PPEM using Homomorphic encryption

Prof. Adi Akavia's Secure Cloud Computing Laboratory, Fall 2022-2023

Supervised by Prof. Adi Akavia

Abed-Elrahman Abu Hussein, Michael Rodel

Abstract

This paper introduces a Privacy Preserving Expectation Maximization (PPEM) protocol that addresses privacy concerns in distributed algorithms. The protocol combines the Expectation-Maximization (EM) algorithm and the Gaussian Mixture Model (GMM) to find maximum likelihood estimates in statistical models with hidden variables. The proposed protocol ensures privacy and integrity of data across distributed parties, such as hospitals collecting disease data. It employs homomorphic encryption, particularly the CKKS encryption, which allows computation on encrypted data. The algorithm divides optimization formulas into sections and uses CKKS encryption to exchange information with a central server. The server combines encrypted intermediate updates, maintaining privacy, and enabling collaborative EM computation.

1 Introduction

1.1 Introduction to Expectation Maximization (EM) Algorithm

The Expectation-Maximization (EM) algorithm using GMM (general mixture model) is a computational technique used for finding maximum likelihood or maximum a posterior estimates in statistical models where some of the variables are hidden or unobserved. It's particularly useful when dealing with problems involving missing data or latent variables. Moreover, it is considered as one of the unsupervised soft clustering algorithms.

1.2 Introduction to Privacy Preserving Expectation Maximization (PPEM) Protocol

Privacy Preserving Expectation Maximization (PPEM) Protocol is a privacy preserving variant of the Expectation Maximization (EM) algorithm that aims to preserve the privacy and integrity of the data when using distributed algorithms (the data is distributed amongst several parties) in which the privacy of each parties data is at risk, thus creating the need for a privacy preserving variant of the distributed em algorithm (consider for instance hospitals across cities and countries collecting data on diseases). It becomes a non trivial matter when we take into account the different data protection laws of each country into consideration. Another example is some research which needs a variety of samples from different regions for the research to be accurate, creating a need for a privacy preserving variant of the EM algorithm.

In this paper we aim to address those privacy issues when joint computations from different parties occur. In our implementation we aim to fit a Gaussian Mixture Model

(GMM) based on a dataset that is distributed among different parties securely. Hence, the parties want to fit a GMM for the combined dataset from all parties without having to disclose each individual parties data or partial data. Most approaches to a PPEM solution can be divided into 3 categories:

1. Homomorphic encryption based approaches: those solutions are known to be computationally demanding and time consuming, see [1, 17, 10].
2. Differentially private approaches: In those approaches there is an inevitable trade-off between privacy and accuracy of the output,see 19-20
3. Secure summation approach: The main idea of the work [13] is to apply a secure summation protocol in each M step for protecting privacy. The drawback is that it requires to first detect a so-called Hamiltonian cycle. In addition, this protocol is very vulnerable to security attacks since it depends on a single node to keep the encryption seed.for more implementations of above method see [15, 18].

In order to address the limitations of the above approaches in the papers above, we propose a new privacy preserving federated EM protocol that puts the burden of the computation on a central server and isolates each party from the other, allowing only communications with the server. When we inspect the updating functions of the EM algorithm, we observe that the M step is the cause of privacy concern as it requires information exchange between different parties. To address this issue we divide the optimization formulas into sections and then use CKKS [6] encryption to send them to the server. The server adds the required fields together and sends them back to each party. The only information that is disclosed between the clients are a summation of vectors, and the server sums those vectors together and sends them back to each client making the protocol a collaborative EM protocol. We will show in this paper the privacy of the protocol and its federated version's weaknesses and strengths in privacy.

2 Preliminaries

2.1 Background and EM Algorithm

In this section, we present some essential background material that lays the foundation for understanding the basics of the EM (Expectation Maximization) algorithm, which is pivotal for our developed PPEM (Privacy Preserving Expectation Maximization) protocol.

2.2 Expectation Maximization

2.2.1 Background and EM Algorithm on 1-Dimentional Data

We aim to divide available information into 2 clusters and characterize each cluster using parameter θ. Our goal is to capture as much information as possible. see [4] we will now explain the algorithm on 2 clusters for an initial understanding of the algorithm.

2.2.2 Probability and Redundancy Function

We start by defining the probability of observing a data point x :

$$Pr(X;\theta)$$

A redundancy function $Pr(X;\theta)$ involves parameter θ and employs a probability model for observed data X.

Let X be the observed data set, divided into 2 clusters, each following a normal distribution $N(\mu,\sigma^2)$. We define the probability of cluster 2 as π.

We have 5 parameters:

- Probability π.
- μ_1.
- σ_1.
- μ_2.
- σ_2.

Collectively, these parameters are denoted as $\theta ::= \pi, \mu_1, \sigma_1, \mu_2, \sigma_2$.

We use the normal distribution function ϕ and define:

$$p(x;\theta) = \pi\phi(x;\mu_1,\sigma_1) + (1-\pi)\phi(x;\mu_2,\sigma_2)$$

The likelihood function is defined as $L(\theta; data)$.

Given observed data $x_1,\ldots,x_n$, we aim to find θ' that maximizes the likelihood of the data set:

$$\theta^{'} = \arg\max_{\phi} Pr(x_1,..,x_n;\theta)$$

To simplify derivatives for products, we work with the log-likelihood function:

$$l(\theta; data) = \log L(\theta) = \sum_{i=1}^{n} \log Pr(x_i;\theta)$$

Our objective is to choose parameters to maximize this.

2.2.3 Connection to K-Means (Unsupervised clustering)

The k-means algorithm, used in data clustering, assigns observations to k clusters based on centroids. This is similar to the EM algorithm's expectation phase.

We introduce a latent variable Δ. If a point is in cluster 2, Δ is 1, else 0. This represents the hard estimation phase.

Updating centroids corresponds to the maximization phase. Maximum likelihood estimates are straightforward when Δ is known. On the other hand, EM uses soft clustering and is not a binary 1.0 function that assigns a data point to a specific cluster, but assigns it with a probability, meaning data point x corresponds to X percent to cluster k.

2.2.4 EM Algorithm Workflow

In our example, we use a soft assignment of Δ, called "responsibility" γ:

$$\gamma_i(\theta) = E[\Delta_i|\theta, data] = Pr(\Delta_i = 1|\theta, data)$$

Algorithm steps:

1. Initialize parameter estimates θ: $\pi, \mu_1, \sigma_1, \mu_2, \sigma_2$. 2. Expectation: Calculate γ_i for each observation. 3. Maximization: Update parameter estimates using maximum likelihood. 4. Repeat steps 2-3 until convergence.

2.2.5 General EM Algorithm

1. We have data X. 2. Assume latent data Δ. 3. Model with parameters θ. 4. Calculate log-likelihood $l(\theta; X, \Delta)$. 5. Calculate conditional distribution $\Delta|X$. 6. Express $P(\Delta, X|\theta) = P(\Delta|X, \theta)P(X|\theta)$. 7. Use logarithms to derive $log[P(X|\theta)] = Q(\theta, \theta') - R(\theta, \theta')$.

2.2.6 Algorithm Correctness

We consider observed data x and latent variables z. Joint probability is $p(x, z|\theta)$. Conditional probability formula implies:

$$log[p(x|\theta)] = log[p(x, z|\theta)] - log[p(z|x, \theta)]$$

Algorithm correctness proof is based on the concept that:

$$Q(\theta_j, \theta_{j-1}) \geq Q(\theta_{j-1}, \theta_{j-1})$$

ensuring the increase in log-likelihood.

2.3 Federated EM Algorithm

The federated learning is motivated by the privacy concern as it does not allow transmitting the private data but only intermediate updates. The concept of federated learning is widely used as a method to train a global model over multiple nodes without sharing each node's local data directly. It does so by each node training a local model and subsequently all local models are combined to a global model. In our learning mission, we aim to learn the parameters of a data generated by a Gaussian mixture model, which is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters.

2.3.1 Fundamentals of GMM and EM over networks

We first introduce the basics of the GMM (Gaussian Mixture Model). Let there be n clients, s.t. each client has its own x_i data, where $i \in 1, 2, ..., n$. The GMM density is given by: $p(x) = \sum \beta_j p(x|\mu_j, \Sigma_j)$ where $p(x|\mu_j, \Sigma_j)$ is the PDF function for the Gaussian distribution, Σ_j is the covariance matrix, μ_j is the covariance and the mixture coefficient is $\beta_j | j \in C$, where C is the number of Gaussian models (clusters). The goal is to estimate Σ_j ,μ_j

and β_j for each $j \in C$. To do that over a network we should first know how to model a network. Networks are generally modeled as undirected graphs $G = V, E$ where $E \subset V \times V$, and $V = 1, 2, .., n$ nodes , s.t. each node can only communicate with another node through an edge that connects it to the other node.
Now, in order to estimate the latent variables through the EM algorithm with the available data of each node we follow the next steps for iteration t:
E-step:

$$P(x_i|N_j^t) = \frac{p(x_i|\mu_j, \Sigma_j)\beta_j^t}{\sum_{k=1}^{c} p(x_i|\mu_k, \Sigma_k)\beta_k^t} \tag{1}$$

M-step:

$$\beta_j^{t+1} = \frac{\Sigma_{i=1}^{n} P(x_i|\mathcal{N}_j^t)}{n} \tag{2}$$

$$\mu^{t+1} = \frac{\Sigma_{i=1}^{n} P(x_i|\mathcal{N}_j^t)x_i}{\Sigma_{i=1}^{n} P(x_i|\mathcal{N}_j^t)} \tag{3}$$

$$\Sigma_j^{t+1} = \frac{\Sigma_{i=1}^{n} P(x_i|\mathcal{N}_j^t)(x_i - \mu_j^t)(\mathbf{x_i} - \mu_\mathbf{j}^\mathbf{t})^\top}{\Sigma_{i=1}^{n} P(x_i|\mathcal{N}_j^t)} \tag{4}$$

where $P(x_i|\mathcal{N}_j^t)$ denotes the conditional probability that data x_i belongs to Gaussian model j. In order to compute the updates in (3) in a distributed manner, we need to collect the data over the network. One straightforward way is for each node to publish its individual dataset to all neighboring nodes. However, directly sharing the individual data would violate the privacy (against passive adversary, but even more against a malicious one).

