UCLouvain

École polytechnique de Louvain

Anomaly detection at the edge

A federated learning approach

Authors: **Colin Evrard, Julien Lienard**
Supervisors: **Prof. Etienne Riviere, Dr. Sarah Klein**
Reader: **Prof. Peter Van Roy**
Academic year 2021–2022
Master [120] in Computer Science

Acknowledgements

First and foremost, we are extremely grateful for the tremendous support of Professor Etienne *Riviere* and Doctor Sarah *Klein* for their support and guidance throughout the whole project. They were a great help in the research and in the development part of this thesis. We also want to thank them for the time they spent with us during the year and for their permanent feedback on the thesis and for their rapid response to our questions.

We want to thank all the people working at *Sirris* that have been involved in this thesis. First, we thank Doctor Sreeraj *Rajendran* for being there to answer our technical questions and for giving us interesting advices. Then, Doctor Nicolás *González-Deleito* for helping understand the problem at the start of our thesis. And Finally, all the people that gave us feedback during the presentation of the thesis at *Sirris*.

We would also want to express our gratitude to Professor Peter *Van Roy* for agreeing to serve as our thesis reader.

Finally, We would like to thank everyone who has assisted us this year, particularly our family and friends. They were really helpful to us while working on our master's thesis.

Colin Evrard would like to thank in particular his parents and brother to help cheer up even during the worst of times. He would also like to thank his friends Antoine, Sandrine, Maxime, Pauline and Maxime for helping him keep his head in the right direction.

Contents

Abstract

In recent years, the development of many IoT applications has risen steadily, leading to many new horizons and possibilities. However, this growth also gave birth to new concerns, among these are privacy concerns [Khan and Salah, 2018]. While ensuring secure and private communication can protect devices from the outside of their networks, some applications such as machine learning still need some form of model centralization and thus puts the data at risk of leaking to actors within the IoT networks. Fortunately, an emerging paradigm, known as federated learning aims to remove the data leak risk from decentralized machine learning [Yang et al., 2019]. Its aim is to make clustering, regression, classification and other machine learning possible without sharing data. This master's thesis explores the task of learning to detect anomalies on a network in a privacy preserving manner. In more details, we intend to tackle the problem of live anomaly detection on a connected fleet of IoT devices' data distribution. The goal is to send an alert to the federated learning network when a new data point from a device does not fit the currently estimated global distribution. Due to the live nature of the problem, we cannot just ignore the temporal factor of the data. More specifically, we are going to use time-series to represent our data and use time series clustering techniques in a federated context. As neural networks tend to lack interpretability, we are going to use more interpretable clustering techniques such as KMeans [Lloyd, 1982] and Agglomerative clustering [Aghabozorgi et al., 2015].

Chapter 1

Problem introduction

In the context of IoT and the generalization of mobile devices, and with the continuous need of creating machine learning models, the need for a technique to create a new model without having to gather the data from the devices is a major challenge. In this chapter, we will introduce the context of our work, what is our goal, how it fits in some paradigms, what challenges we will try to overcome, and which techniques we used to solve those.

1.1 Problem Statement

The problem that is given to us is to find a way to detect anomalies on *edge devices*, which is akin to detecting *out-of-distribution* data points in a network of connected devices. The solutions we will explore will have to be able to tell us when an anomaly happens in the shortest possible time since its appearance, to allow human diagnosis and immediate decision-making. The project for which this problem was created is about detecting anomalies in households energy consumption for energy providers, in order to find emerging patterns and adapt the grid to those. The algorithm should label the data as *normal* (in-distribution) or *anomalous* (out-of-distribution) and should send alerts to an operator that would validate the label and take action accordingly.

Within the scope of this thesis, we will consider the potential adversaries as *honest but curious*. It means that any potential adversary would try to get a hold of private data but would never try to modify it, that would be tackling the problem of *model poisoning*, which is outside the score of this master's thesis. The adversary would only watch the information being transmitted within the network. Thus, our goal would be to reduce to a minimum the amount of private data leaking on the network while still allowing the devices within to collaborate on some form of anomaly detection model.

1.2 Electric consumption

As said before, we will work on data which has the same modality as electric consumption in households in order to detect abnormal patterns in a customer's consumption. The typical modality of such data is *time series*. In our ever more data collecting world, the usage of smart meters becomes more popular to gather all kinds of information, including energy consumption data of households [Zheng et al., 2013]. Electric consumption is something we do not want to leak as it can be used to identify the customer's habits. Indeed, a customer's consumption habits can tell a lot about the customer's behaviour [Steemers and Yun, 2009]. For instance, if a customer is not using electric power, it can mean that the customer is not using the house. The level of electric consumption can also give an idea of how many people are in the house. These pieces of information obviously belong to the *private* sphere, and not only is it ethically questionable to share these, it might also simply be *illegal* in a lot of contexts.

1.3 Anomaly detection

Anomaly detection is the problem of finding a criterion to distinguish between data points that conform to expected behaviours, and those which do not [Chandola et al., 2009]. This last category is often referred to as outliers in the machine learning community. An example of such a distribution can be seen in figure 1.1.

One can distinguish between the following kinds of anomaly detection, this categorization also applies to a myriad of machine learning techniques [Chandola et al., 2009].

1. *Supervised*: When labels for both normal and unexpected data are available.
2. *Semi-supervised*: When the data set only have small amounts of labelled data.
3. *Unsupervised*: When no label is available for any of the data points.

For the scope of our problem, we will assume that labels for normal behaviour will always be available, at least for some part of the data, so it will fall within the semi-supervised category. However, as will be more evident throughout this document, most experiments we have devised do not make use to the full extent of available labels.

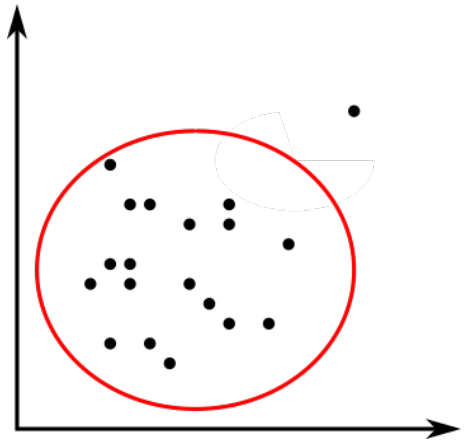

Figure 1.1: *An example of a 2-dimensional distribution of data points with expected behaviour in the lower-left corner and an anomaly on the upper right one.*

1.4 Federated learning and ideal solution sketch

Now that the context has been defined, and the clear need of a machine learning algorithm been stated, we can explain the *federated learning* (FL) paradigm.

With the not-so-new-anymore GPDR and the growing concerns about the influence of big data on the world, it becomes clear that we cannot just centralize all captured data anymore. So, researchers had to find a way to share data useful for building a model without leaking any private information. The solution they came about is named *federated learning*.

The basic premise of FL is to build machine learning models uniquely described by a set of parameters on the edge devices, then to share only these parameters and *aggregate* those in a way that does not allow one to deduce back the original data used for training the models. The aggregation of the parameters should define a *global model* that combines the *knowledge* of all the *local models* learned by the edge devices using their private data.

Now that the paradigm we will be working with has been explained, we can describe what the solution we will arrive to should ideally look like. An intentionally overly generalist overview of this problem is illustrated in figure 1.2, we strongly recommend the reader to consider the illustration as a companion to reading the next paragraphs.

Let us consider a fleet of $\mathcal{D}$ connected IoT devices $d_1, d_2, \cdots, d_{\mathcal{D}}$ collaborating on a global model M. The role of M is to learn the usual distribution of data and

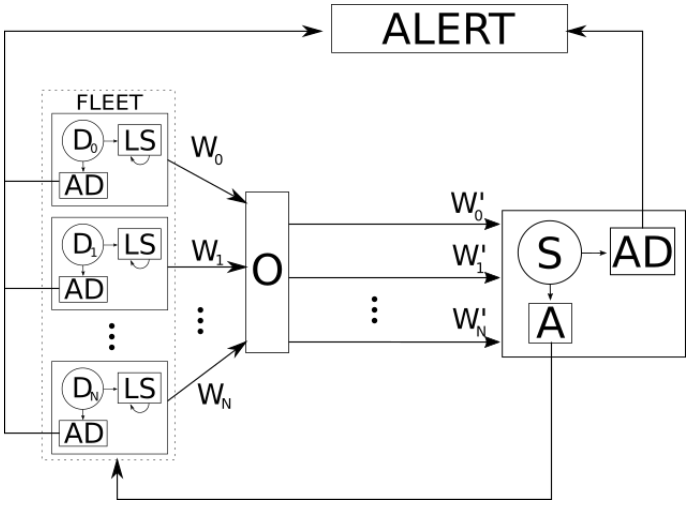

Figure 1.2: *The overview of the problem statement.*

tell whether a new data point p is expected to belong to that distribution or is to be considered unusual, *abnormal*. The federated learning process consists of $\mathcal{R}$ rounds $r_1, r_2, \cdots, r_{\mathcal{R}}$. Each round is initiated by the federated learning server S which collects sets w_i', $i \in \{1, 2, \cdots, N\}$ of non-revealing locally estimated parameters of the model M for each of the $\mathcal{D}$ devices. Each of these parameters set is considered to be the result of a data obfuscation process O on the parameters of the local models such that

$$w_i' = O(w_i) \text{ such that } i \in \{1, 2, \cdots, N\}$$

. Once all the parameters have been received by S, the aggregation scheme A is applied to build the global model parameters. Finally, S updates the parameters of the global model to the ones it just aggregated and sends those to the fleet.

Until now, we have only described the process of collecting the parameters of the local models and aggregating them. However, figure 1.2 also presents the *learning step* (LS) and the *anomaly-detection step* (AD). The learning step is closely related to the type of model in use and the type of data used. The anomaly detection step is the same uses the model just learned by the edge devices to tell whether there are anomalous points within the devices databases. Even if AD is represented the same way both on the clients and server, it is not necessarily the same process in both cases.

1.5 Clustering in federated learning

In order to detect anomalies, we will use a clustering algorithm. We decided to use clustering as it is a very powerful technique that has already been done in the literature [Ahmed et al., 2016a][Münz et al., 2007][Syarif et al., 2012]
[Ahmed et al., 2016b][Issa and Vasarhelyi, 2011]. However, using clustering technique without a neural network in federated learning is a subject that has not been explored as much, and at the time of writing this, there is not really any agreed upon method to perform anomaly detection using *classical* clustering techniques in a federated learning paradigm. There is even fewer literature tackling the specific case of time series data. Indeed, unlike neural networks which are the *de facto* standard in the literature on federated learning [Mansour et al., 2020][Lee et al., 2019][Niknam et al., 2020]
[Ang et al., 2020][Kim et al., 2021][Mukherjee et al., 2019][Briggs et al., 2020]
[Ahmadi et al., 2021], clustering let us have a more human interpretable model. This interpretability is what we are adding to the literature, and it allows us to respond to the given problem as we want a human-readable model. This is especially important as in the cases of critical application, an expert would want to know *why* the model predicts that a data point is abnormal as it could help diagnose quickly potentially large real world problems.

Chapter 2

Background knowledge

In this second chapter, we will introduce the theoretical background necessary to understand the full extent of the solutions that we build. Our solutions are related to several areas of modern computer science, including but not limited to:

1. Federated learning and distributed computing.
2. Time-series clustering.
3. Anomaly detection.

The following sections are devoted to covering, at least partly those background topics.

2.1 Federated Learning

Federated learning (FL) commonly refers to a paradigmatic shift from classical centralized machine learning. Usually, data is collected and brought to the model. In FL, the model is brought to the edge devices, each having its own private data set. Only incremental updates to the model are transmitted from the edges to the rest of the federated network. The strict secrecy of each edge device's data set is usually leveraged to ensure privacy for the users of the edge devices. The collection of the incremental updates by the FL server allows for building a global model which is retransmitted to the edges (see figure 2.1).

As said in the introduction, we decided to use federated learning as it responds to our requirements. Indeed, federated learning lets us create a machine learning model in a decentralized, privacy-preserving way, and it keeps the usage of the bandwidth low.

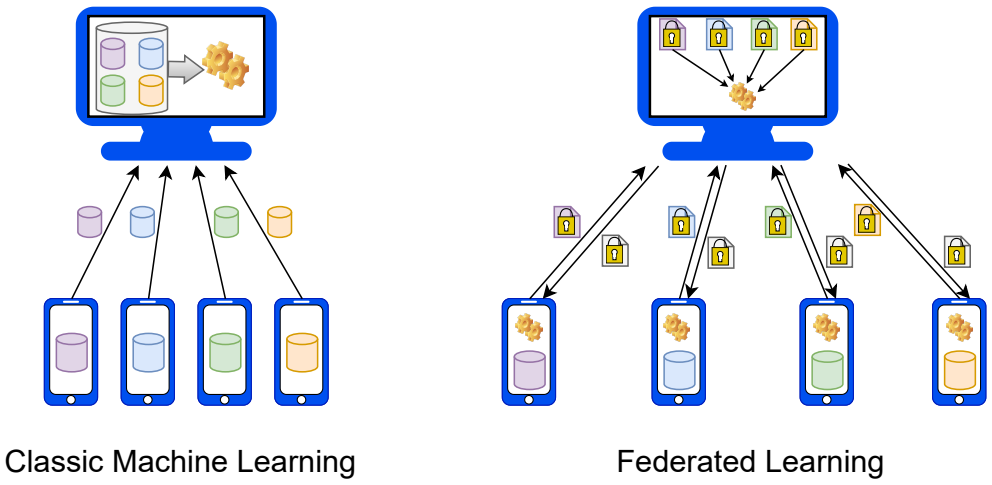

Figure 2.1: Schema of the difference between classic machine learning and the new paradigm of federated learning

2.1.1 An example of FL formalism

The authors of the paper we have picked as a good FL example [Bonawitz et al., 2019], describe a scalable production system for Federated Learning in the domain of mobile devices. What really interests us here is that they also describe the resulting high-level design as well as sketch some challenges and the corresponding solutions. The basic notions associated with it are the following:

1. The participants in the protocol are called *devices*.
2. The devices communicate with the FL server.
3. The work done by the devices is called FL *tasks*.
4. The devices doing the tasks are called the FL *population*.
5. The FL server and FL population set up *rendezvous* points which are called rounds.
6. The computations the FL server tells the FL devices to do are called FL *plan*.

The protocol process consists of a sequence of *rounds*, each consisting of 3 phases.

1. *Selection*: The FL server selects a subset of the devices to perform the tasks.
2. *Configuration*: Based on the server parameters, it sends the devices an FL plan as well as the global model.

3. *Reporting*: The devices report their model updates to the server. The updates should be *aggregated* to the global model by the server in a way that prevents curious but honest attackers from deducing the device's data distribution.

Secure aggregation

This paper also describes a *secure aggregation* [Bonawitz et al., 2017] protocol that is used to aggregate the updates from the devices in a privacy preserving fashion. The presented method is more suited for weights based models.

2.2 Time-series clustering

In order to understand time-series clustering, we will first briefly introduce clustering and come back to it in a later section.

According to the paper *Time-series clustering–a decade review* [Aghabozorgi et al., 2015], clustering is a set of data mining techniques where similar data points are placed into related or homogeneous groups without advanced knowledge of the group's definition. It belongs to the unsupervised category of machine learning algorithms. We used this family of techniques because of its widespread use and fairly understood theory, as well as the potential easy adaptability to a *sem-supervised* context. The most notable example of clustering is Lloyd's algorithm [Lloyd, 1982] implementing the *k-means* method.

2.2.1 Time-series categories

Time-series clustering refers to the application of clustering on time-series data. The review [Aghabozorgi et al., 2015] does the distinction between multiple categories:

1. **Whole time-series clustering.** This consist of multiple time-series clustered as the time is one more dimension (*ex. data from multiple sources*).

2. **Subsequence time-series clustering.** This consist of one long time-series (*ex. data gathered continuously*) that is split according to a sliding window.

3. **Time-point clustering.** It is really close to **Subsequence time-series clustering**, but they are different as **Time point clustering** can consider some points as noise.

