Federated K-Means clustering

Swier Garst[1] and Marcel Reinders[1]

Delft University of Technology, 2600 AA, Delft s.j.f.garst@tudelft.com

Abstract. Federated learning (FL) is a technique that enables the use of distributed datasets for machine learning purposes without requiring data to be pooled, thereby better preserving privacy and ownership of said data. While supervised FL research has grown substantially over the last seven years, unsupervised FL methods remain scarce. This work introduces an algorithm which implements K-means clustering in a federated manner.

Keywords: Federated Learning · K-Means clustering · Distributed machine learning.

1 Introduction

Nowadays, lots of data is being generated in a distributed fashion. Mobile phones and other personal devices such as smart watches enable the collection of massive amounts of data. If made accessible, this data could prove useful for all sorts of classification tasks, thereby improving the performance of the services provided by these devices. However, due to a growing concern on data privacy, more and more users of these devices are hesitant in sharing their data. Furthermore, regulations such as the General Data Protection and Regulation (GDPR) act prevent the collection of data of this kind in bulk. Federated learning (FL) [14] was introduced as a solution to this problem. In short, instead of pooling data to train a single model, instances of a model are being shared to all data owners (clients), which then train the model on their local data. Then, these trained models are sent back to the central server, which aggregates the results. Next, a new round begins with the server sending out the updated models. This cycle continues until convergence. Over the past couple years, research has shown FL to be a promising technique, reaching performances comparable to a central approach (with all the data pooled/collected at a single location) [10] [16]. However, the vast majority of research has been focusing on the supervised learning paradigm. Little work has been done on unsupervised federated learning methods, even less so when specifically looking into clustering techniques [12] [11]. One of these clustering techniques is K-means clustering [5]. In a federated learning setting, K-means clustering can be described as trying to find overarching cluster means over data which is distributed among different datasets (clients). To the best of the authors' knowledge, Only a few papers discuss possible options for federated K-means clustering. [9] introduce a method which assumes availability of some data at the central server to pretrain the model, which is not always

feasible.[6] use secure multiparty computation and blockchain to share encrypted data on which k-means is being performed. However, for some cases (e.g. in the medical domain), even sharing encrypted data might not be acceptable. [13] introduced a method for federated k-means in the cross-device setting, i.e. where each client only holds one sample. [17] proposes a dual averaging approach for k-means clustering. [4] propose a one-shot method by doing a k-means clustering over the cluster means found locally. However, their method does not use a variable amount of local clusters. The method from [4] is a similar approach to ours, and will therefore be the main point of comparison. By introducing a variable amount of local clusters, we show that we can improve from [4], especially in settings where the amount of local clusters is highly varying between clients. Furthermore, we introduce a ruling which can automatically determine the amount of local clusters on each client, reducing the amount of parameters that need manual tuning. We show that in a 2-dimensional setting, a clustering of similar quality can be obtained compared to a conventional K-means clustering (on a pooled dataset). Although our method and experimental validation leaves enough room for further improvement, this work can serve as a basis for exploring clustering in the federated setting.

2 Related Work

This work combines the K-means clustering algorithm and the federated learning paradigm. In this section, both these concepts will be discussed in more detail.

2.1 K-means clustering

The objective of a clustering algorithm is to partition a given dataset into several subsets with similar features. The K-means clustering algorithm does so by trying to minimize the so-called within cluster sum-of-squares criterion:

$$F_{km} = \sum_{j=0}^{m} \min_{\mu_k \in C} (||X_{j,k} - \mu_k||^2) \tag{1}$$

with m the amount of samples, μ_k the cluster mean of cluster k, C the set of all cluster means and $X_{j,k}$ being data point j assigned to cluster k. The procedure in which the k-means algorithm tries to minimize equation 1 consists of two steps. First, all data points get assigned to the cluster mean with the lowest euclidean distance to said point. Then, the mean value from all points assigned to a certain cluster is calculated. This is done for every cluster, creating a new set of means to start the next round with. This process is repeated until the change within these means is smaller than a certain threshold (and the algorithm has reached convergence) [5].

This clustering approach requires a predefined amount of clusters, i.e. a value for k beforehand. Furthermore, it is prone to finding local optima based on the initialization of the mean values, which is classically done by randomly drawing k

samples from the dataset. Recently, more sophisticated methods of initialization have emerged, such as kmeans++ [1], which tries to create a better initialization by asserting a certain minimum distance between means.