2.4 Private data and Adversary models

We define the private data to be the individual data held by each client (i.e. node) i as it often contains sensitive information. For example, a person's health condition indicator, like person's blood pressure. Therefore, each client (i.e. node) i would like to prevent the exposure of its private data x_i to other nodes (the other clients and also the server) during the computation, see [16]. Second, we define the type of the adversary model we should protect against - the passive adversary model:
This model controls a number of corrupted clients who are assumed to follow the protocol but can collect information together. It will then use the collected information to infer the private data of the non-corrupted clients, which we will refer to as honest clients.
As a result, we would like our data proceeded in the protocol to be computational indistinguishable.
First, we will define probability ensemble, as defined in [9]:
probability ensemble is an infinite sequence of random variables $\{X_n\}_{n \in N}$ such that each X_n ranges over strings of length bounded by a polynomial in n.
Second, we will define computational indistinguishability, as defined in [8] : Let $X = \{X_n\}_{n \in N}$ and $Y = \{Y_n\}_{n \in N}$ be probability ensembles such that each X_n and Y_n ranges

over strings of length n. We say that X and Y are computationally indistinguishable if for every feasible algorithm A the difference $d_A(n) \overset{\text{def}}{=} |Pr[A(X_n) = 1] - Pr[A(Y_n) = 1]|$ is a negligible function in n (i.e., $f : \mathbf{N} \rightarrow \mathbf{R}^+$ is negligible if for any polynomial p the function $f(\cdot)$ is bounded above by $\frac{1}{p(\cdot)}$, as defined in [8]).

2.5 Privacy preserving protocol for distributed EM Algorithm

Putting things together, a privacy preserving protocol designed for the distributed EM algorithm in the context of Gaussian Mixture Models (GMM) should adhere to the following pair of conditions:
• Individual privacy: throughout the algorithm execution, each honest client's private data should be protected from being revealed to the adversaries. We assess the individual privacy using mutual information $I(\overline{X};\overline{Y})$, which measures the mutual dependence between two random variables $\overline{X}, \overline{Y}$. We have that $0 \leq I(\overline{X};\overline{Y})$, with equality if and only if $\overline{X}$ is independent of $\overline{Y}$. On the other hand, if $\overline{Y}$ uniquely determines $\overline{X}$ we have $I(\overline{X};\overline{Y}) = I(\overline{X};\overline{X})$ which is maximal.
• Output correctness: the fitted model, i.e., the estimated parameters of GMM, should be the same as using the non-privacy preserving version. In other words, ensuring privacy should not compromise the performance of the GMM.

3 Federated EM and the Privacy Risks It Poses

In the upcoming discussion, we will employ the EM algorithm for GMM as an illustration to demonstrate that federated learning does not consistently ensure privacy.

3.1 Federated EM algorithm for GMM

The goal of the federated learning is to learn the model under the constraint that the data is stored and processed locally, by transmitting intermediate updates at regular intervals to a central server, see [12, 3] Hence, within the scope of the EM algorithm, instead of directly transmitting the data x_i, each client i shares the following intermediate updates only:

$$\begin{aligned} a_{ij}^t &= P(x_i, \mathcal{N}_j^t) \\ b_{ij}^t &= P(x_i, \mathcal{N}_j^t) \cdot x_i \\ c_{ij}^t &= P(x_i, \mathcal{N}_j^t) \cdot (x_i - \mu_j^t) \cdot (x_i - \mu_j^t) \end{aligned} \tag{5}$$

Therefore, all the above updates can also be computed locally at client i. After getting these intermediate updates from all clients, the server will consolidate all updates from local sources and determine the global update, β_j^{t+1}, μ_j^{t+1}, Σj^{t+1}, necessary for the M-step using the following method:

$$\begin{aligned} \beta_j^{t+1} &= \frac{\Sigma_{i=1}^{n} a_{ij}^t}{n} \\ \mu_j^{t+1} &= \frac{\Sigma_{i=1}^{n} b_{ij}^t}{\Sigma_{i=1}^{n} a_{ij}^t} = \frac{b_j^t}{a_j^t} \\ \Sigma j^{t+1} &= \frac{\Sigma_{i=1}^{n} \mathcal{C}_{ij}^t}{\Sigma_{i=1}^{n} a_{ij}^t} = \frac{\mathcal{C}_j^t}{a_j^t} \end{aligned} \tag{6}$$

Following that, the server transmits the global update β_j^{t+1}, μ_j^{t+1}, Σj^{t+1} back to each and every client.

3.2 Privacy vulnerability in Federated EM algorithm

We observe that using the federated EM algorithm described above, despite not directly sharing private data at each node, the data is still disclosed to the server since the intermediate updates $\dashv_j^t$, $\lfloor_j^t$ enable it to get the private data x_i of each client i, because of the following equation: $b_i^t = a_i^t \cdot x_i$.
In other words, during each iteration, the server possesses the subsequent mutual information:

$$I(\overline{X}_i; \overline{A}_i^t, \overline{B}_i^t) = I(\overline{X}_i; \overline{X}_i) \tag{7}$$

which is maximal. This implies that the private data x_i of every client is fully exposed to the server. Consequently, in this scenario, the federated EM algorithm does not provide any privacy protection whatsoever.

3.3 Privacy Preserving EM

3.3.1 Our approach

Our model takes the Federated EM above as a base model which necessitates a server-client approach. We showed above that the privacy issue in the federated em approach is that the exchange of the intermediate updates $a_{ij}^t, b_{ij}^t, C_{ij}^t$ reveals all private data of each client(node) to resolve this issue we propose concealing the data from the eyes of the server and clients by encrypting the data with a Fully Homomorphic encryption (FHE) specifically CKKS as one of the benefits of using CKKS is that addition and subtraction can be done on the encrypted data without revealing the $a_{ij}^t, b_{ij}^t, C_{ij}^t$ of each client (node). Therefore we will explain the CKKS encryption and then show our approach using it in the PPEM that we designed.

CKKS (Cheon-Kim-Kim-Song) [6] Is a cryptographic technique that facilitates computations on encrypted data. With CKKS, real or complex number-based computations can be performed without decryption, preserving data privacy. Encrypted data is represented as vectors of complex numbers, and operations like addition and multiplication are executed on the encrypted data. Noise is introduced during encryption, and while it accumulates and impacts accuracy, techniques like modulus switching and rescaling manage noise. CKKS finds applications in secure computations, enabling privacy-preserving machine learning and collaborative data analysis without revealing sensitive information. However, CKKS entails computational overhead and necessitates parameter tuning for optimal performance. It is based on a mathematical structure called a polynomial ring. This ring is formed using polynomials whose coefficients come from a number of ”slots”. Each ”slot” represents a value, typically a real or complex number. The number of slots determines the capacity for computations in a single ciphertext. It also involves generating a public key and a secret key pair. The public key is used to encrypt data, while the secret key is used to decrypt it.

3.3.2 Polynomial rings

CKKS [6]operates over a polynomial ring with coefficients in the complex numbers. This ring is denoted as

$$R_q = \frac{C(x)}{x^n + 1} \tag{8}$$

where ”n” is the degree of the polynomial. The plaintext message is encoded as a vector of values, each placed in a ”slot” of the polynomial.

3.3.3 Encryption

Given a plaintext message represented as a vector of values ”m” (one for each slot), the encryption process involves several steps:

1. Encoding: Each plaintext value is encoded into a polynomial with coefficients in the ring R_q. This polynomial is represented as

$$f(X) = m_0 + m_1 \cdot X + m_2 \cdot X^2 + ... + m_{n-1} \cdot X^{n-1} \tag{9}$$

, where m_i represents the value in the i^{th} slot.

2. Adding Noise: To ensure security, noise is added to the polynomial representation. This noise makes it difficult for attackers to analyze the encrypted data and gain information about the original plaintext.

3. Encryption: The noisy polynomial is then encrypted using the public key. The resulting ciphertext is denoted as $c(X)$ and is a polynomial in the polynomial ring R_q.

3.3.4 Homomorphic addition and multiplication

Let’s consider two encrypted polynomials $c_1(X)$ and $c_2(X)$ representing plaintext vectors m_1 and m_2.

Homomorphic Addition: To perform addition on encrypted data, you can add the ciphertexts term by term:

$$c_{sum}(X) = c_1(X) + c_2(X) \tag{10}$$

Homomorphic Multiplication: To perform multiplication on encrypted data, you can multiply the ciphertexts element-wise:

$$c_{product}(X) = c_1(X) \cdot c_2(X) \tag{11}$$

Such operations allow computations to be done on the encrypted data directly but they affect the data as they add additional noise to the original data.

