DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, *SWEDEN 2019*

Federated Learning for Time Series Forecasting Using LSTM Networks: Exploiting Similarities Through Clustering

FERNANDO DÍAZ GONZÁLEZ

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Abstract

Federated learning poses a statistical challenge when training on highly heterogeneous sequence data. For example, time-series telecom data collected over long intervals regularly shows mixed fluctuations and patterns. These distinct distributions are an inconvenience when a node not only plans to contribute to the creation of the global model but also plans to apply it on its local dataset. In this scenario, adopting a *one-fits-all* approach might be inadequate, even when using state-of-the-art machine learning techniques for time series forecasting, such as Long Short-Term Memory (LSTM) networks, which have proven to be able to capture many idiosyncrasies and generalise to new patterns. In this work, we show that by clustering the clients using these patterns and selectively aggregating their updates in different global models can improve local performance with minimal overhead, as we demonstrate through experiments using real-world time series datasets and a basic LSTM model.

Keywords

Federated Learning, Time Series Forecasting, Clustering, Time Series Feature Extraction, Recurrent Neural Networks, Long Short-Term Memory

Sammanfattning

Federerad inlärning för tidserieprognos genom LSTM-nätverk: utnyttjande av likheter genom klustring

Federated Learning utgör en statistisk utmaning vid träning med starkt heterogen sekvensdata. Till exempel så uppvisar tidsseriedata inom telekomdomänen blandade variationer och mönster över längre tidsintervall. Dessa distinkta fördelningar utgör en utmaning när en nod inte bara ska bidra till skapandet av en global modell utan även ämnar applicera denna modell på sin lokala datamängd. Att i detta scenario införa en global modell som ska passa alla kan visa sig vara otillräckligt, även om vi använder oss av de mest framgångsrika modellerna inom maskininlärning för tidsserieprognoser, Long Short-Term Memory (LSTM) nätverk, vilka visat sig kunna fånga komplexa mönster och generalisera väl till nya mönster. I detta arbete visar vi att genom att klustra klienterna med hjälp av dessa mönster och selektivt aggregera deras uppdateringar i olika globala modeller kan vi uppnå förbättringar av den lokal prestandan med minimala kostnader, vilket vi demonstrerar genom experiment med riktigt tidsseriedata och en grundläggande LSTM-modell.

Acknowledgements

First of all, I would like to express my gratitude to my industrial supervisors, Tony Larsson and Johan Haraldson, their invaluable assistance and the numerous meetings we had helped me to channel my efforts most efficiently, and allowed me to overcome many obstacles and challenges. Their help has been crucial for the completion of this work.

I would like to thank professor Sarunas Girdzijauskas for his reviews, and professor Henrik Boström for his insightful and interesting pieces of advice which provided me with an opportunity for improving the quality of this thesis.

Thanks also to my colleague Yuntao Li, for his critical ears and continuous discussions. And finally, thanks to my friends Eva Gil and Giorgio Ruffa for their trust and for making this master a life-changing journey.

Author

Fernando Díaz González <fdiaz@kth.se>
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology

Place for Project

Stockholm, Sweden
Ericsson Research

Examiner

Henrik Boström
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology

Supervisor

Sarunas Girdzijauskas
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology

Contents

Acronyms

ACF Autocorrelation Function

DTW Dynamic Time Warping

FFT Fast Fourier Transform

KPI Key Performance Indicator

LSTM Long Short-Term Memory

MASE Mean Absolute Scaled Error

MIMO Multiple Input Multiple Output

MLP Multi Layer Perceptron

MSE Mean Square Error

SGD Stochastic Gradient Descent

SMAPE Symmetric Mean Absolute Percentage Error

SSE Sum of Squares Error

RNN Recurrent Neural Network

UPGMA Unweighted Pair Group Method with Arithmetic Mean

WPGMA Weighted Pair Group Method with Arithmetic Mean

List of Equations

Chapter 1

Introduction

1.1 Background

Edge devices have become more powerful and smarter over the last years. Nowadays their capabilities allow them to fulfil many different roles and they no longer serve as a mere middleman to control the flow of data between two networks [1]. On the other hand, the rise in the volume and variety of the data generated at the edge has exposed the limitations of cloud computing [2]. For certain scenarios, it turns out to be impractical and inefficient to log large volumes of data to a data centre in order to process it [3]. Likewise, it is not possible to upload the raw data to the cloud when the data generated at the edge is private [4]. These kinds of concerns gave rise to the emergence of a new computing framework which aimed to offload cloud computing by moving processing closer to data: edge computing [5].

One of the most promising applications of edge computing is linked to the recent success of artificial deep learning. Machine learning techniques based on neural networks need large datasets and their performance increases with the volume of data available [6], however, this dataset might not be centrally available because of the inherently distributed nature of the data, for example, when data is generated at the edge. According to International Data Corporation (IDC), "By 2019, at least 40% of IoT-created data will be stored, processed, analysed, and acted upon close to, or at the edge of the network" [7]. This growth requires a new learning technique that can be deployed at the edge.

Researchers in this field have proposed different algorithms to overcome the limitations of general distributed deep learning. The most popular nowadays is federated learning [8], a family of algorithms based on differential privacy [9] that attempts to solve the problem described above. The challenge is that this one-fits-all solution might not be ideal when there are different underlying structures present in the data. Take the data generated at base station antennas in a company like Ericsson. Each edge device (antenna) computes different Key Performance Indicators (KPIs) over time that can be fed into models to derive deep insights to optimise and tailor the behaviour of the system [10]. However, different devices gather data with different patterns (e.g., antenna in a rural area vs antenna in urban areas), and each device cares more

about capturing the pattern in the local data than in any other device's data, so the *one-fits-all* solution introduced by this framework might not be sufficient when training on highly non-IID data. In this work, we propose a method to address the system's challenges associated with data heterogeneity.

Federated learning is a learning technique that allows to collectively train a shared model without the need to store the data centrally. In this approach, a server coordinates a loose federation of participating devices (which we refer as *clients* or *nodes* indistinctively throughout this work) to solve the learning task. Each one of the clients has a local dataset which never shares with the coordinating server. Instead, the server sends the latest version of the shared model to the clients and these clients update the model using their local datasets. The clients only communicate back their local updated model to the central server. The server combines the partially trained models to form a federated model. One way to combine the models is to take the average of each coefficient, weighting by the amount of training data available on the corresponding node. This dialogue between clients a server is repeated multiple times to improve the central model maintained by the server.

The goal of federated learning is to fit a model to data generated by a large number of m distributed nodes $\boldsymbol{X}_1; \boldsymbol{X}_2; :::; \boldsymbol{X}_m$. The number of data points on each node $|\boldsymbol{X}_t|$ varies significantly, and so the distribution that generates the data $\boldsymbol{X}_t \sim P_t$. Another property of federated optimisation is that communication with these nodes is frequently expensive or slow; however, in the telecommunication field, communication with edge devices like base stations is not as costly as communication with end-user devices. In the following section, we expose in detail the statistical problem that derives from having heterogeneous data in this setting.

1.2 Problem

Federated learning aims to capture the relationship between the distributions associated with each node by fitting a single global model with the ability to generalise to many patterns present in each node's local dataset. The data at each node X_t is generated by a different distribution X_t P_t, hence, data heterogeneity (i.e., data samples from two nodes are likely to be different) becomes a central property of this setting. However, when fitting the global model, the central aggregator cares more about how well the model captures the patterns present in all the data $\boldsymbol{X}_1 \cup \boldsymbol{X}_2 \cup \cdots \cup \boldsymbol{X}_m$, than in any other node's data $\boldsymbol{X}_t$ individually. This might be a problem when a node not only plans to contribute to the creation of the global model but also plans to apply it on its local dataset. In this case, the local performance (i.e., the performance of the global model on a node's dataset) might be in tension with the global performance (i.e., the performance of the global model on data sampled from all the nodes). For example, if the network is training an image label model and a specific node is more likely to generate nature images, that node cares more about the global model working well with nature images than with any other kind of images. In other words, the data at each node X_t is generated by a different distribution X_t P_t, hence, the data heterogeneity that arises from this setting.

Resolving this tension is the primary goal of this thesis, which has been largely influenced by

previous work that attempted to solve a similar problem. For example, Virginia Smith et al. also identified the very same statistical challenge in federated learning, but they modelled the relationships amongst the nodes in the network via distributed multi-task learning, where each node's model is a task, but there exists a structure that relates the tasks [11]. Bandara et al. performed two studies in which they group time series into similar subgroups before using LSTMs to learn across each subset of time series; in the first study, they make the groups by extracting characteristics from the time series [12]; in their second study, they also group on available domain knowledge [13]. Finally, Aymen Cherif and Hubert Cardot also divide the time series into subsets based on similar characteristics before using a recurrent neural network for prediction [14].

The problem with the aforementioned strategies is that the clustering approach has not been studied under the same considerations of the federated optimisation scenario (i.e., non-IID distributed data in a privacy-preserving scenario). All of these studies assume that the data is centrally available and make use of traditional learning techniques. Besides, the only research that considers the federated scenario models a solution in which there is no notion of similarity between node's datasets. We consider this gap worthy of further investigation.

One way of reducing the tension between the local models and the global model is by fitting separate global models simultaneously. Instead of training a single global model, the idea is to train as many models as data patterns are present in the underlying data distributions in the network. Building upon the previous example, if 50% of the nodes are generating nature images and the other 50% are generating urban landscape images, fitting two separate models can improve local performance by paying a small overhead associated with the clustering process.

In this way, a natural way of clustering the clients in a federated network is by clustering the patterns that each local data distribution exhibits. To this end, each client is represented by a feature vector which captures the idiosyncrasies of the local dataset. The information in this feature vector should not allow the central coordinator to decrypt it to its original form (i.e., the raw time series)., otherwise, we would be leaking user-level privacy [15, 16]. Once we have categorised each client in different groups, we apply the federated optimisation algorithm to each one of the groups separately. This is reasonable considering that some clients might be observing data with similar patterns. For example, concerning radio base stations, the traffic load measured at a base station in the centre of Stockholm could be similar to the traffic load in another base station located in the centre of Gothenburg; however, more likely both will be different from the traffic load in an antenna located in a rural area.

In summary, we propose an approach that differs from the vanilla federated learning model in the sense that, instead of training just one global model, we cluster the clients according to their underlying data distributions and then aggregate the updates of each cluster independently, thus generating as many global models as groups or clusters we first identified. We focus on time series data, where a notion of similarity between groups within the same dataset exists. In chapter 4 - Federated clustering we present this method in detail and later, in chapter 5 - Results, we show that by following this approach, local performance is improved with no significant overhead.

1.3 Purpose

Our research question is mainly motivated by the studies of Bandara et al. [12, 13] and Smith et al. [11]. The former concludes that, in the presence of disparate time series, clustering is an intuitive approach to exploit the similarities between series to train better predictive models. The later also stress the need for capturing the relationship amongst nodes and their associated, distinct distributions in the context of federated learning. Taking into consideration these two studies, a natural question that arises is:

> *To what extent the local performance of the global model generated by the conventional federated averaging algorithm can be improved by building separate models for subgroups of similar clients, specified by a feature-based clustering methodology?*

Which leads us to the following hypothesis:

> *When dealing with heterogeneous data in federated learning, the one-fits-all approach might be limited, and fitting separate models for subgroups of similar local datasets will outperform the original solution due to the inability of the former one to effectively fit many idiosyncrasies.*

The above research question covers two aspects, to be specific, 1) what we define as local performance, and 2) the notion of similarity between datasets.

1. Let us define what local performance is with an example. Suppose a federated network of m distributed nodes $\boldsymbol{X}_1; \boldsymbol{X}_2; ::: ; \boldsymbol{X}_m$ where $\boldsymbol{X}_i$ is the local dataset at node i. Given a neural network with a selection of weight values $\boldsymbol{w}$, we define a performance function $P(\boldsymbol{w}; \boldsymbol{X})$ as the performance (e.g., mean average squared error) of the neural network on dataset $\boldsymbol{X}$. We define local performance as $\frac{1}{m} \sum_{i=1}^{m} P(\boldsymbol{w_j}; \boldsymbol{X}_i)$, where $\boldsymbol{w_j}$ can be a) the single generated global model in the case of vanilla federated learning, or b) the model generated by the cluster to which node t belongs.

2. As it was mentioned in section 1.1 - Background, varying the features extracted from the dataset or the similarity metric used can have a decisive influence on the outcome of the clustering process. Therefore, the notion of similarity is dependent on certain parameters that should be studied.

Answering the previous research question allows building automated forecasting methods that can work with distributed and heterogeneous data.

1.4 Objectives

The goal of this thesis is to evaluate if there is any local performance gain in federated learning when using the previous clustering approach. This goal is in line with the answer to the proposed research question. In order to do so, we further break down the goal in different milestones to effectively tackle the problem:

1. Building a framework for Federated Learning, since there is no out-of-the-box solution available at the moment of writing this document.

2. Propose a feature-based representation for a time-series dataset that does not leak private information about the data.
3. Propose a similarity metric for the proposed representation.
4. Adapt the federated averaging algorithm to allow for the clustering of clients.
5. Evaluate the performance of this approach and compare it with the performance of the original federated learning algorithm.

1.5 Benefits, Ethics and Sustainability

Many forecasting applications in the time series domain can benefit from this work to produce more accurate results. The variety and velocity of the data in edge computing and, in particular, federated learning, pose a fundamental challenge when the goal is to extract actionable information from this data [17]. The suggested approach allows data scientists to focus on the model engineering step, offloading the task of data understanding to the proposed algorithm itself. This is reasonable since, due to privacy concerns, effective data exploration could be arduous or even impossible given the scale, the heterogeneity, and the massively distributed nature of the data.

On the other hand, moving data processing closer to the edge grant different benefits. It lowers communication costs due to the smaller operational costs of processing the data in a local device vs a data centre. Reduces network traffic because less data is sent to the cloud, thus reducing network traffic bottlenecks. It also improves application performance due to the lower latency achieved, allowing for real-time data analysis. The last one is of crucial importance for companies that wish to tailor the behaviour of their systems, as in the case of Ericsson, which applies machine intelligence on radio base station sites to achieve network automation [18]. A clear example of the value of real-time processing is Envision, a power producer company which cut its data analysis from minutes seconds, enabling it to increase the wind turbines production by 15 per cent [19].

Finally, we could also think of the potential ethical benefits that federated learning might create. In the last years, we have heard about numerous scandals involving how companies have been exploiting users private data for the development of AI technologies, for example, the Facebook–Cambridge Analytica data scandal [20]. Federated learning is set to disrupt the current AI paradigm, in which better algorithms comes at the expenses of collecting more personal data. With federated learning, it is possible to learn powerful models without the need for transferring data to a central entity. In this way, this tool can be used to extract insights from sensitive personal data without compromising the confidentiality of the participants.

1.6 Delimitations

This thesis is focused on the concept of local performance introduced in section 1.3 - Purpose. We want to find if the performance is affected after grouping similar time series within a heterogeneous dataset. To do so, some decision must be made, particularly, the choice of feature extracted from the time series, the distance used to compare these representations and the clustering algorithm used to group the representations (and therefore the time series) accordingly. However, this is not an extensive study on time series clustering. Our decision regarding the foregoing aspects is driven by the need for gathering enough empirical evidence. That is why we rely on the work of other authors when deciding on the best strategy to perform time series clustering. In simpler words and to illustrate what has been previously said, if we choose a particular set of features to use with the clustering algorithm, it is because other authors used those characteristics in a similar problem and it worked for them. The same applies when choosing a particular clustering algorithm. This is not a study of the effectiveness of clustering algorithms. We considered that for this specific problem, and given the reduced number of elements to cluster, opting for a hierarchical clustering algorithm due to its flexibility and the ability to guide how many and what kind of groups there are in the dataset is the right decision. Finally, even though we are concerned about protecting user-level privacy, recent studies have concluded that the original federated averaging algorithm is not as secure as previously assumed. For example, Wang et al. propose a framework incorporating GAN with a multitask discriminator, which simultaneously discriminates category, reality, and client identity of input samples (i.e., data leakage) [15]. This makes very difficult to determine with total confidence that the proposed method will not leak any data at all, and it is why we consider that this type of study would entail a completely different line of research.

1.7 Outline

The rest of this work is organised as follows: in chapter 2 we introduce the relevant theory and the related work; chapter 3 presents the research methodology used to validate our hypothesis; chapter 4 presents three different experiments; chapter 5 displays the results of the different experiments and in chapter 6 we discuss the previous evaluation results.

Chapter 2

Extended background

This chapter introduces the relevant theory of our research. The problem at discussion is tightly coupled with multiple areas. First, since the main goal is to improve forecast accuracy, it is mandatory to introduce this problem and talk about the different techniques available for time series forecast. We then introduce the idea of time series clustering as a mean of partition time series data into groups based on similarity. After that, we introduce federated learning as a learning mechanism in the context of distributed deep learning. Finally, we extend on machine learning methods based on neural networks that can be used to solve the forecasting problem.

2.1 The time series forecasting problem

2.1.1 Motivation

Time series forecasting is all about predicting the future. This information can help organisations to make calculated decisions to reduce risks and increase returns. If a company can predict when unwanted events will occur, they can manage their resource accordingly to minimise loss. For example, Ericsson makes forecasts regularly to support internal decisions and planning using historical data from traffic measurements in over 100 live networks covering all major regions of the world [21]. For other companies, like Uber, "forecasting enables to predict user supply and demand in a spatio-temporal fine granular fashion to direct driver-partners to high demand areas before they arise, thereby increasing their trip count and earnings" [22].

Therefore, forecasting is of utmost importance for any business, since it guarantees efficient utilisation of capital and correctness of management decisions.

2.1.2 Time series data

A time series can be seen as a sequence of observations along with some information about when those values were recorded. This historical data can be denoted as $y_1; : : : ; y_T$. For example, the

monthly observations in Table 2.1.1 could be represented as $< 4; 10; 10; 4; 1 >$:

Month	Observation
June	4
July	10
August	10
September	4
October	1

Table 2.1.1: An example of a monthly time series with length 5

Since we have monthly data, with one observation per month, we can unequivocally represent the time component of the time series with either the starting or ending month. Time series can also exhibit some patterns that are useful to describe the data, the two most important for this research being trend and seasonality. It is necessary to define these patterns carefully as we often reference them throughout our work.

A *trend* (T_t) exists when there is a long-term increase or decrease in the data. The trend does not have to be linear; it can also be exponential and can also disappear over time.

Time series can also contain *seasonal* patterns (S_t), a cycle that repeats over time, such monthly or yearly. When we define a time series, we usually denote the frequency with an integer number that represents the number of observations per season. For example, if the time series consists of daily observations and the data has annual seasonality, the frequency value for this data would be 365, as the pattern repeats every 365 observations. However, if the observations were aggregated on a monthly basis, then the frequency would be 12.

When the data present these patterns, it is possible to think of the series as a combination of the previous two components plus the remainder (R_t) which represents the residuals from the seasonal plus trend fit. This is a useful abstraction that can help forecasting models to understand the data. For example, if we assume an additive decomposition, then we can write:

$$y_t = S_t + T_t + R_t; \tag{2.1}$$

where y_t is the data, S_t is the seasonal component, T_t is the trend component, and R_t is the remainder component, all at period t. Alternatively, a multiplicative decomposition would be written as:

$$y_t = S_t \times T_t \times R_t: \tag{2.2}$$

It is also possible to log-transform the data to stabilise the variations over time, and then use an additive decomposition:

$$y_t = S_t \times T_t \times R_t \quad \text{is equivalent to} \quad \log y_t = \log S_t + \log T_t + \log R_t: \tag{2.3}$$

These decompositions are useful when using machine learning models since we can avoid learning the seasonality component by just fitting the remainder plus the trend and adding the seasonal component later. This is advantageous since the seasonality could be unnecessarily challenging to fit. But they are also helpful for statistical methods. For example, each one of the different profiles in Figure 2.1.1 leads to a different model formulation for the application of the exponential smoothing methodology.

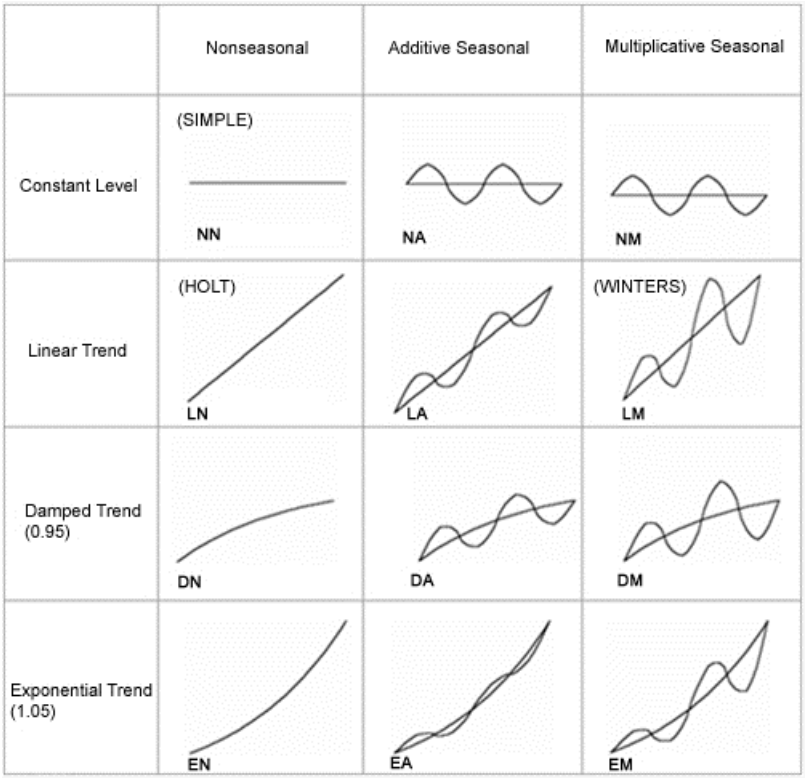


Figure 2.1.1: Forecast profiles for Exponential Smoothing. Adapted from E. Gardner, Journal of Forecasting, Vol. 4 (1985) [23]

2.1.3 Simple statistical methods

We introduce two simple statistical methods for forecasting because they are often reported in multiple competitions [24, 25, 26, 27] as the baseline to beat. Moreover, both of them are used to define the evaluation metrics that we adopt in this work, so it is important to know the underlying theory on which these statistical tools are based.