We will mainly focus on the second item. As we will run our clustering algorithm on each client which gathers the data continuously, we have to separate the data into different time-series using a sliding window. Using whole time-series is not appropriate as clients have only one continuous time-series. And the last technique

implies the fact that we should have taken the noise into account, but this is outside the scope of this thesis.

2.2.2 Similarity measure

In order to perform clustering on time-series, we have to use a good measure of similarity between time-series. As we can see in the paper from Aghabozorgi *et al.* [Aghabozorgi et al., 2015], there are many methods to compare time series. Each method has its advantages and disadvantages. In this thesis, we will use *dynamic time warping*.

2.3 Dynamic time warping

Dynamic time warping (DTW) is a similarity measure that can capture resemblances between time-series even if they have been stretched or shifted. In this section, we will use the notation from the book *Information Retrieval for Music and Motion* [Müller, 2007].

The goal of this technique is to measure the alignment of two time-series. These will be $X := (x_1, x_2, \ldots, x_N)$ with $N \in \mathbb{N}$ and $Y := (y_1, y_2, \ldots, y_M)$ with $M \in \mathbb{N}$. The feature space, in which all elements of X and Y lie, is formalized as $\mathcal{F}$. We then have $x_n, y_m \in \mathcal{F}$ for $n \in [1 : N]$ and $m \in [1 : M]$.

We introduce a local cost measure

$$c : \mathcal{F} \times \mathcal{F} \to \mathbb{R}_{\geq 0}.$$

Which allows us to compare points in the feature space. The cost c can also be considered as a local distance measure. The locality here refers to the time-step-wise nature of these measures. Figure 2.2 illustrates the alignment of two time-series X and Y.

In order to find the optimal alignment, we need to know the cost associated with each pair of time steps. We will store it in the cost matrix $C \in \mathbb{R}^{N \times M}$, it is defined as:

$$C := c(x_n, y_m)$$

Historically, dynamic time warping (from now on, referred to as DTW) has been used to study speech patterns due to its ability to capture similarities between speeches at different utterance rates. The algorithm we are going to describe is referred to as *classical DTW* in the book this section is based on [Müller, 2007].

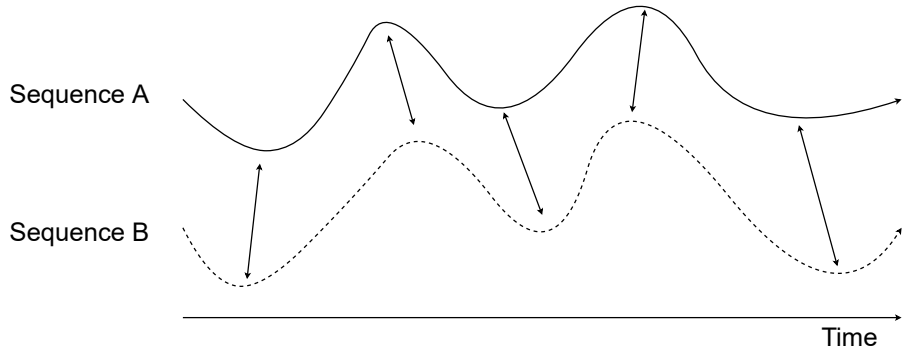

Figure 2.2: *The alignment of the time-series A and B. The arrows between the lines point to the index at which the sequences are aligned. The picture is inspired from the book used in this section [Müller, 2007]*

2.3.1 Classical DTW

The classical DTW algorithm corresponds to the variation of the algorithm that uses dynamic programming in order to reduce the computational cost of DTW. In the book [Müller, 2007], some assumptions are made about the alignments in order to prune the tree of possibilities. Every alignment corresponds to a path. More precisely, these paths are called (N, M)-warping-paths. It is defined as the following:

An (N, M)-warping path is a series $p = (p_1, \ldots, p_L)$ with $p_\ell = (n_\ell, m_\ell) \in [1 : N] \times [1 : M]$ for $\ell \in [1 : L]$ satisfying the following three conditions.

1. *Boundary condition*: $p_1 = (1, 1)$ and $p_L = (N, M)$.
2. *Monotonicity condition*: $n_1 \leq n_2 \leq \ldots \leq n_L$ and $m_1 \leq m_2 \leq \ldots \leq m_L$.
3. *Step size condition*: $p_{\ell+1} - p_\ell \in \{(1, 0), (0, 1), (1, 1)\}$ for $\ell \in [1 : L - 1]$.

More intuitively, the monotonic condition means that the "time" along the path can only go forward. The boundary condition means that the path must start at the beginning of both series and stop at their end. The step size condition means that the path can only go forward in the time of either X or Y by increments of one step at a time. An example of an admissible warping path can be seen in figure 2.3.

Next, we will need to define the cost of a warping path. We will simply take

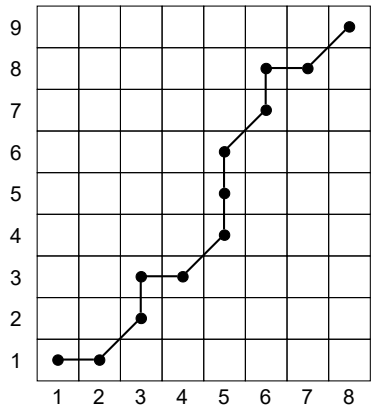

Figure 2.3: *An example of an admissible warping path. The rows and columns correspond to indices of two time-series. The picture was inspired by an illustration in Information Retrieval for Music and Motion [Müller, 2007]*

the sum of all the local costs along the path and denote it by

$$c_p(X,Y) := \sum_{\ell=1}^{L} c\left(x_{n_\ell}, y_{m_\ell}\right)$$

.

Using this definition, we can formalize the optimal warping path. We will denote it by $c_{p^*}(X,Y)$. Then using this definition, we can define the similarity measure:

$$\begin{aligned} \mathrm{DTW}(X,Y) &:= c_{p^*}(X,Y) \\ &= \min\left\{c_p(X,Y) \mid p \text{ is an } (N,M)\text{-warping path }\right\} \end{aligned}$$

We can now start seeing the outline of a dynamic programming solution. To express it, we will need the following notations. We will define sub sequences as $S(i:j)$ of $S = (s_0, s_1, \ldots, s_K)$ as $S(i:j) = (s_i, s_{i+1}, \ldots, s_j)$, where $i \leq j$ and $i, j \leq K$ for $i, j, K \in \mathbb{N}$. Based on this, we can define prefix sequences of S as $S(1:k)$ where $1 \leq k \leq K$ and $k \in \mathbb{N}$.

The next step is to define the matrix $D \in \mathbb{R}^{N \times M}$ as

$$D(n,m) = DTW(X(1:n), Y(1:m))$$

which will be referred to as the *accumulated cost matrix*. In a dynamic programming framework, it would be the table used in a table implementation. Finally, the following propositions from the book [Müller, 2007] gives the Bellman equation of this particular dynamic programming problem.

The accumulated cost matrix D satisfies the following identities: $D(n,1) = \sum_{k=1}^{n} c(x_k, y_1)$ for $n \in [1:N]$, $D(1,m) = \sum_{k=1}^{m} c(x_1, y_k)$ for $m \in [1:M]$, and

$$D(n,m) = \min\{D(n-1,m-1), D(n-1,m), D(n,m-1)\} + c(x_n, y_m)$$

for $1 < n \leq N$ and $1 < m \leq M$. In particular, $\mathrm{DTW}(X,Y) = D(N,M)$ can be computed with $O(NM)$ operations. The algorithm used to compute $DTW(X,Y)$ is thus a classical table dynamic programming.

2.4 DTW barycenters

In order to use DTW similarity measures in a clustering context, we need to have a way to pick a point in time-series space that represents well a set of time-series. This is what is called a *consensus* representation in [Petitjean et al., 2011]. As we will see, finding a consensus representation is not a trivial problem. Fortunately, the authors of a recent paper, *A global averaging method for dynamic time warping, with applications to clustering* [Petitjean et al., 2011] have found a technique called *dynamic time warping barycenter (DBA)* that can be used to find a consensus representation while avoiding most pitfalls. It will be useful to us as it will allow us to re-implement the *time series K-means* algorithm in a way that is adaptable to the FL paradigm.

2.4.1 Motivations

Finding a consensus representation of a set of time-series is at the heart of many algorithms working with time-series. For example, running the K-means algorithm on a set of time-series needs a computationally inexpensive technique that works well with time-series of different lengths and is not too sensitive to shifts and stretches.

A naïve approach would be to use the same technique as in a time-independent context. Usually, the technique used in those contexts would be a simple average of a given set of points. However, if we applied it in a time-dependent context, it would need time-series to be of the same length and would not give a consensus representation that represents well the set of time-series.

Indeed, if we take a set of time-series that are obtained by shifting some original time-series by a random amount picked in a uniform distribution, the time-series obtained by simply averaging all the shifted time-series would be a poor consensus representation of the original set of time-series as it would not take into account the shift and associate points that aren't related in the temporal dimension. This

is just one of the pitfalls of using simple averaging. This very bad case is illustrated and compared to DBA in figure 2.4.

The authors of the paper cited earlier[Petitjean et al., 2011] mention some of the past techniques that have been used to solve this problem as well as their limitations. In particular, they mention that many of the previous solutions, while elegant, were computationally prohibitive.

2.4.2 DBA's inner workings

As the pseudo-code given by the authors is not really self-explanatory, we judged it would be more adapted to explain it briefly using a more natural description.

First, let's introduce the notation. We will work in a space of time-series of length $t \in \mathbb{N}_0$ denoted by E^t. While DBA *does* work well with multivariate time-series, for clarity, we will assume each point of a time-series is just a real value. DBA works with a set of $n \in \mathbb{N}_0$ time-series denoted by $\mathbb{S}$ with no particular constraint on the length of the time-series. It uses DTW's cumulative cost matrix which we will denote by m. The paper [Petitjean et al., 2011] uses the term *coordinate* to denote a particular time step of a time-series.

DBA works by iteratively refining a starting barycenter. A common approach is to start with a time-series from $\mathbb{S}$. The current barycenter is B and for each one of its coordinates, it will save a set of associated coordinates from the coordinates of the time-series in $\mathbb{S}$. Then, at each iteration, we do the following:

1. For each time-series in $\mathbb{S}$, compute the distance between the time-series and the current barycenter $B \in E^t$. Save the warping paths for the next step.

2. For each time-series $S_i \in \mathbb{S}$, follow the warping path between B and S_i and add each traversed coordinate of S_i to the corresponding set of associated coordinates of B.

3. Compute the new barycenter $B' \in E^t$ by computing the average of the associated coordinates of B.

The paper [Petitjean et al., 2011] mention that the convergence of the algorithm is guaranteed as it relies on an inertia parameter α that is always decreasing.

Finally, the authors mention that the algorithm is not that expensive, since DTW has a complexity for time-series of maximal length T of $\Theta(T^2)$, if DBA performs I iterations to find the barycenter of N time-series, it will have a complexity of $\Theta(I \cdot N \cdot T^2)$.

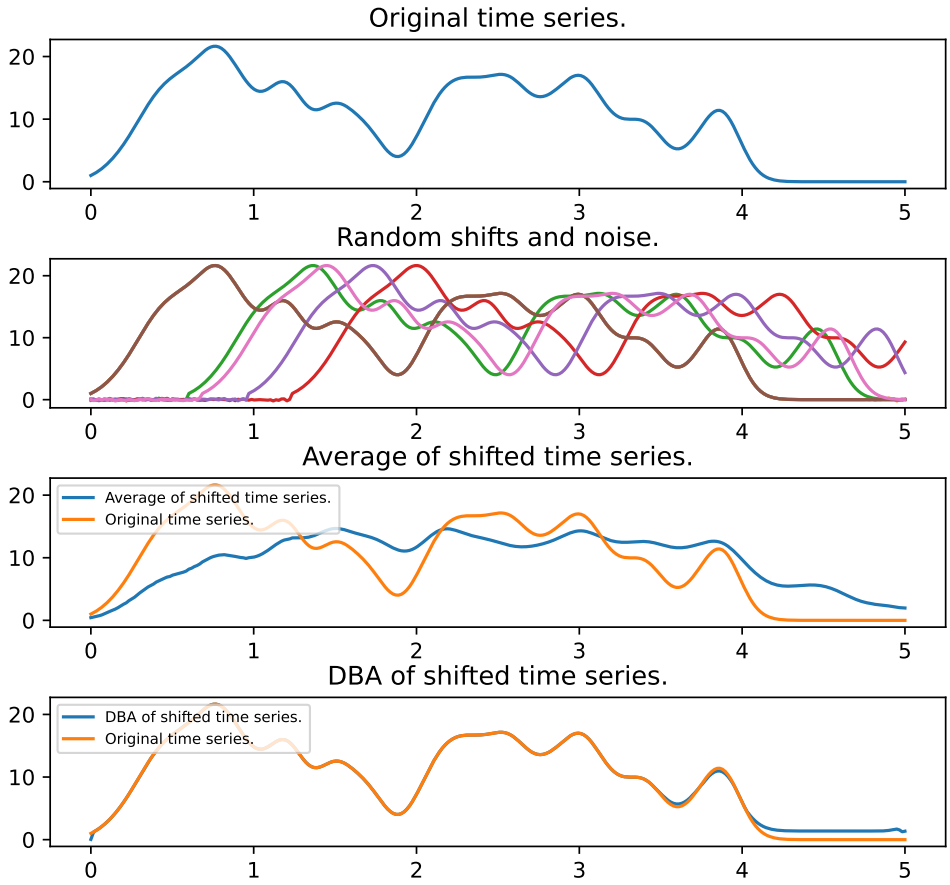

Figure 2.4: *From top to bottom: a synthetic time-series; a set of time-series that are the same as above but with a random shift and some noise; the consensus representation obtained by averaging all the shifted time-series; the consensus representation obtained by applying DBA. We can see that the consensus representation obtained by averaging all the shifted time-series is not a good consensus representation of the original set of time-series. On the contrary, the consensus representation obtained by applying DBA very closely resembles the original one slightly shifted.*

2.5 The K-means algorithm

K-Means of one of the first and of the most known clustering method that has been used for many years now and that has proved its efficiency. Though it has been discovered by multiple people, its discovery is generally attributed to Lloyd during the 1950s [Lloyd, 1982]. His specific version of this algorithm is the one we are going to cover. *K-Means* is the base of many other clustering algorithms such as *K-Medoids* that we will cover later. As we will explain in the *K-Medoids* section (section 2.6), *K-Medoids* is the clustering algorithm that we will use for some of our solutions.

2.5.1 Formalism

The goal of the K-means algorithm is to find a set of $K \in \mathbb{N}$ points in a $D \in \mathbb{N}$ dimensional feature space. Given a dataset of $N \in \mathbb{N}$ points, we want these K points to represent well a group of points in our dataset. Those K points can be arbitrary values within the same space as the data set points. In other words, we want to find a K-partition of the dataset.

The K points mentioned above are called the *centroids* of the K partitions. In order to evaluate the quality of the partition, we can measure the distance from the centroids to the points within their partition. To do so, we need to define a distance function between two points of the feature space. The most basic distance function we can use with K-means is the euclidean distance. Let us define d as the euclidean distance between x and y and $x, y \in \mathbb{R}^D$.

$$d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R} : d(x, y) = \sqrt{\sum_{i=1}^{D} (x_i - y_i)^2}$$

We formalize the dataset S and the centroids C as follows:

$$S = \{x_1, x_2, \ldots, x_N \mid x_i \in \mathbb{R}^D \text{ for } i \in \{1, \ldots, N\}\}$$

$$C = \{c_1, c_2, \ldots, c_K \mid c_i \in \mathbb{R}^D \text{ for } i \in \{1, \ldots, K\}\}$$

For each of the N points of the dataset $(x_0, x_1, \ldots, x_N)$, we want to know to which of the K centroids $(c_0, c_1, \ldots, c_K)$ it is associated.
To do so, we define the association matrix A such that $A \in \mathbb{N}^{N \times K}$ and

$$\sum_{k=1}^{K} a_{i,k} = 1 \quad \text{for } i \in \{1, \ldots, N\}$$

Using this formalism, we define the objective function of the K-means algorithm.

$$SSE(C, S) = \sum_{i=1}^{N} \sum_{k=1}^{K} a_{i,k} d^2(x_i, c_k)$$

If we define the error to be the sum of the distances between each dataset point and their associated centroid, we can think of the above objective function as a sum of squared errors, which is why we will designate it as SSE from now on.

