provided a slight improvement in performance by refining data partitioning, demonstrating its potential to enhance learning outcomes, particularly in scenarios where data distribution is highly imbalanced.

6.1 Future Work

It is clear that dimensionality reduction has an impact on machine learning in federated learning environment. However the experiments do only cover a small area of available scenarios. This project for example only considered three datasets. This can be expanded further into more datasets which are even more complex than the current ones in order for example to understand the gap between PCA and Autoencoder better.

It is also important to notice, that there could be a case where the model has to get more complex too in more advanced experiments with more complex datasets. This opens up another area for research, where the impact of dimensionality reduction is diminished by more complex models.

Another limitation in this project was that the computational resources were limited. This is why the maximal number of clients is ten. In a setup where more computational power is available the number of parameters can be increased to a degree where the results are even more informative especially in determining for example the optimal number of clusters.

Moreover, the clustering method utilized here is greedy-based, meaning that cluster assignment is done sequentially based on the current state of the system, without reviewing all distributions first to obtain an optimal clustering. This approach was pursued to reduce computational costs and minimize kernel crash, but may potentially result in suboptimal clustering. Future work can assess different clustering approaches that yield better assignment while remaining computationally efficient.

Bibliography

- [1] P. M. Mammen, “Federated learning: Opportunities and challenges”, *ACM Conference Proceedings*, 2021. DOI: 10.1145/nnnnnnnn.nnnnnnnn.
- [2] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, “A review of applications in federated learning”, *Computers & Industrial Engineering*, vol. 149, p. 106 854, 2020. DOI: 10.1016/j.cie.2020.106854.
- [3] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, “A survey on federated learning”, *Knowledge-Based Systems*, vol. 216, p. 106 775, 2021, ISSN: 0950-7051. DOI: <https://doi.org/10.1016/j.knosys.2021.106775>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950705121000381>.
- [4] C. Hu, J. Jiang, and Z. Wang, *Decentralized federated learning: A segmented gossip approach*, 2019. arXiv: 1908.07782 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1908.07782>.
- [5] E. T. Martínez Beltrán, M. Q. Pérez, P. M. S. Sánchez, *et al.*, “Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges”, *IEEE Communications Surveys Tutorials*, vol. 25, no. 4, pp. 2983–3013, 2023. DOI: 10.1109/COMST.2023.3315746.
- [6] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, *Communication-efficient learning of deep networks from decentralized data*, 2023. arXiv: 1602.05629 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1602.05629>.
- [7] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, *Advances and open problems in federated learning*, 2021. arXiv: 1912.04977 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1912.04977>.
- [8] S.-A. Wegner, *Mathematische Einführung in Data Science*, German. Berlin, Heidelberg: Springer Berlin Heidelberg, 2024, ISBN: 3662686961, 9783662686966. DOI: 10.1007/978-3-662-68697-3.
- [9] D. Scholz, *Numerik interaktiv*, English. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, ISBN: 3662529408, 9783662529409. DOI: 10.1007/978-3-662-52940-9.
- [10] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006, ISBN: 978-0387310732.
- [11] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks”, *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006. DOI: 10.1126/science.1127647.