Let us recall the primary goal of forecasting. When we forecast, we are trying to determine a h-step ahead prediction $y_{T+1};::::;y_{T+h}$ based on the historical data $y_1;::::;y_T$. We can denote this estimate as $\hat{y}_{T+hjT}$. There are multiple methods for estimating these future observations, some of them extremely simple and surprisingly effective.

Naïve method

A naive forecast sets all the future values to be the value of the last observation. That is,

$$\hat{y}_{T+h|T} = y_T: \tag{2.4}$$

This method works well with economic and financial time series.

The error produced by this method is often compared with the error of more elaborate methods to get an idea of how well the forecast model is. If a method can not beat the naive forecast, or if it can, but the effort put into it is not worth the improvement, then the naive forecast should be used.

Seasonal naïve method

When the time series has a very strong seasonal component, a straightforward way to improve the accuracy of the previous method is to set the value of each forecast to be the last observed value from the same season. For example, with daily data (1 observation per day) and yearly seasonality (pattern repeats every year), the forecast for all future days are equal to the value observed at the same day of the previous year. Formally, this is

$$\hat{y}_{T+h|T} = y_{T+h-m(k+1)}; \tag{2.5}$$

where $m =$ the seasonal period, and k is the integer part of $(h-1)/m$ (i.e., the number of complete years in the forecast period prior to time $T+h$). Building on the previous example, if we suppose that we have 2 years' worth of data and we want to predict the value for the following day (i.e., $h = 1$, $m = 365$, $k = 0$), we have

$$\hat{y}_{T+1|T} = y_{T+1-365}$$

If instead, we want to predict the observation after 366 days (i.e., the following day after 1 year, $h = 366$, $m = 365$, $k = 1$), we have:

$$\hat{y}_{T+366|T} = y_{T+366-730} = y_{T+1-365}$$

Notice how both predictions use the value observed the year before.

Figure 2.1.2 shows a comparison of the two aforementioned forecasts for a series from the NN5 dataset. This series consists of daily data with a weekly pattern, and the goal is to predict the next 56 observations. Hence, $h = 56$, $m = 7$, $k = 0$. Note the accuracy of the seasonal naïve forecast for this series. The naïve method is very inaccurate due to the strong seasonality component.

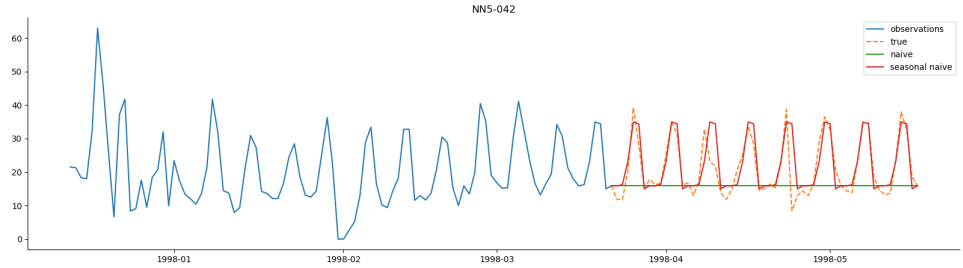


Figure 2.1.2: Naïve, seasonal naïve and ground truth forecasts for series 42 of the NN5 dataset.

2.1.4 Machine learning methods

Forecasting methods based on neural networks are gaining popularity recently as they have been proven able to match traditional statistical methods [28]. Despite their drawbacks, like their more significant computational requirements, they are the only model that can be used with federated learning to scale to millions of time series and thousands of devices. Federated averaging cannot aggregate multiple ARIMA or ER models in order to create a single global model that collects the learning outcomes achieved respectively by each individual statistical model. In computer science terminology, using statistical models with federated learning is implausible because there is no *reduce* function to apply after the *map*. This does not necessarily mean that the predictions of different distributed statistical models could be combined in an ensemble fashion. However, combining the models in such a way would is not equivalent to training one statistical global model across the network, something impossible because, as we just said, it is not feasible to combine the different this type of updates which would be the local statistical models themselves (e.g., parameters p, d, and q in ARIMA), not their outputs.

Machine learning methods have been proved to work with all sort of time series: financial time series [29], the direction of the stock market index [30], macroeconomic variables [31] and balance sheet and profit and loss accounts [32]. There are multiple families of machine learning algorithms that can be used to produce forecast like Multi-Layer Perceptron (MLP), Bayesian Neural Network (BNN), CART regression trees (CART) or Recurrent Neural Networks (RNN). In this work, we focus on two algorithms that have attracted substantial interest in the forecasting field: Recurrent Neural Networks (RNN) [33, 34] and Long-Short Term Memory networks (LSTM) [35, 36].

In section 2.3 we extend on the previous two algorithms and on the properties that differentiate them from other artificial neural networks.

2.2 Time series clustering

Clustering consists of finding similar groups in a dataset in order to gain insight and take advantage of it. Regarding this work, we presume that finding these groups would be beneficial to improve the accuracy of the forecasts since a group of related time series would be easier to fit than a very heterogeneous group.

However, commonly-used distances that tend to perform well with most general *raw-data-based* clustering methods, do not achieve the same level of efficiency when working with time series dataset. For this type of datasets, temporal sequence-based distances such as Dynamic Time Warping (DTW) have been proved to be more suitable [37]. This is why DTW is one of the de-facto algorithms for pattern matching applications such as speech recognition [38, 39].

Nevertheless, distance measures that make use of the raw time series data are not acceptable in the context of federated learning, since we would need to share the original data with the coordinating server, overlooking privacy as a consequence. Instead, it would be more appropriate if we could decompose the time series into different features while doing our utmost so these characteristics do not leak too much information about the original data. Then, we could safely share these features with a central orchestrator that would perform the clustering using feature-based similarity measures.

The previous feature-based approach for time series clustering has already been successfully applied in the literature [40, 41, 42]. In view of this fact and considering that this approach also takes user privacy into consideration, we will ignore any other approach (i.e., raw-data-based) in this review since they are not a central topic of our research.

However, quantifying the similarity of the observations is only one part of clustering analysis. The particular clustering algorithm still ought to be defined. In the following sections, we explain what different features can be extracted from time series data and which clustering algorithm can be used in combination to find the hidden patterns.

2.2.1 Feature extraction

Many researchers have used time series features as a data-mining tool. For example, Nanopoulos et al. used statistical features to classify time series by using a Multi-Layer Perceptron [43]. Fulcher and Jones also extracted multiple of interpretable features from time series to solve a similar problem [44]. Mörchen and Fabian used wavelet and Fourier decompositions as a method for dimensionality reduction in time series datasets.

The problem with the last method (i.e., DWT and DFT) is that they reveal too much information about the original data since the constituent frequencies obtained after applying these transformations can be used to reconstruct the original signal, disregarding privacy again as a consequence. Another problem with the previous methods is that some of them extract as many features as possible, which then they filter using feature selection methods. For example, Fulcher and Jones selected the most informative features over a total of 9000 using greedy forward feature selection with a linear classifier [44]. In our study, it is not possible to perform this last step as we also consider the case of non-labelled data. As a result, methods that extract a large number of features are not practical for our purpose. For this research, it would, therefore, be useful to extract a limited collection of carefully selected features.

Following the previous approach, we have identified several works that propose different sets of describable features that aim to capture the majority of the dynamics that can be observed in time series. In particular, Hyndman, Wang and Laptev compute a set of features that they claim to be

useful for unusual time series detection [45]. Kang, Hyndman and Smith-Miles propose a method for assessing the diversity of a time series dataset using a particular set of features [46]. Bandara et al. use a collection of features for time series clustering in the E-commerce domain [13]. We will often refer to the three previous sets of features as `hwl2015`, `khs2017` and `ban2019` respectively, using the initials of the authors and the date when their research was published. Below, we present a summary of the features extracted for each of the previously mentioned studies.

Hyndman, Wang and Laptev (2015) – `hwl2015`

Feature	Description
Mean	Mean
Var	Variance
ACF1-x	First order of autocorrelation
Trend	Strength of trend
Linearity	Strength of linearity
Curvature	Strength of curvature
Season	Strength of seasonality
Peak	Strength of peaks
Trough	Strength of trough
Entropy	Spectral entropy
Lumpiness	Changing variance in remainder
Spikiness	Strength of spikiness
Lshift	Level shift using rolling window
Vchange	Variance change
Fspots	Flat spots using disretization
Cpoints	The number of crossing points
KLscore	Kullback-Leibler score
Change.idx	Index of the maximum KL score

Table 2.2.1: Features proposed by Hyndman, Wang and Laptev [45]. Table adapted from [12]

Kang, Hyndman and Smith-Miles (2017) – `khs2017`

Feature	Description
Entropy	Spectral entropy
Trend	Strength of trend
Season	Strength of seasonality
ACF1-x	First order of autocorrelation
ACF1-e	First order of autocorrelation of the residuals
BoxCox.lambda	Box-Cox transformation parameter

Table 2.2.2: Features proposed by Kang, Hyndman and Smith-Miles [46]

Bandara et al. (2019) – `ban2010`

Feature	Description
Zero.obs.percentage	Observations sparsity/percentage of zeros
Trend	Strength of trend
Spikiness	Strength of spikiness
Linearity	Strength of linearity
Curvature	Strength of curvature
ACF1-e	First order of autocorrelation of the residuals
Entropy	Spectral entropy

Table 2.2.3: Features proposed by Bandara et al. [13]

2.2.2 Clustering algorithm

There are multiple clustering algorithms, and usually, the decision of choosing one over another is based on empirical evidence, unless we know enough about the shape of our data. For example, k-means cannot handle convex sets, so it is pointless to use this algorithm when dealing with this type of data. Moreover, this decision becomes harder when we are working with high-dimensional data (as in our case), even when using projections to understand the overall distributional structure.

Even though there is not a general set of rules to assist us in choosing between the different alternatives, sometimes there are mathematical reasons to prefer one clustering method over another. Besides, each algorithm has a different time and space complexity, which can also help to make an informed decision. Table 2.2.4 shows a comparison of different clustering algorithm along with suggestions to help in the decision of which algorithm to use.

It is out of the scope of this project to discuss the various methods for obtaining clustering. As we underlined in section 1.6, this work is not an extensive study on time series clustering. For this research and based on the advice of Table 2.2.4, we consider that an agglomerative clustering algorithm is suitable for our use case due to the following reasons: 1) The number of clients to cluster is in the order of thousands (small enough even for a $O(n^3)$ time complexity algorithm). 2) It produces a dendogram which can be very useful in understanding your data set to select k. 3) It is extremely flexible. It can be used with any similarity function and provides multiple linkage algorithms to use (more below).

Hierarchical clustering

Hierarchical clustering is a family of clustering algorithms that build nested clusters by joining smaller clusters (bottom-up or agglomerative) or by splitting bigger groups (top-down or divisive). We focus on the former since it is the most widely used method. Nonetheless, we must be aware that they are not equivalent, and they can generate different results [47].

In agglomerative clustering, each data point is considered as an individual cluster at the beginning. At each iteration, the two most similar or nearest clusters are merged into one cluster. This process repeats until all the data has been aggregated into one cluster. This algorithm is

formalised in Algorithm 1.

Besides, the whole operation can be represented using a dendrogram: a tree diagram that depicts the arrangement of the clusters produced by the algorithm. Figure 2.2.1 shows the dendrogram obtained after using the agglomerative clustering procedure with a dataset containing 10 major US cities. In this dataset, the similarity is represented by the flying mileages (very related to geographical distance).

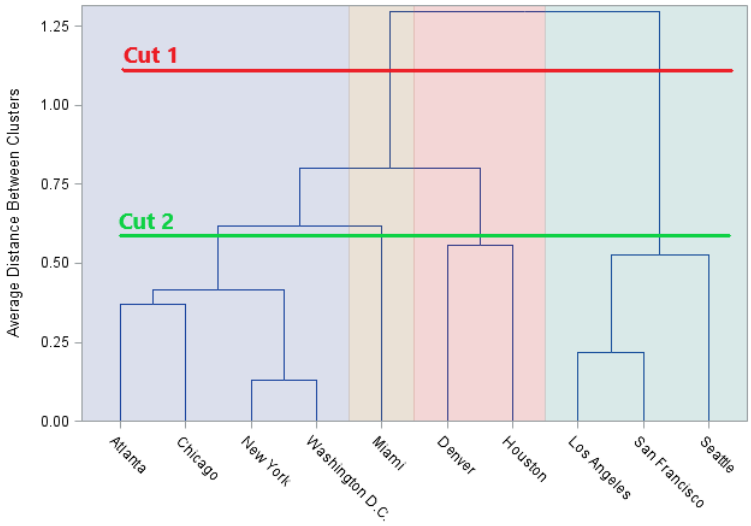


Figure 2.2.1: Dendrogram obtained from clustering 10 US cities. The proximity of two clusters is based on the flying mileages. Source: blogs.sas.com - How to color clusters in a dendrogram

We can also see how the longest vertical distance without any horizontal line passing occurs when merging the last two clusters. If we trace a horizontal line at this point (red line), the data would be split into two clusters: {Atlanta, Chicago, New York, Washington, Miami, Denver, Houston} and {Los Angeles, San Francisco, Seattle}. However, we can also get a suitable clustering if we cut further down (green line), dividing into the western, central, and eastern United States. The previous argumentation clearly demonstrates that dendrogram truncation (i.e., determining the number of clusters) is not trivial. This is the reason why we discourage automatic cut-off selection methods, such as the *inconsistency* method [48] (i.e., finding the longest vertical distance in the dendrogram).

Algorithm 1 Agglomerative clustering

Input: $X = \{x_1; ::: ; x_n\}$
Require: Distance function $dist(c_1; c_2)$
1: $C = \{x_1; ::: ; x_n\}$
2: **while** C:size > 1 **do**
3: $(c_{min1}; c_{min2}) =$ minimum $dist(c_1; c_2)$ for all $c_i; c_j \in C$
4: Merge c_{min1} and c_{min1} in c_{merged}
5: Remove c_{min1} and c_{min1} from C
6: Add c_{merged} to C
7: **end while**
8: **return** set of nested clusters

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters`	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters or distance threshold	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters or distance threshold, linkage type, distance	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes	Distances between nearest points
OPTICS	minimum cluster membership	Very large `n_samples`, large `n_clusters`	Non-flat geometry, uneven cluster sizes, variable cluster density	Distances between points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large `n_clusters` and `n_samples`	Large dataset, outlier removal, data reduction.	Euclidean distance between points

Table 2.2.4: Comparison of clustering algorithms. Source: scikit-learn documentation - Clustering

As we mentioned earlier, one of the benefits of using this algorithm is the ability to use different criteria for the merge strategy. Taking a look at Algorithm 1 (line 3) we can see how the algorithm computes the distance between clusters to merge the nearest ones. Determining the distance between two observation (i.e., clusters of size 1) is trivial. However, when dealing with clusters with many observations, there are multiple methods to measure the distance between two groups. This method is commonly known as *linkage* criteria, and it determines the results of the clustering. The following are linkage methods for computing the distance between clusters u and v.

Single linkage or **nearest neighbour**. The proximity of the two clusters is the distance between their two closest objects.

$$d(u;v) = min(dist(u[i];v[j])) \tag{2.6}$$

for all points i in cluster u and all j in cluster v. In this way, we pay attention to the area where the two clusters are closest, ignoring more distant parts. This tends to produce long and skinny clusters.

Complete linkage or **farthest neighbour**. The proximity of the two clusters is the distance between their two most distant objects.

$$d(u;v) = max(dist(u[i];v[j])) \tag{2.7}$$

This results in a preference for compact clusters. The drawback is that it is susceptible to outliers since a single observation far from the centre can significantly increase the distance between two clusters.

Both single-link and complete-link clustering reduce the comparison to two observations: one from cluster u and another one from cluster v. This cannot adequately reflect the distribution of all the elements in both clusters and, consequently, they often produce undesirable results. In general, there are more useful linkage methods that consider all the points of the candidate merge clusters.

Average linkage or **UPGMA (unweighted pair group method with arithmetic mean)**. The proximity of the two clusters is the average of all pairwise distances between the observations of clusters u on one side, and cluster v, on the other side.

$$d(u;v) = \sqrt{\frac{dist(v;v[j]}{|u| \times |v|}} \tag{2.8}$$

Note that the term *unweighted* indicates that all pairwise distances contribute equally to the result. There is also a *weighted* version denominated **WPGMA**. Both can produce clusters of different shapes.

Ward linkage or **UPGMA (minimise increase of sum of squares)**. Based on Ward's criterion [49]. Minimises the increase in variance that results when merging two clusters.

$$d(u; v) = SSE_{u \cup v} - SSE_u - SSE_v \tag{2.9}$$

Intuitively, this approach is similar to the variance-minimising in k-means but using an agglomerative approach.

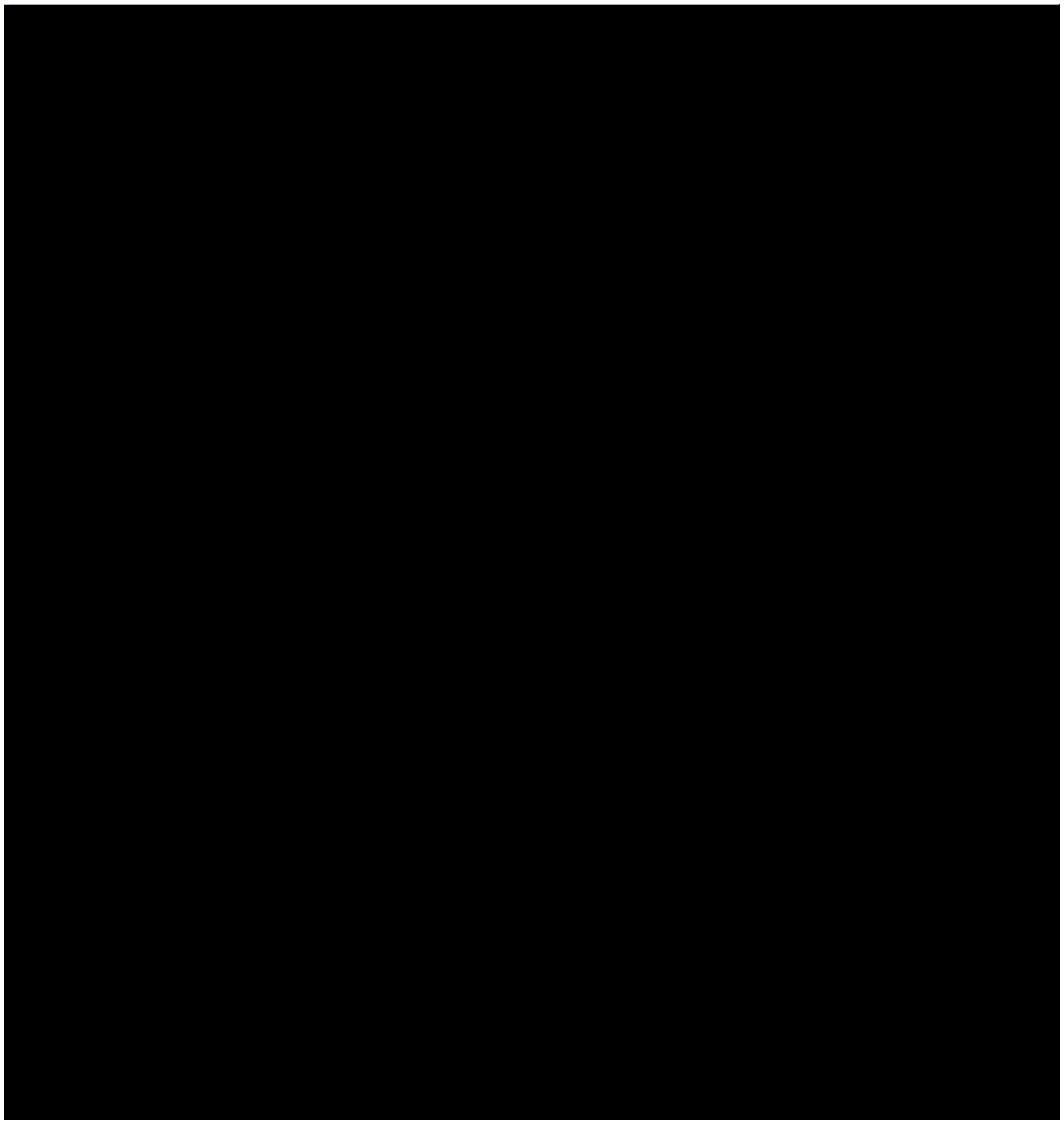


Figure 2.2.2: Linkage method behaviour for different types of datasets. Source: scikit-learn documentation - Hierarchical clustering

In a nutshell, the linkage method determines the distance between a set of observations. Each method has a different effect on the type of clusters produce and we must understand our data to

select the most suitable method.

Before closing this section, we must address one last and very important facet of clustering analysis: the distance measure. We can see in all the previous equations how the linkage method continuously applies a distance function $dist(x; y)$. Below, we examine three of the most common metrics to measure similarity.

Distance metrics

If we need to be careful about the linkage method, by the same token, we must also be cautious when choosing a distance metric, since it also affects the quality of the clustering. For example, the experimental results presented by Kumar et al. reveal that the performance and quality of different distance measures vary with the nature of data as well as clustering techniques [50]. In another study, Kapil and Chawla observe the same effect of the distance function upon k-means clustering. Thereby we cannot overlook this important fact.

Euclidean distance. Given two 1D arrays $x = [x_1; ::: ; x_n]$ and $y = [y_1; ::: ; y_n]$, the Euclidean distance between x and y is defined as

$$dist(x; y) = \sqrt{\sum_{i}^{n} (x_i - y_i)^2} \tag{2.10}$$

Cosine distance is defined as

$$dist(x; y) = 1 - \frac{x \cdot y}{\|x\|_2 \|y\|_2} \tag{2.11}$$

where $x \cdot y$ is the dot product of x and y. The benefit of using cosine similarity over Euclidean distance is that the former is not affected by the magnitude of the vector. Another incentive to use cosine similarity is that it returns a number between -1 (totally dissimilar) and 1 (same vector), which is easier to interpret than the unbounded scalar value returned by Euclidean distance.

Manhattan distance (also know as **City Block**) is defined as

$$dist(x; y) = \sum_{i}^{n} |x_i - y_i| \tag{2.12}$$

When vectors x and y are similar on most of the variables but very disparate in one (or more) of them, Euclidean distance tends to amplify this discrepancy (due to the exponent outside the parentheses), unlike Manhattan distance.