3.4 Proposed PPEM

For iteration t, the encryption unit creates a public key and a secret key as a pair $(Public-k_t, secret-k_t)$ which is then distributed to all trusted nodes (clients) except the server which has no access to any key, each node then calculates it's own E-step (PDF) then each node calculates the partial m-step (intermediate m-step) in which it calculates:

$$a^t_{i,j} = P(x_i, N^t_j) \tag{12}$$

$$b^t_{i,j} = P(x_i|N^t_j)X_i \tag{13}$$

$$c^t_{i,j} = P(x_i, N^t_j)(x_i - \mu^t_j)(x_i - \mu^t_j)^T \tag{14}$$

where $i \in 1, 2, 3..n$ nodes ,$j \in 1, 2, ...k$ Gaussian distributions and assuming that the nodes are honest nodes that are not corrupted then each client(node) prepares a,b,c as three vectors that are made flat because we might be dealing with more than two dimensions , meaning each Gaussian distribution (Cluster would be made up of multiple dimensions . lets consider d dimensions for Data X with k Gaussian mixture models : $a_i \in V^{1xk}$ where V is a vector space ,$b_i \in M^{kxd}$,$c_i \in M^{kxdxd}$ Where M is a matrix space. our implementation takes b_i, c_i and flattens them to be one single vector ,meaning $b'_i \in V^{kd}, c'_i \in V^{kdd}$:

$$a'_i = [a_{i1}, a_{i2}, ..a_{ik}, n_i] \tag{15}$$

where n_i is the number of data points in node i

$$b'_i = [b_{i11}, b_{i12}, ...b_{i1d}, b_{i21}..b_{ikd}] \tag{16}$$

$$c'_i = [c_{i111}, c_{i112}, ..., c_{i11d}, c_{i121}, ...c_{i1dd}, c_{i211}...c_{ikdd}] \tag{17}$$

after defining those vectors we continue with the partial m-step , we apply the CKKS encryption on each vector a'_i, b'_i, c'_i then each node sends them back to the server . The server calculates using property (7) :

$$a^{encrypted}_{total} = \sum_{i=1}^{n} (a^{encrypted}_i)' \tag{18}$$

$$b^{encrypted}_{total} = \sum_{i=1}^{n} (b^{encrypted}_i)' \tag{19}$$

$$c^{encrypted}_{total} = \sum_{i=1}^{n} (c^{encrypted}_i)' \tag{20}$$

The server then sends the results to each client where each client(node) then calculates: $a, b, c = decrypt(a_{total}), decrypt(b_{total}), decrypt(c_{total})$ $n_c = a[k+1]$ $\beta^{t+1} = \frac{a_j}{n_c}$ $\mu^{t+1} = \frac{b_j}{a_j}$ $\Sigma^{t+1} = \frac{c_j}{a_j}$ Thus finishing the m-step , we repeat this process until convergence.

3.5 Initialization

The initialization is done ahead of the first iteration : $\sigma_i \leftarrow$ diagonal matrix of 1's. $\mu_i \leftarrow$ is randomly created. $\pi_i \leftarrow$ is initiated with a Uniform distribution in which each Cluster (Gaussian distribution) has the same probability as the others.

Algorithm 1 PPEM Protocol

```
1: procedure SERVER FOR ITERATION $t$
2:     for each Honest client $i$ do
3:         Initialize $\Sigma_i$, $\mu_i$, $\beta_i$
4:         Perform E-step for client $i$
5:         Create a CKKS keys using the encryption unit and provide it to client $i$ via secure channels
6:         Perform Partial m-step i.e Calculate $a_i$, $b_i$, $c_i$ using (5)(15-17)
7:         Encrypt vectors $a_i$, $b_i$, $c_i$ Using the keys
8:         Send $a_i^{enc}$, $b_i^{enc}$, $c_i^{enc}$ to the server via secure channels
9:     end for
10: end procedure
11: procedure SERVER
12:     Perform Server M-step:
13:     $a_{\text{total}}^{enc}$, $b_{\text{total}}^{enc}$, $c_{\text{total}}^{enc} \leftarrow \sum_{i=1}^{c} a_i^{enc}$, $\sum_{i=1}^{c} b_i^{enc}$, $\sum_{i=1}^{c} c_i^{enc}$ (18)(19)(20)
14:     Send $a_{\text{total}}^{enc}$, $b_{\text{total}}^{enc}$, $c_{\text{total}}^{enc}$ to all trusted clients
15: end procedure
16: procedure CLIENT $i$, ELEMENT $j$
17:     for each client $i$ do
18:         for each cluster j do
19:             $a$, $b$, $c \leftarrow$ decrypt($a_{\text{total}}^{enc}$), decrypt($b_{\text{total}}^{enc}$), decrypt($c_{\text{total}}^{enc}$)
20:             $\beta^{t+1} \leftarrow \frac{a_j}{a_{k+1}}$
21:             $\mu^{t+1} \leftarrow \frac{b_j}{a_j}$
22:             $\Sigma^{t+1} \leftarrow \frac{c_j}{a_j}$
23:         end for
24:     end for
25: end procedure
26: procedure SERVER
27:     for each client $i$ do
28:         Calculate the log likelihoods for client $i$
29:     end for
30:     $Log - Likelihood^{total} \leftarrow \sum_{i=1}^{clients} Log - Likelihood_i$
31:     $Log - Likelihoods.append(Log - Likelihood^{total})$
32:     if $Log - Likelihoods[-1] - Log - Likelihoods[-2] \leq \epsilon$ then
33:         Break
34:     else
35:         Return to step 2 until convergence criteria is met
36:     end if
37: end procedure
```

3.6 Privacy

In this analysis we assume that the nodes are honest nodes and not corrupted. As we showed earlier the Federated em algorithm has a vulnerability that x_i is exposed and can be inferred by an attacker during the partial m-step, Meaning the following mutual information is exposed to the server maximally :

$$I(\overline{X_i}; \overline{A^t_{ij}}, \overline{B^t_{ij}}) = I(\overline{X_i}; \overline{X_i}) \tag{21}$$

We address this vulnerability by encrypting both A^t_{ij}, B^t_{ij}. Assuming that enc(Y) is a CKKS encrypted version of Y and that the server operates as we have detailed then the information that the server holds is:

$$I(\overline{X}; \sum \overline{enc(A^t_i)}, \sum \overline{enc(B^t_i)}, \sum \overline{enc(C^t_i)}) \tag{22}$$

With the assumption that CKKS is an indistinguishable under the Chosen Plaintext Attack (IND-CPA)[11] we can assume that each $enc(A^t_i), enc(B^t_i), enc(C^t_i)$ are indistinguishable from one another. Therefore, as an Honest but Curious adversary, the server cannot distinguish $enc(A^t_i)$ from $enc(B^t_i)$. Moreover, the server does not have the private key of the nodes(clients), thus cannot decrypt the information to find out the true values of A_i, B_i. Therefore addressing the main vulnerability of the original Federated EM algorithm:

$$I(\overline{X_i}; enc(\overline{A}^t_i), enc(\overline{B^t}_i)) =_{IND-CPA} 0 \tag{23}$$

secondly , we get the following mutual information:

$$I(\overline{X}; \sum \overline{enc(A^t_i)}, \sum \overline{enc(B^t_i)}, \sum \overline{enc(C^t_i)}) =_{(10)} I(\overline{X}; enc(\sum \overline{A}^t_i), enc(\sum \overline{B}^t_i), enc(\sum \overline{C}^t_i)) \tag{24}$$

$$= I(\overline{X}; \overline{A}^{encrypted}_{total}, \overline{B}^{encrypted}_{total}, \overline{C}^{encrypted}_{total}) \tag{25}$$

If we use the same principle we get that A,B,C total are indistinguishable from one another thus an honest but curious server cannot infer any information from a combination of any of them. Now the final step is to validate the privacy is the clients side as the server sends the encrypted sums A,B,C the client(node) then decrypts them into $a_{total}, b_{total}, c_{total}$ which does not impose any security concerns when we consider the size of data sets. This is a security concern only when the number of nodes and the data sets is significantly smaller at which using this approach is not recommended nor is it useful. If we look at the whole system as a graph, the server would be the root and all other nodes would be leafs meaning there is no connection between one leaf node to the other and thus the usage of a shared key (public, secret) for all nodes does not pose any breach of privacy between nodes as nodes can only communicate with the server the only way that a leaf node can get any information about any other node is through the server as there is no edge between two clients and the longest path from one leaf to the other is only two edges, meaning the breach of privacy between leaf nodes would come from a transition between the server and the client and we addressed that by addressing the honest but curious server above.

To address an adversary (such as eavesdropping) on the server we change the key at each iteration so that the adversary does not find any patterns in the plaintext that the server receives that can enable him to infer the secret key.

4 Stoppage criteria

The stoppage criteria to our algorithm lies in the difference between the log-likelihoods between each iteration with the ϵ variable as the decision maker of stopping or continuing the algorithm ,in which we observed that the smaller ϵ is the more precise the algorithm is and the bigger it is the less accurate our result would be . the ultimate value for epsilon is absolute zero , yet the algorithm might run until the number of iterations set by the user is met . with our model , the deciding log likelihood would be the sum of all likelihoods from all clients ,and this is where the log portion of the log likelihood function helps as as we don't need to multiply or try to use the responsibilities of all nodes.

