FEDERATED K-MEANS: CLUSTERING ALGORITHM AND PROOF OF CONCEPT

A PREPRINT

Oskar J. Triebe
Stanford University
triebe@stanford.edu

Ram Rajagopal
Stanford University
ramr@stanford.edu

September 4, 2019

ABSTRACT

An algorithm to cluster distributed datasets without communicating data is introduced. It builds upon the concept of federated learning, distributed k-means and mini-batch k-means. Results on synthetic data and real data are presented for a non-iid setting. The algorithm is able to produce a clustering of similar or better quality than central k-means clustering, while preserving privacy.

***K*eywords** Federated Learning · Clustering · Machine Learning · K-Means · Privacy

1 Introduction

Clustering data that is semi-private is a common challenge in many real-world settings. Past work has mainly focused on entirely decentralized protocols or on centralized approaches. A fully decentralized solution to the issue was introduced in 2013[1]. However, fully distributed approaches are prone to communication issues or may be inefficient. Recently an approach to solve such tasks with a trusted third party, without actually sharing the data, called Federated Learning [2] was introduced. We apply this approach to Clustering, as this is most similar to a real-world setting, where privacy is valued, but a partially trusted third party exists. Our algorithm is also inspired by mini-batch K-Means Clustering, first introduced in 2010[3], which was proven to be an effective solution to large scale dataset clustering in 2016[4].

2 Methods

2.1 Data

The data is synthetically created with random noise:

- 50 non-iid client datasets
- each client's dataset is randomly sampled from one of 5 Gaussians with means 1, ..., 5 and scale of 0.2
- 1 dimensional
- 10 samples per client

2.2 Metrics

Mean and standard deviation of mean euclidean intra-cluster distance and of Davies-Bouldin Index are reported for for 10 runs with different random seeds.

2.3 Algorithm

Algorithm 1: `FederatedKMeans`: Federated algorithm to cluster distributed data into k clusters. The M clients are indexed by i. $\mathcal{P}_i$ is the dataset held by client i. C is the fraction of clients sampled in each round. T and E are the number of global rounds and local epochs. B is the local mini-batch size, default is $B = \infty$ for full-batch steps. α and β are the server and client learning rates. Γ depicts the set of K cluster centroids ($\mu_1, ...\mu_k \in \mathbb{R}^m$). Λ depicts the set of K cluster sizes ($\lambda_1, ..., \lambda_k \in \mathbb{R}$). Operations over Γ and Λ are executed element-wise.

Server executes:

initialize cluster centroids $\Gamma_{t=0}$ randomly.
$m \leftarrow max(C \cdot M, 1)$
for *each round* $t = 1, ..., T$ **do**
  $S_t \leftarrow$ (random set of m clients)
  for *each client* $i \in S_t$ ***in parallel*** **do**
    $(\Lambda_t^{(i)}, \Gamma_t^{(i)}) \leftarrow \texttt{ClientKMeans}(i, \Gamma_{t-1})$
  $\Lambda_t = \sum_{i=1}^{m} \Lambda_t^{(i)}$
  $\Gamma_t^* = \frac{1}{\Lambda_t} \cdot \sum_{i=1}^{m} \Lambda_t^{(i)} \cdot \Gamma_t^{(i)}$
  $\Gamma_t = \Gamma_{t-1} + \alpha \cdot (\Gamma_t^* - \Gamma_{t-1})$

Client executes:

`ClientKMeans`(i, Γ)
  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_i$ in batches of size B)
  for *each epoch* $e = 1, ..., E$ **do**
    initialize cluster sizes Λ to zero.
    randomly shuffle order of batches in $\mathcal{B}$
    for *each batch* $b \in \mathcal{B}$ **do**
      for *each data* $x_j \in b$ ***in parallel*** **do**
        $c_j = \arg\min_k ||x_j - \mu_k||^2$
      $\Lambda^b = \sum_{j=1}^{B} 1_{\{c_j = k\}}$
      $\Gamma^b = \frac{1}{\Lambda^b} \cdot \sum_{j=1}^{B} 1_{\{c_j = k\}} \cdot x_j$
      $\Lambda = \Lambda + \Lambda^b$
      $\Gamma = \Gamma + \beta \cdot \frac{\Lambda^b}{\Lambda} \cdot (\Gamma^b - \Gamma)$
  return Λ, Γ

If a cluster has a size smaller than $0.01 \cdot \sum_i^M |\mathcal{P}_i|$ for 20 consecutive rounds, it is randomly reassigned.