We can now express the k-means algorithm as solving a minimization problem. Given S the set of data points, find C^*, a set of centroids such that $SSE(C^*, S)$ is minimal.

$$SSE(C^*, S) = \min_{\text{all possible C}} SSE(C, S)$$

2.5.2 K-means's steps

We will now describe the steps of the most common implementation of the K-means algorithm. K-means can be described as a three steps algorithm.

1. Centroids initialization.
2. Data association.
3. Mass center computation.

Centroids initialization

The first step of the K-means algorithm is to initialize the centroids. The naïve approach is to initialize them at random. However, this is not a good idea as we can stumble upon the case of *lost centroids*. This happens when the centroids are too far from the data points, no point is associated with the centroid, and the centroid is never updated. We thus lose some of the partitioning quality.

In the general case, many issues can arise while deciding on the initial centroids, it has thus been the subject of extensive research. The most common approach is to use the K-means++ [Arthur and Vassilvitskii, 2006] algorithm. However, many other methods exist [Celebi et al., 2013]. The one we ended up using in our implementation was loosely based on K-means++ but was more of a place holder since we wanted to spend a maximum amount of time on the FL aspects of our techniques.

Data association

This step is rather simple. We associate each data point to the closest centroid. Given the set of centroids at iteration t, C_t, we redefine the association matrix as such.

$$a_{ik_t} = \begin{cases} 1 \text{ if } d(x_i, c_{k_t}) = \min_{j=1 \to K} d(x_i, c_{j_t}) \\ 0 \text{ otherwise.} \end{cases}$$

$$\text{Where } i \in \{1, \ldots, N\} \text{ and } k \in \{1, \ldots, K\}.$$

Mass center computation

This step move the centroids closer to their newly assigned data points. We define the values of the centroids for the next iteration $t + 1$ as such.

$$c_{k_{t+1}} = \frac{\sum_{i=0}^{N} a_{ik_t} x_i}{\sum_{i=0}^{N} a_{ik_t}}$$

Which is akin to computing mass centers with equal weights for every x_i associated with some centroid index k.

2.5.3 Algorithm

These steps give way to a rather natural way of laying out the algorithm. The pseudo-code can be found in algorithm 1. First, we initialize the centroids (`line 4`), and then we alternate between the data association (`line 8`) and the mass center computation (`line 10`) steps until some stopping condition is met (`line 7`). That stopping condition is usually when the centroids do not move anymore or when some arbitrary maximum number of iterations is reached. The corresponding pseudo-code is given in Algorithm 1.

2.6 The K-Medoids Algorithm

The *K-Medoids* algorithm is a variant of the *K-means* algorithm. In this section, we will explain the first algorithm of the *K-Medoids* algorithm and the principle behind it. To do so, we will use the book *Finding Groups in Data: An Introduction to Cluster Analysis* [Kaufman and Rousseeuw, 2009]. We decided to use the *K-Medoids* algorithm because it is a variant of the *K-means* algorithm that is less impacted by outliers. Using *K-Means*, if we have an outlier, the center will be shifted in its direction and may misrepresent the cluster. *K-Medoids* cannot shift its center in a direction, this phenomenon is illustrated in the next section. *K-Medoids* will be used to generate the clusters on the client-side.

Algorithm 1: Centralized clustering using k-means

Data: data, nclusters, maxiters, tol
Result: Set of centroids that minimize SSE

```
N = number of data points in data;
K = nclusters;
C_0 = zeros(K, D);
C = InitializeCentroids(data, K);
A = diag(1, N, K);
i = 0;
while ||C − C_0|| > tol ∧ i ≤ maxiters do
    A = ComputeAssociation(data, C);
    C_0 = C;
    C = ComputeCentroids(data, A);
    i = i + 1;
end
return C
```

2.6.1 Formalism

The *K-Medoids* algorithm uses *Medoids* as centers of the clusters. If we want K clusters in the data set S, we will select K medoids m from S such that $m_1, \ldots, m_K \in S$. The partitioning around medoids (PAM) algorithm consists of two steps.

1. The **build** phase.
2. The **swap** phase.

But before starting those two phases we need to compute the dissimilarities between all data points.

Build Phase

As K-means, the PAM algorithm takes k as a parameter for the number of clusters. To select the first medoids, we will take the data point for which the sum of dissimilarities with all other data points is minimal. This is the most centered data point. Then, at each step, until we have k medoids, we will select the next medoids as follows:

1. Select an item i not yet selected

2. Select another item j also not yet selected. Compute the difference between the dissimilarity D_j with the most similar previously selected item and the dissimilarity $d(j, i)$.

3. if $D_j \geq d(j, i)$ j will be useful to select i. So we use the formula:

$$C_{ji} = \max(D_j - d(j, i), 0)$$

.

4. We compute the gain if we select i:

$$\sum_j C_{ji}$$

5. We select the not yet selected data point i that maximizes the gain:

$$maximizes_i \sum_j C_{ji}$$

Swap Phase

The swap phase tries to improve the set of medoids. The algorithm will consider every pair (i, h) where i has been selected and h not. For each pair, we will look if using k instead of i improves the set of medoids. If it does, we will swap i and h. To compute the effect of the swap we will do as follows:

1. Chose a non-selected data j and its contribution C_{jih}:

   (a) If j is closer from any other medoid than i or h,

   $$C_{jih} = 0.$$

   (b) If j is closer from i,

   $$d(j, i) = D_j$$

   . Thus we have two possible situations:

   i. j is closer from h than any other medoid. So the contribution is:

   $$C_{jih} = d(j, h) - d(j, i).$$

   ii. j is closer to than any other medoid. The contribution is:

   $$C_{jih} = E_j - D_j.$$

   Where E_j is the dissimilarity between j and the second closest medoid.

In the second case, the contribution may be negative implicating that the swap is not beneficial for j.

(c) j is closer to any other medoid but closer to h,

$$C_{jih} = d(j,h) - D_j.$$

2. Get the total result of the swap:

$$T_{ih} = \sum_j C_{jih}$$

3. Select the pair (i, h) that minimize the total result:

$$minimizes_{i,h} T_{ih}$$

4. If the minimal T_{ih} is negative, we will swap i and h, and we go to step 1. Then, the algorithm stops.

2.6.2 Improvement

This algorithm has been shown to be very inefficient in practice for large data sets. In the same book [Kaufman and Rousseeuw, 2009], the authors proposed an alternative called CLARA. Instead of using all data points, CLARA will use a sample of those data and applies PAM to it.

Many other papers have tried different techniques to improve the method.

We can cite for example *A simple and fast algorithm for K-medoids clustering* [Park and Jun, 2009] that uses a similar algorithm to k-means. This technique only needs the distances between data only once, reducing the computation time.

The paper *A genetic k-medoids clustering algorithm* [Sheng and Liu, 2006] uses a genetic algorithm to find the best medoids. Genetic k-medoids clustering algorithm (GMCA) randomly selects a set of medoids in a population P and then tries to improve it by swapping medoids with other data points. Then it creates a new generation of size P and repeats the process until a stopping criterion is met.

2.7 The Agglomerative Clustering Algorithm

Agglomerative clustering is a hierarchical clustering algorithm. It is based on the idea of grouping the different clusters in a hierarchical way. To describe this algorithm we will use the book Introduction to HPC with MPI for Data Science by [Nielsen, 2016]. *Agglomerative Clustering* will be used to see how many clusters will be generated at a certain step of the federated learning algorithm. As the algorithm is based on the idea of grouping the different clusters in a hierarchical way, we can use a threshold to define at which point we want a value to be considered as different from existing patterns.

2.7.1 Formalism

Starting with each data point as leaves (interpreted as singleton clusters), hierarchical clustering involves merging the clusters two by two, to the closest clusters until we reach the root of the tree, which contains all the data points. The linkage distance, denoted by $\delta(X_i, X_j)$, is the distance between any two subsets of the data. Agglomerative Clustering will start at the leaves, storing singletons (x_i's), and merge subsets iteratively until we reach the root. We can represent this merging process as a dendrogram or using a Venn diagram (Fig 2.5).

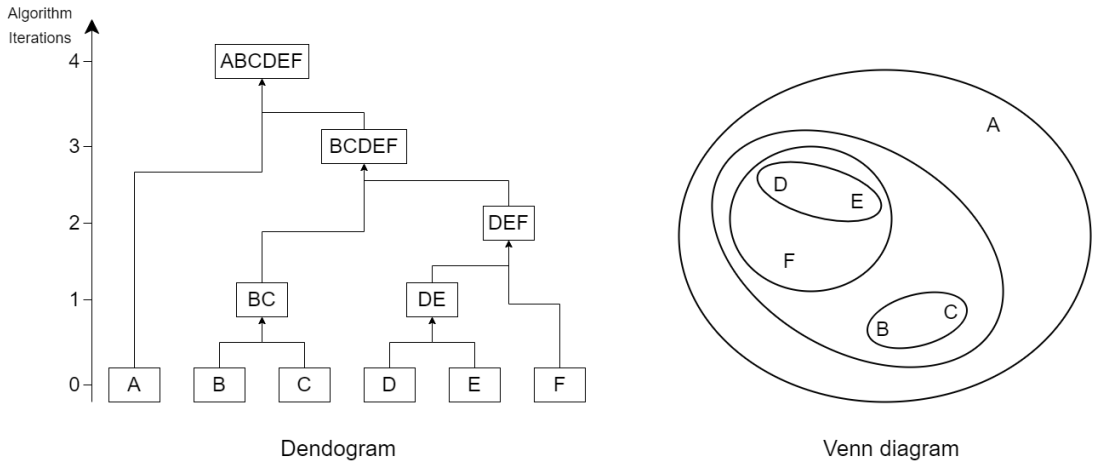

Figure 2.5: Example of a Dendrogram and a Venn diagram of the agglomerative clustering algorithm

The elementary distance between any two clusters of X is denoted by $D(x_i, x_j)$ (in this thesis we will mainly use DTW 2.3). We need to define a subset of distance $\Delta(x_i, x_j)$ between any two subsets of elements in order to select the closest par of subsets at each stage of the algorithm. When both subsets are singletons, we

should have $\Delta(X_i, X_j) = D(x_i, x_j)$ for $X_i = x_i$ and $X_j = x_j$. The following are the most common linkage mehods (Fig 2.6):

- Single linkage:
$$\Delta(X_i, X_j) = \min_{x_i \in X_i, x_j \in X_j} D(x_i, x_j)$$
- Complete linkage:
$$\Delta(X_i, X_j) = \max_{x_i \in X_i, x_j \in X_j} D(x_i, x_j)$$
- Average linkage:
$$\Delta(X_i, X_j) = \frac{1}{|X_i||X_j|} \sum_{x_i \in X_i} \sum_{x_j \in X_j} D(x_i, x_j)$$

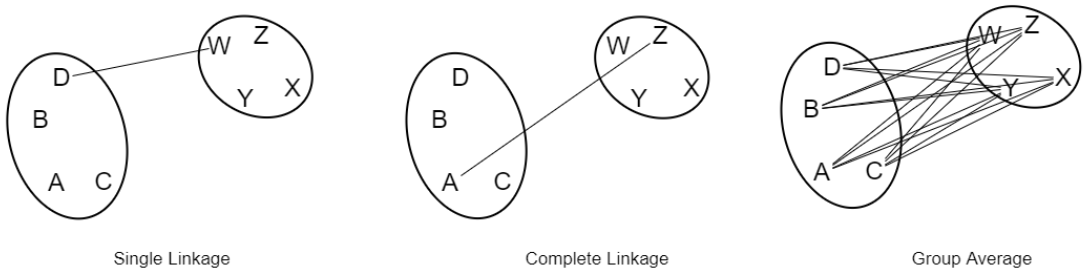

Figure 2.6: Example of the different linkage distances

2.7.2 Algorithm

The algorithm of the agglomerative clustering is given by Algorithm 2. It is quite straightforward. We start with a set of singletons, and merge them iteratively by their distance, choosing the closest pair of clusters. We stop when we reach a single cluster. This algorithm will perform exactly $n - 1$ merge operations because we start with $n = |X|$ singletons and end with a root containing the entire set X.

From this first algorithm, we can implement another algorithm that will stop the agglomeration when we reach a certain distance threshold. The following algorithm is given by Algorithm 3. The algorithm will stop at this threshold and return the resulting clusters.

Algorithm 2: Agglomerative clustering

Data: data, Δ
Result: DendogramRoot
Clusters $= \emptyset$;
forall $x_i \in data$ **do**
  $G_i = \{x_i\}$;
  $Clusters = Clusters \cup G_i$;
end
while $size(Clusters) > 1$ **do**
  $G_i, G_j = \min_{G_i, G_j \in Clusters} \Delta(G_i, G_j)$;
  $G_{i,j} = G_i \cup G_j$;
  $Clusters = Clusters \cup G_{i,j}$;
  $Clusters = Clusters \setminus \{G_i, G_j\}$;
end
return *Clusters*

2.8 Kernel density estimation

The family of *kernel density estimation* (*KDE*) techniques is a set of techniques to estimate the probability density of a multivariate random variable. It is a generalization of the histogram technique and has been around for quite some time (1960s) as a staple of statistics [Chen, 2017]. We will use *KDE* at the edge to generate new data based on the probability density function (pdf) estimated from the locally gathered data. This generated data will have the same distribution as the original data, but we cannot know if the data synthetic or not. This will help us control how much information the edge devices will reveal to the rest of the FL network.

2.8.1 Formulation

According to [Chen, 2017], KDE can be formulated in the following way:

Let $\mathbf{X} \sim P$ be a random variable with a domain $D \subseteq \mathbb{R}^d$. If we know $X_1, \ldots, X_n \in D$, i.i.d to be randomly sampled from the same distribution P with density function p, we can express KDE as

$$\widehat{p}_n(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right)$$

For any $x \in D$.

Algorithm 3: Agglomerative clustering

Data: data, Δ, threshold
Result: DendogramRoots

```
Clusters = ∅;
forall x_i ∈ data do
    G_i = {x_i};
    Clusters = Clusters ∪ G_i;
end
while size(Clusters) > 1 do
    G_i, G_j = min_{G_i,G_j ∈ Clusters} Δ(G_i, G_j);
    if Δ(G_i, G_j) > threshold then
        break;
    end
    G_{i,j} = G_i ∪ G_j;
    Clusters = Clusters ∪ G_{i,j};
    Clusters = Clusters \ {G_i, G_j};
end
return Clusters
```

$K : \mathbb{R}^d \mapsto \mathbb{R}$ is a smooth function called the kernel function, $h > 0$ is called the *bandwidth* and controls the amount of smoothing applied by K.

2.8.2 Familiarization

Intuitively, performing KDE of a random sample can be thought of as picking a value $x \in D$ of which we want to know the probability. We will then let every sample *contribute* to the probability density function (pdf) estimation of x, depending on how far they are from x.

The bandwidth h will control how much the contribution of each sample will decrease with the distance to x. The shape of K will control the global form of the KDE function. An illustration of this process is shown in figure 2.7.