4.1 Correctness

The criteria for correctness in this context is presented in the estimated parameters of the Gaussian mixture model and the degree to which they are impacted by our implementation. What we do differently from the original federated em is that we send the a_i, b_i, c_i parameters encrypted by CKKS, now although CKKS does not change the data drastically the impact of such encryption would be based on the degree to which we choose the precision parameters to be. Specifically on the GMM the precision does impact the final results, as the parameters should be chosen based on the task at hand, with Gaussian mixture models we choose high precision parameters based on the tests done by the TenSEAL team [2]. Moreover, the results obtained throw this scheme are approximate and have some noise. Yet, they do not significantly impact the accuracy of this model if the encryption parameters are chosen correctly, [12]

4.2 Results

4.3 Correctness of the output

We used Parkinson's dataset [14] from the UCI Repository [7]to validate our results against the Federated EM. A data set that contains voice measurements from 31 people of which 23 have the Parkinson's disease and 9 are healthy that consists of 195 samples each person with 6 different samples with 22 features each. This data set was chosen for two reasons , firstly it contains highly sensitive data which are biomedical voice measurements of individuals, secondly we wanted to check a real world scenario . We reduced the dimensions of the dataset using PCA (principle component analysis) and used the two components that we got from PCA for GMM fitting on both Federated EM and our approach. as shown in 16 we can see that on this data set the results are identical which shows that the algorithm has near perfect results on this data set. Hence, the output correctness of our approach is guaranteed on this data set. To guarantee the correctness of our approach we conducted several experiments using random data on a varying number of data sets as shown in figures 345 and we can see that in the included chart 1 of the outputs of the algorithm and the log likelihood value.

We also wanted to insure that the number of nodes in the network doesn't affect the correctness of the output, so we also conducted experiments on both the Parkinson's data

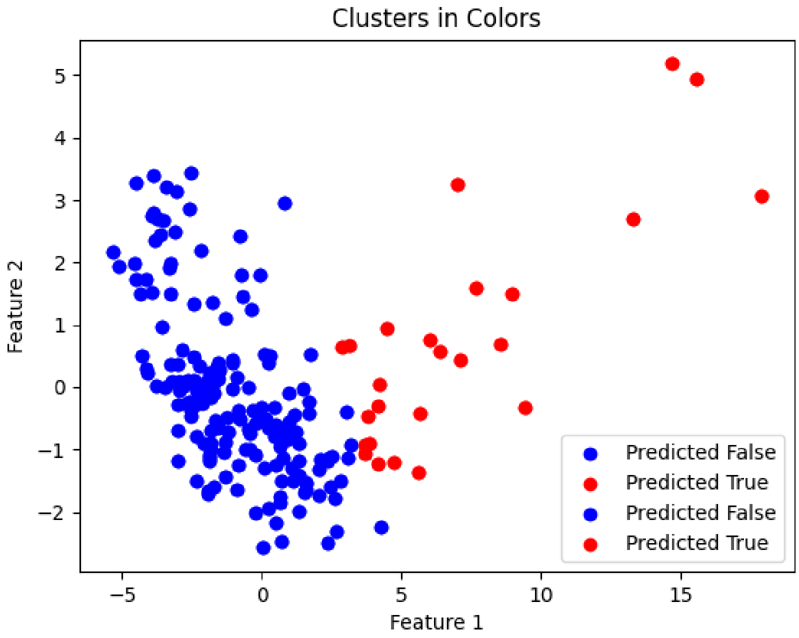

Figure 1: showing Parkinson's data using our approach with PCA applied to the data

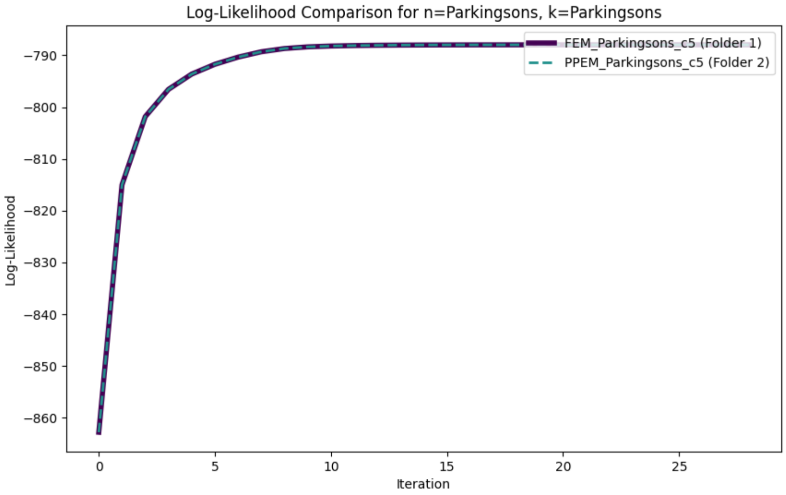

Figure 2: Log likelihood of Parkinson's data compared between our approuch and the federated em

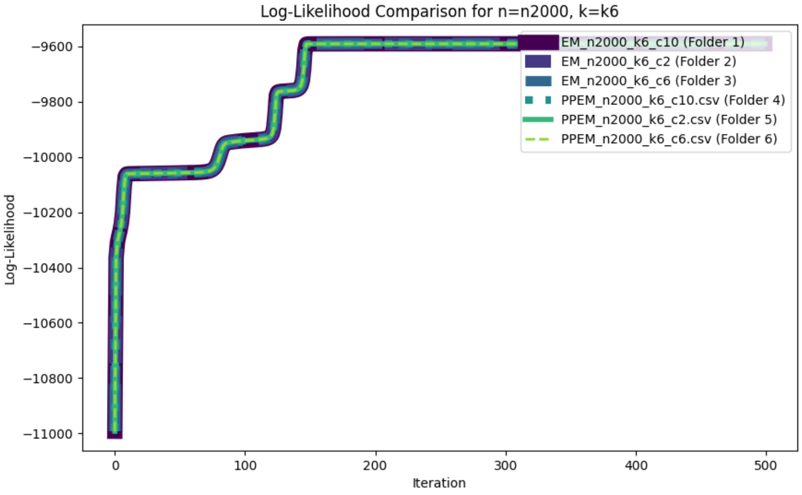

Figure 3: 2000 random data points with 6 clusters with varing number of clients

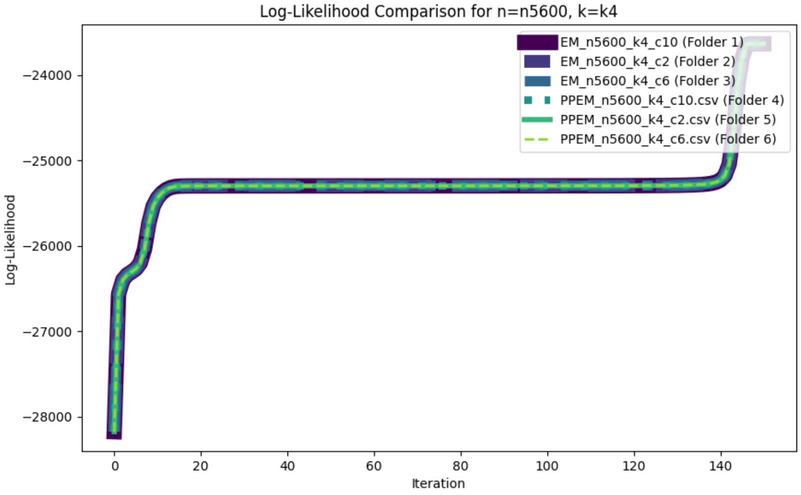

Figure 4: 5600 random data points with 4 clusters with varing number of clients

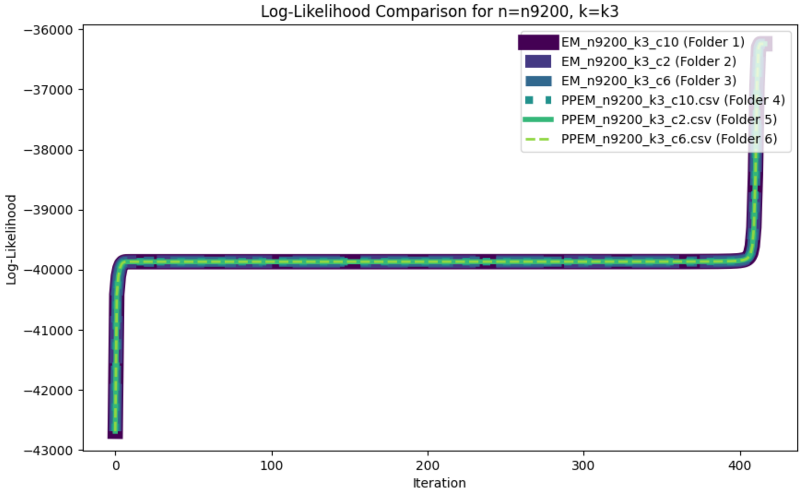

Figure 5: 9200 random data points with 3 clusters with varing number of clients

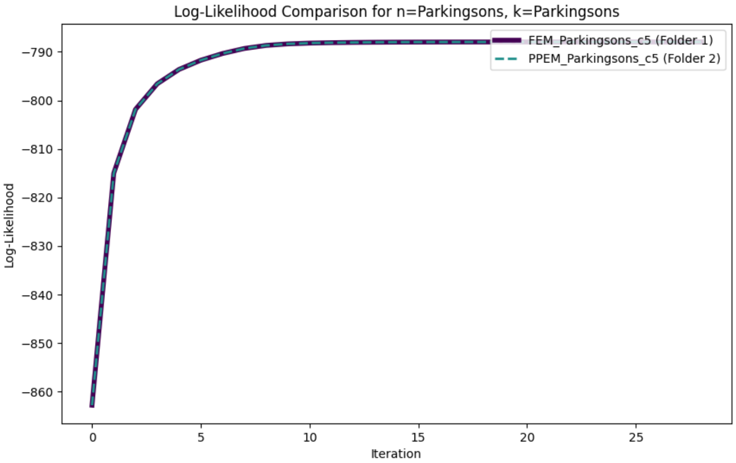

Figure 6: Log likelihood of Parkinson's data compared between our approuch and the federated em