3 Results

3.1 Varying Sample Fraction

The Federated clustering algorithm is able to match or outperform the central clustering algorithm if the hyperparameters are appropriately set. The performance of the central clustering approach is reported as "central" sample-fraction.

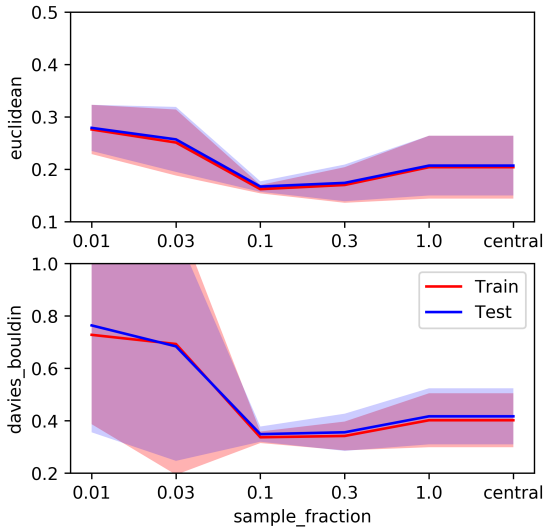
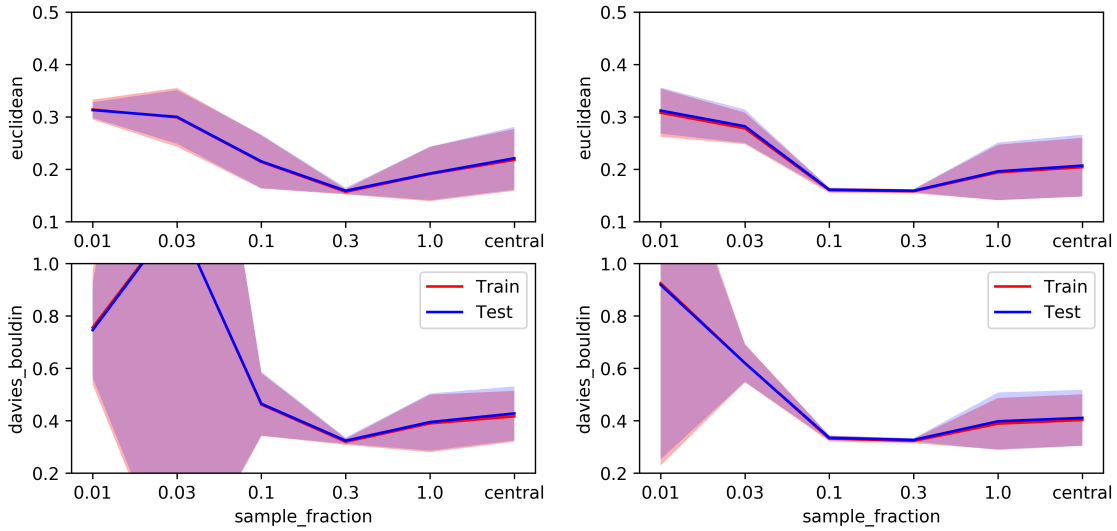
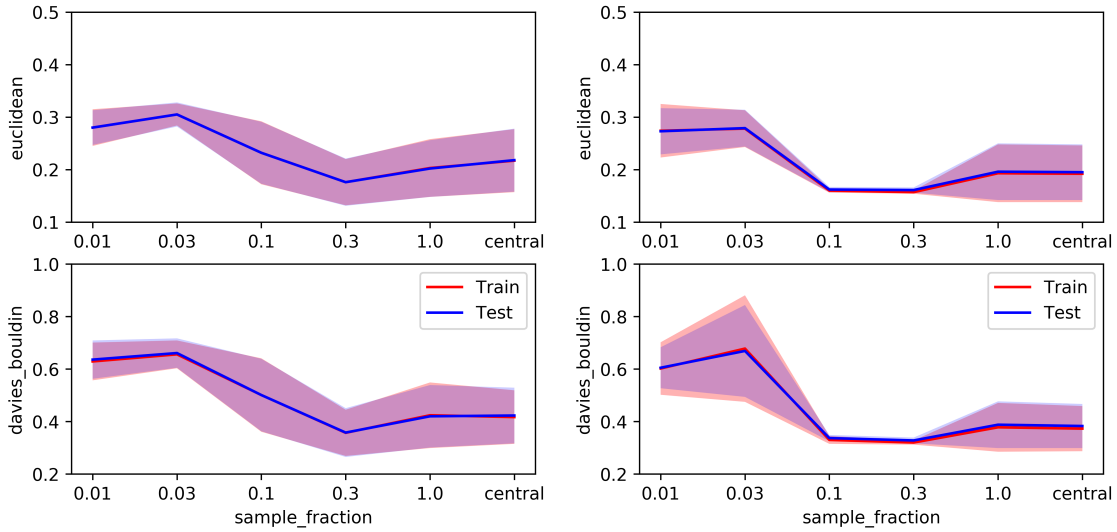