2.8.3 KDE in practice

The earlier formulation of KDE is very practical and gives us a way to very quickly implement KDE, we will thus not detail a pseudo-code. Instead, we will explain some optimizations and the sampling method used in *scikit-learn* [Pedregosa et al., 2011]. One of the optimizations [Pedregosa et al., 2011] used is to store the samples in a tree structure. This allows to perform the KDE on a subset of the samples, and

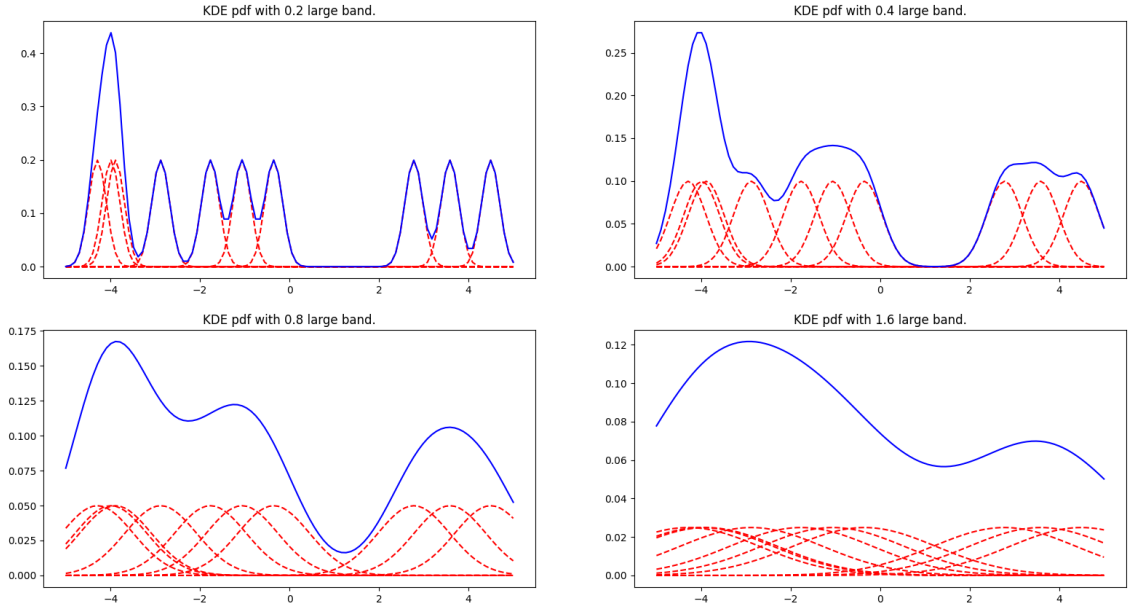

Figure 2.7: *Illustration of the KDE construction with a gaussian kernel under various band parameters. For each of the plots, the same sample of points has been used. In red dotted lines are the contributions of each sample and in blue is the estimated pdf. We can see that the greater the bandwidth, the smoother the pdf estimation.*

thus reduces the number of samples to be processed. Indeed, a lot of kernels are expensive and most don't span the whole real space thus a lot of samples can be skipped in processing. Using a larger bandwidth, however, can lead to larger computational costs due to kernel spanning more of the real space.

The sampling method used in *scikit-learn* is more computationally efficient than an intuitive rejection method. It can be described as follows:

1. Pick a random point from the dataset.
2. Pick a random point in the domain of the kernel function around the picked point.

One of the main drawbacks is that this sampling method stays relatively close to a point of the dataset. Another drawback is that this method is only available for the *gaussian* and *tophat* kernels at the moment of the writing.

2.9 Semi-Supervised Detection of Outliers

Amongst the more advanced anomaly techniques presented towards the end of the *experiments and results*, one of those uses an anomaly scoring mechanism method at its core. This scoring is the one from the *semi-supervised detection of outliers* (SSDO) paper [Vercruyssen et al., 2018]. We used it extensively because of its close relationship with centroid-based clustering which we will detail here.

2.9.1 Formalism

The authors of [Vercruyssen et al., 2018] introduce several hypotheses and notations. Let $\mathcal{F} = \{f_1, \ldots, f_n\}$ be a data set of feature vectors where $f_i \in \mathbb{R}^N$ for $0 \leq i \leq n$. We assume we know a clustering partition of $\mathcal{F}$ with $k \in \mathbb{N}$ clusters, each with their own centroid $\mathcal{C} = \{c_1, \ldots, c_k\}$ where $c_i \in \mathbb{R}^N$ for $0 \leq i \leq k$. We then define the following:

$$C : \mathcal{F} \to \mathcal{F} : C(f) = \{x \text{ such that } c(x) = c(f)\}$$
$$c : \mathcal{F} \to \mathcal{C} : c(f) = c_i$$
$$d : \mathcal{F} \times \mathcal{F} \to \mathbb{R}$$

Where c_i is the centroid of the cluster to which f belongs and d is a distance measure. Now that we have laid the base of the formalism we can build up to the actual scoring method of SSDO.

2.9.2 SSDO Scoring

Point deviation Is related to how much a point deviates from its cluster.

$$point\_dev(f) = \begin{cases} \frac{d(f,c(f))}{\max_{f_i \in C(f)} d(f_i,c(f))} & \text{if } |C(f)| > 1 \\ 1 & \text{otherwise} \end{cases}$$

Cluster deviation Measures how much a given cluster centroid $c(f)$ deviates from the other cluster's centroids.

$$cluster\_dev(f) = \begin{cases} \frac{\min_{1 \leq i \leq k \wedge c(f) \neq c_i} d(c(f),c(i))}{\max_{1 \leq i,j \leq k} d(c(i),c(j))} & \text{if } k > 1 \\ 1 & \text{otherwise} \end{cases}$$

Cluster size Measures how big the cluster of a data point is when compared to other clusters:

$$\text{cluster\_size}(f) = \frac{|C(f)|}{\max_{1 \leq i \leq k} |C(i)|}$$

The squashing function Takes a real γ as a parameter and *squashes* its other parameter between 0 and 1.

$$g : \mathbb{R} \times \mathbb{R} \to \mathbb{R} : g(x, \gamma) = 2^{-\frac{x^2}{\gamma^2}}$$

This lead to the definition of a function:

$$score : \mathcal{F} \to \mathbb{R} : score(f) = 1 - g\left(\frac{point\_dev(f) \times cluster\_dev(f)}{cluster\_size(f)}; \gamma\right)$$

To score a data point f. In practice, γ is chosen as a parameter of *score*. It is chosen such that for a pre-determined proportion p of anomalous data points, *score* will be greater than 0.5 for only those points.

Chapter 3

Previous work

Many techniques already exist to detect anomalies. This subject has been studied in many research papers. In this section, we will try to give you a brief overview of some of the techniques that have been used over the years. And we will also try to present some algorithms that are already used for our problem and some example of what our algorithms could have been used for.

3.1 Anomaly in power consumption

Detection of anomalies in power consumption has already been studied in some papers. Detecting such anomalies can be really useful for many applications. We will explain two of those in this part.

3.1.1 Static model

An approach that is close to the one that we have used is described in the paper [Liu et al., 2021]. The idea is to use a 3-step approach.

1. First, they prepare the data by normalizing it. They used normalization to extract the temporal modification rather than the magnitude of those modifications. They also filled the gaps in the data with an averaging method.

2. The second step consists of discovering the anomalies in the data. They used a similar method to the one that we used. They have a two-step clustering technique. The first clustering is done with DBSCAN. This first step tries to extract the outliers in the data. The second step is utilizing K-Means to group similar patterns and detect various groupings.

3. Finally, they used a classification tree to discover knowledge in the data. In this paper, this tree is used to detect anomalies in building electric usage.

In conclusion, the author of this paper was able to detect anomalies that are learned in a learning step. One of the main problems is that it cannot learn from emerging patterns, only from already seen patterns.

3.1.2 Dynamic model

A second approach to this problem is explained in a recent paper
[Chou and Telaga, 2014]. This paper tries multiple algorithms to detect anomalies in real-time. They tried different methods:

K-Means Simple K-Means, one of the most popular

Artificial Neural Network (ANN) Simple neural network using only feed forward perceptrons.

Auto-regressive integrated moving average (ARIMA) A statistical model for analyzing and predicting data from a temporal or chronological series. Based on the values viewed previously, it determines the values that will be integrated into this one. In a particular context, an autoregressive system proposes that prediction functions are acceptable while limiting the risk of inaccuracy inherent in such a task.

Neural network autoregressive These neural networks are quite similar to recurrent neural networks, which are well-known. They are feed-forward neural networks, which take one additional input to represent the neural network's value for the previous value.

They discovered that combining ARIMA and ANN allowed them to accurately anticipate the building's usual consumption over the next four weeks with an accuracy of 89 to 96 percent. To detect anomalies they simply compute the difference between the real-time consumption and the predicted consumption. If the difference is greater than some threshold, then the consumption is considered to be abnormal.

3.2 Anomaly detection using clustering

Different clustering techniques have been used for anomaly detection. This section will be dedicated to giving the reader a brief overview of some of these used in different situations.

3.2.1 Network anomalies

A survey on network anomaly detection [Ahmed et al., 2016a] describes three main assumptions (see Figure 3.1) when dealing with network anomaly detection using clustering techniques:

1. Data that does not fit well in existing clusters of normal data are considered anomalies.

2. When they are normal and abnormal data in the same cluster, the normal data are closer to the centroid and the abnormal data are further from the centroid.

3. If clusters vary in size, the smaller and sparser clusters can be deemed aberrant, while the thicker clusters can be considered normal.

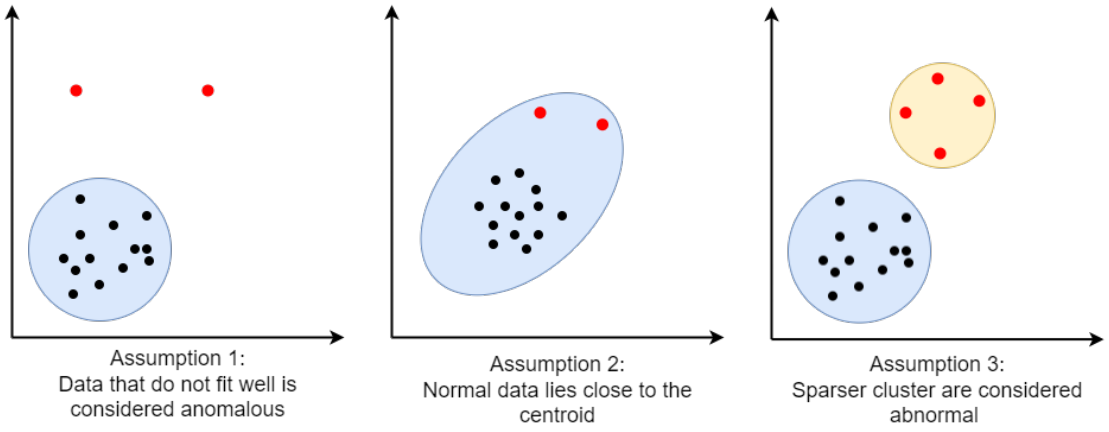

Figure 3.1: Visualization of the three assumptions for network anomaly detection using clustering techniques. Red dots are data that will be considered anomalies.

In anomaly detection, *K-Means* is also a popular clustering algorithm. A paper [Münz et al., 2007] used it to detect network anomalies. The main idea is to build a model with a training set containing normal and abnormal data. Then, the algorithm will create clusters around them, thus having normal clusters and abnormal clusters. Using this model, they use two different distance-based methods and a combination of the two.

- The first method is **Classification**. It is the most straightforward, we compute the distance between the closest normal cluster and the closest abnormal cluster. The point is then labelled according to the closest one.

- The second one is the **Outlier detection**. If the distance to the closest normal cluster is greater than a predefined threshold, then the point is considered an anomaly (as in assumption 1).

- Finally, the combination of the **Classification** and **Outlier detection**. To detect the anomaly, we use both techniques. We compute the distance between the closest normal cluster and the closest abnormal cluster. If the distance with the abnormal is greater, the point is considered an anomaly. Then, if the distance with the normal is greater than the threshold, the point is considered an anomaly. And, if the distance with the normal is less than the threshold and the distance with the abnormal cluster, the point is considered normal.

Unsupervised clustering approach for network anomaly detection
[Syarif et al., 2012] compares the performance of five different clustering algorithms:

- K-Means
- improved K-Means
- K-Medoids
- EM clustering
- Distance-based outlier detection

They tested the algorithms on 22 different types of intrusion with 10-fold cross-validation. We referenced their results in table 3.2.1 to give you a brief overview of the results. We can see that they have good performance with the fifth algorithm. But the authors also explain that they have approximately 20% of false positives.

Algorithm	**Accuracy**
K-Means	57.81%
improved K-Means	65.40%
K-Medoids	76.71%
EM clustering	78.06%
Distance-based outlier detection	80.15%

Table 3.1: Results of the clustering algorithms for network anomaly detection. The algorithms are tested on 22 different types of intrusions with 10-fold cross-validation.

3.2.2 Semi-supervised anomaly detection using time series clustering

In the paper by Vercruyssen *et al.* [Vercruyssen et al., 2018], researchers have devised a way to detect anomalies in time series data representing water consumption. for the *Colruyt* group. They used a semi-supervised approach based on the *COP-K-Means* [Wagstaff et al., 2001] algorithm. As a quick explanation, *COP-K-Means* is used to integrate expert knowledge in the *K-Means* algorithm in the form of *must* and *must not* be in the same cluster constraints. The main part that sets this paper apart is the design of a new anomaly scoring method: *SSDO*. Previous works had already proposed scoring methods that were based on some distance measure between the points and the clusters centers, but these required more parameters or could not use the labelling of the data. As the methodology to obtain the *SSDO* score is already described in section the background section of this document, the details needed to compute it won't be repeated here. Instead, we will focus on how it compares with previous solutions. The authors of [Vercruyssen et al., 2018] compare their solutions against the following.

1. *kNNo* An unsupervised distance-based outlier detection technique.
2. *LOF* A density-based unsupervised outlier detection technique.
3. *CBLOF* Is a cluster-unsupervised outlier detection technique.
4. *SSAD* Is a semi-supervised anomaly detection technique based on support vector machines (SVM).

With their results, amongst all these methods the researchers show that only *SSAD* is competitive with *SSDO*. All the other mentioned methods end up being beaten by *SSDO* by a significant margin.

3.3 Federated Learning

In a survey on federated learning [Zhang et al., 2021], we can see that federated learning has been used in many applications. We will summarize some of those in this section to give you a brief overview of what we have seen.

3.3.1 Personalization

A paper from 2020 [Mansour et al., 2020] describes how they used federated learning to predict the next word written on a mobile keyboard by gathering data from client devices but preserving confidentiality. To do so, they created three different algorithms:

User clustering* or *hypothesis-based clustering They group users in different clusters and then create a model for each group.

Data interpolation They gather data and then train a model with all the data.

Model interpolation They create a local model for each user and a global model. They then use the combination of those models to create a new model.

To demonstrate their algorithms, they used synthetic data with a different number of clients and a different number of data per client. They also used the well known **EMNIST** dataset to train their model and have pretty good results with all the three proposed algorithms.

3.3.2 Cancer detection

Lee et al. [Lee et al., 2019] used federated learning on data that are considered really sensitive, medical data. They used an interconnected medical system (APOLLO) to gather data in a federated way, thus preserving the privacy of the data. Using the gathered data set, they created a model to assist the doctors in the diagnosis of cancer. For now, given the fact that many algorithms and techniques are used in the medical field, the tools are only used as a help for the doctors and never used by themselves.

3.3.3 Wireless machine learning

Traditional models-based approaches are no longer appropriate with the growing number of wireless devices in the market, such as smartphones or tablets. In the context of the growing usage of 5G networks, transferring all the data from devices to the cloud is a very expensive operation. Therefore, the concept of federated learning appears to be evident for the lighter use of data transfer.

In a paper from 2020 by Nikmnam *et al.* [Niknam et al., 2020], they analyzed the importance of federated learning in such networks and drive simulations to prove the efficiency of federated learning, but using a wireless network as a learning environment brings new challenges.

One of these challenges is to eliminate noise in the data. This problem has already been studied for traditional machine learning models, but for federated learning, this challenge is new. *Robust Federated Learning With Noisy Communication* [Ang et al., 2020] proposes a robust way of performing federated learning, thus reducing as much as possible the effect of the noise. They tested the robustness of their approach in 3 ways:

- **Robust Design Under the Expectation-Based Model**: They regularized the loss function to handle the statistical property of the noise. This improved the accuracy and performance of the model.
- **Robust Design Under the Worst-Case Model**: To handle the worst-case scenario, they used a sampling method and the successive convex approximation (SCA) algorithm [Razaviyayn, 2014].
- **Convergence Analysis for the Proposed Designs**: Finally, cancelling noise in the data can lead to a different convergence behaviour. They used a convergence analysis to verify the robustness of the proposed designs.

3.4 Clustering with federated learning

A large portion of the literature that uses federated learning is based on deep neural networks, but we can find some other papers that use clustering algorithms. Unfortunately, even those papers use some kind of neural network. The scientific literature dedicated to classic (not using neural networks) clustering in a federated context is indeed very restricted. This section will be dedicated to summarizing some of that literature so that the reader will be a bit more familiar with the context of federated clustering.