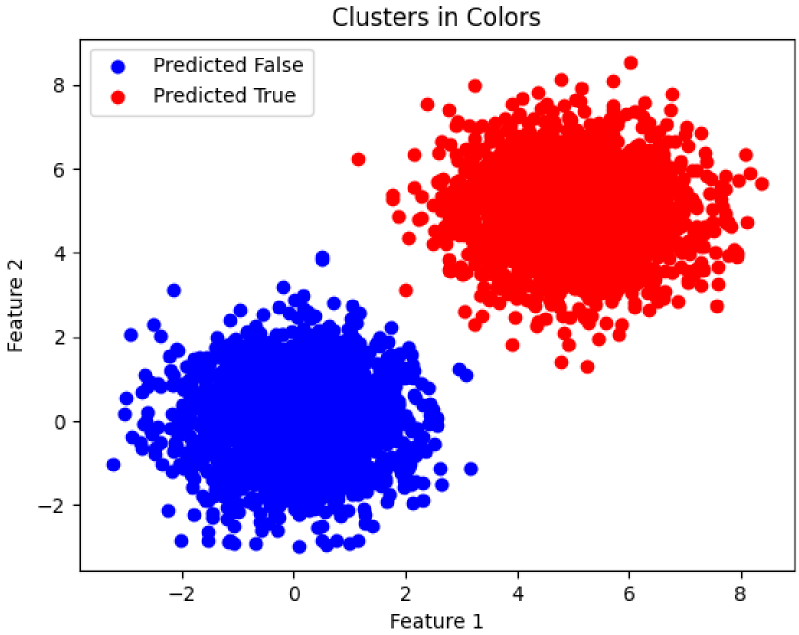

Figure 7: Showing Result of 4700 samples with 2 clusters and 2 clients

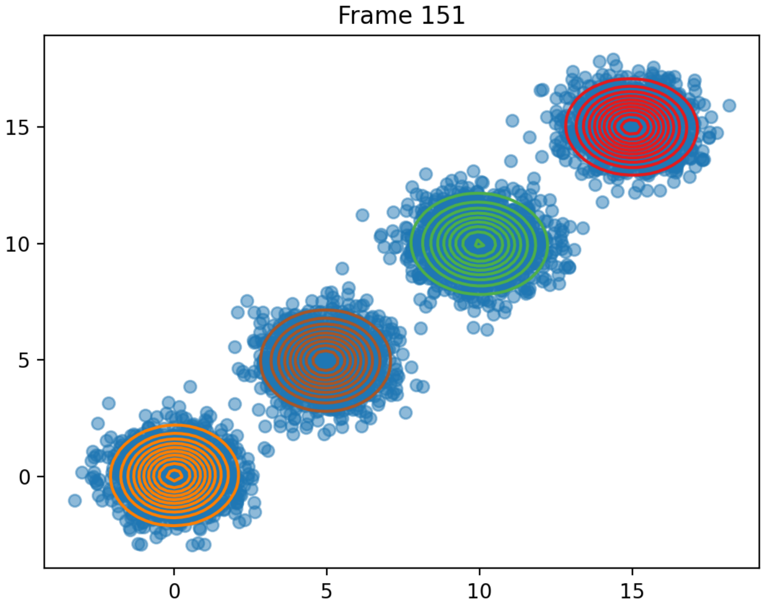

Figure 8: Showing Result of 5700 samples with 4 clusters and 10 clients

base with multiple participants and on the randomly generated data with varying number of nodes (clients) as shown in chart 3456

4.3.1 Individual Privacy

Firstly we proved that Federated EM is not privacy Preserving from the fact that $a_i = b_i x_i$ with i as the node i. While we can show that the privacy level of the proposed approach is similar to the existing approach's because both of them only reveal the sum of the three elements a, b, c. Yet our approach differs by adding the fact that the server is not allowed to know what the content of a, b, c is while only allowing the nodes to know such information, thereby preventing the server the access to each node's individual data x_i.

4.3.2 Communication overhead

Homomorphic encryption introduces a communication overhead based on the parameters of the encryption method ,a full analysis on the effects of CKKS encryption on the size of the encrypted word based on the parameters chosen has been done in which they test each polynomial degree with coefficient modulus sizes based on what keys are generated ,in our implementation we chose parameters that would both grant us fast results and less encrypted data size based on the chart provided by the TenSEAL library[2].

4.3.3 Scalability

The model that we provided can be scaled to any number of clients(nodes) as they do not increase nor decrease the privacy aspect of the algorithm ,from a performance standpoint the scalability factor does not play a role according to our results see 5 1 .

4.3.4 Convergence behaviour

The convergence behaviour throughout our testing was basically the same see 2 5the convergence behavior based on the input type.

4.4 Optimizations and enhancements

In this section we provide what we see fit as a feature study case that can improve our approach.

4.4.1 CKKS implementation

a. A combination of those two options: 1. Our implementation of ckks can be altered to accommodate the malicious adversary by using the Galois keys's rotation property, as we can rotate the data for each client differently (when its already encrypted) giving each client in each iteration a rotation that it should do on the data, and then the server would rotate each one of the inputs before it adds them together while making sure that the rotation on each encrypted data is done according to the rotations done in each clients environment while not allowing the data's orientation to return to it's original form. after the addition the server would rotate the resulting sum differently for each client. 2. Replacing the vectors a,b,c into a new vector that mixes the a,b,c vectors into one vector and then using a higher modulus degree to accommodate for the vectors size. b. There is a new implementation of CKKS that allows addition between two different encrypted values that have a different private key called multi-key CKKS variant see [5] . in which they shed a new light to linear keys that could also help with the speed of encryption and addition.

4.4.2 Implementing a threshold encryption method

Add another layer of security to check the validity of the client and the validity of the server such as Elliptic Curve Digital Signature Algorithm or Edwards-curve Digital Signature Algorithm to make sure that the server or client is who they say they are.

4.4.3 Initialization and pre-processing

Throughout our testing we tried different methods of initializing both the data and σ, μ, π , and we found it more challenging to initialize them without getting a singular covariance matrix when applying the algorithm to high dimensional data. and our solution to this issue is shown in the initialization section above. Pre-proccessing with a hard clustering algorithm such as k-means on the data can alter the actual results and result in a hard clustered result. meaning in a dataset such as Parkinson's it would lead to false estimations and inaccurate results, that said when we used EM as a preprocessing algorithm in each clients environment we observed that:
1. it doesn't deviate the results.
2. it can make the Proposed PPEM and the Federated EM converge faster, although it always depends on the input, so if the nature of the input is known to the user we do recommend preprocessing with EM.

4.4.4 Open questions for further research

1. One of the main weaknesses of our approach is the time it takes for each client to encrypt the data.
2. Our implementation doesn't take into account the malicious adversary. 3. Lets consider a situation where an adversary can be malicious such that it can manipulate the distribution of the data and the resulting Clustering, how can we combat such an adversary?

4.5 Conclusions

In this paper, we consider the privacy preserving distributed Expectation maximization on GMM problem. We gave theoretical privacy analysis to the existing approach of distributed EM algorithm, showed the criteria for such privacy and proved the weakness in it. Then we proposed a Privacy-Preserving EM algorithm that uses a Fully Homomorphic scheme called CKKS improved on the weakness found in the Distributed EM algorithm and proved with theoretical analysis that we have overcome that privacy weakness. We provided Numerical results As well as a full sweet of testing results to demonstrate the correctness of the output, then we provided improvement strategies for feature work.

	Log Likelihoods		Iterations		Ticks	
Identifier	FEM	PPEM	FEM	PPEM	FEM	PPEM
n200_k2_c2	-685.275	-685.275	52.0	52.0	0.703	69.812
n200_k2_c6	-685.275	-685.275	52.0	52.0	0.562	195.448
n200_k2_c10	-685.275	-685.275	52.0	52.0	0.833	325.358
n200_k3_c2	-839.393	-839.393	134.0	134.0	1.504	184.353
n200_k3_c6	-839.393	-839.393	134.0	134.0	2.438	518.811
n200_k3_c10	-839.393	-839.393	134.0	134.0	2.651	849.381
n200_k4_c2	-821.071	-821.071	92.0	92.0	1.54	132.292
n200_k4_c6	-821.071	-821.071	92.0	92.0	1.717	369.971
n200_k4_c10	-821.071	-821.071	92.0	92.0	2.102	605.782
n200_k5_c2	-912.215	-912.215	180.0	180.0	2.477	269.087
n200_k5_c6	-912.215	-912.215	180.0	180.0	3.442	764.005
n200_k5_c10	-912.215	-912.215	180.0	180.0	5.038	1258.293
n200_k6_c2	-962.493	-962.493	146.0	146.0	2.877	225.755
n200_k6_c6	-962.493	-962.493	146.0	146.0	3.505	631.135
n200_k6_c10	-962.493	-962.493	146.0	146.0	6.042	1030.984
n1100_k2_c2	-3853.107	-3853.107	184.0	184.0	6.393	248.166
n1100_k2_c6	-3853.107	-3853.107	184.0	184.0	7.883	684.927
n1100_k2_c10	-3853.107	-3853.107	184.0	184.0	7.11	1118.505
n1100_k3_c2	-4285.445	-4285.445	81.0	81.0	4.714	114.819
n1100_k3_c6	-4285.445	-4285.445	81.0	81.0	4.411	316.279
n1100_k3_c10	-4285.445	-4285.445	81.0	81.0	5.046	514.523
n1100_k4_c2	-4606.801	-4606.801	62.0	62.0	4.442	91.017
n1100_k4_c6	-4606.801	-4606.801	62.0	62.0	4.809	251.041
n1100_k4_c10	-4606.801	-4606.801	62.0	62.0	4.336	411.394
n1100_k5_c2	-5092.974	-5092.974	316.0	316.0	23.691	488.138
n1100_k5_c6	-5092.974	-5092.974	316.0	316.0	25.008	1334.075
n1100_k5_c10	-5092.974	-5092.974	316.0	315.0	29.449	2180.295
n1100_k6_c2	-5449.522	-5449.522	500.0	500.0	43.064	799.13
n1100_k6_c6	-5449.522	-5449.522	500.0	500.0	43.549	2185.977
n1100_k6_c10	-5449.522	-5449.522	500.0	500.0	43.999	3571.023
n2000_k2_c2	-7046.019	-7046.019	188.0	188.0	8.735	257.923
n2000_k2_c6	-7046.019	-7046.019	188.0	188.0	9.057	706.403
n2000_k2_c10	-7046.019	-7046.019	188.0	188.0	9.534	1155.559
n2000_k3_c2	-7844.166	-7844.166	142.0	142.0	9.629	204.749
n2000_k3_c6	-7844.166	-7844.166	142.0	142.0	10.038	555.569
n2000_k3_c10	-7844.166	-7844.166	142.0	142.0	10.528	908.755
n2000_k4_c2	-8427.783	-8427.783	98.0	98.0	9.247	148.813
n2000_k4_c6	-8427.783	-8427.783	98.0	98.0	9.162	403.987
n2000_k4_c10	-8427.783	-8427.783	98.0	98.0	9.833	654.837
n2000_k5_c2	-9321.779	-9321.779	500.0	500.0	59.89	792.812
n2000_k5_c6	-9321.779	-9321.779	500.0	500.0	57.836	2142.922