2.2 Federated learning

The term Federated Learning (FL) was first coined in 2015 by Google [14] as they were looking into ways of using smartphone text data for their natural language processing models, without having to download said data. The result was a paradigm shift in which the model moves to the data instead of the other way around. In FL, there is a notion of a central server (without any data), which is connected to a set of local users (clients with data). The server is controlled by someone who wants to train a machine learning model, while the clients are devices owning a certain amount of data. First, the server sends out a certain machine learning model to (a subset of) clients, with weights randomly initialized. Then, clients train said model for one or more epochs on their local data. These trained models are sent back to the server. The server aggregates the models, before sending them out to the clients again, and the cycle continues. The key idea is that by aggregating models from multiple clients into a single model, the resulting model is better than one of its parts. The most widely used algorithm for aggregating these models is the one introduced by Google in 2015, called "federated averaging" [14]. In federated averaging, model parameters are aggregated by taking the average for all model parameters, weighted by the sample size of the respective client. Once converged, the resulting model usually gives comparable results compared with a model trained on all data pooled together, while the server has not had access to a single sample of data [10] [16].

3 Methods

A federated k-means clustering algorithm can be approached in a similar way compared to supervised federated learning methods as described in section 2.2. Clients perform k-means clustering on their local datasets, after which they send the resulting means to a central server. The server then aggregates these means in a certain way, before sending back a set of global means. clients can then use these means as a starting point for their next iteration of k-means. However, compared with the supervised federated learning methods, several key challenges arise:

- **Aggregating cluster means.** This cannot be done by simply taking the average of all clusters received by the server, as this would mix cluster means from completely different clusters.
- **Cluster alignment** As the server receives the values for cluster means from different clients, there is not a set way of knowing which of these means can be assumed to be from the same global cluster. This is in contrast with averaging on a model such as a neural network, where due to the structure of the model, parameters can be averaged between models.

– **Imbalance in local cluster amounts.** Datasets could be split in such a way that one client holds another set of clusters than another client. Not only is the local cluster amount something that needs to be detected without looking at the data (as we cannot access said data from the central server), we also need to ensure that, when a client receives the entire set of new global means from the server, it only uses the ones corresponding to its' clusters.

3.1 The algorithm

Table 1. notations used

symbol	description
X^i	data on client i
N_c^g	global number of clusters
N_c^i	number of clusters on client i
C^g	global cluster means
C^{g*}	updated global cluster means
C^i	cluster means of client i
C^{i*}	updated cluster means of client i
$D^i(C^g, C^i)$	distances between global clusters and local clusters on client i
s^i	amount of samples for each cluster on client i

Algorithm 1 The federated K-means algorithm

inputs: N_c^g

Initialization:
On Server do: t = Determine_t()
On all clients i **do:** N_c^i, C^i = determine_local_clusters(t)

Iterative K-means:
For all rounds r **do:**
  On all clients i **do:** $C^{i*} = kmeans(X^i, C^i)$
  On all clients i **do:** $s^i = [s^j]$ for each cluster $j \in C^{i*}$
  On all clients i **do: if** $1 \in s^i$ **then** remove corresponding cluster from C^{i*} and s^i
  On all clients i **do:** send C^{i*}, s^i to server
  On server do: $C^g = [C^1|C^2|..|C^m], s^g = [s^1|s^2|..|s^m]$
  On server do: $C^{g*} = kmeans(C^g, N_c^g, s^g)$
  On server do: send C^{g*} to all clients
  On all clients i **do:** $D^i = euclidean\_distance(C^{g*}, C^i))$
  On all clients i **do:** C^i = select N_c^i from C^{g*} with lowest D^i

Here, we present a federated K-means algorithm that addresses these challenges. Notation used throughout this section is found in table 1. The pseu-

Algorithm 2 determine_local_clusters()

1: inputs: t

2: $N_c^i = N_c^g$
3: **for** $j = 0$ to N_c^g **do**:
4: $C = kmeans++(X, N_c^i)$
5: $A = euclidean\_distance(C)$
6: **if** any value in $A_i < t$:
7: $N_c^i = N_c^i - 1$
8: **else** break
9: **return** N_c^i, C

docode for the federated K-means algorithm can be found in algorithm 1. The algorithm can be divided into two parts: an initialization step (lines 3 and 4), in which we determine the amount of clusters to expect on each client, and an iterative k-means step in which clients communicate their cluster means to the server, which aggregates these means into a 'global' set of means, which then gets redistributed to the clients for the next k-means iteration. This procedure is heavily inspired by the original federated averaging [14] approach for supervised federated learning.