- [12] H. Bourlard, *Auto-association by multilayer perceptrons and singular value decomposition*, 2000. [Online]. Available: <https://infoscience.epfl.ch/handle/20.500.14299/227943>.
- [13] A. Morales-Forero and S. Bassetto, “Case study: A semi-supervised methodology for anomaly detection and diagnosis”, in *2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 2019, pp. 1031–1037. DOI: 10.1109/IEEM44572.2019.8978509.
- [14] J. Kwon and H. Park, “Efficient low-rank federated learning based on singular value decomposition”, in *Proceedings of the Twenty-Third International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc ’22)*, New York, NY, USA: ACM, 2022, pp. 285–286. DOI: 10.1145/3492866.3561258. [Online]. Available: <https://doi.org/10.1145/3492866.3561258>.
- [15] B. Liu, B. Pejó, and Q. Tang, “Privacy-preserving federated singular value decomposition”, *Applied Sciences*, vol. 13, no. 13, p. 7373, 2023. DOI: 10.3390/app13137373. [Online]. Available: <https://doi.org/10.3390/app13137373>.
- [16] F. Artoni, A. Delorme, and S. Makeig, “Applying dimension reduction to eeg data by principal component analysis reduces the quality of its subsequent independent component decomposition”, *NeuroImage*, vol. 175, pp. 176–187, 2018. DOI: 10.1016/j.neuroimage.2018.03.016.
- [17] S. C. Ng, “Principal component analysis to reduce dimension on digital image”, in *Procedia Computer Science*, vol. 111, Elsevier, 2017, pp. 113–119. DOI: 10.1016/j.procs.2017.06.017.
- [18] A. Grammenos, R. Mendoza-Smith, J. Crowcroft, and C. Mascolo, “Federated principal component analysis”, in *NeurIPS*, Conference on Neural Information Processing Systems, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/federated_pca.pdf.
- [19] H. S. Sikandar, S. ur Rehman Malik, A. Anjum, A. Khan, and G. Jeon, “Cohort-based kernel principal component analysis with multi-path service routing in federated learning”, *Future Generation Computer Systems*, vol. 149, pp. 518–530, 2023. DOI: 10.1016/j.future.2023.07.037.
- [20] M. Al-Digeil, Y. Grinberg, D. Melati3, *et al.*, *Pca-boosted autoencoders for nonlinear dimensionality reduction in low data regimes*, 2022. arXiv: 2205.11673 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2205.11673>.
- [21] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, *Deep learning for classical japanese literature*, cite arxiv:1812.01718Comment: To appear at Neural Information Processing Systems 2018 Workshop on Machine Learning for Creativity and Design, 2018. DOI: 10.20676/00000341. [Online]. Available: <http://arxiv.org/abs/1812.01718>.
- [22] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks”, *Information Processing and Management*, vol. 45, no. 4, pp. 427–437, 2009. DOI: 10.1016/j.ipm.2009.03.002.

List of Figures

2.1	A central server with 5 clients	5
2.2	The clients exchange parameters with their neighbors	6
4.1	The first column represents the modern hiragana characters	23
4.2	Example of the titanic dataset	24
5.1	Evaluation metrics for the MNIST dataset across different techniques and for different alphas	38
5.2	Evaluation metrics for the Kuzushiji-MNIST dataset across different techniques and for different alphas	39
5.3	Evaluation metrics for the Fashion-MNIST dataset across different techniques and for different alphas	40
5.4	Evaluation metrics for all datasets across different techniques and for different alphas	42
5.5	Evaluation metrics for the MNIST dataset clustered across different techniques and for different alphas	45
5.6	Evaluation metrics for the Kuzushiji-MNIST dataset clustered across different techniques and for different alphas	46
5.7	Evaluation metrics for the Fashion-MNIST dataset clustered across different techniques and for different alphas	47
5.8	Evaluation metrics for all datasets clustered across different techniques and for different alphas	49
5.9	Evaluation metrics for all datasets for different clusters and $\alpha = 0.1$. .	52
5.10	Evaluation metrics for all datasets for different clusters and $\alpha = 10$. .	53

List of Tables

4.1	Labels and categories in the Fashion-MNIST dataset	22
4.2	Comparison between MNIST and Fashion-MNIST datasets	22
4.3	CIFAR-10 Categories	23
4.4	Activation Functions for Different Output Layers	25

Appendix A

Installation Guidelines

A.1 Introduction

This guideline explains how to install and run the experiment associated with the project. The experiment is implemented in Python using Jupyter Notebook and requires libraries like pytorch, torchvision and sklearn.

A.2 Prerequisites

1. Python: Install from <https://www.python.org>
2. Jupyter Notebook: Install via pip or Anaconda (<https://www.anaconda.com/>)

A.3 Setup the project

1. Clone the repository or download the ZIP file: <https://github.com/Seouless29/Federated-Dimensionality-Reduction.git>
2. `pip install requirements.txt`
3. adjust the relative path for importing models, datasets, and functions

A.4 Folders

1. The folder Final Experiments consists of files considered for the results
2. The data folder consists of datasets

3. The function folder consists of functions used for the project such as pca, FedAvg etc.
4. The folder model consists of models used for the project. For the final results, only model 2 and 4 are used as training model. The Autoencoder 2 is used for dimensionality reduction.
5. The Playground folder consists of all Experiment conducted over the duration of the project which are not being used for the final results