Figure 1: Hyperparameters: $B = \infty, E = 1, T = 200, \alpha = (1, 0.5), \beta = 1.0$.



(a) Hyperparameters: $B = \infty, E = 10, T = 200, \alpha = (1, 0.5), \beta = 1.0$.

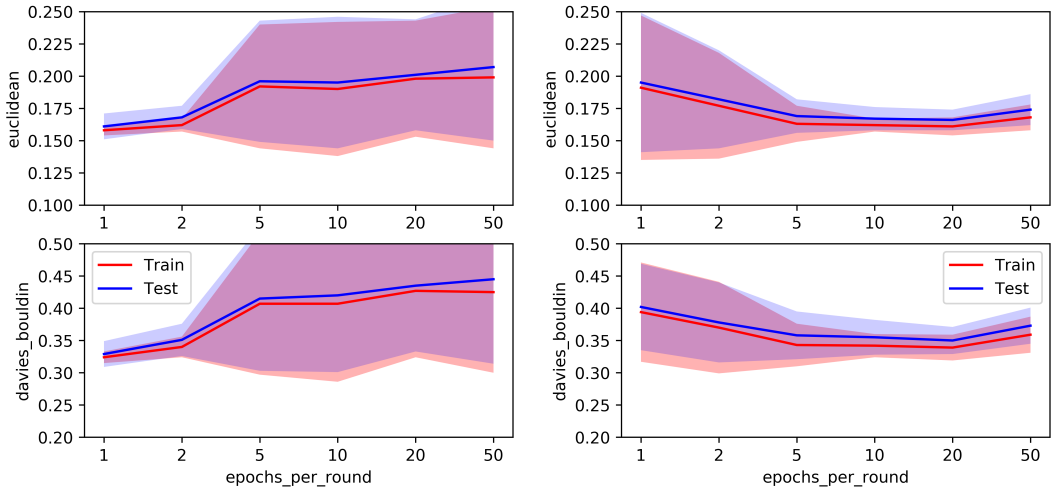
(b) Hyperparameters: $B = \infty, E = 10, T = 200, \alpha = (1, 0.5), \beta = 0.1$.



(a) Hyperparameters: $B = \infty, E = 5, T = 200, \alpha = (1, 0.5), \beta = 1.0$.

(b) Hyperparameters: $B = \infty, E = 5, T = 200, \alpha = (1, 0.5), \beta = 0.2$.