A visual demonstration of the effects of the previous effects on agglomerative clustering can be seen in the following scikit-learn code example.

In summary, in this section, we have examined two crucial components of clustering analysis: the measure to quantify metrics between observations, and the algorithm used to generate the clusters.

2.3 Artificial Neural Networks

Neural networks are a set of algorithms loosely modelled after the biological connections of the human brain. The most fundamental unit of a neural network is called an *artificial neuron*, a mathematical function that receives one or more inputs and produces a result equal to a weighted sum of these inputs. This result is passed through a non-linear function known as activation function to produce the final output. The first artificial neuron, called *perceptron*, was proposed by Frank Rosenblatt in 1958 [51] and later refined by Minsky and Papert in 1969 [52]. The model proposed by Minsky-Papert introduced the concept of numerical weights. However, the artificial neurons that we use today to build artificial neural networks are slightly different from this redefined perceptron. The difference is the non-linear activation function, which allows overcoming the limitations of the original model, which used a Heaviside step function (zero for negative arguments and one for positive arguments) and could only learn to approximate functions for linearly separable datasets.

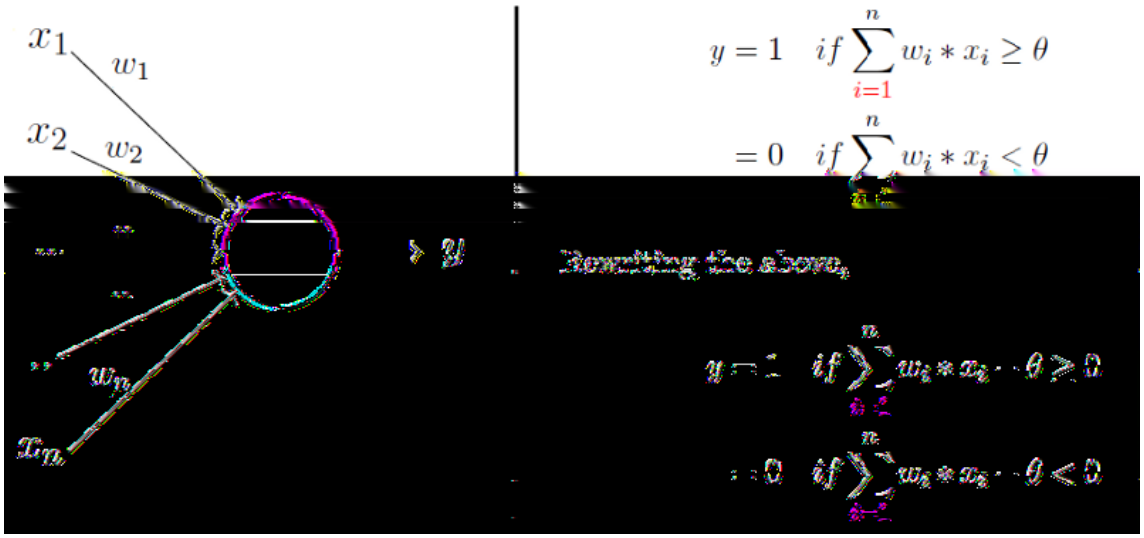


Figure 2.3.1: Minsky-Papert perceptron model. Source: Medium - Perceptron Learning Algorithm

Deep neural networks are composed of multiple layers are composed of multiple perceptrons stacked one after the other in a layerwise fashion. The goal of the network is to approximate a function f^*. For example, for regression, $y = f^*(\mathbf{x})$, maps an input $\mathbf{x}$ to a real value y. A feedforward neural network defines a mapping $\mathbf{y} = f(\mathbf{x};\ )$ and learns the value for the weights that reduce the error of the approximation.

These models are called feedforward because the information flows from input to output, but there are no feedback connections in which the outputs are fed as inputs again. Later, in

subsection 2.3.1 we will explore a variety of neural network with this kind of connections, called recurrent neural networks (RNNs).

The layers in a neural network can be seen as functions themselves, with multiple inputs and multiple outputs. The neural network is then is created by composing multiple functions. For example, we might compose three functions $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$ to form $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$. In the previous case, $f^{(1)}$ would be the input layer, $f^{(2)}$ the second layer (also termed hidden layer) and $f^{(3)}$ the output layer. The term "deep learning" derives from the length of the chain of functions. And usually, neural networks are not called "deep" unless they have at least one hidden layer.

During training, we drive $f(\mathbf{x})$ (the output of the real function) to match $f^*(\mathbf{x})$ (the desired output), this is accomplished by using the training data. Each training sample $(\mathbf{x}; \mathbf{y})$ indicates how the output layer of the neural network should behave in order to generate $\mathbf{y}$, given the input $\mathbf{x}$. However, the behaviour of the rest of the layers is not specified in the training data. Instead, the learning algorithm must decide how to use these layers to implement an approximation of f^*.

The conventional method to train a neural network is by using the stochastic gradient descent optimisation algorithm. In the context of this algorithm, as we previously mentioned, the goal is to reduce the error between f^* and its approximation f. The error is usually measured by a loss function like the following one:

$$L(\mathbf{y}; \hat{\mathbf{y}}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y_i})^2 \tag{2.13}$$

Where $\hat{y}$ is the output of the neural network, y is the desired output (obtained from the training data) and m is the number of sample in the training dataset. The previous loss function is known as Mean Square Error (MSE) and is the most commonly used regression loss function. This function tells us "how good" is our neural network at making predictions. The loss function has its own shape and gradients. This shape tells the optimisation algorithm how to update the weights of the neural network to reduce the error between the actual output and the desired value. This is achieved by computing the partial derivatives of the cost function with respect to each weight and adjusting them in a direction that reduces the loss [53].

2.3.1 Recurrent Neural Networks

Recurrent Neural Networks or RNNs are a variety of artificial neural networks well suited to process sequence data $\mathbf{x}^{(1)}; : : : ; \mathbf{x}^{(\tau)}$. The main difference compared to the model presented in the last section (MLP), is that RNNs share parameters across different parts of the model. If this were not the case, it would be impossible to generalise to sequence lengths not seen during training and to share statistical strength across different positions in time. For example, given the two following sentences: "I went to Gothenburg in 2018"and "In 2018 I went to Gothenburg". If we ask a machine learning model to read both sentences and extract the year in which I went to

Gothenburg, it should not matter whether the year appears in the six or second position in the sentence. An MLP would need to learn different weights for both sentences. On the contrary, an RNN can share the same weights across different time steps.

Similar to the mapping defined for MLPs, recurrent neural networks use the following mapping to define the output of their artificial neurons:

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}; \mathbf{x}^{(t)}; \ ) \tag{2.14}$$

where $\mathbf{h}^{(t-1)}$ represents the output of the previous hidden neuron. At each time step t, the recurrent network process the input $\mathbf{x}^{(t)}$ by incorporating the state from the previous time step $\mathbf{h}^{(t-1)}$. Previously, the state $\mathbf{h}^{(t-1)}$, was the result of incorporating the state $\mathbf{h}^{(t-2)}$ to the input $\mathbf{x}^{(t-1)}$. This unfolding equation is continuously applied, as depicted in Figure 2.3.2.

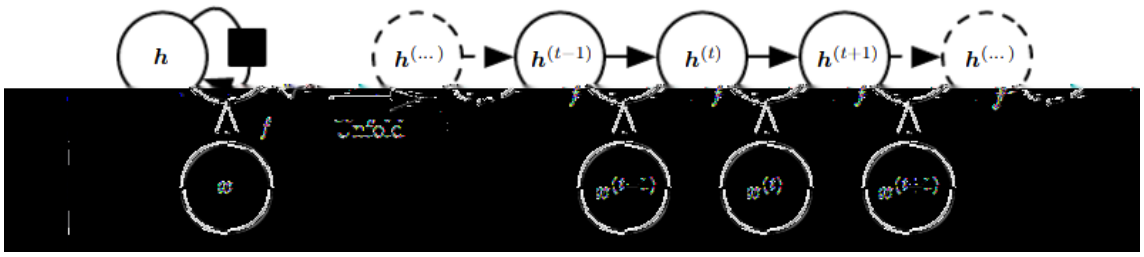


Figure 2.3.2: A Recurrent Neural Network with no outputs. The black square (left) indicates a delay of a single timestep. Source: Goodfellow, Bengio, and Courville [54], page 370

There are three different architectures for RNNs, depending on the problem to solve:

- RNNs that produce an output at each time step and have recurrent connections between hidden units.
- RNNs that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step.
- RNNs with recurrent connections between hidden units, that read an entire sequence and then produce a single output.

We will focus on the last since they are the ones used in this research. Their architecture is very similar to the recurrent design that we have already seen. The only difference is that we add a single output in the end, as illustrated in Figure 2.3.3. Such a network can be used to read part of a time series and predict a 1-step-head forecast. If instead, we want to predict an n-step-ahead forecast, we would to append the first forecast to the neural network input, ask the model again for the second forecast and so on. Another way would have been to use a multi-input-multi-output RNN.

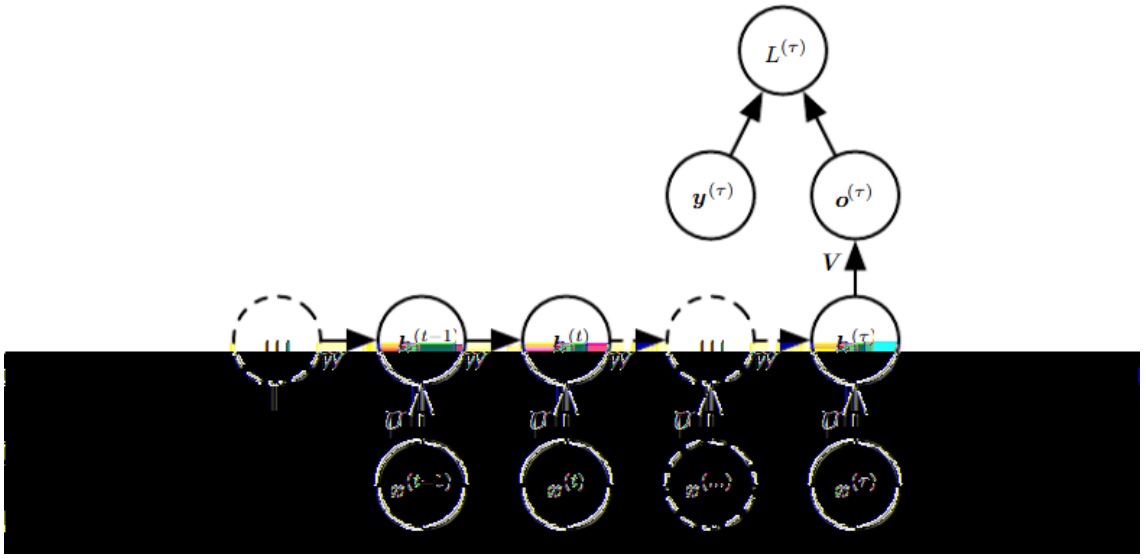


Figure 2.3.3: A Recurrent Neural Network with a single output at the end of the sequence. Source: Goodfellow, Bengio, and Courville [54], page 375

In this type of network, the forward propagation process starts initialising the initial state $\mathbf{h^0}$, and, for every time step from 1 to , the next states are computed using the following equation

$$\mathbf{h}^{(\mathbf{t})} = \tanh(\mathbf{b}_h + \mathbf{W}\mathbf{h}^{(\mathbf{t-1})} + \mathbf{U}\mathbf{x}^{(\mathbf{t})}) \tag{2.15}$$

where the parameters are the bias vector $\mathbf{b}_h$ along with the weight matrices $\mathbf{W}$ and $\mathbf{U}$, respectively, for hidden-to-hidden and input-to-hidden connections. Finally, once the last state $\mathbf{h}^{(\mathbf{t})}$ has been computed, the final output can be calculated as:

$$\mathbf{o}^{(\tau)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{\tau} \tag{2.16}$$

$$\hat{y^{(\tau)}} = \mathrm{softmax}(\mathbf{o}^{(\tau)}) \tag{2.17}$$

where the parameters are the bias vector $\mathbf{c}$ along with the weight matrix $\mathbf{V}$ for hidden-to-output connections. The final output $\mathbf{o}^{(\tau)}$ can be used with a target $\mathbf{y}^{(\tau)}$ and a loss function $\mathbf{L}^{(\tau)}$ to compute the gradients. In this way, the gradients at other time step t can be calculated by back-backpropagating from further downstream modules. This algorithm of back-propagation applied to the unfolded graph is called backpropagation-through time.

In summary, we have seen how RNNs have recurrent loops that let them maintain information about past observations. However, for problems that require to capture long-term temporal dependencies, this model is not useful. The reason is that the gradient of the loss function decays exponentially with time, causing long-term dependencies to be forgotten. Fortunately, there is a select type of RNN design that can solve the vanishing gradient problem: Long Short-Term

Memory architectures.

2.3.2 Long Short-Term Memory

In recent years, LSTM has become the de-facto standard for learning from sequence data. Since its inception in 1997 [55], several architectures have been proposed. E.g., Dilated LSTMs by Chang et al (2019) [56]. However, a study by Greff et al. demonstrated that most of the variants could not improve the original LSTM architecture significantly and proved the forget gate and the output activation function to be its most critical components [57].

LSTM preserve information and propagate errors for a much longer chain in the network when compared to standard RNN models, overcoming the problem of vanishing gradients. One of the components that enables LSTM to retain long-term dependencies is the design of its fundamental block, shown in Figure 2.3.4.

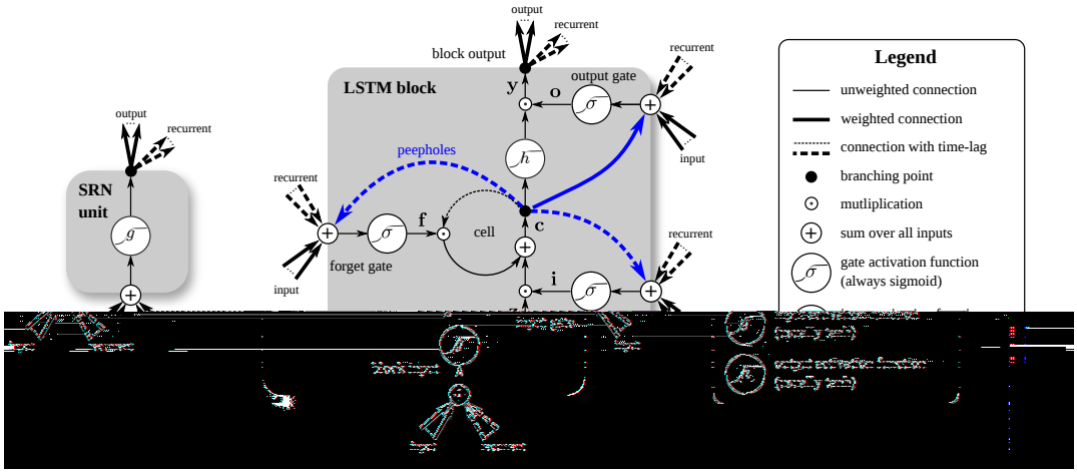


Figure 2.3.4: Detailed schematic of a simple RNN unit (left) and a LSTM block (right). Some studies ignore peephole connections (blue links). Source: Greff et al. [57]

The formulas that control the feedforward process in an LSTM are:

$$\bar{\mathbf{z}}^t = \mathbf{W}_z \mathbf{x}^t + \mathbf{R}_z \mathbf{y}^{t-1} + \mathbf{b}_z \tag{2.18}$$

$$\mathbf{z}^t = g\left(\bar{\mathbf{z}}^t\right) \tag{2.19}$$

$$\bar{\mathbf{i}}^t = \mathbf{W}_i \mathbf{x}^t + \mathbf{R}_i \mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i \tag{2.20}$$

$$\mathbf{i}^t = \quad \left(\bar{\mathbf{i}}^t\right) \tag{2.21}$$

$$\bar{\mathbf{f}}^t = \mathbf{W}_f \mathbf{x}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f \tag{2.22}$$

$$\mathbf{f}^t = \quad \left(\bar{\mathbf{f}}^t\right) \tag{2.23}$$

$$\mathbf{c}^t = \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t \tag{2.24}$$

$$\bar{\mathbf{o}}^t = \mathbf{W}_o \mathbf{x}^t + \mathbf{R}_o \mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o \tag{2.25}$$

$$\mathbf{o}^t = \quad \left(\bar{\mathbf{o}}^t\right) \tag{2.26}$$

$$\mathbf{y}^t = h\left(\mathbf{c}^t\right) \odot \mathbf{o}^t \tag{2.27}$$

Here, $\mathbf{i}$, $\mathbf{f}$ and $\mathbf{o}$ are called input, forget and output gates, respectively. Observe that these equations are very similar, and the only difference is the weight matrices used ($\mathbf{W}$ and $\mathbf{R}$). They are called gates because the sigmoid function squashes the value of these vectors to the [0, 1] range, and by multiplying these values with another vector, the LSTM controls how much of the vectors should be let pass through. For example, the output gate defines how much of the internal state is exposed to the output of the block.

It is also important to notice that $\mathbf{c}^{(t)}$ is a hidden state that is calculated using the previous hidden state $\mathbf{c}^{(t-1)}$ and the current input. $\mathbf{c}^{(t)}$. Notice how the forget gate defines how much the previous hidden state affects the current one. Intuitively, $\mathbf{c}^{(t)}$ is the memory of the LSTM block and is a combination of the previous memory and the current input. In this way, the LSTM could decide to ignore the input (input gate all 0's) or, on the contrary, forget everything (forget gate all 0's) and consider only the new input. But most likely, something in between will be more beneficial.

Intuitively, plain RNNs could be considered a special case of LSTMs. If fix the input gate all 1s, the forget gate to all 0s (say, always forget the previous memory) and the output gate to all 1s (say, expose the whole memory), it will almost get a standard RNN.

2.4 Distributed deep learning

Deep neural networks trained on large datasets have achieved convincing results and are currently the state-of-the-art approach for many different tasks: object detection [58, 59], language models [60], continuous control [61], time series classification [62], and so forth. However, training large neural networks is computationally demanding, which limits their deployability on resource-constrained devices. Fortunately, in order to tackle this challenge and allow researchers and practitioners to learn bigger models, multiple frameworks have introduced innovations to enable distributed machine learning. Perhaps the most known of these is Tensorflow, which

already in a primitive version (DistBelief) supported a variety of distributed algorithms for large-scale training [63].

The two most common methods that these frameworks use to achieve high scalability are data parallelism and model parallelism.

Conceptually, the first paradigm is straightforward:

1. Run multiple copies of the model, and for each one:
   (a) Read a partition of the data.
   (b) Run the data through the model.
   (c) Compute model updates (gradients)
2. Reduce all the updates in a single update (aggregate).
3. Update the model using the previous update.
4. Repeat from 1a).

It is not very difficult to realise that the previous paradigm is very similar to the MapReduce programming paradigm [64]. Using MapReduce, one can effortlessly schedule parallel tasks onto multiple processors, as well as distributed environments. The previous design has been applied to parallelise the Stochastic Gradient Descent algorithm to speed it up further. There are multiple implementations of Parallel Stochastic Gradient Descent [65, 66, 67], but all of them share the same underlying procedure. The model's replicas communicate the updates through a centralised parameter server, which keeps the current state of the global model. After receiving an updated copy of the parameters, the central server can either 1) apply the update immediately to the global model (asynchronous); or 2) wait to receive other replicas' updates, to later aggregate them and update the model. As presented in subsection 2.4.1, this approach is very similar to the one used in federated learning.

The second approach to parallelise deep networks training is model partitioning. This strategy splits the network across a number m of machines. Unlike in the previous approach, it is necessary to copy the same minibatch to all participant, since each one is responsible for storing and applying updates to $1/m$ th of the model parameters. In this way, updates to different parts of the network are computed separately, and all the updates are communicated to a central aggregator that has a copy to the entire model. Nevertheless, neural network architectures create layer interdependencies, which, in turn, generate communication that determines the overall performance (bold lines in Figure 2.4.1 - right).

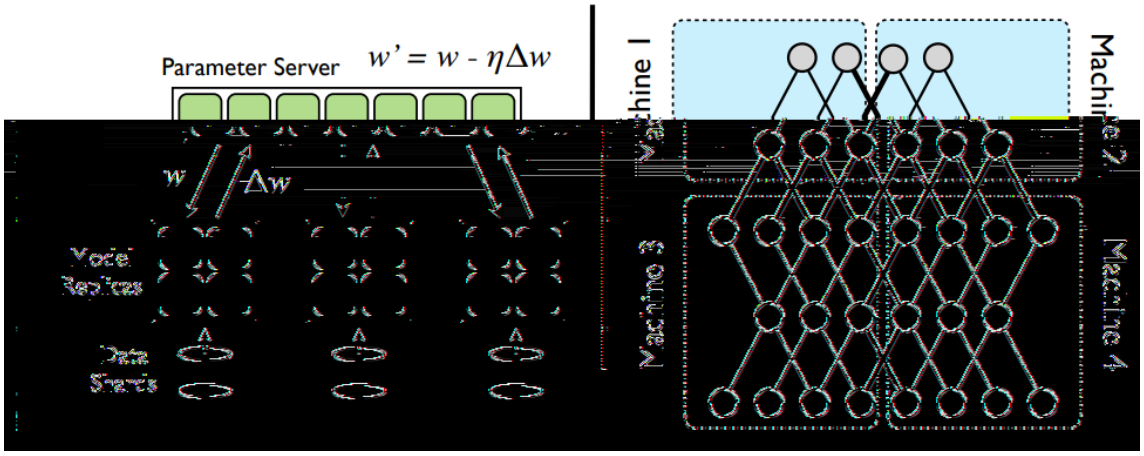


Figure 2.4.1: Data parallelism (left) vs model parallelism (right). Source: Dean et al. [63]

We have seen how distributed deep learning is not free. Distributed the workload over a network of machines has a significant overhead. In the data parallelism approach, we must be aware of the communications costs resulting from sending the updates back and forth from the replicas to the server. When using model parallelism, some links will cross partition boundaries, and it will be necessary to transmit state across machines. So, for distributed training to be worthwhile, we need the computational benefit to outweigh these overheads.

2.4.1 Federated Learning

Federated learning allows learning from decentralised data, without the need of centrally store it [8]. The data remains where it was generated, which guarantees privacy and reduces communication costs.