3.4.1 ClusterGAN

The first algorithm that we will see, from *Dynamic Clustering in Federated Learning* [Kim et al., 2021], uses a *generative adversarial network* (GAN) to create clusters using a technique called **ClusterGAN** [Mukherjee et al., 2019]. They create their global model in 3 steps:

1. **Create local models** with **ClusterGAN**. The **ClusterGAN** network consists of 3 subnetworks. The data transferred between the client and the server consist of the weight of those subnetworks.
2. **Train the global model** using the **ClusterGAN** networks of the clients. The global model is trained by the data generated by those gathered networks.
3. **Split clusters** of the global model. They try to minimize the average loss and the variance loss of the global model.

They compared their technique with 2 other algorithms on 4 datasets. And they improved the accuracy by 29% to 45% depending on the dataset.

3.4.2 Agglomerative Clustering

Agglomerative clustering has been used extensively in the literature. In this paper by Briggs *et al.* [Briggs et al., 2020] they used this method with federated learning on the **MNIST** and **FEMNIST** dataset. Each client creates a local model using deep neural networks and sends the weights to the server. The server will merge those weights to create a global model. Agglomerative clustering is used to split the clients into clusters. Their algorithm has multiple global models, one for each client cluster. To cluster the clients, they run some setup round and fit the agglomerative model only once using those data. On both datasets, they get an average accuracy of over 90% on all clients and up to 98% in certain cases.

3.4.3 Spectral Clustering

Ahmadi *et al.* [Ahmadi et al., 2021] proposed in their paper to use Deep-Q-Reinforcement learning on the clients while using spectral clustering to determine which clients must be rewarded and which do not. As we can see in table 3.2, they get pretty good accuracy on the **MNIST** and **FEMNIST** datasets but perform poorly on the **CIFAR-10** dataset. Thus concluding that their technique can work pretty well, but the data has a big impact on the accuracy.

Dataset	Accuracy
MNIST	97.21%
FEMNIST	88.11%
CIFAR-10	58.42%

Table 3.2: Comparison of the accuracy of the algorithms on the 3 datasets.

3.5 Anomaly detection with federated learning

It didn't take so long after the emergence of federated learning to see papers that use this brandy new technique for anomaly detection. As said before, many of them, if not all, are based on neural networks. We will give you a quick look at some of them.

3.5.1 Deep anomaly on IoT

In a paper from 2021 [Liu et al., 2020], the authors used Deep Learning to detect anomalies in IoT data. For the client, they used a *convolutional neural network* with *long short-term memory* units. They used the *CNN* on each data of their

time series and passed the result of this first neural network into an *LSTM* model to have a prediction on the time series (see figure 3.2). When sending the local models to the server, they send the weights of the *CNN* and the *LSTM* model to the server. This one will aggregate the values of those models and send them back to all the clients. With this technique, they detected anomalies in the **MNIST** and **CIFAR-10** datasets. They have more than 90% accuracy on both datasets. In addition, they compared the method to 3 other techniques, *GRU*, *SAEs* and *SVM* on 4 datasets and on all of them they get better accuracy.

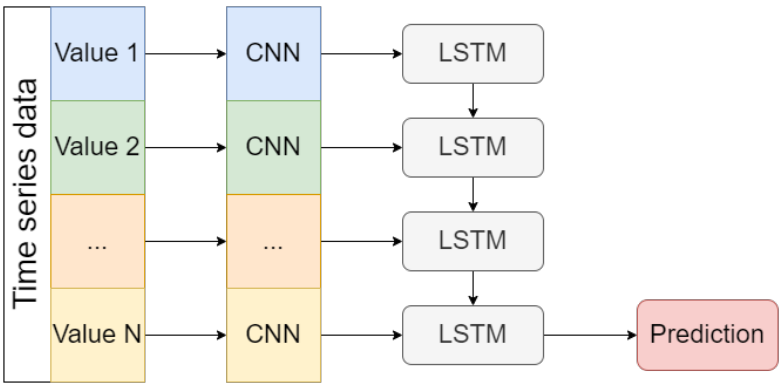

Figure 3.2: Schema of the technique used in the paper [Liu et al., 2020]. Each data of the time series is passed to a *CNN* and the result is passed to the *LSTM* model. The result of this last model is the prediction.

3.5.2 Anomaly Detection using blockchain

In the paper *Chained Anomaly Detection Models for Federated Learning: An Intrusion Detection Case Study* [Preuveneers et al., 2018], they developed a model that uses a blockchain method called **MultiChain**. When a node sends the weights of its local neural network to the server, it also pushes those weights to the blockchain. The server will only merge weights that have been validated by the blockchain. It ensures that the model cannot be modified by someone outside the network. Using blockchains might slow down each epoch, but they showed that the time necessary to use that blockchain is approximately 5 to 15% worse than a normal model as the time that they need to do the epoch. Thus, it does not have a huge impact on the time performance of the model as the server will also have to wait for the clients to send the weights of the next epoch. With the proposed technique they got the same accuracy as a normal model using an auto-encoder, which is around 97% on the **CICIDS2017** dataset.

Chapter 4

Experiments and results

4.1 Experimental setup

In this section, we will explain how we performed our experiments and what kind of data we used during this thesis.

4.1.1 Setup

First, we have to explain how we test the algorithms. To simulate the edge devices we decided to use Raspberry Pi as an edge client. Raspberry Pi seemed a shred of evidence to use as they can easily simulate small edge devices. To standardize our Raspberry we use the same image for each raspberry (Raspberry Pi 3 Model B with Raspberry Pi OS lite (32-bit)). All of our implementations are made to be running on Python 3.9.10 [vanRossum, 1995].

The server will be running on a laptop but as the data is only collected on edge, the machine used for the server does not have any impact on the data. Many algorithms that we used come from *scikit-learn* [Pedregosa et al., 2011] as it is the main toolbox used in the world of data science for python.

For the federated part, we looked for different frameworks like Tensorflow [Abadi et al., 2015], PySyft with PyGrid [Ziller et al., 2021] or flower. We used the flower [Beutel et al., 2020] framework because it is the simplest one to adapt to clustering and to the kind of experiments that we want to do as it allows us to define a custom aggregation function and custom learning algorithms for the clients. The other federated learning frameworks we reviewed were either not popular enough or not modular enough. Unlike most of its competitors, flower does not make the assumption that all the models are based on neural networks.

To gather data we used the python library psutil as it is really easy to use and matches our needs (see figure 4.1). To store data and read it in a convenient way, we used pandas [Wes McKinney, 2010] [pandas development team, 2020] as it is a performant staple of data science. Finally, to compute DTW and DBA, we used respectively the *python-dtw* [Giorgino, 2009] package and the implementation from DBA paper [Petitjean et al., 2011].

For most of the experiments, we will gather two values (see figure 4.2):

1. The CPU percentage usage over 1 sec
2. The memory used at each second

Those values are used over a 10-second time series to keep the overall pattern of each device (see section 2.2.1). We decided to go for CPU measurement because it is close to energy consumption. Indeed, energy consumption is long time series that we need to split (as explained in Section 2.2.1), and CPU data follows the same structure. It's simple to monitor CPU and memory utilization and produce controlled anomalies. Additionally, in edge devices such as the Raspberry Pi, CPU usage is directly linked to energy consumption [Kaup et al., 2014].

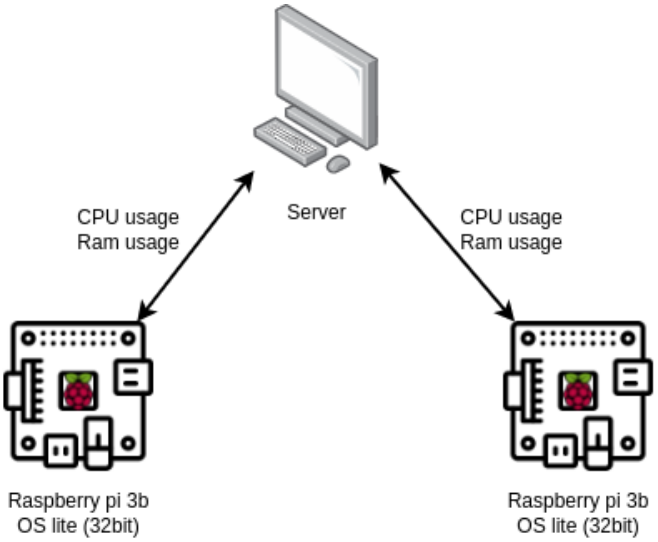

Figure 4.1: *Schematic of the experimental setup with two Raspberry Pi communicating with the server*

4.1.2 Anomalies

In this section, we will consider anomalies as any new pattern that can emerge in the time series. As we explained in our problem statement, we want to detect new

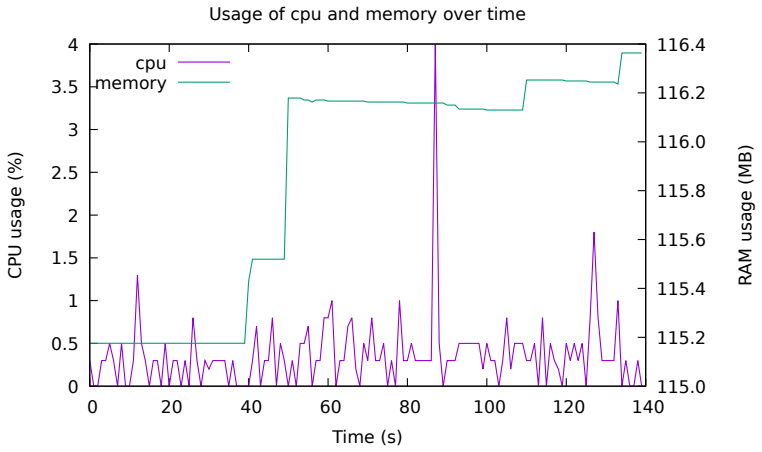

Figure 4.2: *Example of CPU and ram usage for a normal behaviour*

emerging patterns for any energy provider and, then, it's up to the energy provider to identify those patterns and label them as normal or abnormal.
To create anomalies we created a simple program that will run on the Raspberry Pi. The program computes matrix multiplication on pure python to create a huge CPU load. This type of attack is easy to detect, and thus we use it to create anomalies that are easy to detect (see figure 4.2 and figure 4.3). With this program, we can easily decide when to create an anomaly and when to stop it.

4.2 Algorithms

We tried multiple algorithms over the time of this thesis. We will describe those and show the associated results.

4.2.1 *K-Medoids* with a fixed number of clusters

This is the first algorithm we have thought of (see Figure 4.4). It is a decent start towards fixing the situation we have been handed. We will go through how it works on both the client and server sides. We will call it *Fixed Federated Learning K-Medoids or FFLKM*

Client

The first algorithm we tried is a simple *K-Medoids* on each device (see Algorithm 4). We first thought of using an algorithm with a dynamic number of clusters and especially *Agglomerative Clustering* for the clients. But we rapidly noticed that this

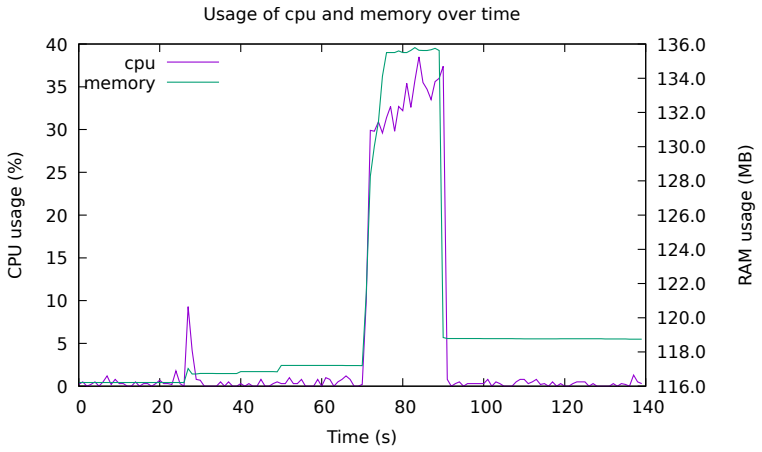

Figure 4.3: *Example of a CPU load attack between 70s and 90s.*

method does not let us share any interesting information with the server and the other clients, so we fall back to a *K-Medoids* algorithm and use a defined number of clusters. As our Raspberry Pi does not have many patterns of usage, we decided to use a relatively low number of 4 medoids. For this first iteration, the clients do not learn from the other edge devices and the anomalies are detected on the server-side. We used DTW for the clustering similarity measure. As explained in section 2.3 we use DTW as a similarity measure for our time series because it can compare time series by taking into account the shifts and stretches in the edge devices patterns. With *DTW* we can match pikes and valleys in the time series even if they are not at the same time. We compute the distance matrix using DTW (line 1) and then use it to fit our *K-Medoids* model (line 2). After that, we send the medoids to the server (line 3).

Algorithm 4: Client side clustering with *K-Medoids*

```
   Data: data, KMedoids
1  matrix = computeDTWDistanceMatrix(data);
2  KMedoids.fit(matrix);
3  medoids = KMedoids.medoids;
4  respond(medoids);
```

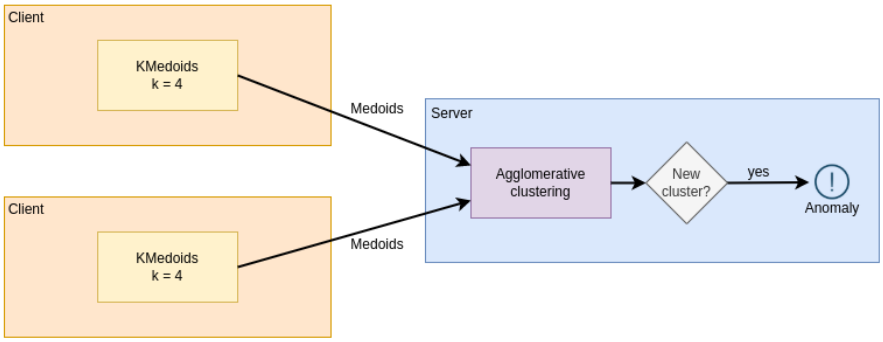

Figure 4.4: *First algorithm where edge clients perform a K-Medoids with 4 medoids. Those medoids are sent to the server. The server then uses those medoids to do an Agglomerative Clustering. If a new cluster emerges from this clustering, the server has detected an anomaly*

Server

To detect emergent patterns from the edge devices we used Agglomerative Clustering (see Algorithm 5). For the same reason as for the clients, we use DTW for the metrics. To detect the anomalies we gather the medoids from all the clients (lines 1-3). With those medoids and the preceding gathered medoids, we create a machine learning model (line 5). We only extract the number of clusters created as it is the number of different patterns of the edge devices (line 6). We keep the medoids over iterations to save the already seen patterns. The number of patterns seen may vary over time and so would the number of clusters. If this number goes up, it means that we have a new cluster emerging (lines 7-9), and it may be an anomaly. If this number goes down, it means that two patterns are close enough to be considered roughly the same. Then we save the number of clusters (thus patterns) for the next iteration (line 10).

Results

As we can see in figure 4.5, this algorithm can already detect anomalies. Before the anomaly, each client has 4 clusters and server 1 as all devices are the same and run the same algorithm. When an anomaly happens at a client, this client does not detect it, but we can see that with the sent medoids the server has detected the anomaly. The server will then create a new cluster for this anomaly. As discussed before, we can see that this technique does not share information between the edge devices. Only the server will detect the upcoming anomalies.

Algorithm 5: Server-side clustering with Agglomerative Clustering

```
Data: clients, AgglomerativeClustering, ncluster, medoids
Result: Detect possible anomalies
1  forall client in clients do
2  |  medoids = medoids ∪ request(client);
3  end
4  matrix = computeDTWDistanceMatrix(med);
5  AgglomerativeClustering.fit(matrix);
6  newNcluster = AgglomerativeClustering.ncluster;
7  if newNcluster > ncluster then
8  |  alert(possible anomaly);
9  end
10 ncluster = newNcluster;
```

4.2.2 *K-Medoids* with a dynamic number of clusters

As we also want to have a dynamic number of clusters for the clients, we tried to improve our first technique by changing a bit the algorithm (see Figure 4.6). We named it *Dynamic Federated Learning K-Medoids or DFLKM*.