Determining the amount of local clusters. While the global amount of clusters is set (main parameter k of the k-means procedure), it is not a given that each client has data in each of these clusters. In other words, the amount of clusters between clients can differ, and is not necessarily equal to the amount of clusters in the pooled data. In order to determine the amount of local clusters (N_c^i for each client i), we initialize the amount of clusters in the local dataset to the number of clusters to be found in the pooled data (input parameter k for k-means).This initialization is done using the kmeans++ algorithm [1], which ensures a certain distance between means. We then calculate distances between the cluster means. If one of these distances is smaller than a certain threshold (t), we lower the amount of local clusters by one and repeat the procedure. This process continues until all distances between the cluster means are bigger than t, at which point the amount of local clusters (N_c^i) is determined for the remaining part of the algorithm. The cluster values found in this final iteration of this procedure are used as initial values for the iterative k-means procedure. Algorithm 2 gives the pseudocode for this initialization of the local amount of clusters.

Threshold t is a function of the variance in all dimensions, as well as the global amount of clusters to be found (N_c^g):

$$t = \left(\frac{\prod_{j=0}^{d} 2\sigma_j}{N_c^g}\right)^{\frac{1}{d}} \tag{2}$$

with d the dimension of the data, and σ_j the standard deviation of feature j. The factor 2 ensures 95% of data is covered, so roughly speaking, equation 2 describes the volume of the data divided by the preset number of clusters. We then

take the $d^t h$ root of this volume, in order to arrive at a value that is comparable to the one-dimensional distance that t is being compared with in algorithm 2. Note that this threshold can be determined in a federated way, as the variance over features can be easily determined federatively. Further, we would like to stress that the initialization procedure does not perfectly find the amount of clusters present at each client. In fact, due to the stochasticity of initializing cluster means, the amount of local clusters found when running the algorithm multiple times can vary substantially.

Iterative K-means The iterative K-means algorithm (Shown in Algorithm 1 first starts by determining the local amount of clusters. That procedure also returns the cluster means of the local clusters After that the iterative communication loop starts, which resembles the structure of an original supervised federated learning paradigm. Each client calculates one iteration of K-means on their local data (each with their own amount of local clusters), after which they send their cluster means and amount of samples per cluster back to the server. The server then concatenates all cluster means, and aggregates them. It does so by running a K-means clustering on the received local means, to align clusters from different clients to each other. This global k-means step is weighted by the amount of samples per cluster found, such that a cluster with lots of samples in it will have a bigger impact on the aggregation step compared to a cluster with fewer samples. That is, we modify the objective function from equation 1 into:

$$F_{km} = \sum_{j=0}^{m} \min_{\mu_k \in C} (||s_k * X_{j,k} - \mu_k||^2) \tag{3}$$

where s_k is the amount of samples corresponding to cluster k. Note that the *cluster means* sent back by the clients are at the server used as the *samples* for clustering using k-means. Doing the aggregation with a k-means clustering, we solve both the aggregation as well as the cluster alignment problem, since similar clusters will be close to each other and thus merged by the global k-means step.

After the server creates a new set of global means, it sends these values back to the clients. These clients then use these global means as initial values for a new k-means step on their local data. However, the server calculates N_c^g new cluster means, while a client only expects N_c^i (the local amount of) clusters. Each client therefore selects the N_c^i most suitable cluster means from the list of N_c^g means that were sent by the server. It does so by calculating the distance between each global mean sent by the server and its set of local cluster means from previous iteration.

Because the amount of samples have to be reported to the central server, there exists a privacy risk if a client finds a cluster with only one sample in it, since in this case the sample mean reported to the server would be the same as the values of said sample. Since it is also known (at the server) that this cluster holds only one sample, the server thereby learns the exact values of this sample. To prevent this, a check is done to prevent any clusters holding only one sample

to be send to the server; these cluster means are simply omitted from the list of means sent to the server.