	Log Likelihoods		Iterations		Ticks	
Identifier	FEM	PPEM	FEM	PPEM	FEM	PPEM
n2000_k5_c10	-9321.779	-9321.779	500.0	500.0	59.684	3483.985
n2000_k6_c2	-9590.976	-9590.976	500.0	500.0	64.804	860.832
n2000_k6_c6	-9590.976	-9590.976	500.0	500.0	68.354	2188.58
n2000_k6_c10	-9590.976	-9590.976	500.0	500.0	71.741	3557.217
n2900_k2_c2	-10200.346	-10200.346	316.0	316.0	20.293	394.18
n2900_k2_c6	-10200.346	-10200.346	316.0	316.0	20.798	1173.492
n2900_k2_c10	-10200.346	-10200.346	316.0	316.0	21.564	1912.396
n2900_k3_c2	-11361.236	-11361.236	183.0	183.0	17.542	265.227
n2900_k3_c6	-11361.236	-11361.236	183.0	183.0	17.974	712.645
n2900_k3_c10	-11361.236	-11361.236	183.0	183.0	18.74	1155.797
n2900_k4_c2	-12202.963	-12202.963	85.0	85.0	11.474	128.46
n2900_k4_c6	-12202.963	-12202.963	85.0	85.0	11.346	346.489
n2900_k4_c10	-12202.963	-12202.963	85.0	85.0	11.708	560.504
n2900_k5_c2	-12849.743	-12849.743	360.0	360.0	62.264	572.1
n2900_k5_c6	-12849.743	-12849.743	360.0	360.0	57.882	1507.571
n2900_k5_c10	-12849.743	-12849.743	360.0	360.0	60.262	2739.166
n2900_k6_c2	-13897.603	-13897.603	500.0	500.0	98.11	718.095
n2900_k6_c6	-13897.603	-13897.603	500.0	500.0	96.091	2003.693
n2900_k6_c10	-13897.603	-13897.603	500.0	500.0	99.379	3052.369
n3800_k2_c2	-13452.228	-13452.228	282.0	282.0	23.673	321.929
n3800_k2_c6	-13452.228	-13452.228	282.0	282.0	24.758	855.237
n3800_k2_c10	-13452.228	-13452.228	282.0	282.0	24.809	1386.65
n3800_k3_c2	-14983.354	-14983.354	197.0	197.0	24.547	232.821
n3800_k3_c6	-14983.354	-14983.354	197.0	197.0	24.883	622.494
n3800_k3_c10	-14983.354	-14983.354	197.0	197.0	25.746	1008.962
n3800_k4_c2	-16081.152	-16081.152	203.0	203.0	33.368	254.223
n3800_k4_c6	-16081.152	-16081.152	203.0	203.0	33.898	669.794
n3800_k4_c10	-16081.152	-16081.152	203.0	203.0	34.862	1084.023
n3800_k5_c2	-16929.679	-16929.679	291.0	291.0	58.958	383.993
n3800_k5_c6	-16929.679	-16929.679	291.0	291.0	67.247	1004.027
n3800_k5_c10	-16929.679	-16929.679	291.0	291.0	62.389	1626.837
n4700_k2_c2	-16641.217	-16641.217	180.0	180.0	18.835	203.34
n4700_k2_c6	-16641.217	-16641.217	180.0	180.0	18.853	545.521
n4700_k2_c10	-16641.217	-16641.217	180.0	180.0	19.448	884.329
n4700_k3_c2	-18535.617	-18535.617	124.0	124.0	21.334	149.073
n4700_k3_c6	-18535.617	-18535.617	124.0	124.0	23.452	395.199
n4700_k3_c10	-18535.617	-18535.617	124.0	124.0	20.861	638.994
n4700_k4_c2	-19893.635	-19893.635	92.0	92.0	18.184	118.213
n4700_k4_c6	-19893.635	-19893.635	92.0	92.0	18.396	306.882
n4700_k4_c10	-19893.635	-19893.635	92.0	92.0	19.022	493.189
n4700_k5_c2	-21990.205	-21990.205	500.0	500.0	120.72	683.54

	Log Likelihoods		Iterations		Ticks	
Identifier	FEM	PPEM	FEM	PPEM	FEM	PPEM
n4700_k5_c6	-21990.205	-21990.205	500.0	500.0	134.129	1747.585
n4700_k5_c10	-21990.205	-21990.205	500.0	500.0	131.067	2818.247
n5600_k2_c2	-19766.288	-19766.288	195.0	195.0	22.787	222.804
n5600_k2_c6	-19766.288	-19766.288	195.0	195.0	23.131	593.78
n5600_k2_c10	-19766.288	-19766.288	195.0	195.0	23.585	961.726
n5600_k3_c2	-22021.374	-22021.374	181.0	181.0	31.216	222.385
n5600_k3_c6	-22021.374	-22021.374	181.0	181.0	31.824	580.194
n5600_k3_c10	-22021.374	-22021.374	181.0	181.0	32.14	938.742
n5600_k4_c2	-23635.158	-23635.158	151.0	151.0	34.826	198.462
n5600_k4_c6	-23635.158	-23635.158	151.0	151.0	35.31	507.595
n5600_k4_c10	-23635.158	-23635.158	151.0	151.0	36.215	817.973
n6500_k2_c2	-22937.709	-22937.709	356.0	356.0	54.907	412.297
n6500_k2_c6	-22937.709	-22937.709	356.0	356.0	55.07	1082.207
n6500_k2_c10	-22937.709	-22937.709	356.0	356.0	56.285	1755.912
n6500_k3_c2	-25559.145	-25559.145	204.0	204.0	46.58	254.718
n6500_k3_c6	-25559.145	-25559.145	204.0	204.0	42.592	656.683
n6500_k3_c10	-25559.145	-25559.145	204.0	204.0	42.982	1057.657
n7400_k2_c2	-26162.33	-26162.33	349.0	349.0	55.401	414.751
n7400_k2_c6	-26162.33	-26162.33	349.0	349.0	55.578	1081.33
n7400_k2_c10	-26162.33	-26162.33	349.0	349.0	56.433	1747.054
n7400_k3_c2	-29152.201	-29152.201	153.0	153.0	36.154	197.151
n7400_k3_c6	-29152.201	-29152.201	153.0	153.0	36.235	501.202
n7400_k3_c10	-29152.201	-29152.201	153.0	153.0	37.512	806.996
n8300_k2_c2	-29361.698	-29361.698	344.0	344.0	60.25	415.72
n8300_k2_c6	-29361.698	-29361.698	344.0	344.0	59.846	1058.735
n8300_k2_c10	-29361.698	-29361.698	344.0	344.0	61.546	1712.621
n8300_k3_c2	-32716.3	-32716.3	159.0	159.0	40.898	206.822
n8300_k3_c6	-32716.3	-32716.3	159.0	159.0	41.889	520.217
n8300_k3_c10	-32716.3	-32716.3	159.0	159.0	42.161	831.673
n9200_k2_c2	-32523.923	-32523.923	422.0	422.0	87.873	508.324
n9200_k2_c6	-32523.923	-32523.923	422.0	422.0	87.733	1310.76
n9200_k2_c10	-32523.923	-32523.923	422.0	422.0	89.209	2107.598
n9200_k3_c2	-36243.374	-36243.374	417.0	417.0	131.225	551.847
n9200_k3_c6	-36243.374	-36243.374	417.0	417.0	133.811	1372.212
n9200_k3_c10	-36243.374	-36243.374	417.0	417.0	135.927	2197.736

Table 1: Chart with all results of random input , formated as n:number of input in dataset k: number of clusters(Gaussian distributions) , c: number of clients

References

[1] Abdulatif Alabdulatif, Ibrahim Khalil, Albert Y Zomaya, Zahir Tari, and Xun Yi. Fully homomorphic based privacy-preserving distributed expectation maximization on cloud. *IEEE Transactions on Parallel and Distributed Systems*, 31(11):2668–2681, 2020.