In federated learning, a federation of participants (also called *clients*) cooperates to solve the learning task while a central coordinator (also called *server*) oversees the process. Each client has a local dataset which never shares with the central server, instead, the client calculates an update to the current shared model maintained by the server, and only this update is communicated. This approach is very similar to the data parallelism paradigm explained in the previous section, the only peculiarity is that, in this case, the data is inherently parallel and there is no need to distribute the data over the clients.

This approach is especially well-suited when training on real-world data from mobile devices and when the data is privacy sensitive. For example, Hard et al. applied this method for the purpose of next-word prediction in a virtual keyboard for smartphones [68]. Moreover, the federated optimisation algorithm has different properties which differentiate it from the standard distributed optimisation algorithm:

- **Non-IID**: the dataset in the device (client) might not be representative of the population.
- **Unbalanced**: varying amount of training samples between devices.
- **Massively distributed**: number of participants $\gg$ number of samples per participant.

- **Limited communication**: expensive connections. Communication might not be possible.

Let us be more specific about the details of the federated optimisation algorithm since it is of crucial importance to understand this research.

The FederatedAveraging algorithm

This algorithm relies on a parallelised variant of the SGD algorithm, similar to the one proposed by Chen et al. [69]. This approach starts by selecting a fraction of C clients. These clients, as mentioned earlier, compute a gradient update on their respective local datasets. Finally, all the updates (hence, the algorithm is synchronous) are aggregated by computing a weighted average. Each update is weighted according to the number of samples in each client's local dataset. So for example, if out of 10 clients, only 2 are selected (i.e., $C = 0{:}2$), and c_1 has 100 examples in its local dataset and c_2 has 200; when the central server aggregates the updates from these two clients, it will give more importance to the update of the last client ($\times 2$). The algorithm is formally represented in Algorithm 2.

Algorithm 2 FederatedAveraging. Clients are indexed by k, B is the local minibatch size, E is the number of local epochs and is the learning rate

Server executes:
  initialise weights w_0
  for each round $t = 1; : : : ; r$ **do**
    $m \leftarrow max(C \cdot K; 1)$
    $S_t \leftarrow$ (sample of m of clients)
    for each client $k \in S_t$ in parallel **do**
      $w_{t+1}^k \leftarrow$ ClientUpdate$(k; w_t)$
    end for
    $w_{t+1} \leftarrow \sum_{k=1}^{k} \frac{n_k}{n} w_{t+1}^k$
  end for

ClientUpdate($k; w_t$**):**
  $\mathcal{B} \leftarrow$ split $\mathcal{P}_k$ into batches of size B
  for each local epoch $i = 1; : : : ; E$ **do**
    for batch $b \in \mathcal{B}$ **do**
      $w \leftarrow w - \nabla`(w; b)$
    end for
  end for

Observe how, when aggregating the updates, the weights of the weighted arithmetic mean are directly proportional to the number of samples on each client k ($\frac{n_k}{n}$ where n is the count of all the examples in the federated training set). Also, notice the three parameters that drive the algorithm: C, B and E (if we exclude the learning rate). These parameters play a similar role as the hyperparameters of a neural network, in the sense that they are not learned from the training set, but instead they are set by the practitioner depending on the problem. As shown by McMahan et al. [8], these parameters have a high impact on the convergence of the learning algorithm. For example, increasing parallelism (i.e., increasing C) usually reduces the number of communication rounds needed to converge. However, increasing C also results in higher communication costs, so it becomes necessary to find a good balance between computational efficiency and convergence

rate.

In conclusion, even being a relatively new learning technique, federated learning has attracted the attention of the research community and holds itself out as the best setting when training on decentralised data. Since its inception, multiple researchers have introduced improvements to the protocol, for example, by improving the security of the model aggregation [70], or by enhancing the communication efficiency [71]. So, if edge computing can reach peak productivity in the near future, it is undeniable that federated learning will play a key role.

Chapter 3

Methodology

The first question to address is which is the most appropriate research method for this work. According to Anne Håkansson, "The most common research methods are: Experimental or Quantitative, and Non-experimental" [72]. The non-experimental research method questions existing scenarios but it does not test relationships between variables. On the contrary, the experimental method aims to find these relationships and also the relation of cause and effect between relationships. In other words, this method tends to modify one variable while keeping the rest constant, to observe how this change influences the result.

Since this work aims to study the performance of a system, i.e., the accuracy of federated learning "with clustering" when groups of clients have similar data distributions, the choice of the empirical method is more than justified. Furthermore, the following example clearly illustrates why the methodology mentioned earlier well suits this thesis: if we would like to know how a particular feature-based representation affects the final performance, it would be necessary to fix the rest of variables (e.g., similarity metric, the model being trained, hyperparameters of the optimisation process) before coming to a conclusion. Otherwise, if we alter more than one variable, it would be challenging to know which one influenced the result. The previous reasoning is systematically applied throughout all the experiments.

Moreover, the choice of this particular research method is highly motivated by the recent literature for instance, McMahan et al., the authors that introduced the concept of federated learning, conducted an extensive experimental study of how different parameters in the federated averaging algorithm affected the convergence speed of the algorithm [8]. In the same way, Wang et al., proposed, in a later study, a modification to the federated averaging algorithm to improve the performance under constrained resource constrained edge computing systems, backing up their discoveries with via extensive experiments with real datasets [73]. Then, it is reasonable for us to follow the same path as these researchers, and back any claim with experimental results. This suggests again that the most appropriate research method is the experimental approach, in which we manipulate one or more variables (cause), and controls and measures the change in other variables (effect). The ultimate goal is to identify this cause-effect relationship by observing the obtained empirical results.

The rest of this section is devoted to describing the structure of the roadmap of the research. We use the framework proposed by Holtz et al. [74] to organise the cycle of our investigation. To be precise, the authors encourage to answer the following questions, which define an iterative process that allows us to refine our findings:

a) What do we want to achieve?

b) Where does the data come from? How and what to collect.

c) What do we do with the data? (e.g., identify patterns or trends)

d) Have we achieved our goal? Draw conclusion and evaluate results.

Both section 1.3 - Purpose and section 1.4 - Objectives already answered question a). This section focuses on questions b) and c), which describe the experimental setup and the validation test. Question d) is addressed later in chapter 6 - Summary and discussion, where the conclusion is drawn to answer the research question. Moreover, questions b) and c) clearly separate data collection from data analysis. We also keep this distinction and address each question separately in the two following sections. However, we defer the explanation about how the results are obtained until chapter 4 - Federated clustering, where we describe the artefact used to perform the experiments and which generates the data that is evaluated in the end.

3.1 Data collection

Given the quantitative nature of this study, we use the *experiments* data collection method to collect data for our research. We take advantage of the public datasets of time series available, in particular, we centre our attention in popular forecasting competitions because it allows us to compare and reproduce the experimental results of other researchers. Additionally, the University of California Irvine Machine Learning Repository has numerous time series datasets which are tagged by task [75]. We examine all the univariate time series datasets labelled either with the `clustering` or `classification` task, since knowing the true clusters can be very beneficial to understand the efficacy of the algorithm, as we show in chapter 5 - Results. Given the nature of the data (i.e., time series observations), it is clear that all the data collected at this step is quantitative.

As for the data generated by the experiments, we store all the data that is necessary to validate our assumptions, namely, the performance of each node in the network across time (i.e., local performance) and the distances or similarities between these participants, as it was introduced in section 1.2 - Problem. Again, all this data is quantitative, since the performance that we aim to measure (e.g., error in the predictions) in inherently numeric.

The subsequent analysis of this data is necessary to evaluate the proposed model and to answer the research question. Nevertheless, it is of vital importance to understand what kind of information is desired or valid for this research, since otherwise we would miss relevant data. As we mentioned in section 1.3 - Purpose, the model that we present would benefit from databases of heterogeneous time series where a notion of similarity between groups exists, in other words,

it would turn to good account homogeneous time series groups within a heterogeneous dataset. The following section is devoted to determining the validity of the data collected and how the results are evaluated via data analysis.

3.2 Data analysis

A random data collection would miss relevant data for this study. Not every available time series dataset is valid, as it has to comply with our original hypothesis of similar time series within the dataset. For this reason, it is necessary to assure the validity of the data collected. Besides that, it is also essential to define the approach to evaluate the results from the experiments. Again, we refer to the work of Anne Håkansson to support our choice of the data analysis method.

3.2.1 Verifying the validity of the collected data

We inspect the data to find if the time series dataset is composed of different subgroups. Sometimes, this is possible to achieve without the need for statistical models by means of visual methods, for example, by using a line plot representing the relationship between time and observed values in a time series, as we can see in Figure 3.2.1. However, considering that one of the goals of the suggested approach is to find these groups automatically, it would be convenient to use a "computational mathematics" method (i.e., a numerical method), to discover these groups without human intervention.

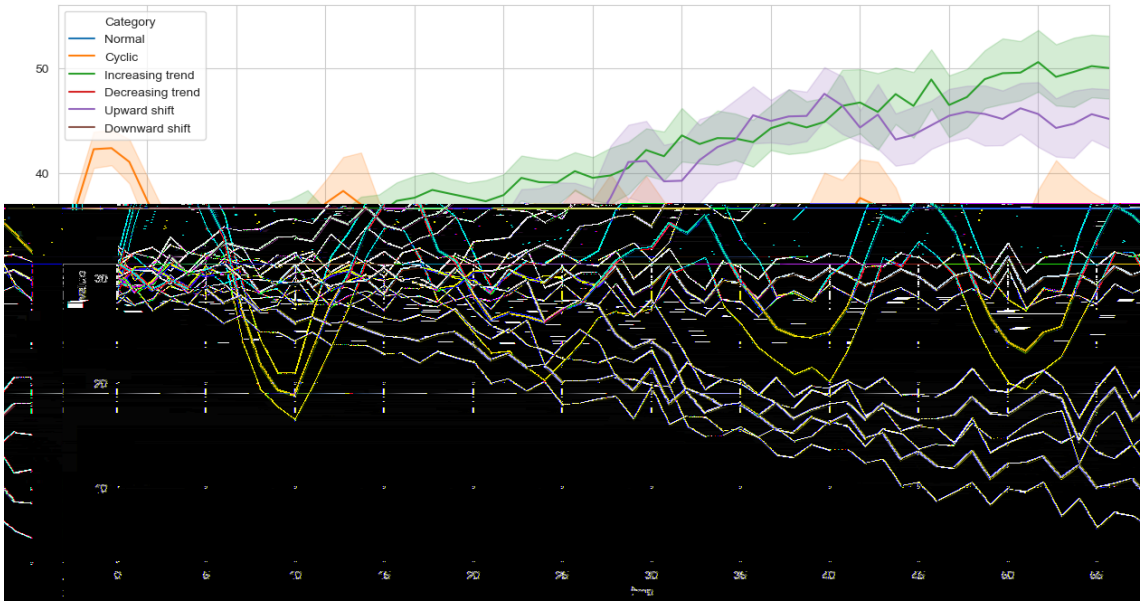


Figure 3.2.1: Synthetic Control Chart Time Series Data Set. Each class is depicted by a different colour. The highlighted line represents an estimation of the central tendency (mean) of all the series in the same category. The faded area represents a confidence interval for that estimate.

Given that our proposed model (which will be addressed later in chapter 4) makes use of a

clustering algorithm to identify the different subgroups, it is advisable and coherent to use the same algorithm. The reason is that if we cannot recognise any group during the initial validation of the data, neither the proposed model will when we feed the same data to it.

Using the approach mentioned above, we found two datasets which are worth considering. In particular, the NN5-Competition dataset [26], and the Synthetic Control Chart Time Series dataset [76].

3.2.2 Evaluating the efficiency of the proposed model

To evaluate the performance of our approach, we use the "statistical" method. We noted that most of the state-of-the-art studies compare the performance of two different models just by taking account of the difference in performance of the two models, for example, by computing the percentage decrease in the error. For instance, Makridakis, along with other authors, performed two different studies of the results of the M3 competition. The first of these analysed the results of the submissions to the competition [24]. A later study extended the previous one, focusing the discussion on why the statistical models outperformed the machine learning models [77]. Both of these studies compare the accuracy of different families of machine learning models by ranking the methods according to their accuracy. We use the same approach in this work. Still, we need to discuss what we refer to as "performance", and what metrics are we going to use to evaluate the performance.

In this work we are concerned about the local performance in a federated learning network, this is, how well the model (i.e., a neural network with a selection of weights) that the proposed algorithm generates performs on the local dataset of each of the nodes in the network. Let us formalise this by using the already introduced notion of federated optimisation. Given a federated network of m distributed nodes, where each node i stores a local dataset $\boldsymbol{X}_i$, we can represent the entire federated dataset as $\boldsymbol{X}_1 \cup \boldsymbol{X}_2 \cup ::: \cup \boldsymbol{X}_m$. Furthermore, given a neural network j with a selection of weight values $\boldsymbol{w}_j$, we define a performance function $P(\boldsymbol{w}_j; \boldsymbol{X})$ as the performance (the particular metric will be defined later) of the neural network with parameters $\boldsymbol{w}_j$ on dataset $\boldsymbol{X}$. In practice, it is convenient to apply the performance function on a subset of the dataset reserved for evaluation (i.e., the *validation dataset*). This defines a train/evaluation split that helps us to prevent both underfitting and overfitting. Finally, we define a one to one mapping $k : i \longmapsto \boldsymbol{w}_j$, which given a node i, returns the neural network parameters that node i contributed to (remember that, when using the clustering approach, different groups of nodes contribute to the creation of different models).

To get a single metric representing the local performance overall (`LP`), we apply the following formula.

$$LP = \frac{1}{m} \sum_{i=1}^{m} P(\boldsymbol{w_j}; \boldsymbol{X}_i) \quad \text{where} \quad \boldsymbol{w_j} = k(i) \tag{3.1}$$

This formula computes an average over all the clients to obtain a mean performance. In a similar

way, it is also possible to compute a median over all the clients by ranking the clients according to their performance first. This last metric allows us to get a better understanding of the performance of the network since outliers may dominate the former evaluation.

Lastly, it remains to answer which metric we are going to use to evaluate the performance. We use the Symmetric Mean Absolute Percentage Error (SMAPE), one of the most used metrics to evaluate forecasting models [78]. It is calculated according to:

$$\mathrm{SMAPE} = \frac{200\%}{n} \sum_{t=1}^{n} \frac{|F_t - Y_t|}{|F_t| + |Y_t|} \tag{3.2}$$

where Y_t is the actual value, F_t is the respective forecast and n represents the number of observations in the test set.

The problem with SMAPE is that, despite its name, this error is not symmetric and over- and under-forecasts are not treated equally. It also lacks interpretability and is unstable with values close to zero. To address some of these shortcomings, we use a second evaluation metric proposed by Koehler & Hyndman, the Mean Absolute Scaled Error (MASE) [79]. For non-seasonal time series is defined as:

$$\mathrm{MASE} = \frac{1}{h} \frac{\sum_{t=1}^{h} |F_t - Y_t|}{\frac{1}{n-1} \sum_{t=2}^{n} |Y_t - Y_{t-1}|} \tag{3.3}$$

Before reasoning about the previous equation, it is important to define the difference between *in-sample* and *out-of-sample*. Suppose that in our sample, we have a sequence of 10 data points. This data can be divided into two parts - e.g. first 7 data points for estimating the model parameters and the next 3 data points to test the model performance. Using the fitted model, the predictions made for the first 7 data points will be called *in-sample* forecast, while the prediction for the last 3 data points will be called *out-of-sample* forecast. It is similar to splitting the dataset in train and validation.

Regarding Equation 3.3, it is important to notice that the numerator is the absolute error produced by the actual forecast, clearly, the *out-of-sample* (h points) forecast, while the denominator is the mean absolute error produced by a naive forecast (i.e., the previous value is used as the next forecast), this is, the *in-sample* forecast (m points). The denominator is calculated using the *in-sample* instead of the *out-of-sample* data because we only know how well a naive forecast performs in the input. The output is unknown until we apply the model.

As a result, MASE represents a ratio between the actual forecast and a naive forecast. When $\mathrm{MASE} < 1$, the proposed forecasting model gives, on average, smaller errors than the naive one-step method. On the contrary, when $\mathrm{MASE} > 1$, the proposed forecasting model does worse than the naive forecast and suggest that the actual forecast should be discarded in favour of the naive one. Both assumptions expect the *out-of-sample* data to be like the *in-sample*.

Analogously, for seasonal time series, is possible to compute the MASE with a different lag in the

naive forecast. Now, the last period's observation are used as this period's forecast:

$$\mathrm{MASE} = \frac{1}{h} \frac{\sum_{t=1}^{h} |F_t - Y_t|}{\frac{1}{n-M} \sum_{t=M+1}^{n} |Y_t - Y_{t-1}|} \tag{3.4}$$

Where M is the seasonal period of the time series. It is easy to see that Equation 3.3 is a particular case of Equation 3.4 where $M = 1$.

To conclude, in this section we have defined how are we going to evaluate our model. We began by defining the concept of local performance, and then we established which metric are we going to use to measure that performance. In the next section, we describe the preprocessing techniques used in our forecasting framework.

3.3 Data preprocessing

Time series datasets usually include various difficulties of real-world forecasting tasks, such as missing values, handling seasonality and trend, and multi-step ahead forecast. In this section, we address these challenges to sanitise our data before we feed it into our data model.

3.3.1 Missing values

To impute the missing values, we use linear interpolation, although it is true that assigning missing points as predictions of more complex models usually shows better results (e.g., fuzzy c-mean and autoregressive models [80]). The reason to use this simple approach is that, for our problem and dataset, performance is good using, as there are no many series with consecutive missing values. We decided to drop series with consecutive missing values since in those cases the number of missing values was higher than 50% of the number of total observations, as represented in Figure 3.3.1.

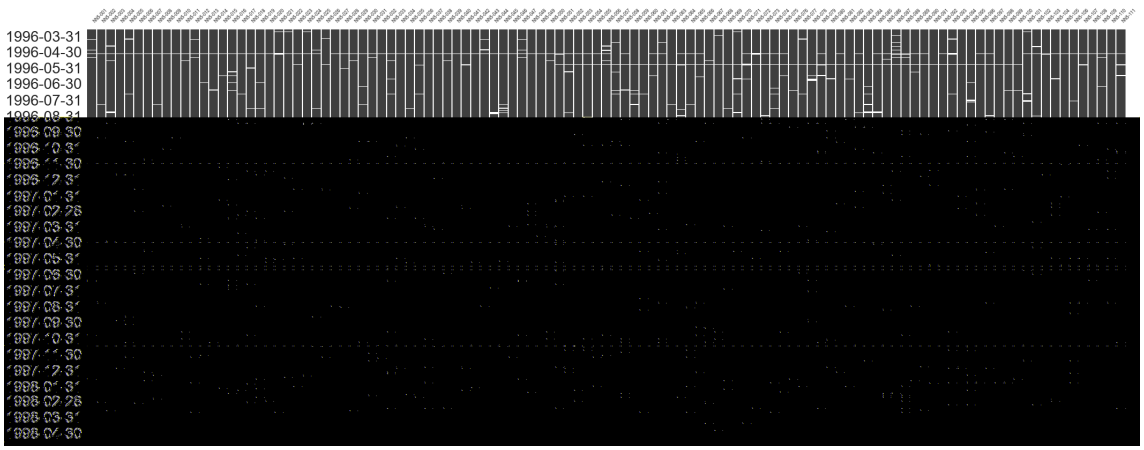


Figure 3.3.1: Visualisation of the number of missing values in the NN5 dataset. The horizontal axis represents individual time series; the vertical axis represents observations within the time-series. White spaces denote missing values for a particular observation. We can observe how most series have missing values around 1997-06

3.3.2 From time series to supervised

Federated learning enables us to train a neural network for time series prediction. Nevertheless, before using this machine learning approach, we must frame the time series forecasting problem as a supervised learning problem. A neural network computes a function that maps inputs to output; hence, it is mandatory to transform time series data from a sequence, to pairs of input and output sequences.

Recurrent Neural Networks and its variants have been proven to perform well on temporal data. However, different problems require different input-output architectures (i.e., many-to-many, many-to-one, one-to-many, as well as one-to-one). For the problems at hand, since all of them require multi-step-ahead forecasting, we decided to choose a many-to-many approach, even though it is also possible to tackle this problem with a *recursive* strategy [81]. We modelled the data as a *Multi-Input Multi-Output* (MIMO) function, since this model has the benefit of keeping the stochastic dependencies between predicted values [82, 83].

Given a time series $[y_1; y_2; \dots; y_N]$ with length N, an input sequence with length L, and the forecasting horizon H (i.e., the number of points to predict, the output sequence), the MIMO strategy learns a multiple-output model F as:

$$[\hat{y}_{t+L+1}; \dots; \hat{y}_{t+L+H}] = F(y_{t+1}; \dots; y_{t+L}) \tag{3.5}$$

with $t \in \{0; \dots; N-L-H\}$. Notice that N and H are defined by the problem (the length of the time series and the required length of the forecasting respectively), while L is a design choice.

Let illustrate the previous logic with an example. Let us suppose a time series $[1; 2; 3; 4; 5; 6]$, an input sequence with length 2 and a forecasting horizon of 3. The input-output $(x; y)$ pairs that we can extract from the previous series are $\{([1; 2]; [3; 4; 5]); ([2; 3]; [4; 5; 6])\}$.

In summary, a time series of length N is converted to $(x; y)$ patches where $|x| = L$ and $|y| = H$ using a sliding window approach (stride = 1). In total, there are $N - L - H$ of such patches. We use the last one of these for validation, and the rest to train the neural network.

3.3.3 Modelling trend and seasonality

Deep neural networks have been shown to be universal approximators [84], which means that they are suitable to model the trend and seasonality in time series data. However, some studies show that normalising the time series by removing seasonal and cyclic effects it is necessary to produce accurate forecasts for instance, Nelson et al. compared the accuracy of neural networks model trained with deseasonalized data and non-deseasonalized data, using various series from the M-competition [85]. Their results indicate that performance can be improved by removing seasonality and trend from the raw time series data. In a similar way, Ben Taieb et al. determine that deseasonalization leads to uniformly improved forecast accuracy [86].