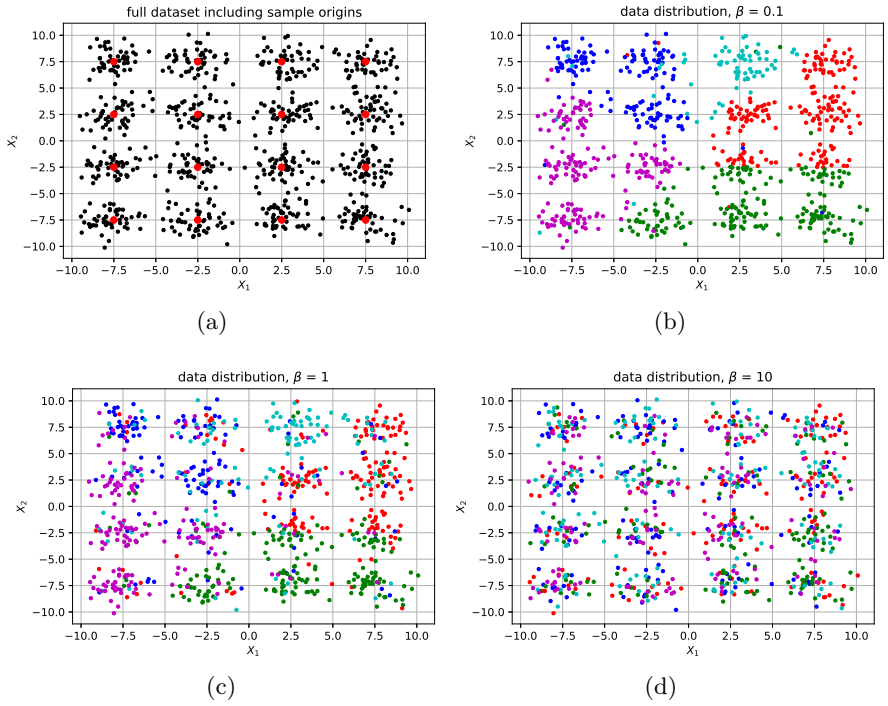


Fig. 1. The dataset used. (a) shows the original sampling of the dataset, with the sampling origins in red. (b) until (d) show how this dataset is separated into five different subsets for different values of β. Different colors denote the different clients

3.2 Experiments

In order to validate the federated K-Means algorithm, two synthetic two-dimensional datasets were generated, one of which was distributed (among the five clients) in four different ways. Furthermore, to test performance on higher dimensional data, an experiment with a subset of FEMNIST [2] was performed. For the two-dimensional datasets, performance was measured using the Adjusted Rand Index (ARI) [8]. The silhouette score [15] was used as a performance metric for the FEMNIST dataset, as there is no labelling for the clustering. The experiments compared our method with the method from [4], as well as a central clustering with all data available at one location.

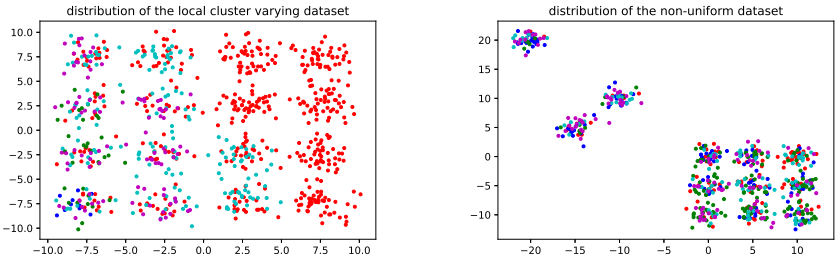
Regular synthetic dataset. In order to validate the federated K-Means algorithm, a synthetic two-dimensional dataset was generated. The generation

procedure is taken from [17]. Sixteen cluster centers were chosen with an equal distance from one another, see fig. 1(a). Then, 50 data points were sampled around each cluster center using a normal distribution (with variance 1). This data was then distributed among four clients in the following way: First, each client is assigned a 'location' within the field ($X_1, X_2 \in (-12.5, 12.5)$). From there, the probability P that a data point would be assigned to a certain client scales inversely with the euclidean distance d to said datapoint [17]:

$$P = 1 - exp(-\frac{\beta}{d}) \tag{4}$$

where β is a parameter which can be tuned to promote more or less heterogeneity in the data separation. Differing from [17], if a data point happens to be assigned to multiple clients, it instead gets assigned at random. Three different distributions of the data were generated, for $\beta = 0.1, 1, 10$. See figure 1 for the final distributions.