Client

To do so, we decided to add a second clustering technique, *Agglomerative Clustering*, to the client-side. This will help us group the data points that are close to each other with a predefined threshold. This number of groups will be given to the *K-Medoids* algorithm with the aim of grouping anomalies together. In this second algorithm (see algorithm 6), as for the first algorithm, we compute the distance matrix with DTW (line 1). Then we fit the *Agglomerative Clustering* with the distance matrix (line 2) and extract the number of clusters generated (line 3). After that, we fit a *K-Medoids* model with the same number of clusters previously extracted with the distance matrix (lines 4-5). Finally, we send the medoids to the server (line 10).

Server

For *DFLKM*, we didn't have to change the server-side algorithm as the only thing that changed is the number of medoids received from the clients. But this number does not have any impact on the *Agglomerative Clustering* algorithm. We can detect the anomaly in the same way as explained before (see section 4).

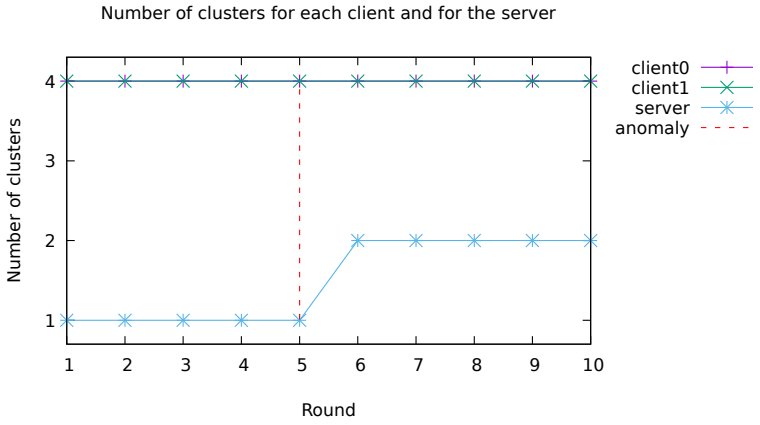

Figure 4.5: *Number of cluster on the clients and the server over 10 iterations of the federated algorithm (± 100 seconds). Where the clients use K-Medoids with k = 4 and the server uses Agglomerative Clustering*

Algorithm 6: Client Side clustering with *K-Medoids* & *Agglomerative Clustering*

```
Data: data, KMedoids, AgglomerativeClustering
matrix = computeDTWDistanceMatrix(data);
AgglomerativeClustering.fit(matrix);
nclusters = AgglomerativeClustering.nclusters;
model = KMedoids(nclust=nclusters);
model.fit(matrix);
medoids = model.medoids;
respond(medoids);
```

Results

Figure 4.7 shows us that with *DFLKM*, the edge devices now have a dynamic number of clusters. Unlike the first one, we can see the anomalies emerging on the client at the same time as the server. But we can also see that the edge devices do not share the information between them. Now, the clients and the server can detect the anomalies.

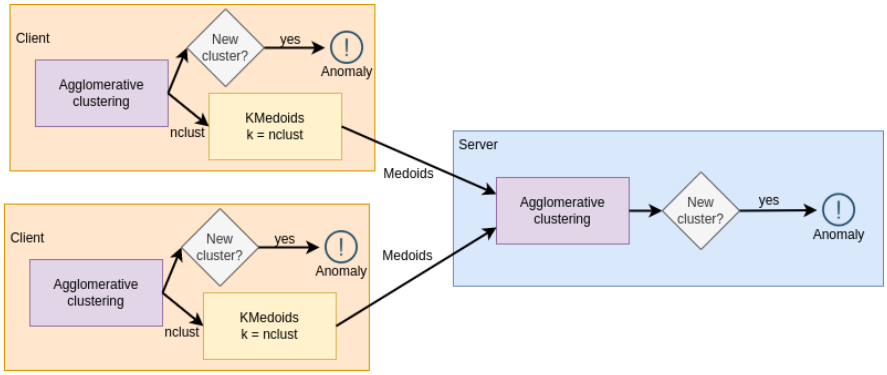

Figure 4.6: *DFLKM refines FFLKM by using an Agglomerative Clustering to determine the number of clusters k for the K-Medoids algorithm. Then the server will use the medoids to create an Agglomerative Clustering model. The server will then detect emerging clusters as in the first algorithm.*

4.2.3 *K-Medoids* with dynamic number of clusters and client knowledge update

The usage of *federated learning* allows us to have a global model that will be shared by all the devices. The clients will use this global model to update their knowledge of good and bad behaviours. In this third iteration of our algorithm, we will share this global model with the clients (see Figure 4.8). We will refer to this model as *Dynamic Federated Learning K-Medoids with client knowledge update or DFLKM+.*

Client

To update the client's knowledge of the different normal patterns we use the same algorithm as for *DFLKM* in section 4.2.2. The only difference is that the server will send data to be added to the gathered data that will force a cluster around those (see Algorithm 7 line 1).

Server

To generate the knowledge that we want to send to the edge clients we have to choose a clustering technique where we can send the clusters through the network. As said previously, *Agglomerative Clustering* does not give us proper centroids we can share, so we decided to use the same technique as for the client. We used *Agglomerative Clustering* to determine the number of clusters for a *K-Medoids* algorithm (see algorithm 8). Then, we gather the medoids from the edge clients

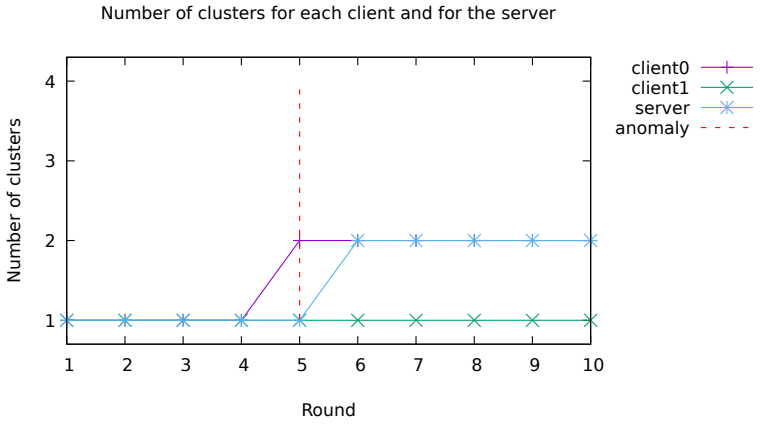

Figure 4.7: *The number of clusters on the clients and the server over 10 iterations of the federated algorithm (± 100 seconds). Where the clients use K-Medoids with k determined by an Agglomerative Clustering and the server uses Agglomerative Clustering*

(lines 1-3), create the distance matrix with DTW (line 4) and put it into an Agglomerative Clustering to determine the number of clusters of the *K-Medoids* (lines 5-6). We still have the anomaly detection part on the *Agglomerative clustering* (lines 7-9). Then we fit the KMedoid with the distance matrix (lines 11-12) and send the medoids back to the clients (lines 14-16).

Results

With this algorithm, we can see in Figure 4.9 that when an anomaly appears on an edge device that device will detect it. After sending his medoids to the server, the server will detect it as well and send the new pattern to all other edge devices. Those clients will learn the new pattern using the medoids of the first client. In this figure, we can see the propagation of the new pattern to the server and then to the other edge device, all in 3 steps.

4.2.4 Federated time-series K-Means with KDE obfuscation

This section is dedicated to introducing what we think is our most interesting algorithm for a federated learning approach. Indeed, where data privacy was regarded as a secondary concern for our previous attempts, here it was at the very center of our efforts. So as we will first introduce the key differences between our previous attempts.

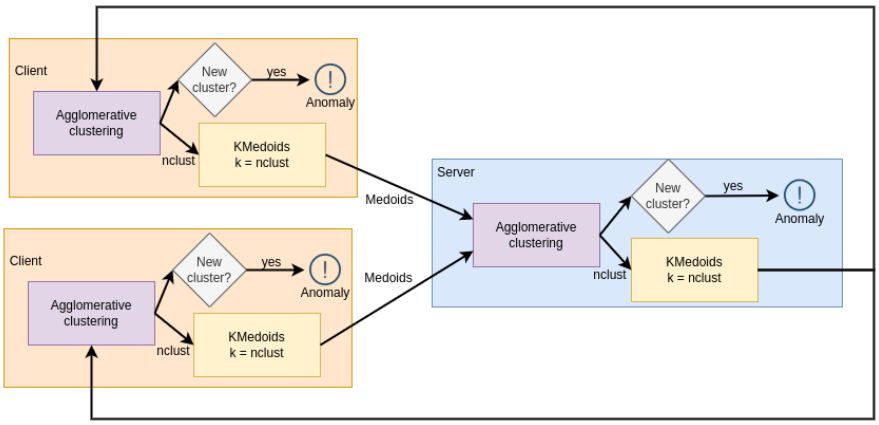

Figure 4.8: *DFLKM+ uses roughly the same algorithm for the clients and the server. The server uses his K-Medoids algorithm to send the medoids to the clients to update their local knowledge of anomalies. Therefore, the client will not detect emerging patterns that have been flagged as normal by the server.*

Privacy enhancements

So far, the only privacy-preserving measure we have taken can be summarized by just not sharing every data point. While training a global detection model without sharing every data point is already a step in the right direction, edge devices still have to share some real-world data from their private database. In many cases, even sharing one data point can be problematic. To convince ourselves of that, we can just think of the data collected on edge in [Mansour et al., 2020], even one text message could be very sensitive.

The natural answer to that problem would be to use a *centroid-based* clustering algorithm instead of a *medoid-based* algorithm, which differ from each other in the fact that the first use barycenters of data points as representative points for the clusters while the latter uses actual points from the dataset as representative points for the clusters.

While this would already increase greatly the data privacy, a centralized FL server could potentially be able to learn parts of the distribution of a targeted edge device over time. We want to make that task the most difficult possible for the server. However, in most of clustering literature, we have to pick either synthetic points, barycenters or database points to share for the global model to be able to take advantage of the FL network's knowledge.

A solution that could even work for *medoid-based* algorithms would be to project data points onto a global space specified during FL rounds. This could

Algorithm 7: Client Side clustering with *K-Medoids* & Agglomerative Clustering and knowledge update

```
Data: data, KMedoids, AgglomerativeClustering, serverMedoids
data = data ∪ serverMedoids;
matrix = computeDTWDistanceMatrix(data);
AgglomerativeClustering.fit(matrix);
nclusters = AgglomerativeClustering.nclusters;
model = KMedoids(nclust=nclusters);
model.fit(matrix);
medoids = model.medoids;
respond(medoids);
```

be achieved in multiple ways, we could use an auto-encoder neural network to learn a latent space common to all devices. However, we wanted to stay away from neural networks as much as possible due to their *black-box* nature. Another way to hide the authentic data from the rest of the FL network would be to learn a principal component analysis projection in a federated context [Wang et al., 2021]. Unfortunately, this technique is very recent, so we deemed it still too experimental to base our work on. The technique we ended up picking to enhance privacy is based instead on much more solid work.

KDE obfuscation

The idea we used was with what we will call *KDE obfuscation* from now on. The idea behind it is very simple. We make the assumption that we have a way to sample a pdf estimated using KDE and a given point c for which we want to control how much information about c we will share. Then, all we have to do is sample N points from the estimated pdf and pick the one that is closest to the target according to some similarity measure d and call it p. We can control how much information about c we give out by controlling n. The process is fleshed out in Algorithm 9. An illustration of the process as well as the effect of n can be found in Figure 4.10.

FL network description

This part will be dedicated to describing the FL network we have used in our experiments. We will start by describing its architecture.

The FL network will simply be a centralized network where all edge devices communicate with the FL server each round, sending their incremental updates,

Algorithm 8: Server side clustering with Agglomerative Clustering & *K-Medoids*

Data: clients, AgglomerativeClustering, ncluster, medoids, KMedoids
Result: Detect possible anomalies & update clients

```
forall client in clients do
    medoids = medoids ∪ request(client);
end
matrix = computeDTWDistanceMatrix(med);
AgglomerativeClustering.fit(matrix);
newNcluster = AgglomerativeClustering.ncluster;
if newNcluster > ncluster then
    alert(possible anomaly);
end
ncluster = newNcluster;
model = KMedoids(ncluster=ncluster);
model.fit(matrix);
serverMedoids = model.medoids;
forall client in clients do
    send(client, serverMedoids);
end
```

the server will collect those updates, proceed to server side anomaly detection and send back the federated model to the edge devices.

The FL model that will be shared between the server and the edge devices will be a *time series K-means* model. This is a slightly modified version of *K-means*, described in algorithm 10.

The model elements shared on the FL network will be the KDE obfuscated centroids of the edge devices and their anomalies. We will describe the anomalies in more details later but all we need to know for now, is that both the server and the edge devices will share an *anomaly score alert threshold* parameter.

Client description

Each FL round, the client will wait for the updated cluster centroids from the server if needed. It will then proceed to add the data points collected during the current round and add it to its own private database. Then it will proceed to the learning step. It is constituted of three phases:

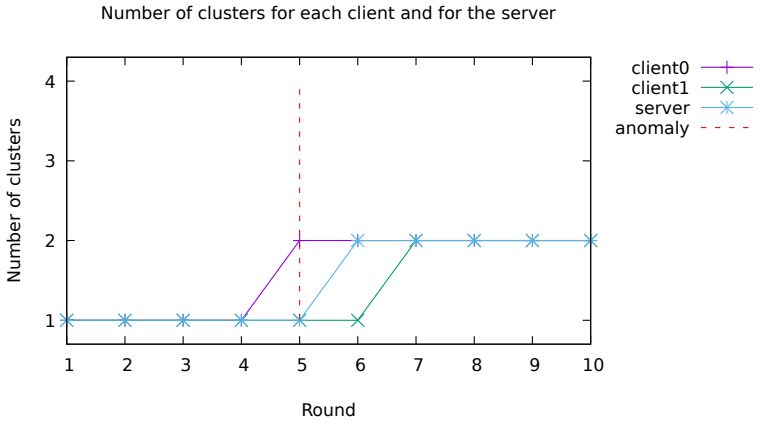

Figure 4.9: *The number of clusters on the clients and the server over 10 iterations of the federated algorithm (± 100 seconds). Where the clients use K-Medoids with k determined by an Agglomerative Clustering and the server uses Agglomerative Clustering*

1. Learn the KDE pdf of the whole database, including the new points.
2. Update the local copy of the local model with new data points using *time series K-means*.
3. Perform SSDO anomaly scoring on each point of the database using the new centroids.
4. Send back the KDE-obfuscated centroids of the local model to the server.
5. Send back tuples of (point score, point index) to the server for every database point that has an anomaly score ≥ 0.5.
6. If some of the anomalies tuples's scores go over the *anomaly score alert threshold*, send back the corresponding tuples to the server and an anomaly reporting system to the server.
7. Should the server need it, estimate the ideal number of clusters using *agglomerative clustering*.

An overview of the client's system can be seen in figure 4.11. A pseudo-code for the client estimation of the number of clusters can be found in algorithm 11.

Algorithm 9: KDE obfuscation algorithm

```
   Data: pdf, c, n
   Result: p
 1 min_dist = ∞;
 2 p = ∅;
 3 forall i in 1..n do
 4 |   candidate = sample(pdf);
 5 |   if p == ∅ then
 6 |   |   p = candidate;
 7 |   end
 8 |   else if d(c, candidate) < d(c, candidate) then
 9 |   |   p = candidate;
10 |   |   min_dist = d(c, p);
11 |   end
12 end
13 return p
```

The basic idea is that the client will see how many more clusters n_c the *agglomerative clustering* would need. However we can't simply take from the last n_c clusters as we have no idea which of these clusters are new. Fortunately, by construction of the SSDO anomaly score, we can guess that clusters with few points are more likely to be anomalies. Thus we can say that if the agglomerative clustering model wants new clusters, the clusters with the least points will be the ones that are most likely to be considered new.

This is to be taken with a grain of salt however as it is purely a heuristic and even the concept of *new* cluster is not really clear to begin with.