For this work, we use seasonal and trend decomposition implemented by the Python `rstl` library [87]. This library allows us to decompose the original time series in `seasonality`, `trend` and `remainder` (Seasonal Decomposition of Time Series by Loess, abb. STL). However, since the previous library implements an STL decomposition algorithm using an additive method [88], we must make sure that the seasonality of the data is additive. This is possible to achieve via a power transformation, which enforces a constraint on the balance of the seasonality (i.e., additive or multiplicative) [89]. A popular case is the Box-Cox transform [90]. However, we decided on a more straightforward version proposed by Bandara et al. which already enforces additive seasonality and avoid problems for zero values [12]:

$$w_t = \begin{cases} \log(y_t) & \text{if } \min(y) > \epsilon \\ \log(y_t + 1) & \text{if } \min(y) \leq \epsilon \end{cases} \tag{3.6}$$

where y_t is the original time series and ϵ is a small number to prevent computing the logarithm of 0 (e.g., 10^{-3}).

After applying this log transformation, the time series is decomposed in seasonality, trend and remainder. We apply the window approach explained in subsection 3.3.2 - From time series to supervised to the sum of the trend and the remainder. Again, motivated by the work of Bandara et al., the trend in each input window is used for local normalisation. This value of the trend is subtracted from each input and output window. Continuing with the previous example, if we obtained $\{([1; 2]; [3; 4; 5]); ([2; 3]; [4; 5; 6])\}$ after transforming our time series to a supervised learning dataset, and we know that the trend of the original time series was $[0{:}1; 0{:}2; 0{:}3; 0{:}4; 0{:}5; 0{:}6]$, local normalisation would yield the following dataset $\{([0{:}8; 1{:}8]; [2{:}8; 3{:}8; 4{:}8]); ([1{:}7; 2{:}7]; [3{:}7; 4{:}7; 5{:}7])\}$. According to the authors, this procedure allows the RNN to model even fast-growing, exponential trends.

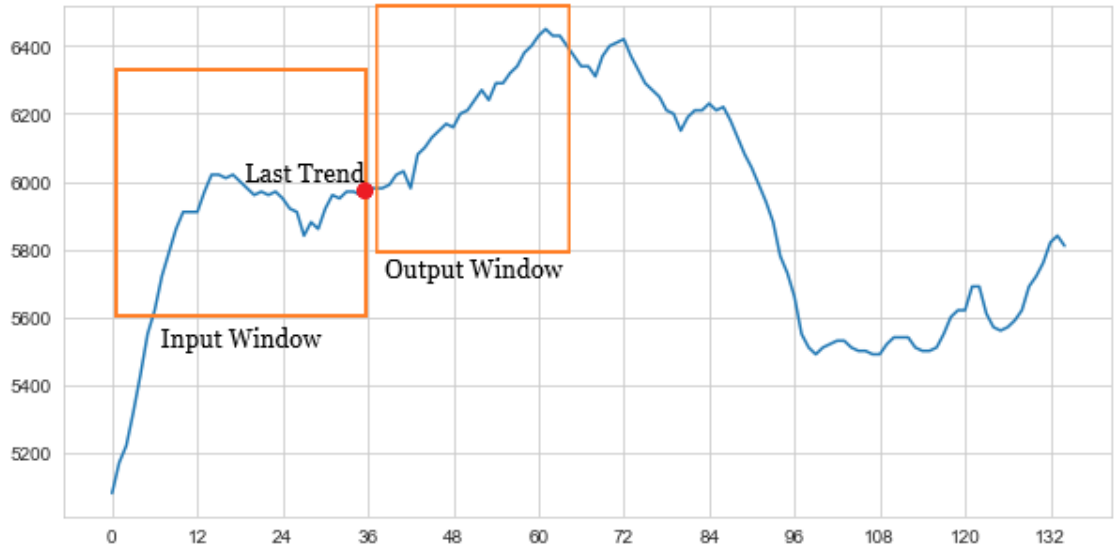


Figure 3.3.2: Example of series normalisation. After deseasonalizing the time series, the last trend value of the input window (red filled dot) is subtracted from the input and output windows.

Chapter 4

Federated clustering

In this chapter, we describe a novel method to cluster the clients in a federated learning network and improve local performance. Similar to the `FederatedAveraging` algorithm, the clients share an updated version of the global model after training on local data. However, in this approach, we also allow the client to shares a feature vector capturing the patterns of its local data distribution. In subsection 2.2.1, we already proposed three methods to extract these features. We follow the strategies of Hyndman, Wang and Laptev [45], Kang, Hyndman and Smith-Miles [46] and Bandara et al. [13] to extract three different sets of features that aim to capture the majority of the dynamics that can be observed in time series. We name these extractors `hwl2015`, `khs2017` and `ban2019` respectively.

The procedure is as follows. The coordinating server first clusters all the clients applying an agglomerative clustering algorithm to the extracted feature vectors (using one of the aforementioned feature extractors). Then the algorithm proceeds as usual, selecting a subset of clients to update the shared model. However, when the server aggregates the clients' updates, it merges the updates of each group separately, in a different copy of the shared model. In subsequent steps, when a client is selected to participate in a communication round, the server has to find first this client's cluster label, since it needs to send to model to which this client is contributing. The server now maintains k models.

We use this a hierarchical clustering algorithm because, generally, it is easier to decide on the number of clusters just by programmatically looking at the resulting dendrogram. Even though other algorithms, such as k-Means, have better time complexity, we found this algorithm suitable for our particular problem as the number of clients to the cluster was not very large. We also decided to use this algorithm due to the flexibility that provides (distance metrics and linkage methods). We must also mention that we use the scikit-learn library to perform the clustering [91].

Therefore, we decided to focus our attention on two parameters: the feature extractor used and the selection of k (the number of clusters). The goal is to understand the cause and effect relationship (if any) between these two parameters and the local performance in the client. Exploring this association with other parameters, such as the distance metrics or the linkage

method was not possible on account of time constraints.

Moreover, to get comparable results, we also fix the hyperparameters of the model trained across experiments, and we make sure that the weight initialisation across experiments is not random, what removes the possibility of external factors influencing the cause-effect relation between parameters that is the object of this study. Finally, regarding the meta-parameters that control the federated averaging algorithm (i.e., C, B and E), we indicate in each experiment what values are used, but again, we make sure that they take the same values within the same experiment (dataset).

Model parameter	Model value
LSTM-cell-neurons	8
Batch size	4
Learning rate	0.001
Optimizer	RMSProp
L2-regularization-weight	0.0005
Gaussian noise	False

Table 4.0.1: Hyperparameters of the 2-layer NN trained. The first hidden layer is an LSTM layer. The size of the output layer is a dense layer whose size (number of units) depends on the forecasting horizon of a given problem.

4.1 Implementation details

Class diagram

The goal of this section is to clarify the proposed algorithm by studying the interactions between the components that make up the developed artefact. In Figure 4.1.1 we can see a class diagram reporting the agents involved in the algorithm.

As expected, there are two primary components whose interactions drive the execution of the algorithm: the `Server` and `Client`. The former holds a reference to a collection of clients, and, when needed, uses the `update` method of the client to update the global shared model. In response to this call, the client returns the update weights to the server. The client also has some utility methods to inform the server about the number of samples in its local dataset (the `nk`) and to evaluate the performance of a given model weights using its local data (`evaluatePerformance`). It is important to notice that both client and server use the same interface when communicating to the model that is being trained (whether the model is the local model in the client or the shared one in the server). This makes it easier for client and server to change its weights.

On the other hand, the server also holds a reference to a `ClusteringStrategy` to cluster the clients in the federated network. This class implements the Strategy Design Pattern [92], which enables the server to select a specific clustering strategy (e.g., hierarchical, precomputed) at runtime.

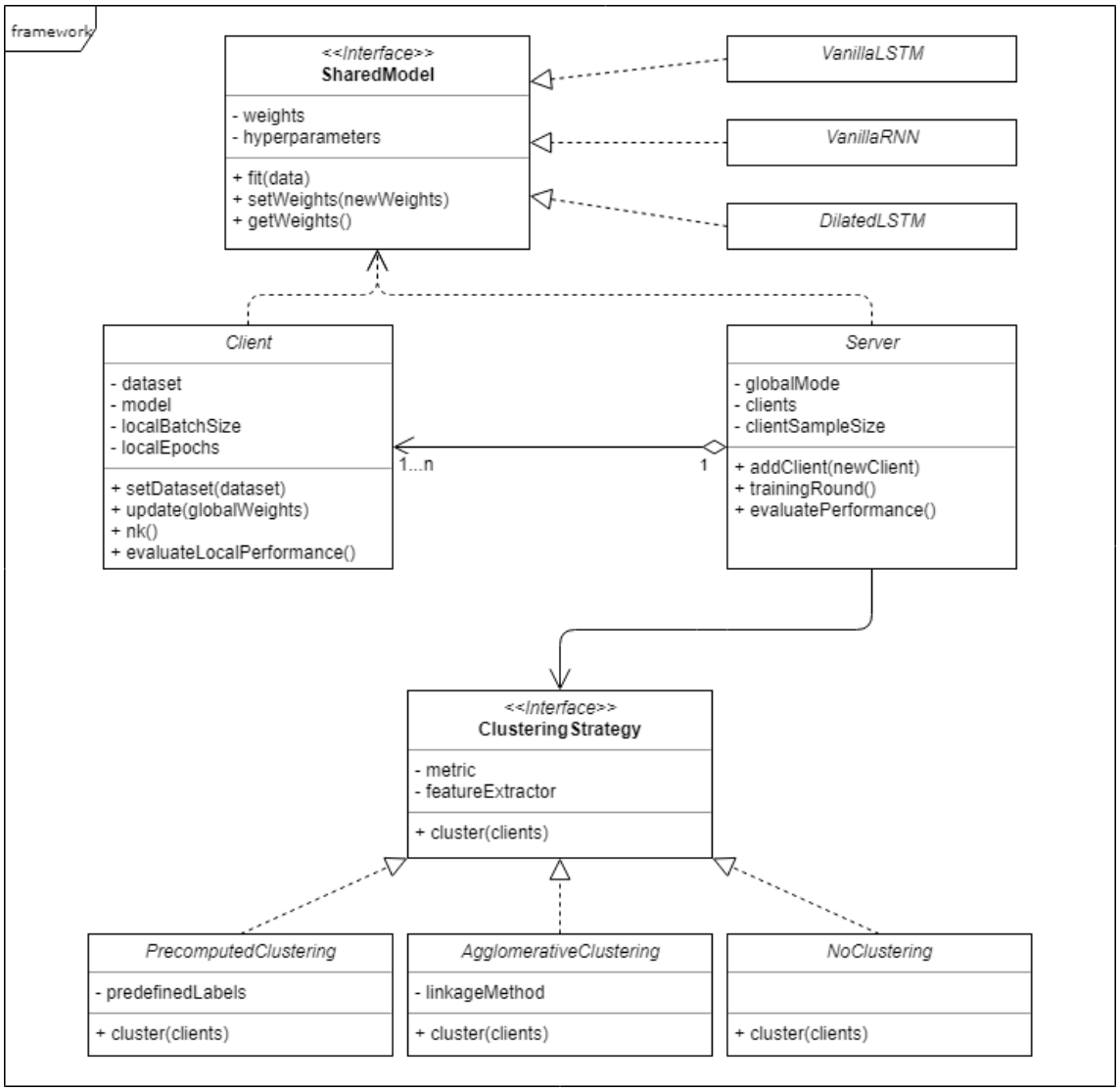


Figure 4.1.1: Each class in the diagram represents an agent, and relationships express communications between agents

Sequence diagram

In Figure 4.1.2 we show the sequence diagram of the above mentioned interactions. We consider the example of one server and three clients. The algorithm will cluster the clients into two groups: {client1, client2} and {client3}. Notice how the training process is the client is asynchronous. The server does not wait for an immediate answer after asking a client to update the global weights.

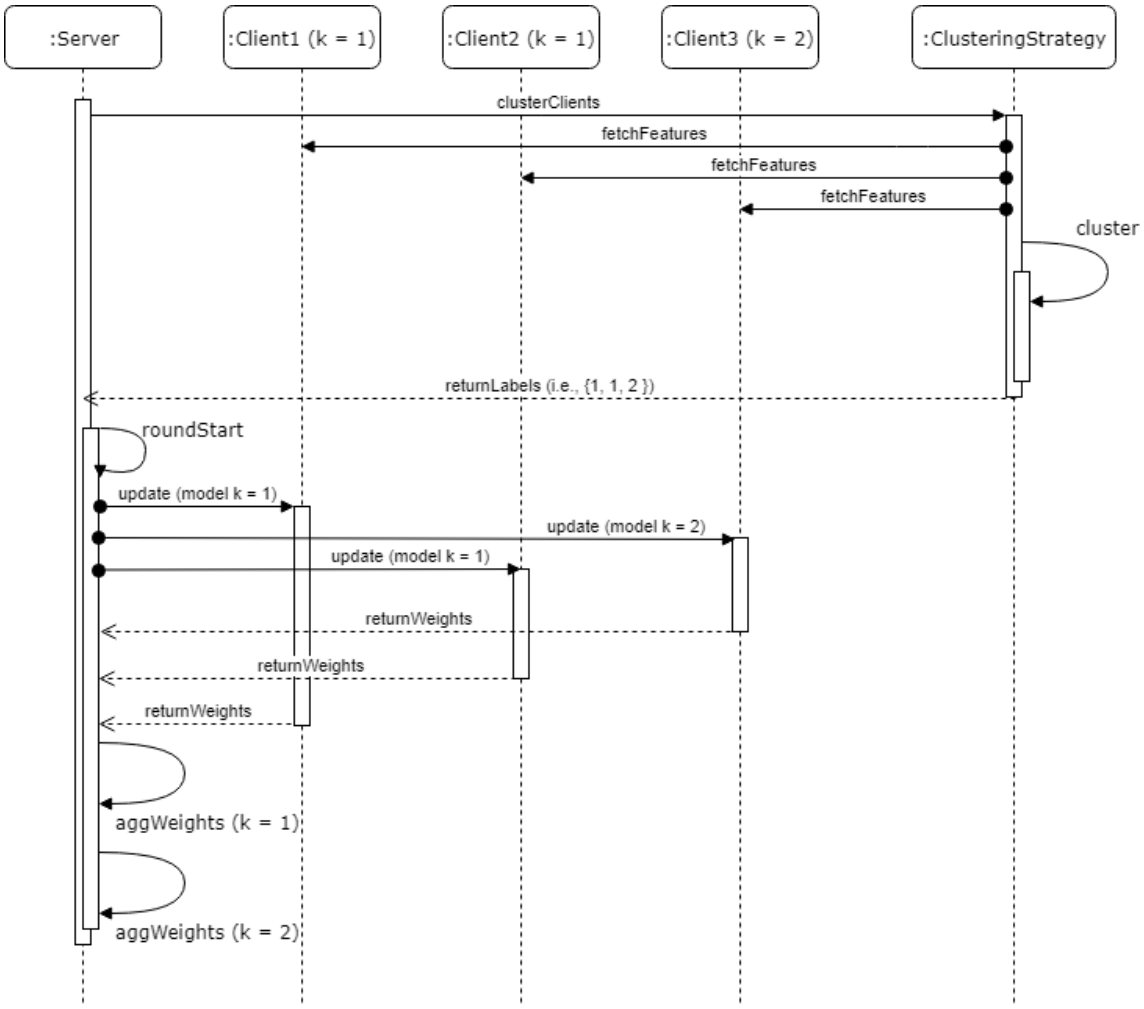


Figure 4.1.2: Sequence diagram of the federated clustering algorithm

Chapter 5

Results

In this section, we present the results of the experiments on the three different datasets. We must emphasise again that, given the large number of configuration parameters, we decided to fix most of them since our main concern is finding how much the original algorithm can be improved by building separate models for subgroups of similar clients. However, in order to collect enough empirical evidence, we decided to investigate the effect of two parameters, namely, the feature based representation and the choice of k. In chapter 4 - Federated clustering you can find a description of the fixed parameters and their values.

5.1 Case 1: synthetic dataset – Disparate time series

As we previously presented, this dataset consists of synthetically generated control charts with 6 visible, distinct patterns. These patterns divide the series into 6 different categories: normal, cyclic, increasing trend, decreasing trend, upward shift and downward shift. The goal is to forecast the next 10 values using the previous 14 observations ($10 \times 1{:}4$ in consonance with our heuristic).

For this particular case, we decided not to apply any preprocessing except for the missing values, mostly because an STL decomposition would be useless given that the vast majority of these time series have no trend or seasonal component. Instead, we fit the model using raw data. We train the LSTM presented in chapter 3, or k LSTMs with the same weight initialisation when applying the clustering algorithm. We use 120 time series sampled homogeneously over the different time series subgroups (i.e., 30 series from each subgroup). The time series are distributed over 120 clients (i.e., 1 time series per client). Regarding the configuration parameters of the federating averaging algorithm, we set $C = 0{:}3$ (the fraction of selected clients), $E = 2$ (the number of local epochs), $B = 8$ (the local batch size) and we train for 200 communication rounds.

Table 5.1.1 shows the evaluation results. The first row corresponds to the case of no-clustering; in other words, the vanilla federated averaging algorithm, that we use as the baseline to beat. The last row corresponds to an optimal grouping using the ground-truth categories where we train an LSTM for each time series subgroup. The rest of the rows corresponds to cases where we cluster

the time series in k groups using the features obtained from a particular feature extractor.

		sMAPE			MASE		
		mean	median	p90	mean	median	p90
k	feature extractor						
1	NA (baseline)	0.232	0.208	0.365	1.409	1.065	2.839
2	hwl2015	0.223	0.181	0.341	1.316	1.045	2.384
	khs2017	0.246	0.212	0.354	1.441	1.182	2.796
	ban2019	0.240	0.209	0.370	1.456	1.084	2.691
6	hwl2015	**0.181**	**0.132**	**0.309**	**0.951**	**0.824**	**1.249**
	khs2017	0.245	0.196	0.472	1.462	1.053	3.004
	ban2019	0.226	0.175	0.448	1.228	1.029	1.846
*	NA (expert)	0.179	0.133	0.286	0.922	0.869	1.312

Table 5.1.1: Results of evaluation using sMAPE and MASE measure for 120 time series of the synthetic dataset. For each column, the results of the best performing configuration are marked in bold (excluding last row, which represents optimal grouping).

We see that the only cases in which the proposed method outperforms the baseline is in the case of $k = 6$ using the features proposed by Hyndman, Wang and Laptev (that we shorten as `hwl2015`), and when doing an optimal grouping. A brief analysis of the clusters identified in the case of $k = 6$ showed that the previously mentioned feature extractor was able to identify 90% of the ground truth subgroups. However, it is important to notice that, for the rest of the cases, our method was not proved to be as successful. This underlines the importance of using the appropriate feature extractor and the choice of k. We can see that, even when using the same feature extractor but a different value for k, the improvement is not so remarkable. Below, we analyse the case in which our method beat the baseline.

Figure 5.1.1 shows that is not trivial to choose k even when looking at a dendrogram (right subfigure). Even though this experiment shows that the best choice of k is 6, the dendrogram suggest the existence of 2 distinct clusters. These two clusters are represented in the left subfigure.

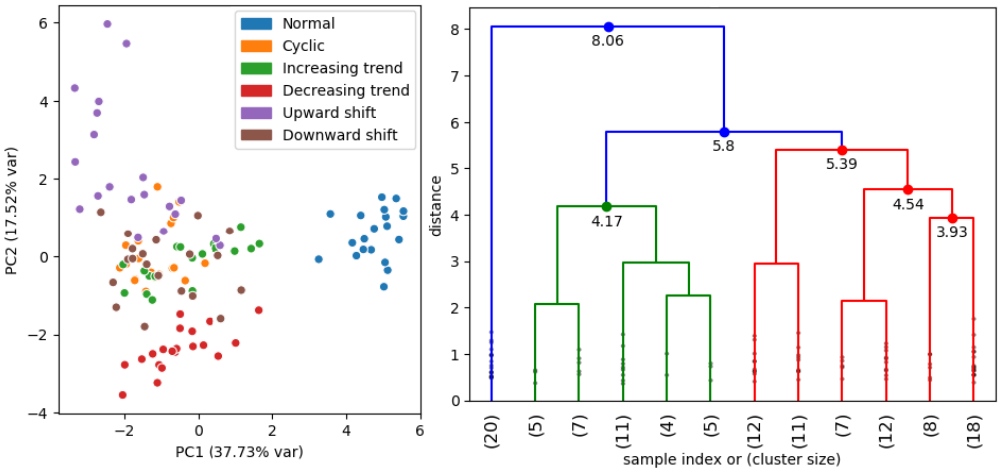


Figure 5.1.1: Two-dimensional representation of the Principal Component Analysis (PCA) of the features proposed by Hyndman, Wang and Laptev (`hwl2015`), extracted from the 120 time series (left); and the dendrogram obtained from the same features (right) using the ward method and euclidean distance.

It is also very interesting to look at the performance of each cluster separately. Figure 5.1.2 shows the evolution of the in-cluster mean sMAPE (i.e., an average of the sMAPE score of the clients in each cluster).

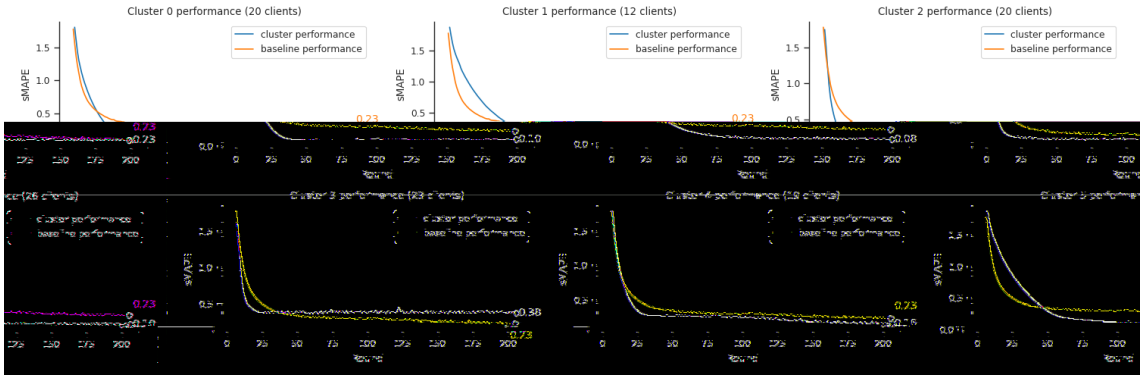


Figure 5.1.2: sMAPE vs communication round for the synthetic dataset experiment using the `hwl2015` feature extractor and $k = 6$. The orange line represents the baseline performance of the federated averaging algorithm. The blue line represents the average of the sMAPE score of the clients in the obtained cluster.