High variability in local clusters. Next we wanted to explore the effect of having a variable local k. The hypothesis was that this effect would be best noticable on a dataset where the amount of clusters that each client has access to, is highly variable. we therefore used the same data as generated for the regular synthetic dataset, but distributed even more heterogeneously, such that each client only had data from 1, 4, 7, 10, 16 clusters, respectively. see fig. 2(a).



(a) (b)

Fig. 2. The two additional two-dimensional synthetic datasets. (a) shows a high variability in local clusters, whereas (b) is constructed to 'break' the automatic determination of t in our algorithm.

Non-uniform synthetic dataset. Until now, datasets have had their cluster centers uniformly distributed among the 2d plane. This is the ideal case given how t is determined in our algorithm. We therefore wanted to see how our algorithm would perform if cluster centers were a bit more scattered around. We used the same procedure as for the first dataset, however cluster centers were chosen to not be in an uniform grid. see fig. 2(b).

FEMNIST. In order to determine performance on a higher dimensional dataset, the Federated Extended MNIST (FEMNIST) from LEAF [2] was used, which separates the original Extended MNIST [3] handwritten numbers and letters based on the person who wrote them. Only 10 clients were used from the original FEMNIST, as this drastically sped up the experiments, while keeping enough data for a meaningful assessment.

4 Results

The federated approach with the local datasets (referred to as 'own method') is compared to the scenario where a K-means clustering is executed on all data centrally, as well as to the method of Dennis et. al., for which we tuned their parameter the local number of clusters (k_{local}) for each experiment. We calculate the Adjusted Rand Index (ARI) for both central and federated approaches with respect to the labelled samples data. Since there are no labels for the clustering in the FEMNIST experiment, the silhouette score was used instead.

Regular synthetic dataset. We generated three versions of this dataset as described in section 3.2. Note that β only changes which points get assigned to which client, meaning that it does not influence the performance for the central case. Figure 3(a) shows that, for higher values of β, our method decreases in variance compared to the method from Dennis et. al. . However, both methods still underperform compared to centralized clustering.

High variability in local clusters. The previous experiment hints at the usefulness of having a local k that can vary between clients. We wanted to explore this further, for which we redistributed the previous dataset in an even more imbalanced way, as shown in figure 2(a). Figure 3(b) shows indeed an increase in performance compared to the method from [4].

Non uniform clustering. The last synthetic dataset was made to explore the limits of how parameter t is determined in our algorithm, (equation 2). This equation suggests that when clusters do not lie evenly spaced, the algorithm might perform worse. We first ran our own algorithm using the value of t corresponding to equation 2 (resulting in $t = 5.29$). Then, we optimized t manually to the value for which the best performance was found, which is $t = 1$. Figure 3(c) shows the expected performance drop for $t = 5.29$, while being able to attain a performance similar to central when tuning for t.

FEMNIST. Finally, we wanted to test our algorithm on real-world, higher dimensional data. For this, a subset of the FEMNIST dataset from the LEAF federated benchmark was used. Similar to the last experiment on synthetic data, first the 'natural' value for t (according to equation 2) was used, after which it was tuned for best performance (which was at $t = 3$). 3(d) shows that, when allowed tuning for t, our method slightly outperforms the method from Dennis et. al., while still under-performing compared to central clustering.