[2] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. Tenseal: A library for encrypted tensor operations using homomorphic encryption. *arXiv preprint arXiv:2104.03152*, 2021.

[3] Kanishka Bhaduri and Ashok N Srivastava. A local scalable distributed expectation maximization algorithm for large peer-to-peer networks. In *2009 Ninth IEEE International Conference on Data Mining*, pages 31–40. IEEE, 2009.

[4] Ravi Charan. Expectation Maximization Explained — towardsdatascience.com. `https://towardsdatascience.com/expectation-maximization-explained-c82f5ed438e5`. [Accessed 10-09-2023].

[5] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 395–412, 2019.

[6] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Paper 2016/421, 2016. `https://eprint.iacr.org/2016/421`.

[7] Dheeru Dua, Casey Graff, et al. Uci machine learning repository. 2017.

[8] Oded Goldreich. Modern cryptography, probabilistic proofs and pseudorandomness - weizmann, May 2000.

[9] Oded Goldreich. Basics of probability theory (for theory of computation courses) - weizmann, Nov 2008.

[10] Sharon X Lee, Kaleb L Leemaqz, and Geoffrey J McLachlan. Ppem: Privacy-preserving em learning for mixture models. *Concurrency and Computation: Practice and Experience*, 31(24):e5208, 2019.

[11] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. In *Annual International Cryptology Conference*, pages 560–589. Springer, 2022.

[12] Qiongxiu Li, Jaron Skovsted Gundersen, Katrine Tjell, Rafal Wisniewski, and Mads Græsbøll Christensen. Privacy-preserving distributed expectation maximization for gaussian mixture model using subspace perturbation. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4263–4267. IEEE, 2022.

[13] Xiaodong Lin, Chris Clifton, and Michael Zhu. Privacy-preserving clustering with distributed em mixture modeling. *Knowledge and information systems*, 8:68–81, 2005.

[14] Max Little, Patrick Mcsharry, Stephen Roberts, Declan Costello, and Irene Moroz. Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection. *Nature Precedings*, pages 1–1, 2007.

[15] Mijung Park, James Foulds, Kamalika Choudhary, and Max Welling. Dp-em: Differentially private expectation maximization. In *Artificial Intelligence and Statistics*, pages 896–904. PMLR, 2017.

[16] Amir Hossein Poorjam, Yordan P Raykov, Reham Badawy, Jesper Rindom Jensen, Mads Græsbøll Christensen, and Max A Little. Quality control of voice recordings in remote parkinson's disease monitoring using the infinite hidden markov model. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 805–809. IEEE, 2019.

[17] Bin Yang, Issei Sato, and Hiroshi Nakagawa. Privacy-preserving em algorithm for clustering on social network. In *Advances in Knowledge Discovery and Data Mining: 16th Pacific-Asia Conference, PAKDD 2012, Kuala Lumpur, Malaysia, May 29-June 1, 2012, Proceedings, Part I 16*, pages 542–553. Springer, 2012.

[18] Zhe Zhang and Linjun Zhang. High-dimensional differentially-private em algorithm: Methods and near-optimal statistical guarantees. *arXiv preprint arXiv:2104.00245*, 2021.

A Expectation maximization

A.1 Motivation

We want to divide the information we have into 2 clusters, and characterize each cluster by a parameter θ, with this parameter we want to be able to "catch" as much information as possible.

First, we will define the probability of observing a datapoint with value x:

A redundancy function $Pr(X;\theta)$ is a function of parameter θ, which uses a probability model for the observed data X.

We will define the observed data to be the set X. Suppose this set is divided into 2 clusters, each of which is normally distributed $N(\mu,\sigma^2)$. We will define the probability that the observed data is in cluster 2 to be π.

In total we have 5 parameters:

1. Probability π.

2. μ_1.

3. σ_1.

4. μ_2.

5. σ_2.

So we define π to be the probability, μ as the expectancy and σ as standard deviation for each cluster. We will collectively define them as θ:

$$\theta := \pi, \mu_1, \sigma_1, \mu_2, \sigma_2$$

We will define the normal distribution function as ϕ.

We will define:

$$p(x;\theta) = \pi\phi(x;\mu_1,\sigma_1) + (1-\pi)\phi(x;\mu_2,\sigma_2)$$

We will define $L(\theta; \text{data})$ to be the likelihood function of observing the whole dataset.

In addition, given the observed data $x_1, \ldots, x_n$, we want to find the parameter that maximizes the Likelihood of observing our whole dataset, let's call it θ':

$$\theta' = \arg\max_{\theta} Pr(x_1, \ldots, x_n; \theta)$$

We get that:

$$L(\theta; \text{data}) = \prod_{i=1}^{n} Pr(x_i; \theta)$$

Since it is expensive to calculate a derivative for a product, we will calculate its log function:

$$l(\theta; \text{data}) = \log L(\theta) = \log Pr(x_1, \ldots, x_n; \theta) = \log \prod_{i=1}^{n} p(x_i; \theta) = \sum_{i=1}^{n} \log Pr(x_i; \theta)$$

Our goal is to select the parameters so that we examine all the information, and maximize this choice.

To get an intuition of the purpose of the algorithm and how it works, you can look at the k-means algorithm that is a popular method for analyzing mining data (clustering data). Its purpose is to divide the observations into k-clusters according to centers of gravity (mean). Each observation is associated with one of the "centers of gravity." By correctly selecting the centers of gravity, it is possible to locate the different groups.

In the k-means algorithm, for each point we decide on which center of gravity is the closest one – which is actually the expectancy phase.

For this we will define a latent variable Δ. If a point is in cluster 2, then the value of the variable will be 1.

Succinctly, this is actually the stage at which a hard estimate is made. This is hard because Δ is 1 for one of the clusters, and 0 for the others.

We will then update the centers of gravity to be the expectancy of the points in the cluster – which is basically the maximization phase.

This is the maximum likelihood estimate for μ.

In the k-means algorithm, the ”pattern” of information does not have a standard deviation. The pattern in double quotation marks because it is not generative (constantly created / ”multigenerational”).

If we know Δ, then it will be easy for us to calculate the values of the maximum likelihood of our parameters.

Therefore:

$$l(\theta; \text{data}, \Delta) = \sum_{\text{data}} [(1-\Delta_i) \log(x_i; \mu_1, \sigma_1) + (\Delta_i) \log \phi(x; \mu_2, \sigma_2)] + \sum_{\text{data}} [(1-\Delta_i) \log(1-\pi) + \Delta_i \log \pi]$$

We will notice that the second sum (the second $\sum$) is ”extra” for the purpose of measuring the likelihood of all Δ.

We will notice that assuming that we have already found Δ, the maximum likelihood estimates will be easy to detect.

For μ, we will use the average in each cluster.

For σ, we will use MLE.

For π, we will take their ratio in the second cluster.

The bottleneck here is finding Δ, and this is the point of the EM algorithm.

A.2 How the Algorithm Works

In our example, we'll do a soft placement of Δ. Sometimes we will call it ”responsibility” (how ”responsible” is each cluster for each observation?). We will define the responsibility as γ:

$$\gamma_i(\theta) = E[\Delta_i | \theta, \text{data}] = Pr(\Delta_i = 1 | \theta, \text{data})$$

Let's quickly go over the symbols we've defined so far:

$$\begin{aligned}
\theta &: \text{Parameter vector} \\
\pi &: \text{Probability} \\
\mu &: \text{Expectancy} \\
\sigma &: \text{Standard deviation} \\
\phi &: \text{Normal distribution function} \\
x &: \text{Data point} \\
\Delta &: \text{Latent variable} \\
\gamma &: \text{Responsibility}
\end{aligned}$$

And here is the algorithm:

1. We will initialize the estimates for:

$$\theta \coloneqq \pi, \mu_1, \sigma_1, \mu_2, \sigma_2$$

2. **Expectancy stage:** Calculate the responsibility for each observation:

$$\gamma_i = \frac{\pi\phi(x_i;\mu_2,\sigma_2)}{(1-\pi)\phi(x_i;\mu_1,\sigma_1)+\pi\phi(x_i;\mu_1,\sigma_1)}$$

   (i.e., what is the degree of (relative) proximity to cluster 2? The larger γ_i gets, the closer point is / the more responsible it is to cluster 2.)

3. **Maximization stage:** We will update the estimates of the parameters by using the maximum likelihood function. All summations taken for the indexed observations are only expectancies/standard deviations that are weighted by their degree of responsibility γ_i:

$$\begin{aligned}\mu_2 &= \frac{\Sigma\gamma_i x_i}{\Sigma\gamma_i},\\ \sigma_2 &= \frac{\Sigma\gamma_i(x_i-\mu_2)^2}{\Sigma\gamma_i},\\ \pi &= \frac{1}{n}\cdot\Sigma\gamma_i\end{aligned}$$

4. We will repeat steps 2-3 until we get a local optimum.

Now we have reached the stage of the general algorithm. We will connect everything we have done now and use more complicated EM variables.

A.3 The Correctness of the Algorithm

We will use two vectors:

1. The observed data x.

2. The vector of unobserved variables z.

We denote their joint by $p(x,z|\theta)$ where θ is a vector of parameters to be estimated.