We can see that in some cases, like for cluster 1, the improvement is quite significant. Also note that, if you average the results of the 6 clusters, you get the same value for the mean sMAPE presented in Table 5.1.1 ($\approx$ 1:8). Let us look at how the forecasts differ for this particular cluster when applying the model trained using the baseline method and when applying the model trained using the proposed method.

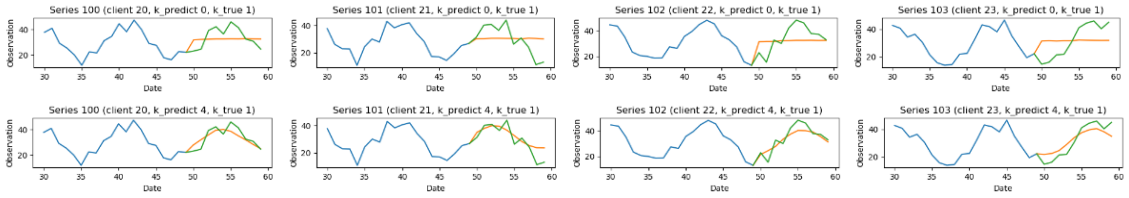


Figure 5.1.3: Forecasts for series 100, 101, 102 and 103 of the synthetic dataset obtained applying the model trained using the baseline method (top) and when applying the model trained using the proposed method (bottom).

Figure 5.1.3 shows how the baseline method is not able to learn the wave pattern in the "cyclical" series, mostly due to the noise introduced by the rest of the time series. On the other hand, since our method isolates (most) of the cyclical time series in their own cluster, the neural network can detect this pattern since it is only trained with this kind of data.

In Appendix A, we present supplementary plots for some of the cases that were not covered in this section. For example, it is possible to see how the forecasts look like for the rest of the clusters.

5.2 Case 2: NN5 dataset – Homogeneous time series

Unlike the previous dataset, the NN5 dataset does not seem to exhibit visually recognisable patterns, at least at first glance. The goal of using this dataset is twofold: finding if the proposed method can find related series even when the similarities are not so apparent, and finding if using the algorithm with a homogeneous datasets results in much weaker performance.

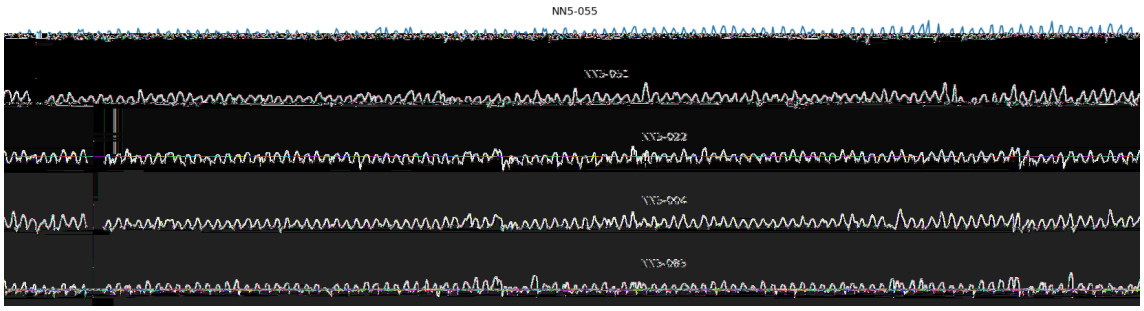


Figure 5.2.1: Sample of 5 series from the NN5 dataset.

In this case, we perform an STL given that the time series in this dataset have a strong seasonality component which may be unnecessarily difficult to learn for the LSTM. Again, we use 1 or k LSTMs depending if we cluster or not the clients. We train using the entire dataset (111 time series), where we try to predict the next 56 values using the previous 78 observations (the forecasting horizon is the same as the one used in the official competition). The time series are distributed over 111 clients (i.e., 1 time series per client). For the federating averaging algorithm, we set C = 0:3 (the fraction of selected clients), E = 1 (the number of local epochs), B = 64 (the local batch size) and we train for 300 communication rounds.

Table 5.2.1 shows the evaluation results for this dataset. The first row corresponds to the baseline. Note that there is no optimal grouping in these experiments because there is no domain knowledge available to group the time series (unlike in the previous case, where we could group using the ground-truth category).

		sMAPE			MASE		
		mean	median	p90	mean	median	p90
k	feature extractor						
1	NA (baseline)	0.237	0.219	0.317	0.924	0.861	1.152
2	hwl2015	0.237	0.221	0.323	0.927	0.867	1.163
	khs2017	0.237	0.220	0.322	0.925	0.875	1.162
	ban2019	0.237	0.218	0.322	0.929	0.875	1.158
6	hwl2015	0.238	0.221	0.324	0.932	0.847	1.199
	khs2017	0.241	0.221	0.325	0.944	0.878	1.212
	ban2019	0.240	0.223	0.319	0.939	0.867	1.178

Table 5.2.1: Results of evaluation using sMAPE and MASE measure the 111 time series of the NN5 dataset.

We can see straight away how the results are very similar for all the possible configurations. One possible reason that would explain these results is the lack of similar groups in this dataset. However, there is also a possibility that the algorithm was not able to find these similarities. This question is left to discussion for a later section.

In one instance or another, we can see that at least the proposed method did not perform worse than the baseline. Nevertheless, it is also true that, in the case that the dataset is truly homogeneous, we would be doing unnecessary work since we would be training almost identical models with subsets of the data.

Furthermore, Figure 5.2.2 shows the representation of the extracted features in a 2-dimensional space (the first two principal components). Again, it is challenging by looking at this graph to know if our method failed because the extracted features were not relevant for this dataset, or because the dataset was indeed homogeneous.

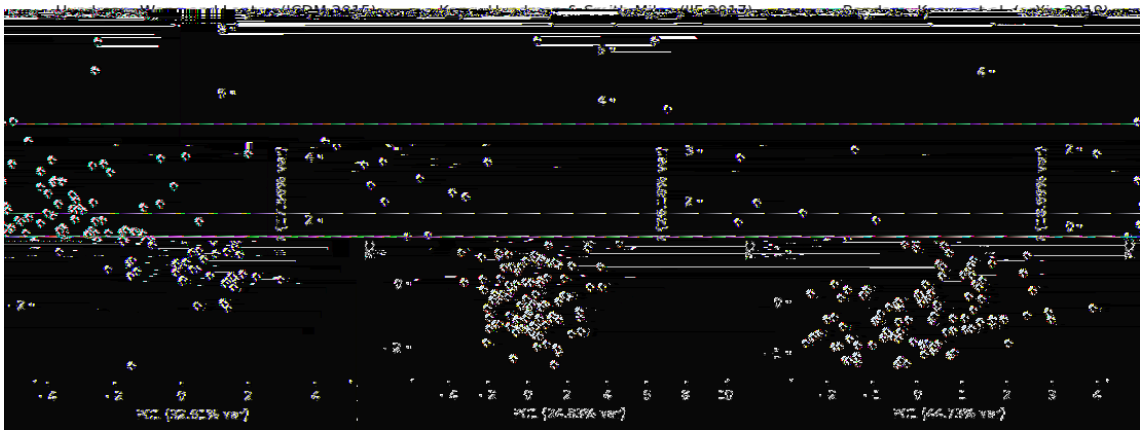


Figure 5.2.2: Two-dimensional representation of the Principal Component Analysis (PCA) of the features extracted from the NN5 dataset using the three proposed feature extractors.

Besides, if we look again at the evolution of the performance per cluster (Figure 5.2.3), we can see how the performance of some of them is way worse than the baseline performance (e.g., second-row, second-column subgraph), while in other cases is considerably better (e.g., second-row, first-column subgraph). This result could mean that there are, in fact, clusters in the dataset, but we are not able to extract them using these particular set of features. If the dataset was homogeneous, we should not see significant differences, no matter how we group the data.

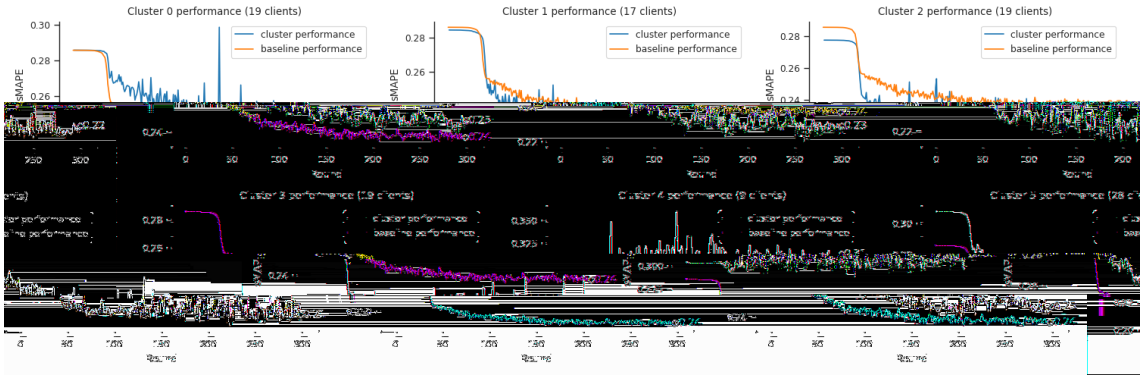


Figure 5.2.3: sMAPE vs communication round for the NN5 dataset experiment using the `ban2019` feature extractor and $k = 6$. The orange line represents the baseline performance of the federated averaging algorithm. The blue line represents the average of the sMAPE score of the clients in the obtained cluster.

To summarise, we have seen how no matter the feature extractor used, it was not possible to beat the baseline performance, which could be caused by inadequate feature extraction. We decided not to include additional graphs in the annex since most of them showed similar behaviour to what we already presented in this section.

5.3 Case 3: Ericsson dataset – Same pattern, different scale

This last case study reports data that is at a mid-point between the two previous cases. It does not exhibit distinct patterns, but neither is entirely homogeneous. In this dataset, all the time series present the same pattern, but the magnitude of this pattern clearly changes between series, as we can see in the next figure.

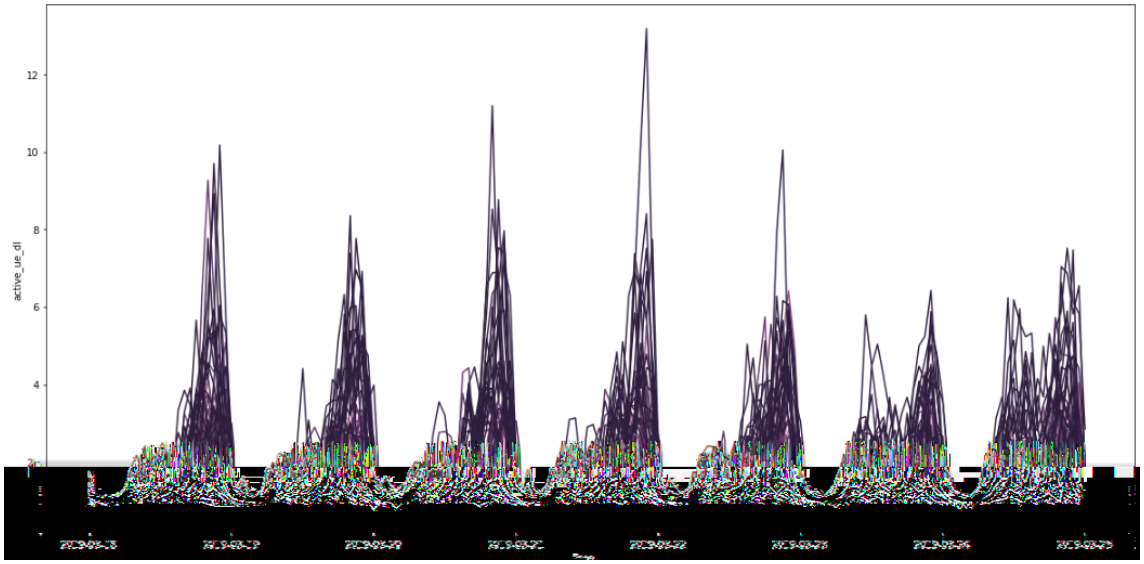


Figure 5.3.1: Observations of a particular Key Performance Indicator at 120 different Ericsson's base stations. Colour represents the average magnitude of the observation for a particular series. Hence, darker lines represent base stations with a higher "load".

As you can see, the seasonal component is the same (24 days), the spikes are usually located at the same points, and all of them tend to follow the same oscillations. The only difference is the amplitude, so to speak, of the time series. Let us take advantage of this fact to group the series as we did in the first case: using domain knowledge. In this case, what we are going to do is:

1. For each time series t_i, sum all the observations, getting $\mathtt{tsum}_i$.
2. Rank the series in ascending order using $\mathtt{tsum}_i$ as the score.
3. Compute the 0.1, 0.25, 0.4, 0.55, 0.7 and 0.8 quantiles.
4. Label each series with the lowest quantile to which it belongs.

In this way, we group the data in 6 groups, as we can see in Figure 5.3.2. The main purpose of this manual grouping is to see how well this precomputed clustering performs when compared to the feature-based clustering.

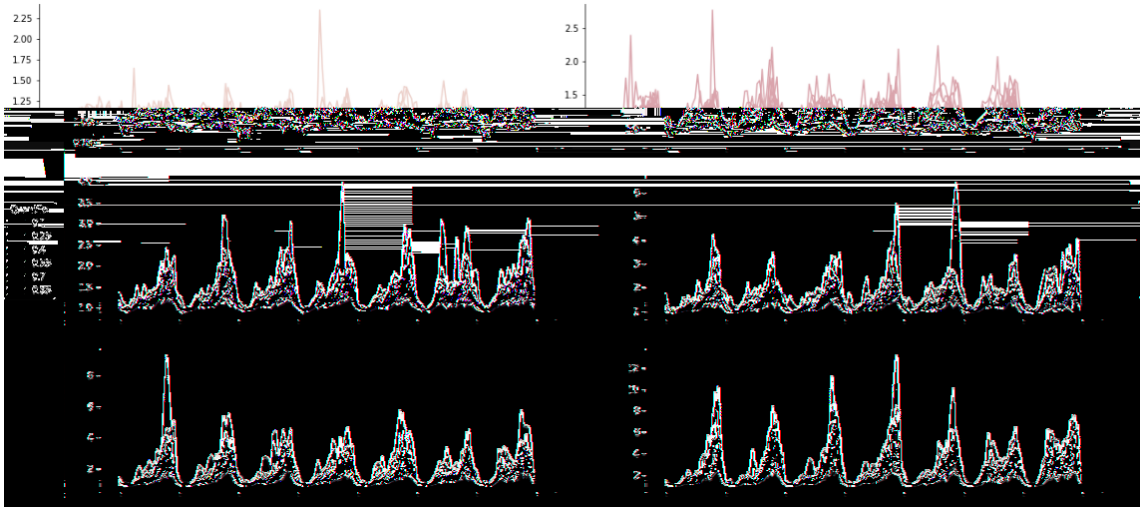


Figure 5.3.2: Same time series shown in Figure 5.3.1, but divided by quantile.

For this particular case, we decided again not to apply any preprocessing except for the missing values. We train using the entire dataset (120 time series), where we try to predict the next 8 hours using the previous 12 observations. The time series are distributed over 120 clients (as if we deployed the training agent in the base station). For the federating averaging algorithm, we set C = 0:3 (the fraction of selected clients), E = 2 (the number of local epochs), B = 32 (the local batch size) and we train for 200 communication rounds.

Table 5.3.1 shows the evaluation results for this dataset. The first row corresponds to the baseline. The last line corresponds to the precomputed clustering using the quantiles as we mentioned earlier.

		sMAPE			MASE		
		mean	median	p90	mean	median	p90
k	feature extractor						
1	NA (baseline)	0.355	0.268	0.717	3.836	2.947	7.701
2	hwl2015	0.292	0.251	0.571	2.996	2.692	4.714
	khs2017	0.302	0.243	0.657	3.643	2.723	7.529
	ban2019	0.314	0.250	0.640	3.554	2.585	7.384
4	hwl2015	0.300	**0.217**	0.639	**2.868**	2.470	**4.667**
	khs2017	0.327	0.236	0.634	3.758	2.903	6.412
	ban2019	0.329	0.266	0.662	3.571	2.663	7.451
6	hwl2015	**0.294**	0.256	**0.556**	2.917	**2.399**	4.980
	khs2017	0.329	0.263	0.692	3.791	2.900	7.765
	ban2019	0.313	0.273	0.584	3.580	2.741	7.420
*	NA (expert)	0.230	0.186	0.462	2.211	1.910	3.794

Table 5.3.1: Results of evaluation using sMAPE and MASE measure the 120 time series of the Ericsson dataset. For each column, the results of the best performing configuration are marked in bold (excluding last row, which represents manual grouping).

The `hwl2015` feature extractor seems to yield again the best results. It is also important to highlight the huge improvement in the percentile 90 error, both for sMAPE and MASE metrics when compared with the baseline (around a 40% decrease in the best case). The other two feature extractors do not seem to improve accuracy in a significant way. Additionally, unlike in the first dataset where the `khs2017` and `ban2019` performed worse than the baseline, in this case, they were both able to beat it by a considerable amount. Nonetheless, the manual grouping (last row) outperformed every feature-based clustering again. This is impressive if we consider that the idea behind the manual grouping (i.e., using the quantiles) is very unsophisticated.

Let us try to understand why, for this dataset, all of the clustering configurations beat the baseline performance by looking at the extracted features in a 2-dimensional space. Let us also mark each point with a colour representing the quantile to which it belongs.

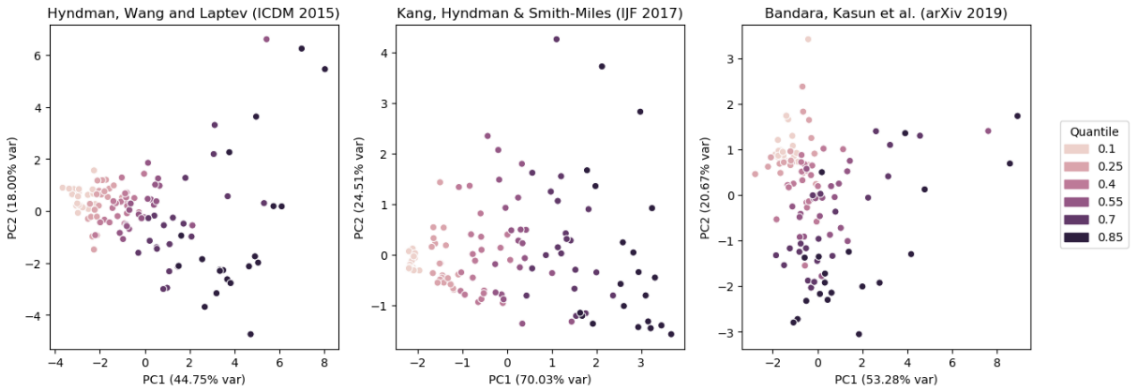


Figure 5.3.3: Two-dimensional representation of the Principal Component Analysis (PCA) of the features extracted from the Ericsson dataset using the three proposed feature extractors. Each point (a time series) is coloured according to the quantile that the original time series belonged.

We can see in Figure 5.3.3 that, even though there are not clear separable clusters in this representation, time series belonging to the same quantile are close to each other. Also, note how the first two principal components of the middle subplot (features from `khs2017`) retain 94,54% of the variance, in other words, is likely that the features from this feature extractor are highly correlated. If that would be the case, we could remove most of the features from that particular extractor without affecting the performance, which could also improve the interpretability of the clustering.

Let us show again the performance of each cluster individually. In this case, we will compare the baseline with the case of manual grouping using the quantiles.