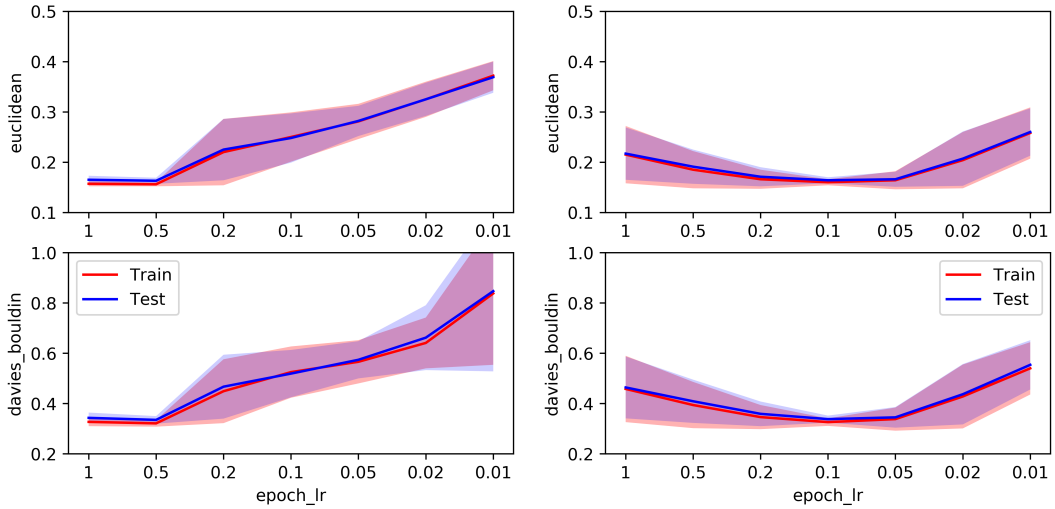
3.2 Varying Client Epochs



(a) Hyperparameters: $B = \infty, C = 0.1, T = 200, \alpha = (1, 0.5), \beta = 1.0$.

(b) Hyperparameters: $B = \infty, C = 0.1, T = 200, \alpha = (1, 0.5), \beta = 0.1$.

3.3 Varying Client Learning Rate



(a) Hyperparameters: $B = \infty, C = 0.1, T = 200, \alpha = (1, 0.5), E = 1$.

(b) Hyperparameters: $B = \infty, C = 0.1, T = 200, \alpha = (1, 0.5), E = 10$.

3.4 Progress Visualization

3.4.1 Central Cluster Learning

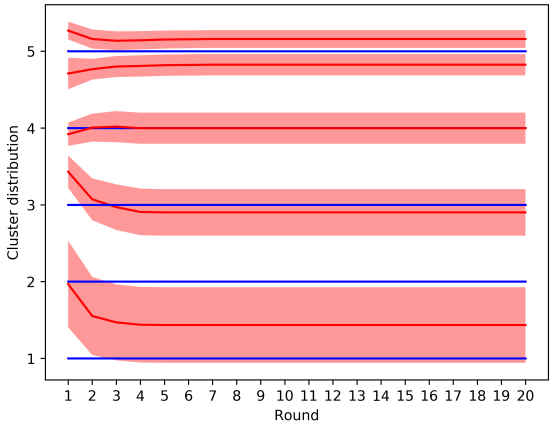
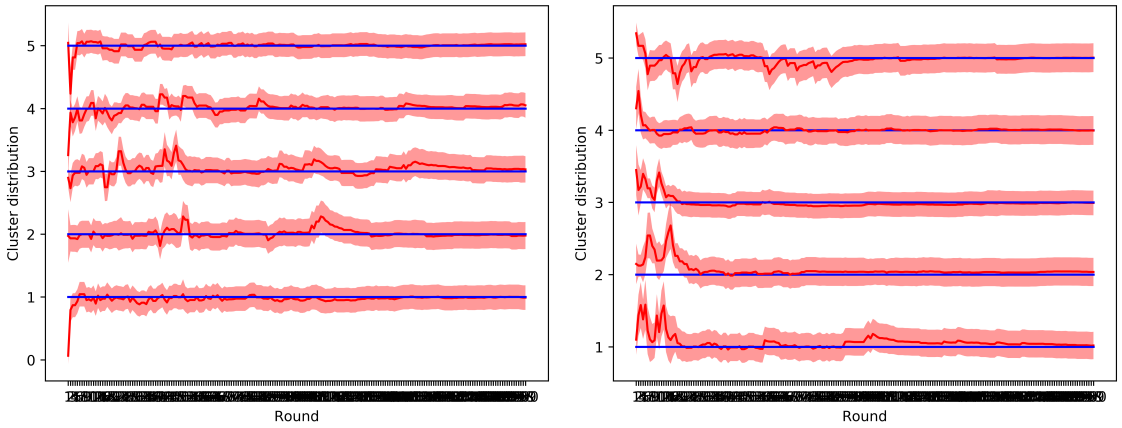


Figure 6: K-means Algorithm. Hyperparameters: $B = \infty, E = 1, T = 20, \alpha = NA, \beta = NA$.