Server

This section is dedicated to the FL server. The role of the server is two-fold:

1. It will receive and update the global *time series k-means* model using DBA.
2. It will record the centroids sent by each device to perform SSDO anomaly scoring on those.

The server side anomaly detection is performed using *time series K-means* with a number of cluster corresponding to that of the global's model. The reasoning behind it is that the global model should be the most descriptive of the global dataset, and thus, all the centroids transmitted to the server should be close to

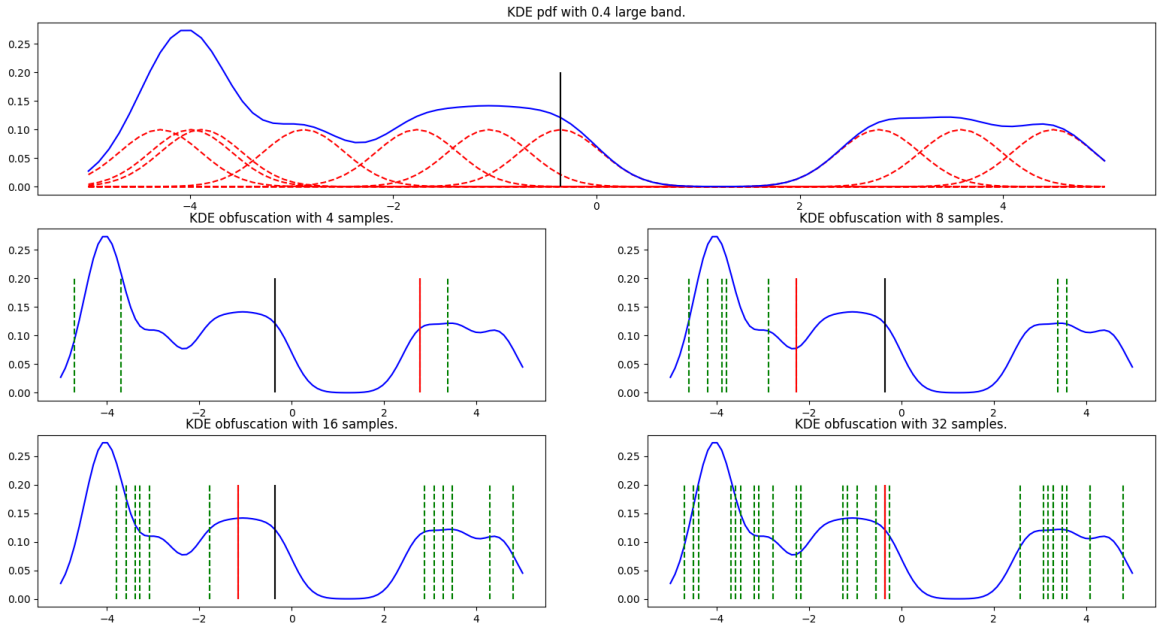

Figure 4.10: *From up left to down right, the first plot represents a KDE estimation of a pdf given some points, the contributions of the points are in red dotted lines and the estimated pdf in a filled blue stroke. The point we want to obfuscate is at the vertical black line. For the following plots, the estimated pdf is still in blue and the obfuscated point in a black line. However, the candidate points are in green dotted lines and the picked point in a filled red line. We can see that increasing the number of candidates gives an obfuscated closer and closer to the target point. For the last plot, it is even right on the target point.*

one of the clusters. This is again a purely heuristic technique, there is no formal proof behind it.

Here is a short description of the server's routine. To follow along, there is a representation of the server's data-flow at figure 4.12. During the first round, the server will tell every device to take part to the FL process with an arbitrary number of starting clusters. This is a simplified version as in actual FL, not necessarily all clients will participate in every round. Then, all the client perform their incremental update of the centroids, in the case of the first round, they also initialize the centroids using the best of a few random sampling of the database.

Then, the server collects each KDE-obfuscated centroid for each device and aggregates them using DBA. Next, the server collects the anomaly scores of the clients and sends those to the warning system if their score is above the *anomaly*

Algorithm 10: Time series K-means update

```
   Data: database, centroids
   Result: new_centroids
 1 clusters = (∅ for c ∈ centroids );
 2 forall x ∈ database do
 3 |   cluster_points = ∅;
 4 |   min_centroid_dist = ∞;
 5 |   nearest_centroid = ∅;
 6 |   forall c ∈ centroids do
 7 |   |   dist = DTW_dist(c, x);
 8 |   |   if dist < min_centroid_dist then
 9 |   |   |   min_centroid_dist = dist ;
10 |   |   |   nearest_centroid = c ;
11 |   |   end
12 |   end
13 |   clusters(nearest_centroid) = clusters(nearest_centroid) ∪ x ;
14 end
15 forall c ∈ centroids do
16 |   new_centroids = new_centroids ∪ DBA(clusters(c));
17 end
18 return new_centroids;
```

score alert threshold. The server will ask the client which has the most anomalous point to estimate the number of clusters and use it as the global number of cluster from now on.

Finally, the server will perform the SSDO anomaly scoring for its own database of collected centers as explained earlier and may raise another alert. Once all of it is done, the server can send the updated model to the edge devices and continue for the next round. An overview of this process is detailed in algorithm 12

Experiments

The main take away of *federated time series k-means* is the method for sharing centroids without sharing too much information as according to our literature review, there isn't really any agreed upon method for doing so at the time of writing this thesis. However, to keep our work in line with the problem we are tackling, we added the anomaly detection part, as well as a rough way to adjust the number of clusters dynamically. All of this was explained thoroughly in the previous sections, so we won't be repeating it again.

Algorithm 11: Estimation of the number of clusters

```
   Data: database, model_params, kmeans_centroids
   Result: num_clusters, new_clusters
 1 n_current_clusters = length(kmeans_centroids);
 2 model = AgglomerativeClustering(database, model_params);
 3 dist_matrix = DTW_dist_matrix(database);
 4 model.fit(dist_matrix);
 5 num_clusters = model.n_clusters;
 6 if num_clusters ≤ n_current_clusters then
 7 |   new_clusters = model.cluster_centers;
 8 |   return num_clusters, ∅;
 9 end
10 else
11 |   new_clusters = ∅;
12 |   sort_by_number_of_cluster_points(model.cluster_centers);
13 |   δc = num_clusters - n_current_clusters;
14 |   forall i in 1..δc do
15 |   |   centroid = pick_point_from_cluster(model.cluster_centers[i]);
16 |   |   new_clusters = new_clusters ∪ centroid ;
17 |   end
18 |   return num_clusters, new_clusters;
19 end
```

Algorithm 12: Server routine

```
   Data: database, round_number, devices, n_clusters
 1 if round_number == 1 then
 2 |   forall d ∈ devices do
 3 |   |   init_clusters(d, n_clusters);
 4 |   end
 5 end
 6 collected = ∅;
 7 forall d ∈ devices do
 8 |   collected = collected ∪ collect_centroids(d);
 9 end
10 database = database ∪ collected;
11 updated_clusters = DBA(collected, n_clusters);
12 anomalies = ∅;
13 forall d ∈ devices do
14 |   anomalies = anomalies ∪ (d, updated_clusters);
15 end
16 most_anomalous_device = argmax(scores(anomalies));
17 if max_score(most_anomalous_device) ≥ anomaly_score_alert_threshold
    then
18 |   n_clusters = estimate_num_clusters(most_anomalous_device);
19 end
20 forall d ∈ devices do
21 |   send_centroids(d, updated_clusters);
22 |   set_num_clusters(d, n_clusters);
23 |   next_round(d);
24 end
25 perform_local_anomaly_scoring(database, n_clusters);
26 next_round();
```

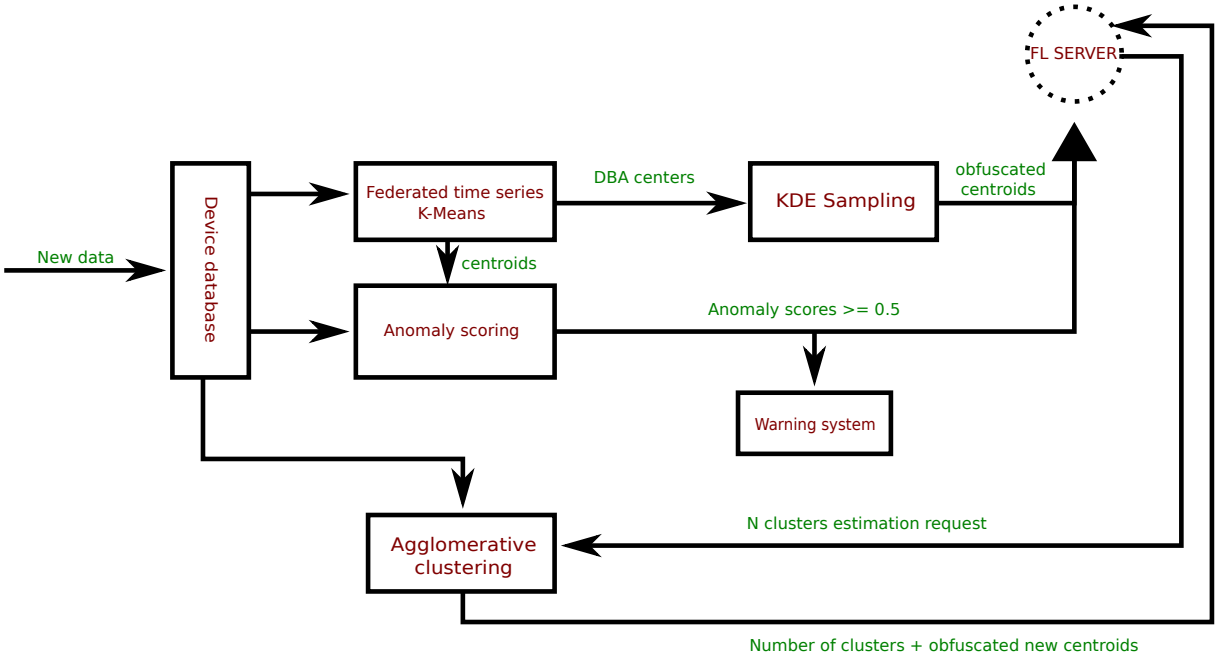

Figure 4.11: *A representation of the dataflow in a FL client, i.e. an edge device. We can see that the entry point is when new data arrives but it might as well be when a round ends. The exit point is when the client sends the updated centroids to the server. The possibility for the server to ask the client to evaluate the number of clusters is also represented in the lower part of the figure.*

First, we will define the experimental setup. For easier implementation, we didn't use a real-world federated setup like in previous experiments. Instead, we used a simple simulation of a federated setup.

The simulation is based on a simple network topology, that is a few devices connected to a central server. This was simulated simply by having separated databases for each fictional device. At each round, the local learning steps would be performed on each database, and aggregated into one using the FL aggregation strategy mentioned earlier. The main disadvantage of such a setup is that the experiments are slower than they would be in real world as each fictional update is performed sequentially rather than in parallel.

Secondly, the experiments will be evaluated against real world data. The dataset we will use is called
skoltech anomaly benchmark (SKAB) [Katser and Kozitsin, 2020] and is meant to be a benchmark evaluating anomaly detection techniques. Indeed, it has labels for both *change points* and *anomalies*. We will use the `valve1` part of the dataset.

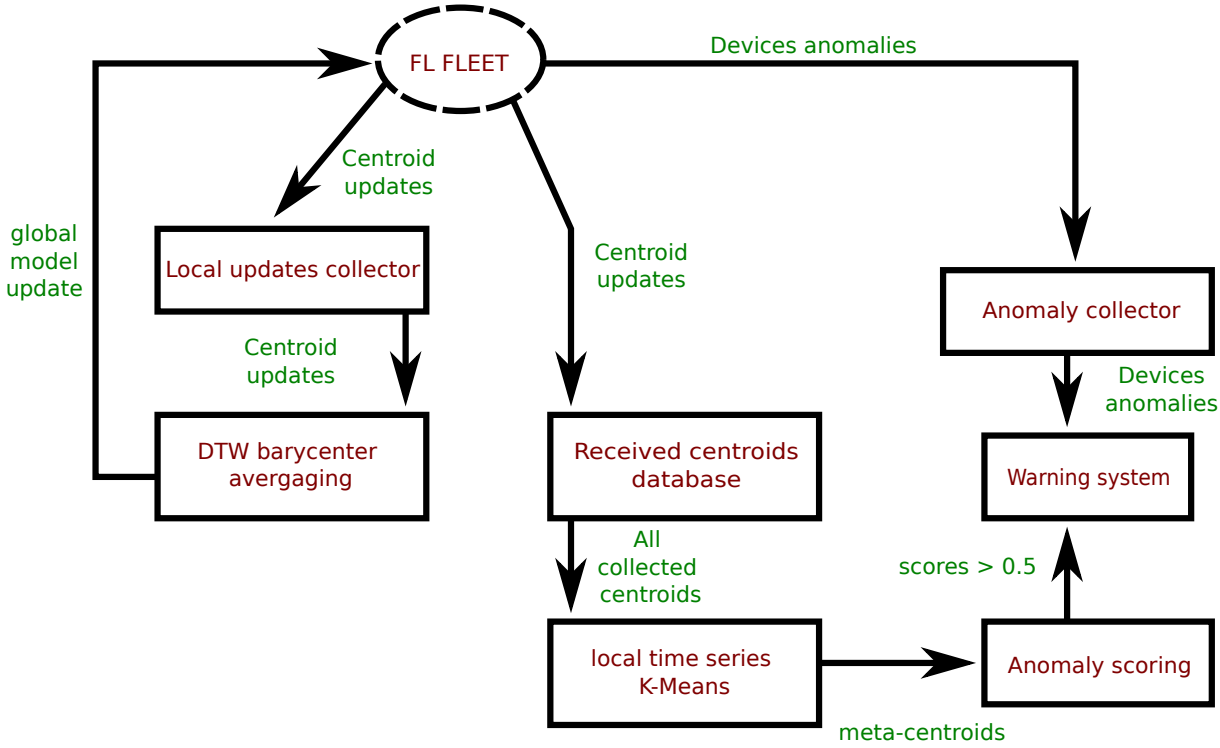

Figure 4.12: *A representation of the dataflow in a FL server. The entry point is not represented as it is when the first round starts. The new data arrives from the FL fleet, which is simply the group of all edge devices taking part to the FL process.*

Each point of the feature space will contain 8 features, excluding the timestamps and labels. As our technique is meant to be used on time series data, we will re-label whole time series by marking a time series as anomalous if one of its points is an anomaly.

Finally, we need to describe the parameters of the experiment:

- Preprocessing: $[0, 1]$-scaling of numerical features.
- Number of simulated devices: 3.
- Similarity measure: Normalized DTW.
- Agglomerative Clustering: Average linkage, distance threshold of 0.5.
- Initial clusters: 5.
- # Initial time series in device database: 30.

- # New time series per round: 25.
- Window size: 25.
- Number of rounds: 5.

It is to be noted that the *# initial time series in device database* are the time series added during the first round. This is to control how much knowledge the devices have of the distribution.

The experiment took about an hour to run. The exact runtime is not important here as it is just a simulation. However, the scale of the runtime is worrisome. One of the explanation could be the almost pure python nature of the implementation even if native C extensions have been utilized when available. One other reason could be the large feature space, $8 \times 25 = 200$ features to consider for *each* time series. The combination of these two reasons could probably explain the huge processing time. After running the experiment with a threshold of 0.7, the prediction label for each time series of each device compared with the ground truth can be viewed in figure 4.13. For reproducibility, we took care to have a random seed governing each. The F1-Scores of all the experiments are shown in table 4.1.

Threshold	0.5	0.6	0.7	0.8	0.9	1.0
Device 0 F1-score	0.179	0.210	0.233	0.106	0.200	0.035
Device 1 F1-score	0.114	0.236	0.121	0.230	0.063	0.199
Device 2 F1-score	0.000	0.103	0.423	0.175	0.054	0.281
All devices average	0.097	0.1834	0.259	0.171	0.106	0.870

Table 4.1: *A table summarizing the F1-Scores of the experiments with a threshold ranging from 0.5 to 1.0 the averages of all device's F1-scores for each experiments can be found on the bottom row.*

As expected, the F1-score are not exactly great, but they seem to follow a predictable pattern. The F1-score of the experiment with a threshold of 0.7 is the best and both other ends are much worse. This suggests that this technique could be used to detect anomalies in time series data with the use of semi-supervised learning and a bit of hyperparameter tuning.