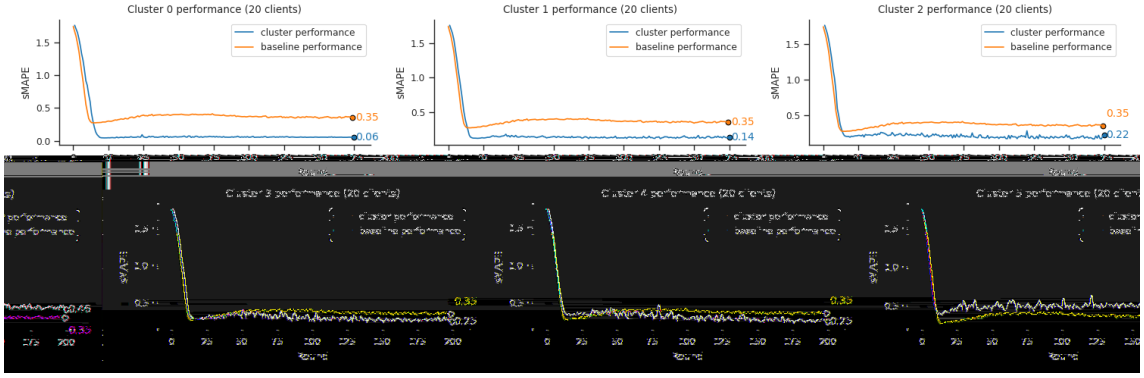
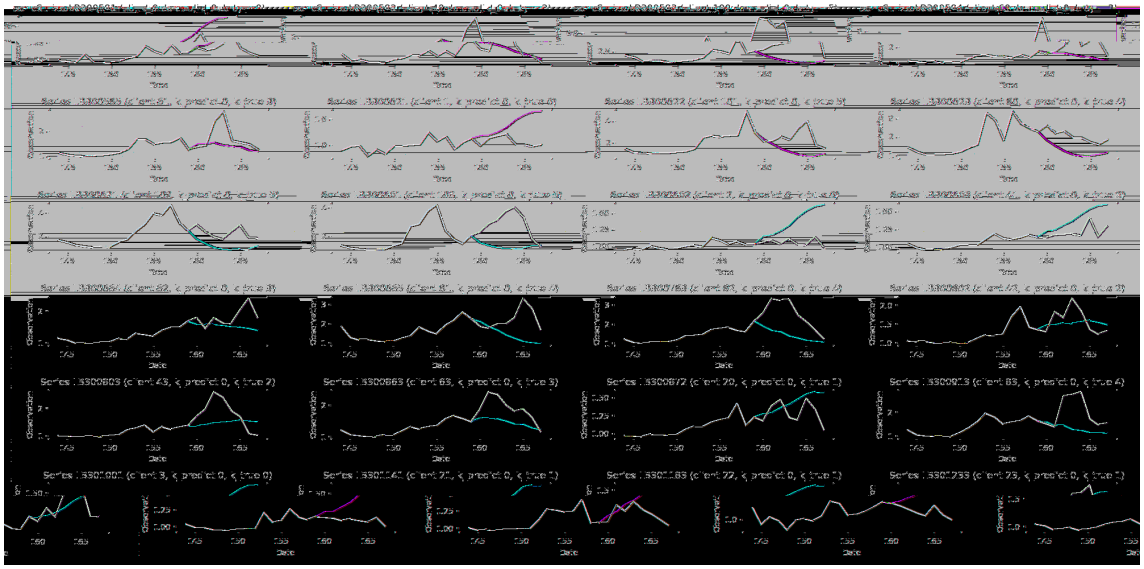


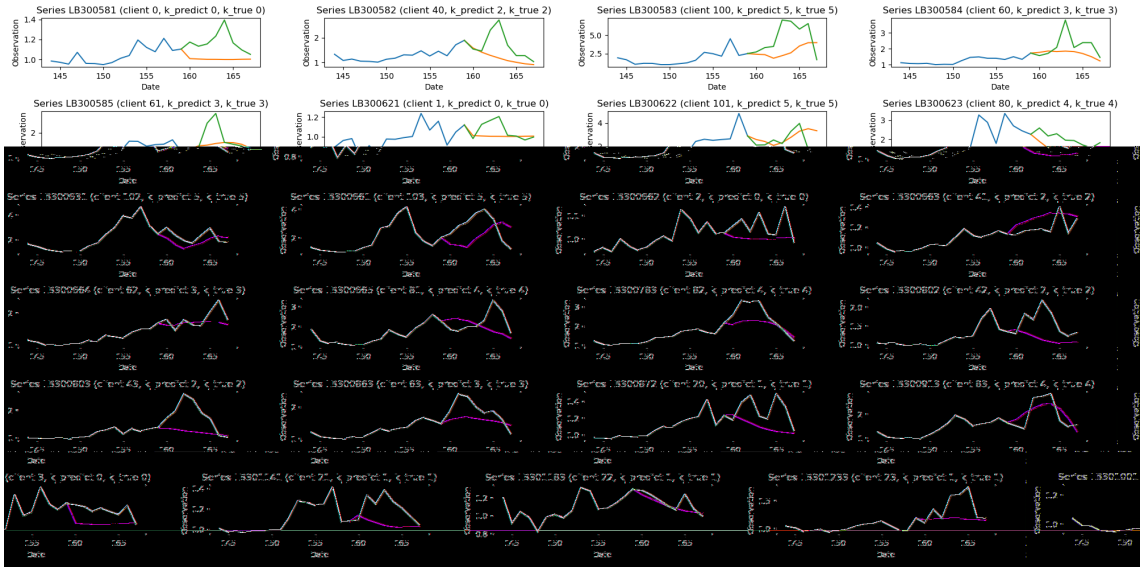
Figure 5.3.4: sMAPE vs communication round for the Ericsson dataset experiment grouping by quantile. The orange line represents the baseline performance of the federated averaging algorithm. The blue line represents the average of the sMAPE score of the clients in the obtained group.

The best case is shown in the first subplot, with an error decrease of around The worst case is displayed in the last subplot, with an error increase of 31%. However, in most of the cases, the error is reduced. This suggests again the idea of a better grouping that would allow reducing the error for all the clusters.

Finally, let us look at the actual forecasts when using the baseline model and when using the model trained using the proposed method (quantile grouping).



(a) Forecasts using the baseline method



(b) Forecasts using the proposed method

Figure 5.3.5: Forecasts for a sample of 24 series from the Ericsson dataset obtained applying the model trained using the baseline method (top) and when applying the model trained using the proposed method (bottom).

Both Figure 5.3.5a and Figure 5.3.5b show the same series, the only difference is the forecast (orange line). Let us index each graph by $(\mathrm{row}; \mathrm{column})$ to better reference them. We can see that some predictions that were completely wrong using the baseline method, like for example $(2;2)$, $(3;3)$ and $(6;3)$, are considerably improved when training with our method.

Chapter 6

Summary and discussion

6.1 Discussion

The problem of the optimal grouping of time series, although being able to improve the capability of recurrent neural network algorithms, is not trivial to solve given the large number of design parameters. On top of that, the federated averaging algorithm also has multiple knobs to tweak, even when not taking into consideration the hyperparameters of the trained model. This makes it particularly challenging to come up with a fully automatic procedure that can identify subgroups of similar time series, and that can be used with any dataset, even when they have similar characteristics. Since many aspects affect the overall performance, the scope of this project has been to find a suitable configuration that was able to exploit similarities in the presence of disparate time series, and that could improve the baseline performance of conventional federated optimisation.

The proposed model has been proven effective to exploit similar time series when there are distinct differences between the time series, as in the synthetic dataset or the Ericsson dataset. However, even in this situation, the importance of choosing one parameter configuration or the other does not become negligible. Besides, when the differences between the time series are not so apparent or non-existent, as in the NN5 dataset, finding similarities by extracting features is an arduous task, and, although the suggested model did not perform worse than the baseline, it is true that it is possible to lose generalisation capability when grouping similar time series in different clusters. However, the question of whether the algorithm was not able to find the similarities or whether, on the contrary, these similarities did not exist might also be relevant for future works. Finally, when using a real-world dataset as the one provided by Ericsson, this method was able to identify again similar, related time series and reduce the error in the forecast. Nonetheless, although not as apparent as in the first case, the similarities were also evident to the naked eye after performing an exploratory data analysis.

We have shown that one of the most crucial decision is the choice of the feature extractor. In the general case, calculating extra features rarely hurt the performance of the algorithm, even when those features were unnecessary to group the series (e.g., computing the mean after scaling all

the series to zero mean and unit variance). Also, although there must be a limit to the number of irrelevant features computed before they start to damage the performance of the model, the benefit that new features can bring overshadows the penalty paid for calculating useless features. This is why we believe that the features used in Hyndman, Wang & Laptev (ICDM 2015) are a good starting point. Nevertheless, we must not forget that grouping using the domain knowledge available has also shown impressive results, outperforming the fully automated method in some cases. This is why we must not overlook the effectiveness of data analysis. The additional benefit of identifying groups using domain knowledge is that we do not incur the additional overhead of computing the features and applying the clustering algorithm. Sometimes, using the domain knowledge can be as simple as separating the time series using a category label (although it is true than in this case it would also be possible to use the same label as an input feature of the neural network model).

Another factor that affected the final performance was the normalisation of the time series dataset and scaling the computed features before applying the clustering algorithm. In the case of the former, different datasets needs different preprocessing methods. For example, it is not possible to make an STL decomposition when the frequency is unknown. It is also useless when there are no trend and seasonal components (such in some series in the synthetic dataset). Choosing an adequate method is the goal of the data scientist. In this work, we have experienced with parametric methods, STL decomposition in particular, but it would also be reasonable to use non-parametric methods such as exponential smoothing. Our conclusion regarding this aspect is that, again, there is not a single normalisation method that works well in all cases. Same is the case of feature scaling before clustering. We think it is wise to standardise the range of the features to a common space before clustering. However, clustering algorithms are highly sensitive to scaling, and rescaling the data can ruin the result if it is not done correctly.

Regarding the network architecture, we do not consider that it had a significant effect on the results, primarily because the same architecture was across experiments. Probably improvements can be made by using a more complex neural network (e.g., Dilated RNN) but we do not think that if we had used a different network architecture, the results presented here and the overall ranking would have changed dramatically.

To conclude, there exists a high potential to exploit similarities between time series while training using the federated averaging algorithm. We presented a method to cluster the clients in a federated learning network to train various global models in order to exploit those similarities and improve the accuracy of the forecasts. To assess the effectiveness of the algorithm, we use different datasets with different characteristics. The results have shown that when diverse time series groups are present, the method is able to outperform the original federated averaging algorithm. However, we must not forget that there are essential design decision that drive the performance of the proposed model (primarily, the extracted features) and that is advisable to perform a previous data analysis to discover useful information to support these decisions.

6.2 Future work

As we previously said, correct feature selection is crucial to identify subgroups of similar time series. When meta-information is available, for example, when the dataset is labelled, it would be convenient to compute as many features as possible, and then perform filtering to remove weakly relevant features. The idea is to select features that are significant for predicting the target meta-information (significance testing). The previous approach, or any other feature weighting algorithm, could improve the performance of the subsequent clustering and even reduce the dimensionality of the feature vector (the "curse of dimensionality" also negatively affects the performance of the clustering algorithm).

Another drawback of the proposed model is that all the clients in the network are grouped before the training stage. In this work, this was reasonable, given the relatively small size of the network. However, this same algorithm might show scalability issues when dealing with a vast federation of clients. Additionally, it might be impossible to communicate with every single client before the training round starts. A natural way of solving this problem is by computing similarities between the clients that participate in the training round and then build the models accordingly.

Finally, an interesting approach would be to use a method that does not require to compute features since, in the context of time series, feature extraction is an arduous task, and it is challenging to find a set of features that work well in the general case. However, instead of manually engineering these features, we could use features that are already present in the client's model, and that represent the client's data distribution: the gradients of the model. It is likely that data corresponding to the same distribution or class generate similar gradients when it is fed to the model. This fact was also identified by Fung et al. when they presented a method to identify malicious clients in a federated learning network [93]. These authors presented a method to identify poisoning sybils based on the diversity of the client updates (the gradient) in the distributed learning process. In a similar way to these authors, we could cluster the clients using the difference in the gradient, for example, by using the cosine distance between two client updates.

Bibliography

[1] Gusev, Marjan and Dustdar, Schahram. "Going Back to the RootsThe Evolution of Edge Computing, An IoT Perspective". In: *IEEE Internet Computing* 22.2 (Mar. 2018), pp. 5–15. ISSN: 1089-7801.

[2] Manzurul, Mohammad, Morshed, Sarwar, and Goswami, Parijat. "Cloud Computing: A Survey on its limitations and Potential Solutions". In: *IJCSI International Journal of Computer Science* 10.4 (July 2013), pp. 159–163.

[3] Hofmann, Paul and Woods, Dan. "Cloud Computing: The Limits of Public Clouds for Business Applications". In: *IEEE Internet Computing* 14.6 (Nov. 2010), pp. 90–93. ISSN: 1089-7801.

[4] Arinze, Bay and Anandarajan, Murugan. "Factors that Determine the Adoption of Cloud Computing: A Global Perspective". en. In: *International Journal of Enterprise Information Systems* 6.4 (Oct. 2010), pp. 55–68. ISSN: 1548-1115, 1548-1123.

[5] Shi, Weisong et al. "Edge Computing: Vision and Challenges". In: *IEEE Internet of Things Journal* 3.5 (Oct. 2016), pp. 637–646. ISSN: 2327-4662.

[6] Sun, Chen et al. "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era". In: *arXiv:1707.02968 [cs]* (July 2017). arXiv: 1707.02968.

[7] MacGillivray, Carrie et al. *IDC FutureScape: Worldwide Internet of Things 2017 Predictions*. en. Nov. 2016. URL: https://www.idc.com/research/viewtoc.jsp?containerId=US40755816 (visited on 03/21/2019).

[8] McMahan, H. Brendan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *arXiv:1602.05629 [cs]* (Feb. 2016). arXiv: 1602.05629.

[9] Dwork, Cynthia. "Differential Privacy". In: *Automata, Languages and Programming*. Ed. by Michele Bugliesi et al. Springer Berlin Heidelberg, 2006, pp. 1–12. ISBN: 978-3-540-35908-1.

[10] Thorström, Marcus. "Applying machine learning to key performance indicators". en. Masters thesis in Software Engineering. Gothenburg: Chalmers University of Technology, 2017.

[11] Smith, Virginia et al. "Federated Multi-Task Learning". In: *arXiv:1705.10467 [cs, stat]* (May 2017). arXiv: 1705.10467.

[12] Bandara, Kasun, Bergmeir, Christoph, and Smyl, Slawek. "Forecasting Across Time Series Databases using Recurrent Neural Networks on Groups of Similar Series: A Clustering Approach". In: *arXiv:1710.03222 [cs, econ, stat]* (Oct. 2017). arXiv: 1710.03222.

[13] Bandara, Kasun et al. "Sales Demand Forecast in E-commerce using a Long Short-Term Memory Neural Network Methodology". In: *arXiv:1901.04028 [cs, stat]* (Jan. 2019). arXiv: 1901.04028.

[14] Cherif, Aymen, Cardot, Hubert, and Boné, Romuald. "SOM time series clustering and prediction with recurrent neural networks". en. In: *Neurocomputing* 74.11 (May 2011), pp. 1936–1944. ISSN: 09252312.

[15] Wang, Zhibo et al. "Beyond Inferring Class Representatives: User-Level Privacy Leakage From Federated Learning". In: *arXiv:1812.00535 [cs]* (Dec. 2018). arXiv: 1812.00535.

[16] Hitaj, Briland, Ateniese, Giuseppe, and Perez-Cruz, Fernando. "Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning". In: *arXiv:1702.07464 [cs, stat]* (Feb. 2017). arXiv: 1702.07464.

[17] Bermúdez-Edo, María, Della Valle, Emanuele, and Palpanas, Themis. "Semantic Challenges for the Variety and Velocity Dimensions of Big Data". In: *International journal on Semantic Web and information systems* 12 (Jan. 2016), pp. v–xviii.

[18] Ericsson. *Making waves with AI*. en. June 2018. URL: https://www.ericsson.com/en/mobility-report/reports/june-2018/applying-machine-intelligence-to-network-management (visited on 03/31/2019).

[19] RTInsights. *Value of Real-Time Data Is Blowing in the Wind*. en. Sept. 2015. URL: https://www.rtinsights.com/value-of-real-time-data-is-blowing-in-the-wind/ (visited on 03/31/2019).

[20] The Guardian. *The Cambridge Analytica Files*. en. Mar. 2018. URL: https://www.theguardian.com/news/series/cambridge-analytica-files (visited on 03/31/2019).

[21] Ericsson. *Forecast methodology*. en. 2019. URL: https://www.ericsson.com/en/mobility-report/mobility-visualizer/forecast-methodology (visited on 23/05/2019).

[22] Franziska Bell and Slawek Smyl. *Forecasting at Uber: An Introduction*. en. Dec. 2018. URL: https://robjhyndman.com/hyndsight/seasonal-periods/ (visited on 23/05/2019).

[23] Gardner, Everette S. "Exponential smoothing: The state of the art". en. In: *Journal of Forecasting* 4.1 (1985), pp. 1–28. ISSN: 02776693, 1099131X.

[24] Makridakis, Spyros and Hibon, Michèle. "The M3-Competition: results, conclusions and implications". en. In: *International Journal of Forecasting* 16.4 (Oct. 2000), pp. 451–476. ISSN: 01692070.

[25] Makridakis, Spyros, Spiliotis, Evangelos, and Assimakopoulos, Vassilios. "The M4 Competition: Results, findings, conclusion and way forward". In: *International Journal of Forecasting* 34.4 (2018), pp. 802–808. ISSN: 0169-2070.

[26] *Time Series Forecasting Competition for Computational Intelligence 2008*. en. 2008. URL: http://www.neural-forecasting-competition.com/NN5/results.htm (visited on 23/05/2019).

[27] Institute for Research and Applications of Fuzzy Modeling, University Ostrava. *Computational Intelligence in Forecasting (CIF) Competition 2016*. en. 2016. URL: http://irafm.osu.cz/cif/main.php?c=Static&page=results (visited on 23/05/2019).

[28] Makridakis, Spyros, Spiliotis, Evangelos, and Assimakopoulos, Vassilios. "Statistical and Machine Learning forecasting methods: Concerns and ways forward". en. In: *PLOS ONE* 13.3 (Mar. 2018). Ed. by Alejandro Raul Hernandez Montoya, e0194889. ISSN: 1932-6203.

[29] Wang, Jie and Wang, Jun. "Forecasting stochastic neural network based on financial empirical mode decomposition". en. In: *Neural Networks* 90 (June 2017), pp. 8–20. ISSN: 08936080.

[30] Qiu, Mingyue and Song, Yu. "Predicting the Direction of Stock Market Index Movement Using an Optimized Artificial Neural Network Model". en. In: *PLOS ONE* 11.5 (May 2016). Ed. by Zhen Wang, e0155133. ISSN: 1932-6203.

[31] Bredahl Kock, Anders and Teräsvirta, Timo. "Forecasting Macroeconomic Variables Using Neural Network Models and Three Automated Model Selection Techniques". en. In: *Econometric Reviews* 35.8-10 (Nov. 2016), pp. 1753–1779. ISSN: 0747-4938, 1532-4168.

[32] Gabor, Manuela Rozalia and Ancuta Dorgo, Lavinia. "Neural Networks Versus Box-Jenkins Method for Turnover Forecasting: a Case Study on the Romanian Organisation". In: *Transformations in Business and Economics* 16 (Mar. 2017), pp. 187–211.

[33] Lipton, Zachary C., Berkowitz, John, and Elkan, Charles. "A Critical Review of Recurrent Neural Networks for Sequence Learning". In: *arXiv:1506.00019 [cs]* (May 2015). arXiv: 1506.00019.

[34] Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. "On the difficulty of training Recurrent Neural Networks". In: *arXiv:1211.5063 [cs]* (Nov. 2012). arXiv: 1211.5063.

[35] Gers, Felix A., Schmidhuber, Jürgen, and Cummins, Fred A. "Learning to Forget: Continual Prediction with LSTM". In: *Neural Computation* 12 (2000), pp. 2451–2471.

[36] Gers, Felix A., Schraudolph, Nicol N., and Schmidhuber, Jürgen. "Learning Precise Timing with Lstm Recurrent Networks". In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pp. 115–143. ISSN: 1532-4435.

[37] Iglesias, Félix and Kastner, Wolfgang. "Analysis of Similarity Measures in Times Series Clustering for the Discovery of Building Energy Patterns". en. In: *Energies* 6.2 (Jan. 2013), pp. 579–597. ISSN: 1996-1073.

[38] Yadav, Munshi and Alam, Afshar. "Dynamic Time Warping (DTW) Algorithm in Speech: A Review". In: (Apr. 2018).

[39] Amin, T. B. and Mahmood, I. "Speech Recognition using Dynamic Time Warping". In: *2008 2nd International Conference on Advances in Space Technologies*. Nov. 2008, pp. 74–79.

[40] Wang, Xiaozhe et al. "A scalable method for time series clustering". In: (May 2019).

[41] Wang, Xiaozhe, Smith, Kate, and Hyndman, Rob. "Characteristic-Based Clustering for Time Series Data". en. In: *Data Mining and Knowledge Discovery* 13.3 (Sept. 2006), pp. 335–364. ISSN: 1384-5810, 1573-756X.

[42] Guijo-Rubio, David et al. "Time series clustering based on the characterisation of segment typologies". In: *arXiv:1810.11624 [cs, stat]* (Oct. 2018). arXiv: 1810.11624.

[43] Nanopoulos, Alex, Alcock, Rob, and Manolopoulos, Yannis. "Feature-based Classification of Time-series Data". In: *International Journal of Computer Research* 10 (2001), pp. 49–61.

[44] Fulcher, Ben D. and Jones, Nick S. "Highly comparative feature-based time-series classification". In: *IEEE Transactions on Knowledge and Data Engineering* 26.12 (Dec. 2014). arXiv: 1401.3531, pp. 3026–3037. ISSN: 1041-4347, 1558-2191, 2326-3865.

[45] Hyndman, R. J., Wang, E., and Laptev, N. "Large-Scale Unusual Time Series Detection". In: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. Nov. 2015, pp. 1616–1619.

[46] Kang, Yanfei, Hyndman, Rob, and Smith-Miles, Kate. "Visualising forecasting algorithm performance using time series instance spaces". In: *International Journal of Forecasting* 33 (Apr. 2017), pp. 345–358.

[47] Roux, Maurice. "A comparative study of divisive hierarchical clustering algorithms". In: *arXiv:1506.08977 [cs, q-bio]* (June 2015). arXiv: 1506.08977.

[48] Matlab documentation. *Inconsistency coefficient in hierarchical clustering*. en. URL: https://se.mathworks.com/help/stats/inconsistent.html (visited on 25/05/2019).

[49] Jr., Joe H. Ward. "Hierarchical Grouping to Optimize an Objective Function". In: *Journal of the American Statistical Association* 58.301 (1963), pp. 236–244.

[50] Kumar, Vijay, Chhabra, Jitender Kumar, and Kumar, Dinesh. "Performance evaluation of distance metrics in the clustering algorithms". In: *INFOCOMP* 13.1 (2014), pp. 38–52.

[51] Malsburg, Christoph von der. "Frank Rosenblatt: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms". In: *Brain Theory* (Jan. 1986), pp. 245–248.

[52] Minsky, M. and Papert, S. *Perceptrons*. Cambridge, MA: MIT Press, 1969.

[53] Rumelhart, David E., Hinton, Geoffrey E., and Williams, Ronald J. "Neurocomputing: Foundations of Research". In: ed. by James A. Anderson and Edward Rosenfeld. Cambridge, MA, USA: MIT Press, 1988. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. ISBN: 0-262-01097-6.

[54] Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[55] Hochreiter, Sepp and Schmidhuber, Jürgen. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80.

[56] Chang, Shiyu et al. "Dilated Recurrent Neural Networks". In: *arXiv:1710.02224 [cs]* (Oct. 2017). arXiv: 1710.02224.

[57] Greff, Klaus et al. "LSTM: A Search Space Odyssey". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (Oct. 2017). arXiv: 1503.04069, pp. 2222–2232. ISSN: 2162-237X, 2162-2388.

[58] Li, Yanghao et al. "Scale-Aware Trident Networks for Object Detection". In: *arXiv:1901.01892 [cs]* (Jan. 2019). arXiv: 1901.01892.

[59] Singh, Bharat, Najibi, Mahyar, and Davis, Larry S. "SNIPER: Efficient Multi-Scale Training". In: *arXiv:1805.09300 [cs]* (May 2018). arXiv: 1805.09300.