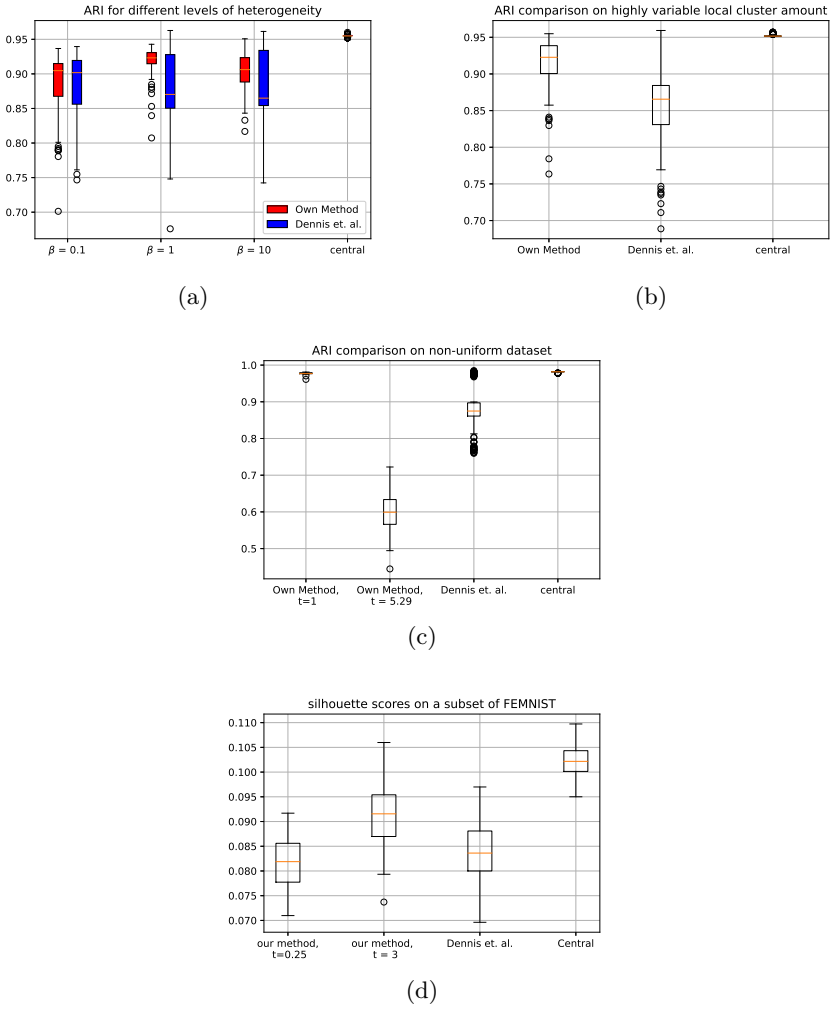


Fig. 3. results from all experiments. (a), (b), and (c) are from 200 runs, (d) only used 100 runs. (a) shows the ARI for the 'original' synthetic dataset, with different distributions amongst clients. (b) shows this same dataset, but with an even more aggressive distribution compared to the different settings in (a). (c) shows the ARIs on the synthetic dataset where clusters aren't evenly spaced, while (d) shows the silhouette scores on (a subset of) FEMNIST.

5 Discussion and conclusion

This work describes the implementation and validation of a federated K-means clustering algorithm, enabling clustering over multiple datasets without sharing the underlying data. Our results show performances approaching a central

method, in which all data is brought into a single location. There is still a gap between federated and central ARI scores. However, when allowing tuning for hyperparameter t instead of using equation 2, a performance comparable to central clustering can be found in two-dimensional data. This suggests that our automatic rule for determining this hyperparameter (or k_{local}, which is indirectly determined by t) might not be optimal. Automatic detection of amount of clusters present in a dataset is a hard problem, on which a lot of literature has been produced already. Our method tries to take advantage of the information present due to the federated setting, i.e. the amount of global clusters present. Nevertheless, finding a better way to determine the amount of local clusters present seems a promising avenue for improving this algorithm in future work, as tuning a parameter such as t manually might not always be feasible in practice.

The FEMNIST experiments use the silhouette score as their performance metric. The silhouette score involves calculating distances from each point in a dataset to each other point in a dataset. This means that, to calculate a 'global' silhouette score, distances between datapoints from different clients need to be determined, something that can not be done in a straightforward manner. In our case, the simulated federated environment used made it possible to calculate the silhouette score for evaluation purposes. In a real-life setting, the simplified silhouette score [7] could be a suitable alternative, as it only calculates distances between datapoints and cluster means, something which can be done on all clients separately.

One of the advantages of [4] is that it is a one-shot method. Although our method is iterative, a quick inspection of the ARI scores over time showed no particular increase after the first or second round. As our method has resemblances to [4], this observation was not too surprising. Nevertheless, there might be situations in which being able to iterate increases performance, something which our method is capable of doing.