An interesting pattern that can be seen in figure 4.13 is that the anomaly detection seems to *propagate* across the devices, suggesting that the devices are indeed learning from each other without sharing actual data, which was the primary goal.

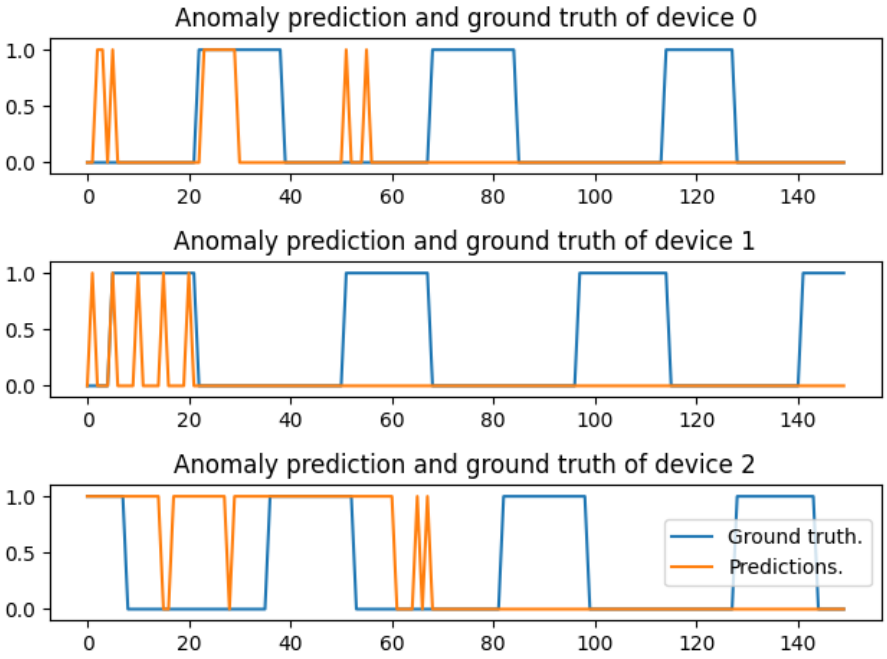

Figure 4.13: *Representations of the results from the experiment with a 0.7 threshold, the blue plot lines reach 1 if a time series is anomalous, and 0 otherwise. The orange plot lines represent the anomalies the devices have detected. Again, it is 1 if anomalous, 0 otherwise. The horizontal axis is just the index of each time series*

This is both a good and a bad thing as the device #2 is wrongfully predicting that the time series are anomalous, probably because of the anomalies detected by device #1 during previous rounds.

Finally, due to the fact that we put the anomalous time series in the learning set after they have been detected, this suggests the use of this technique for change *point detection* rather than *anomaly detection*.

Chapter 5

Conclusion

In this thesis, we propose a new approach to the detection of anomalies in federated learning. We used a clustering approach with clustering algorithms that can have a different number of clusters over the iterations. We used that to detect emerging patterns in CPU usage data sets. With tried different approaches with each iteration meeting more and more requirements (see Table 5.1).

Algorithm	Clustering	Anomaly Detection		Knowledge Sharing	Privacy Preserving
		Server Side	Client Side		
FFLKM	✓	✓	✗	✗	✗
DFLKM	✓	✓	✓	✗	✗
DFLKM+	✓	✓	✓	✓	✗
KDE obfuscation	✓	✓	✓	✓	✓

Table 5.1: *Comparison of our different algorithms regarding the requirements. Each iteration validates more and more requirements.*

5.1 Conclusion

We used a different clustering algorithm with the *federated learning* framework. We showed that our first solutions can detect anomalies in data relatively rapidly. Anomalies are detected on the client the round that they appear and one round after on the server. The data that we used to check our algorithm was the CPU usage of the Raspberry pis that we used for our experiments. We simulated attacks by running a matrix multiplication program on the clients. The first algorithm we had implemented,*FFLKM*, can detect such anomalies on the server but does not preserve any data sent as we simply send the computed medoids over the network, even this naive approach gives good results but leaves privacy aside.

The next algorithm, *DFLKM*, allows us to detect the same anomalies but directly on the client using the same technique as the anomaly detection on the server, but it also puts aside privacy.

The third algorithm, *DFLKM+*, refines the other two by creating a global model on the server and then sharing it with the client. This technique should permit a reduction of false positives for the clients as if they have an emerging pattern that has already appeared in another client. As explained before, and as we can see in table 5.1, the main problem of those algorithms is that it does not preserve any privacy over the data used. The usage of *K-Medoids* forces us to send the medoids which are gathered data points directly.

To answer this last problem we devised what we called *federated time series K-means with KDE obfuscation* which uses the well known *dynamic time warping* and the much more recent *barycenter averaging* technique associated with that similarity measure. On top of all of that is the use of *kernel density estimation* sampling to transmit the centroids in the network in privacy-preserving manner.

We tested it on a well known real world anomaly detection dataset with labeled example. We found out that this technique indeed seems to allow a federated network to learn a global centroid based clustering model in a more privacy preserving manner, more experiments to quantify the impact of the learning process and what is the effect of the number of samples used in *KDE obfuscation*.

Another point to improve is the accuracy of that technique, the F1-Score of the anomaly classification task is far from being on par with state-of-the-art methods and thus can't be used in real world yet. A lead to explore in order to improve the classification score would be to revise the way to set the number of clusters dynamically, potentially use other clustering algorithms even if using non centroid-based algorithms could be less privacy preserving. Another would be to fully use a semi-supervised learning technique to learn the global model.

To summarize, we are very happy with the technique that we created, and we think that we managed to respond to the needs of the initial problem even if some points could be improved. But with the usage of distance-based *clustering techniques*, we found that preserving the data privacy is a real challenge as *clustering algorithms* need the data to create clusters. As far as we managed to go, the more *privacy preserving* technique we found still allows us to reduce the amount of information shared by the edge devices to the federated learning network, however we cannot quantify how much privacy is acquired. This would need to be studied and tested in a more mathematics oriented work. All things considered, working on time series data proved to be a very interesting problem to tackle, the literature regarding the related spanned quite an impressive time period, some articles dating

back to the 1960s. Another positive note is that we managed to avoid working with neural networks which comes with its own *vast* set of problems and challenges, mainly related their lack of interpretability and their potential high computational costs, especially in the case of recurrent neural networks which are the typical ways to solve problems with time series data. We hope that some of our work can be used to help others working with federated clustering on time series data.

5.2 Future work

We think that one of the major work to do is to test our last algorithm with more datasets or even put it in a real environment. Comparing it with other anomaly detection techniques to see if it can detect the same anomalies but in a federated way. Finding ways to improve even more the privacy of the data that we share may be a good improvement in the direction of *federated learning*. Using different *clustering algorithm* and comparing them to find the best fit may be interesting. And lastly we would try to improve the computational performance as it was the main bottleneck of our testing process. There is probably some inefficiencies in the code that we could improve, but in the span of our master's thesis, we were not able to find the culprits.

We learned a lot throughout the year we spent working on this master's thesis. First, we had never heard of federated learning, its capabilities, or its possible applications before this master thesis. So, to build our solutions, we had to figure out how to put them to use. This led to one of the most important skills we learned: doing literature research. To understand how to apply different algorithms and for which problems we may use them, we had to look for references in the literature.

Bibliography

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Aghabozorgi et al., 2015] Aghabozorgi, S., Shirkhorshidi, A. S., and Wah, T. Y. (2015). Time-series clustering–a decade review. *Information Systems*, 53:16–38.

[Ahmadi et al., 2021] Ahmadi, M., Taghavirashidizadeh, A., Javaheri, D., Masoumian, A., Ghoushchi, S. J., and Pourasad, Y. (2021). Dqre-scnet: a novel hybrid approach for selecting users in federated learning with deep-q-reinforcement learning based on spectral clustering. *Journal of King Saud University-Computer and Information Sciences*.

[Ahmed et al., 2016a] Ahmed, M., Mahmood, A. N., and Hu, J. (2016a). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31.

[Ahmed et al., 2016b] Ahmed, M., Mahmood, A. N., and Islam, M. R. (2016b). A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55:278–288.

[Ang et al., 2020] Ang, F., Chen, L., Zhao, N., Chen, Y., Wang, W., and Yu, F. R. (2020). Robust federated learning with noisy communication. *IEEE Transactions on Communications*, 68(6):3452–3464.

[Arthur and Vassilvitskii, 2006] Arthur, D. and Vassilvitskii, S. (2006). k-means++: The advantages of careful seeding. Technical report, Stanford.

[Beutel et al., 2020] Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Parcollet, T., and Lane, N. D. (2020). Flower: A friendly federated learning research framework. *CoRR*, abs/2007.14390.

[Bonawitz et al., 2019] Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, B., et al. (2019). Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, 1:374–388.

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

[Briggs et al., 2020] Briggs, C., Fan, Z., and Andras, P. (2020). Federated learning with hierarchical clustering of local updates to improve training on non-iid data. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE.

[Celebi et al., 2013] Celebi, M. E., Kingravi, H. A., and Vela, P. A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, 40(1):200–210.

[Chandola et al., 2009] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3).

[Chen, 2017] Chen, Y.-C. (2017). A tutorial on kernel density estimation and recent advances. *Biostatistics & Epidemiology*, 1(1):161–187.

[Chou and Telaga, 2014] Chou, J.-S. and Telaga, A. S. (2014). Real-time detection of anomalous power consumption. *Renewable and Sustainable Energy Reviews*, 33:400–411.

[Giorgino, 2009] Giorgino, T. (2009). Computing and visualizing dynamic time warping alignments in r: The dtw package. *Journal of Statistical Software*, 31(7):1–24.

[Issa and Vasarhelyi, 2011] Issa, H. and Vasarhelyi, M. A. (2011). Application of anomaly detection techniques to identify fraudulent refunds. *Available at SSRN 1910468*.

[Katser and Kozitsin, 2020] Katser, I. D. and Kozitsin, V. O. (2020). Skoltech anomaly benchmark (skab).

[Kaufman and Rousseeuw, 2009] Kaufman, L. and Rousseeuw, P. J. (2009). *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons.

[Kaup et al., 2014] Kaup, F., Gottschling, P., and Hausheer, D. (2014). Powerpi: Measuring and modeling the power consumption of the raspberry pi. In *39th Annual IEEE Conference on Local Computer Networks*, pages 236–243. IEEE.

[Khan and Salah, 2018] Khan, M. A. and Salah, K. (2018). Iot security: Review, blockchain solutions, and open challenges. *Future generation computer systems*, 82:395–411.

[Kim et al., 2021] Kim, Y., Al Hakim, E., Haraldson, J., Eriksson, H., da Silva, J. M. B., and Fischione, C. (2021). Dynamic clustering in federated learning. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE.

[Lee et al., 2019] Lee, J. S., Darcy, K. M., Hu, H., Casablanca, Y., Conrads, T. P., Dalgard, C. L., Freymann, J. B., Hanlon, S. E., Huang, G. D., Kvecher, L., et al. (2019). From discovery to practice and survivorship: Building a national real-world data learning healthcare framework for military and veteran cancer patients. *Clinical pharmacology and therapeutics*, 106(1):52.

[Liu et al., 2021] Liu, X., Ding, Y., Tang, H., and Xiao, F. (2021). A data mining-based framework for the identification of daily electricity usage patterns and anomaly detection in building electricity consumption data. *Energy and Buildings*, 231:110601.

[Liu et al., 2020] Liu, Y., Garg, S., Nie, J., Zhang, Y., Xiong, Z., Kang, J., and Hossain, M. S. (2020). Deep anomaly detection for time-series data in industrial iot: A communication-efficient on-device federated learning approach. *IEEE Internet of Things Journal*, 8(8):6348–6358.

[Lloyd, 1982] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137.

[Mansour et al., 2020] Mansour, Y., Mohri, M., Ro, J., and Suresh, A. T. (2020). Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619*.

[Mukherjee et al., 2019] Mukherjee, S., Asnani, H., Lin, E., and Kannan, S. (2019). Clustergan: Latent space clustering in generative adversarial networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4610–4617.

[Müller, 2007] Müller, M. (2007). Information retrieval for music and motion. *Information retrieval for music and motion*, pages 69–84.

[Münz et al., 2007] Münz, G., Li, S., and Carle, G. (2007). Traffic anomaly detection using k-means clustering. In *GI/ITG Workshop MMBnet*, volume 7, page 9.

[Nielsen, 2016] Nielsen, F. (2016). *Hierarchical Clustering*, pages 195–211.

[Niknam et al., 2020] Niknam, S., Dhillon, H. S., and Reed, J. H. (2020). Federated learning for wireless communications: Motivation, opportunities, and challenges. *IEEE Communications Magazine*, 58(6):46–51.

[pandas development team, 2020] pandas development team, T. (2020). pandas-dev/pandas: Pandas.

[Park and Jun, 2009] Park, H.-S. and Jun, C.-H. (2009). A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*, 36(2, Part 2):3336–3341.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Petitjean et al., 2011] Petitjean, F., Ketterlin, A., and Gançarski, P. (2011). A global averaging method for dynamic time warping, with applications to clustering. *Pattern recognition*, 44(3):678–693.

[Preuveneers et al., 2018] Preuveneers, D., Rimmer, V., Tsingenopoulos, I., Spooren, J., Joosen, W., and Ilie-Zudor, E. (2018). Chained anomaly detection models for federated learning: An intrusion detection case study. *Applied Sciences*, 8(12):2663.

[Razaviyayn, 2014] Razaviyayn, M. (2014). *Successive convex approximation: Analysis and applications*. PhD thesis, University of Minnesota.

[Sheng and Liu, 2006] Sheng, W. and Liu, X. (2006). A genetic k-medoids clustering algorithm. *Journal of Heuristics*, 12(6):447–466.

[Steemers and Yun, 2009] Steemers, K. and Yun, G. Y. (2009). Household energy consumption: a study of the role of occupants. *Building Research & Information*, 37(5-6):625–637.

[Syarif et al., 2012] Syarif, I., Prugel-Bennett, A., and Wills, G. (2012). Unsupervised clustering approach for network anomaly detection. In *International conference on networked digital technologies*, pages 135–145. Springer.

[vanRossum, 1995] vanRossum, G. (1995). Python reference manual. *Department of Computer Science [CS]*, (R 9525).

[Vercruyssen et al., 2018] Vercruyssen, V., Meert, W., Verbruggen, G., Maes, K., Baumer, R., and Davis, J. (2018). Semi-supervised anomaly detection with an application to water analytics. volume 2018-November, pages 527–536. IEEE.

[Wagstaff et al., 2001] Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S., et al. (2001). Constrained k-means clustering with background knowledge. In *Icml*, volume 1, pages 577–584.

[Wang et al., 2021] Wang, Y., Bennani, I. L., Liu, X., Sun, M., and Zhou, Y. (2021). Electricity consumer characteristics identification: A federated learning approach. *IEEE Transactions on Smart Grid*, 12(4):3637–3647.

[Wes McKinney, 2010] Wes McKinney (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

[Yang et al., 2019] Yang, Q., Liu, Y., Chen, T., and Tong, Y. (2019). Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19.

[Zhang et al., 2021] Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., and Gao, Y. (2021). A survey on federated learning. *Knowledge-Based Systems*, 216:106775.

[Zheng et al., 2013] Zheng, J., Gao, D. W., and Lin, L. (2013). Smart meters in smart grid: An overview. In *2013 IEEE Green Technologies Conference (GreenTech)*, pages 57–64. IEEE.

[Ziller et al., 2021] Ziller, A., Trask, A., Lopardo, A., Szymkow, B., Wagner, B., Bluemke, E., Nounahon, J.-M., Passerat-Palmbach, J., Prakash, K., Rose, N., Ryffel, T., Reza, Z. N., and Kaissis, G. (2021). *PySyft: A Library for Easy Federated Learning*, pages 111–139.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**