[60] Radford, Alec et al. "Language models are unsupervised multitask learners". In: ().

[61] Hafner, Danijar et al. "Learning Latent Dynamics for Planning from Pixels". In: *arXiv:1811.04551 [cs, stat]* (Nov. 2018). arXiv: 1811.04551.

[62] Kusupati, Aditya et al. "FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 9017–9028.

[63] Dean, Jeffrey et al. "Large Scale Distributed Deep Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1223–1231.

[64] Dean, Jeffrey and Ghemawat, Sanjay. "MapReduce: Simplified Data Processing on Large

[76] *Synthetic Control Chart Time Series Data Set*. en. 1999. URL: https://archive.ics.uci.edu/ml/datasets/synthetic+control+chart+time+series (visited on 23/05/2019).

[77] Makridakis, Spyros, Spiliotis, Evangelos, and Assimakopoulos, Vassilis. "The Accuracy of Machine Learning (ML) Forecasting Methods versus Statistical Ones: Extending the Results of the M3-Competition". In: (Oct. 2017).

[78] Shcherbakov, Maxim et al. "A survey of forecast error measures". In: *World Applied Sciences Journal* 24 (Jan. 2013), pp. 171–176.

[79] Hyndman, Rob J. and Koehler, Anne B. "Another look at measures of forecast accuracy". In: *International Journal of Forecasting* 22.4 (2006), pp. 679–688. ISSN: 0169-2070.

[80] Pratama, Irfan et al. "A review of missing values handling methods on time-series data". In: Oct. 2016, pp. 1–6.

[81] Taieb, Souhaib Ben and Hyndman, Rob J. "Recursive and direct multi-step forecasting : the best of both worlds". In: 2012.

[82] Taieb, Souhaib Ben et al. "Long-term prediction of time series by combining direct and MIMO strategies". In: *2009 International Joint Conference on Neural Networks* (2009), pp. 3054–3061.

[83] An, N. H. and Anh, D. T. "Comparison of Strategies for Multi-step-Ahead Prediction of Time Series Using Neural Network". In: *2015 International Conference on Advanced Computing and Applications (ACOMP)*. Nov. 2015, pp. 142–149.

[84] Hornik, Kurt. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080.

[85] Nelson, Michael et al. "Time series forecasting using neural networks: should the data be deseasonalized first?" en. In: *Journal of Forecasting* 18.5 (Sept. 1999), pp. 359–367. ISSN: 0277-6693, 1099-131X.

[86] Taieb, Souhaib Ben et al. "A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition". In: *arXiv:1108.3259 [cs, stat]* (Aug. 2011). arXiv: 1108.3259.

[87] Eric Rauch. *A Python port of R's stl function*. en. 2018. URL: https://pypi.org/project/rstl/ (visited on 04/04/2019).

[88] J Hyndman, Rob and Athanasopoulos, George. "6.1 Time series components". en. In: *Forecasting: Principles and Practice*. 2nd edition. Melbourne, Australia: OTexts, 2018.

[89] Proietti, Tommaso and Riani, Marco. "Transformations and seasonal adjustment". en. In: *Journal of Time Series Analysis* 30.1 (Jan. 2009), pp. 47–69. ISSN: 01439782, 14679892.

[90] Box, George E. P. and Cox, David R. "An Analysis of Transformations". In: 1964.

[91] Pedregosa, Fabian et al. "Scikit-learn: Machine Learning in Python". In: *arXiv:1201.0490 [cs]* arXiv: 1201.0490[((visited)-Add(r)1-WF29eslet for

[93] Fung, Clement, Yoon, Chris J. M., and Beschastnikh, Ivan. "Mitigating Sybils in Federated Learning Poisoning". In: *arXiv:1808.04866 [cs, stat]* (Aug. 2018). arXiv: 1808.04866.

Appendices

Contents

Appendix A

Additional result plots for the synthetic dataset

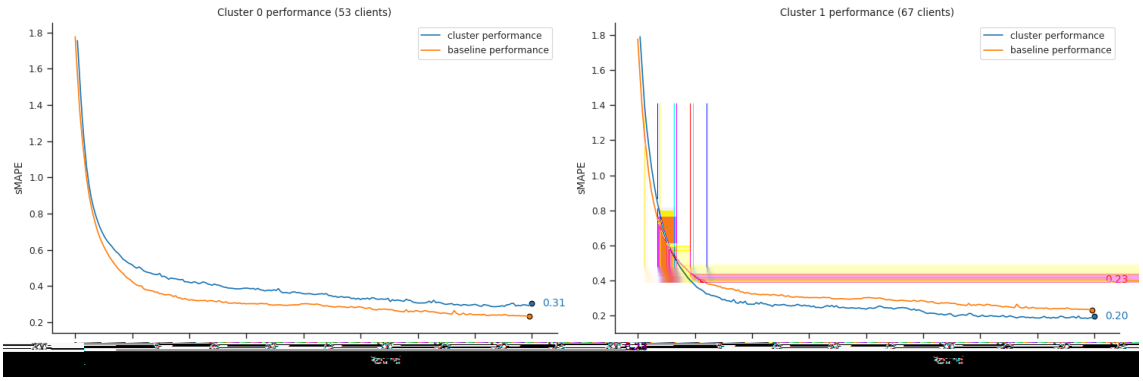


Figure A.0.1: sMAPE vs communication round for the synthetic dataset experiment using the `khs2017` feature extractor and k = 2. The orange line represents the baseline performance of the federated averaging algorithm. The blue line represents the average of the sMAPE score of the clients in the obtained cluster.

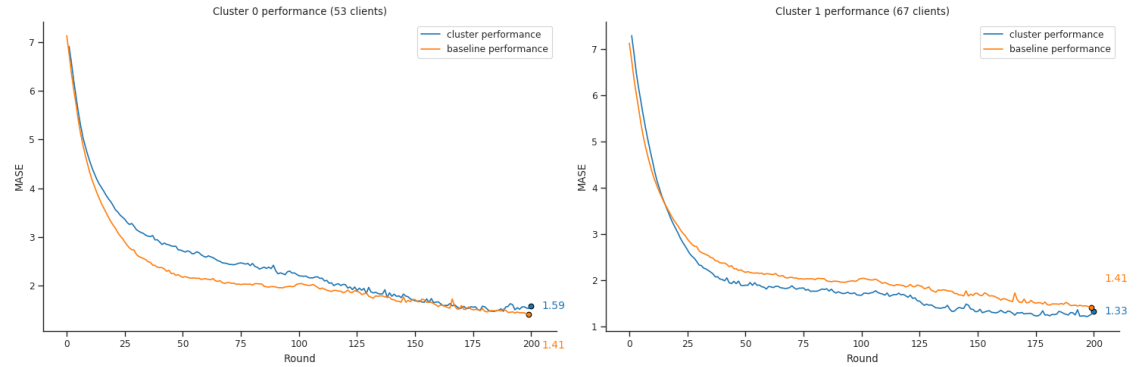


Figure A.0.2: MASE vs communication round for the synthetic dataset experiment using the `khs2017` feature extractor and k = 2.

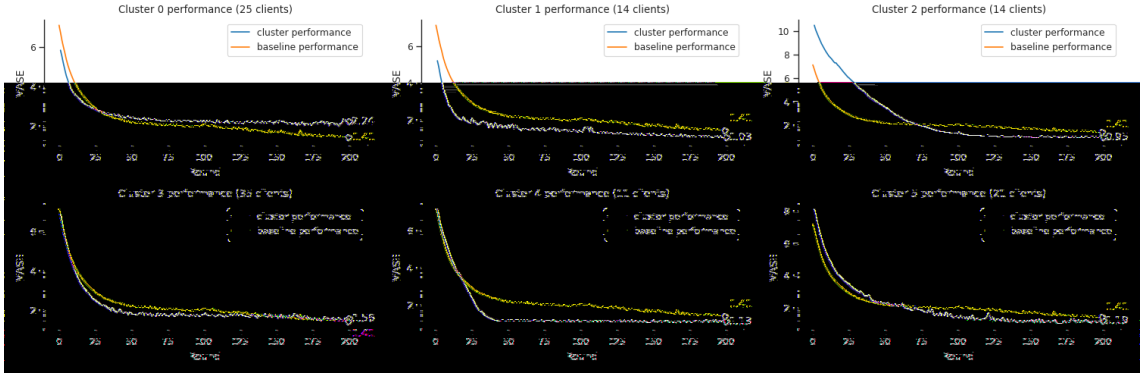


Figure A.0.3: sMAPE vs communication round for the synthetic dataset experiment using the `khs2017` feature extractor and k = 6.

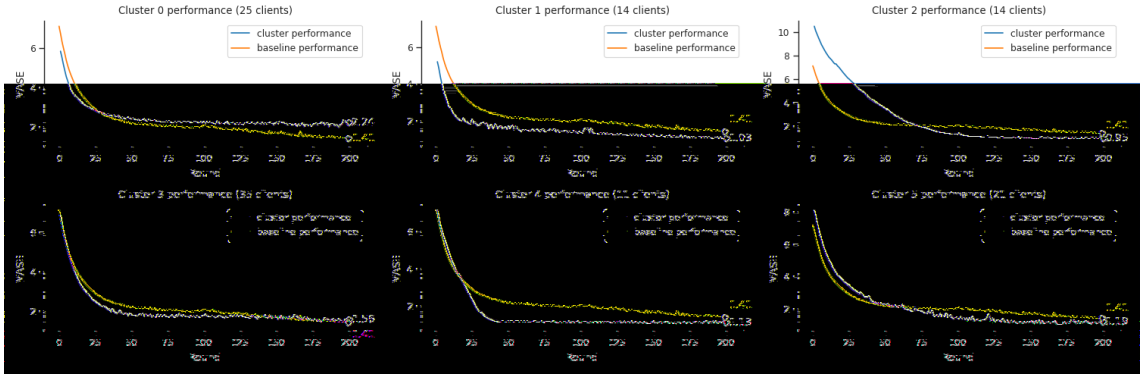


Figure A.0.4: MASE vs communication round for the synthetic dataset experiment using the `khs2017` feature extractor and k = 6.

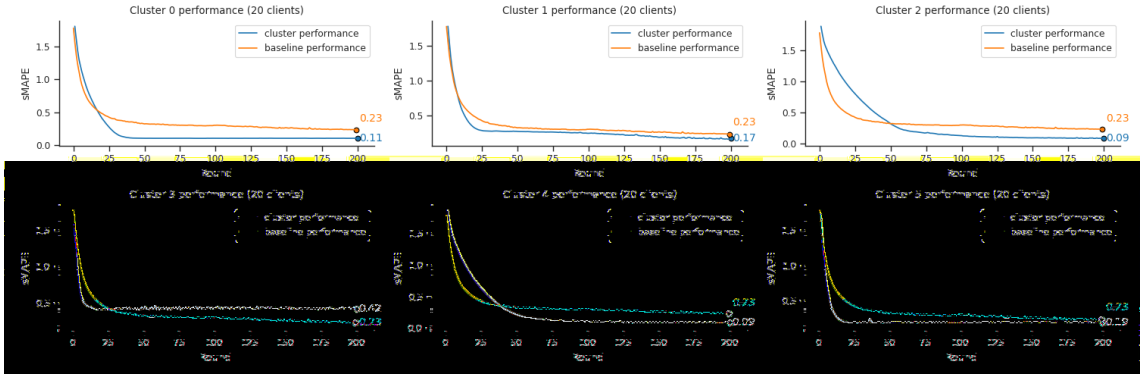


Figure A.0.5: sMAPE vs communication round for the synthetic dataset experiment using optimal clustering (grouping by category).

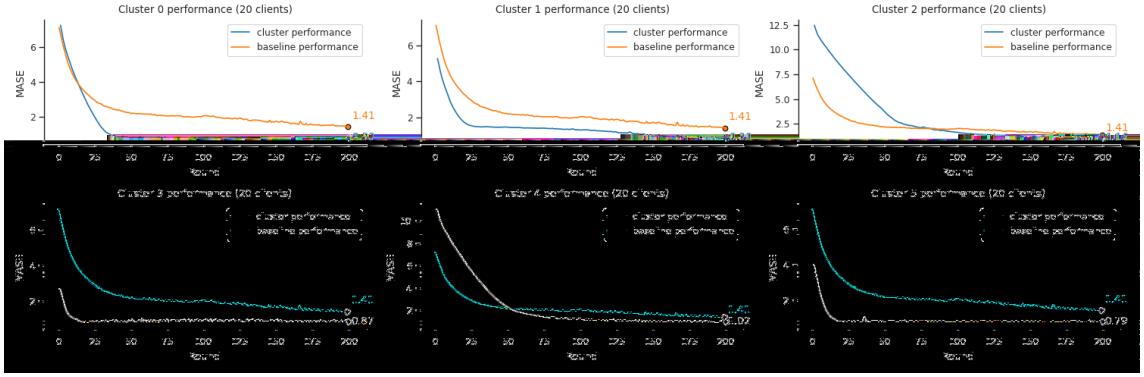


Figure A.0.6: MASE vs communication round for the synthetic dataset experiment using optimal clustering (grouping by category).

Appendix B

Additional result plots for the Ericsson dataset

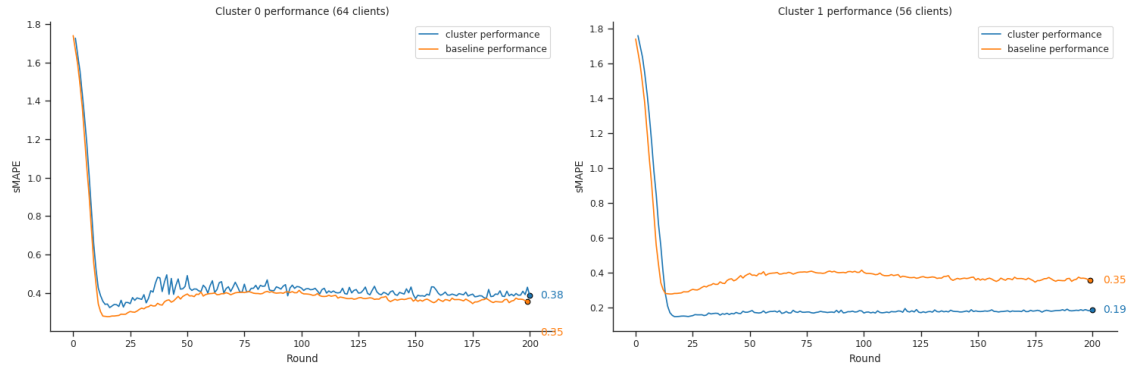


Figure B.0.1: sMAPE vs communication round for the Ericsson dataset experiment using the `hwl2015` feature extractor and k = 2. The orange line represents the baseline performance of the federated averaging algorithm. The blue line represents the average of the sMAPE score of the clients in the obtained cluster.

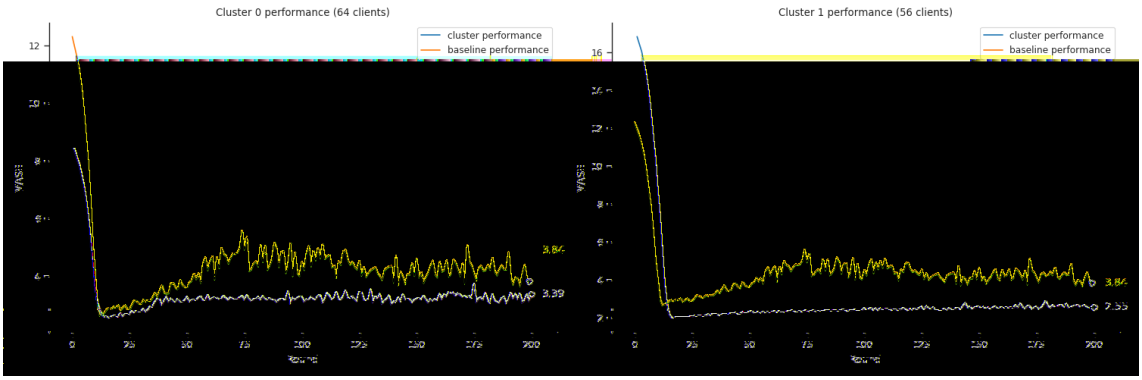


Figure B.0.2: MASE vs communication round for the Ericsson dataset experiment using the `hwl2015` feature extractor and k = 2.

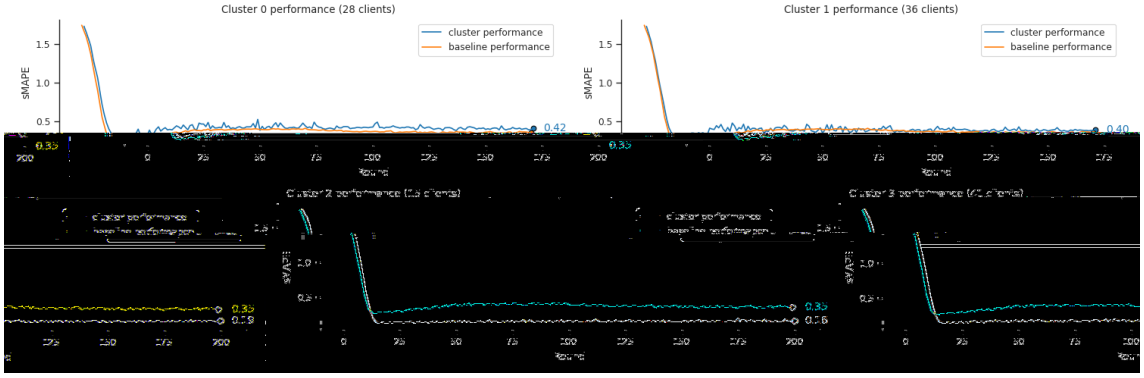


Figure B.0.3: sMAPE vs communication round for the Ericsson dataset experiment using the `hwl2015` feature extractor and k = 4.

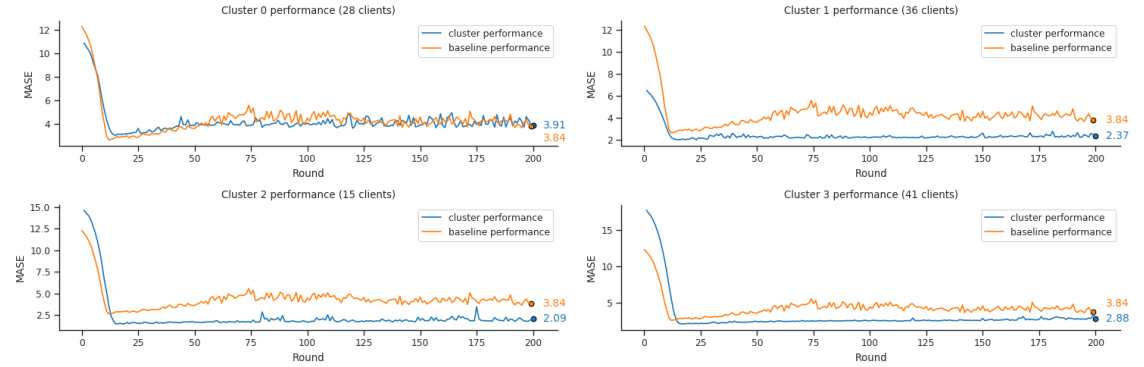


Figure B.0.4: MASE vs communication round for the Ericsson dataset experiment using the `hwl2015` feature extractor and k = 4.

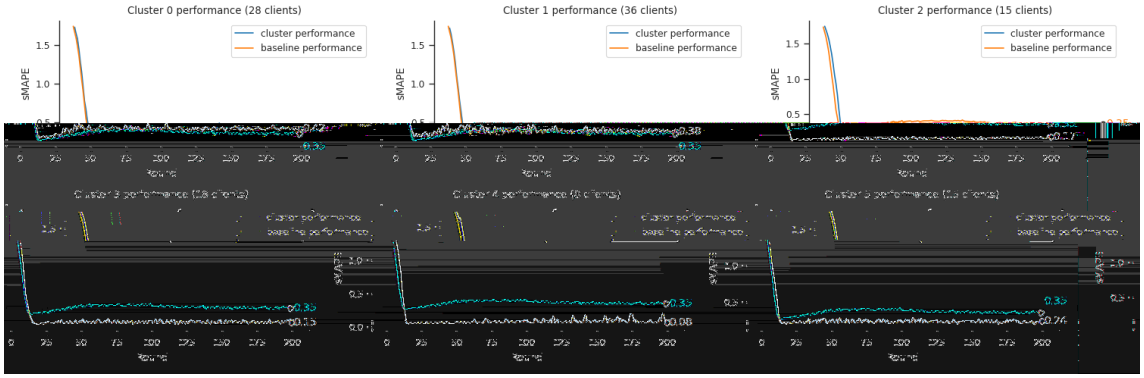


Figure B.0.5: sMAPE vs communication round for the Ericsson dataset experiment using the `hwl2015` feature extractor and k = 6.

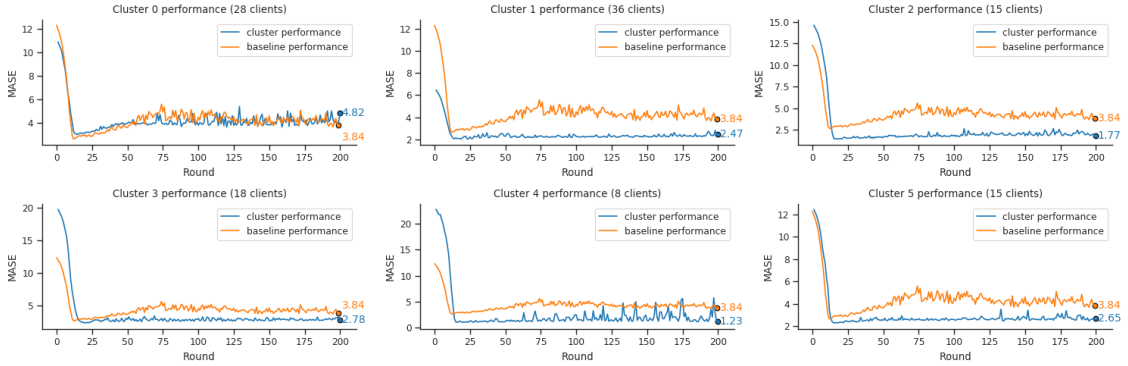


Figure B.0.6: MASE vs communication round for the Ericsson dataset experiment using the `hwl2015` feature extractor and k = 6.

TRITA-EECS-EX-2019:308