3.4.2 Federated Cluster Learning



(a) Hyperparameters: $B = \infty, E = 1, T = 200, \alpha = (1, 0.5), \beta = 1$.

(b) Hyperparameters: $B = \infty, E = 10, T = 200, \alpha = (1, 0.5), \beta = 0.1$.

Figure 7: Federated Clustering Algorithm.

4 Conclusion

The Federated clustering algorithm is able to match or outperform the central clustering algorithm if the hyperparameters are appropriately set.

Further preliminary experiments on real energy datasets indicated that the Algorithm is simple and effective to apply to industrial problem settings (results not included).

Note This is unpolished work, done in May to September 2019. Nevertheless, we decided to share the findings already with the broader science community as others might find use in our simple algorithm.

Acknowledgment The work presented herein was funded in part by Total S.A in a research agreement with Stanford University. The views and opinions of authors expressed herein do not necessarily state or reflect those of the funding source.

References

[1] Gabriele Oliva, Roberto Setola, and Christoforos N. Hadjicostis. Distributed k-means algorithm, 2013.

[2] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2016.

[3] D. Sculley. Web-scale k-means clustering, 2010.

[4] Cheng Tang and Claire Monteleoni. Convergence rate of stochastic k-means, 2016.

5 Appendix

5.1 Algorithm (non-vectorized)

Algorithm 2: `FederatedKMeans`: Federated algorithm to cluster distributed data into k clusters. The M clients are indexed by i. $\mathcal{P}_i$ is the dataset held by client i. C is the fraction of clients sampled in each round. T and E are the number of global rounds and local epochs. B is the local mini-batch size, default is $B = \infty$ for full-batch steps. α and β are the server and client learning rates. The K clusters have centroids $\mu_1, ...\mu_k \in \mathbb{R}^m$ and cluster sizes $(\lambda_1, ..., \lambda_k \in \mathbb{R})$.

Server executes:

initialize k cluster centroids $(\mu_{1,t=0}, ..., \mu_{k,t=0} \in \mathbb{R}^m)$ randomly.
$m \leftarrow max(C \cdot M, 1)$
for *each round* $t = 1, ..., T$ **do**
  $S_t \leftarrow$ (random set of m clients)
  for *each client* $i \in S_t$ ***in parallel*** **do**
    $(\lambda_1^{(i)}, ..., \lambda_k^{(i)}), (\mu_1^{(i)}, ..., \mu_k^{(i)}) \leftarrow$ `ClientKMeans`$(i, (\mu_{1,t-1}, ...\mu_{k,t-1}))$
  for *each cluster* $k = 1, ..., K$ ***in parallel*** **do**
    $\lambda_{k,t} = \sum_{i=1}^{m} \lambda_k^{(i)}$
    $\mu_k^* = \frac{1}{\lambda_{k,t}} \cdot \sum_{i=1}^{m} \lambda_k^{(i)} \cdot \mu_k^{(i)}$
    $\mu_{k,t} = \mu_{k,t-1} + \alpha \cdot (\mu_k^* - \mu_{k,t-1})$

Client executes:

`ClientKMeans`$(i, (\mu_1, ..., \mu_k))$
  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_i$ in batches of size B)
  for *each epoch* $e = 1, ..., E$ **do**
    initialize cluster sizes $(\lambda_1, ..., \lambda_k)$ to zero.
    randomly shuffle order of batches in $\mathcal{B}$
    for *each batch* $b \in \mathcal{B}$ **do**
      for *each data* $x_j \in b$ ***in parallel*** **do**
        $c_j = \arg\min_k ||x_j - \mu_k||^2$
      for *each cluster* $k = 1, ..., K$ ***in parallel*** **do**
        $\lambda_k^b = \sum_{j=1}^{B} 1_{\{c_j = k\}}$
        $\mu_k^b = \frac{1}{\lambda_k^b} \cdot \sum_{j=1}^{B} 1_{\{c_j = k\}} \cdot x_j$
        $\lambda_k = \lambda_k + \lambda_k^b$
        $\mu_k = \mu_k + \beta \cdot \frac{\lambda_k^b}{\lambda_k} \cdot (\mu_k^b - \mu_k)$
  return $(\lambda_1, ..., \lambda_k), (\mu_1, ..., \mu_k)$