The conditional probability formula $p(z|x,\theta) = \frac{p(x,z|\theta)}{p(x|\theta)}$ implies that $\log[p(x|\theta)] = \log[p(x,z|\theta)] - \log[p(z|x,\theta)]$. If we multiply both sides by $p(z|x,\theta_{j-1})$, we obtain:

$$p(z|x,\theta_{j-1})\log[p(x|\theta)] = p(z|x,\theta_{j-1})\log[p(x,z|\theta)] - p(z|x,\theta_{j-1})\log[p(z|x,\theta)]$$

Then, we can sum over the support of z:

$$\sum_{z\in R_z} p(z|x,\theta_{j-1})\log[p(x|\theta)] = \sum_{z\in R_z} p(z|x,\theta_{j-1})\log[p(x,z|\theta)] - \sum_{z\in R_z} p(z|x,\theta_{j-1})\log[p(z|x,\theta)]$$

Since the log-likelihood $\log[p(x|\theta)]$ does not depend on z and $\sum_{z\in R_z} p(z|x,\theta_{j-1}) = 1$, we have:

$$\log[p(x|\theta)] = \sum_{z\in R_z} p(z|x,\theta_{j-1})\log[p(x,z|\theta)] - \sum_{z\in R_z} p(z|x,\theta_{j-1})\log[p(z|x,\theta)]$$

Moreover,

$$Q(\theta,\theta_{j-1}) = \sum_{z\in R_z} p(z|x,\theta_{j-1})\log[p(x,z|\theta)]$$

Thus,

$$\log[p(x|\theta)] = Q(\theta,\theta_{j-1}) - \sum_{z\in R_z} p(z|x,\theta_{j-1})\log\left[\frac{p(z|x,\theta)}{p(z|x,\theta_{j-1})}\right]$$

By Jensen's inequality, we have:

$$\sum_{z\in R_z} p(z|x,\theta_{j-1})\log\left[\frac{p(z|x,\theta)}{p(z|x,\theta_{j-1})}\right] \le \log\left[\sum_{z\in R_z} p(z|x,\theta_{j-1})\left(\frac{p(z|x,\theta)}{p(z|x,\theta_{j-1})}\right)\right] = \log\left[\sum_{z\in R_z} p(z|x,\theta)\right] = \log(1) = 0$$

Therefore,

$$\log[p(x|\theta)] - \log[p(x|\theta_{j-1})] \ge Q(\theta,\theta_{j-1}) - Q(\theta_{j-1},\theta_{j-1})$$

which, for $\theta = \theta_j$, becomes:

$$\log[p(x|\theta_j)] - \log[p(x|\theta_{j-1})] \ge 0$$

Or

$$\log[p(x|\theta_j)] \ge \log[p(x|\theta_{j-1})]$$

A.4 Data Sources

1. Charan, R. (2020). Expectation Maximization Explained. Towards Data Science. https://towardsdatascience.com/expectation-maximization-explained-c82f5ed438e5

2. k-means clustering. (n.d.). Wikipedia. https://en.wikipedia.org/wiki/K-means_clustering

3. Voronoi diagram. (n.d.). Wikipedia. https://en.wikipedia.org/wiki/Voronoi_diagram

4. Taboga, Marco (2021). "EM algorithm", Lectures on probability theory and mathematical statistics. Kindle Direct Publishing. Online appendix. https://www.statlect.com/fundamentals-of-statistics/EM-algorithm.

B Privacy Preserving Expectation maximization with Generalized Linear Mixed Model

B.1 Enhancing GWAS Sensitivity and Statistical Power

Many computational methods have been developed to enhance the sensitivity and statistical power of GWAS (genome-wide association studies). Two notable methods are:

B.2 Linear Mixed Model (LMM)

The LMM aims to explain the variation of a target phenotype in a population by a mixture of fixed effects (variables of interest) and random effects (unknown variables). It has been shown to improve the identification rate of potentially causal variants of human diseases. However, LMM is designed for quantitative traits and cannot be directly applied to categorical phenotypes.

B.3 Generalized Linear Mixed Model (GLMM)

B.4 Introduction

Many computational methods have been developed to enhance the sensitivity and statistical power of GWAS (genome-wide association studies) like:

1. The linear mixed model (LMM) aims to explain the variation of a target phenotype in a population by a mixture of fixed effects (variables of interest) and random effects (unknown variables), which are shown to improve the identification rate of potentially causal variants of human disease. However, LMM is designed for quantitative traits and cannot be directly applied to categorical phenotypes.

2. The generalized linear mixed model (GLMM) extends the LMM to support both categorical and quantitative phenotypes and is frequently used in GWAS.

In a generic setting, a GLMM can be constructed using human genomic data (i.e., genotypes inferred from genome sequences) from a cohort of phenotyped human individuals, which requires data analysts to have direct access to the individual-level genomic data for every member in the cohort.

B.5 GLM

- The GLMM extends LMM by incorporating a link function to convert a continuous outcome into a categorical outcome. Here, we use the logit function:

  $$f(z) = \log\left(\frac{z}{1-z}\right)$$

  as the link function to deal with the binary outcome often used in the case-control GWAS:

  $$\log\left(\frac{P(Y=1|X)}{1-P(Y=1|X)}\right) = \beta X + Zu$$

  where X represents fixed effects, Z represents the random effect, and β and u are the coefficients for the fixed and random effects, respectively.

- The EM algorithm is used for GLMM parameter estimation from a total of N input records, each with the given fixed and random effects as well as the desirable outcome (0 or 1).

- We use an EM algorithm to estimate the parameters (Z and β) of an optimal GLMM.
- Let T be the number of random effect categories and M be the number of fixed effect variables.
- We consider $t \in [1, \ldots, T]$ as one of the random effect categories and $m \in [1, \ldots, M]$ as one of the fixed effect variables.

B.6 The Algorithm

We present a privacy-preserving method for constructing a GLMM using a collaborative Expectation–Maximization (EM) algorithm that combines the Metropolis–Hasting (MH) algorithm in the E-step and the Newton–Raphson (NR) algorithm in the M-step to estimate parameters of the GLMM.

B.7 Estimating Parameters in GLMM

In the MH algorithm, we start from randomly selected σ_i for each category i, and then sample random effects $Z = (z_1, \ldots, z_T)$, where each z_i is randomly sampled from $N(0, \sigma)$. In each MH sampling step, we first sample new random effects z' in the neighborhood of $z \in Z$ based on the current $N(0, \sigma)$, and then compute a proposal probability A between z' and z using the current model parameters β to decide if z should be replaced by z' in Z for the next step. The proposal function is defined as:

$$A(z, z') = \min\left(1, \frac{p^*(z)q(z')}{p^*(z')q(z)}\right) \in [0, 1]$$

where $p^*(\cdot)$ is the likelihood of random effect according to the current GLMM model, and $q(\cdot)$ represents the likelihood of observing a record according to the current $N(0, \sigma)$.

After the MH algorithm completes, the variance of the final samples is used to compute the updated σ.

In the M-step of the EM algorithm, we use the NR algorithm to compute the fixed effect coefficients β based on the current random effects Z and the parameters σ (updated in the E-step).

B.8 Privacy-Preserving GLMM Construction on Horizontally Partitioned Data

The general idea of collaborative computation is to partition a target algorithm (e.g., to construct the GLMM) into two parts: the first part of private computation can be performed on the partial data held by each participating party that generates intermediate results required for the second part of computation, and the second part joint computation has to be performed on a central server using the intermediate results from the private computation of all parties. In this scenario, the central server is considered to be semi-trusted (honest-but-curious), and thus the sensitive raw data cannot be directly sent to the central server by each participating party.

Consider the input data matrix X containing the fix effects (e.g., the genotypes) on a total of N records (disease patients) in each of the T random effect categories. In the horizontal data partition scenario, the entire data matrix was not held by a single user, but instead by K different parties: the ith party holds a subset of $T \times P$ records. (Again, for the convenience of presentation, here we assume each party holds the same number (P) of records in each of the T random effect categories; as a result, $N = T \times P \times K$.

However, in our implementation, we can handle the situation where different parties hold a different number of records, and also the records may be distributed unevenly among random effect categories.) Using the separately held input data matrix, our goal is to develop the collaborative computation algorithm for building a GLMM model among the K participating parties, through the partition of private and joint computation for the EM algorithm as laid out above.

B.9 Collaborative MH Algorithm

Given the horizontally partitioned data, we need to modify the MH algorithm into a collaborative version that consists of the private computation, in which each participating party and the joint computation.

The intermediate result A_k computed in the private computation by the kth party is then sent to the central server, where the joint computation is performed for computing the proposal probability density A, to decide if the previous samples of the random effect z_{i-1} should be replaced by the new estimates z_i. The proposal probability A acts like a filter to replace the less likely (i.e., fitting improperly to the outcome according to the current estimates of fixed effect parameters β) random effects parameters sampled in the previous step: if $A = 1$, the previous parameters are definitively replaced; otherwise, it is replaced with the probability A.

After the update of iteration i, the random effects Z_i are stored in the central server for the subsequent iterations in the MH algorithm. The iteration of the algorithm continues until the parameter estimation converges. In our experiments, the MH algorithm usually converges after 1000 iterations.

After convergence, the central server will retain a pool of random effects Z^* obtained in each iteration. In order to better estimate the parameters Z^*, we adopted the burn-in strategy commonly used in MH algorithms, which eliminates some sampled parameters in the initial steps of MH sampling. Based on the final sample pool of random effects Z^*, the central server updates the parameters σ, which will be used in the M-step (see the collaborative NR algorithm) for estimating the fixed effect coefficients β.

The key idea of the collaborative MH algorithm presented here is the partition of the computation of the proposal probability into two components, the private and joint computation, respectively.

B.10 Collaborative NR Algorithm

In the M-step, we use a NR algorithm to estimate the fixed effect coefficients (β) based on the current estimates of the random effect parameters σ. Similar to the E-step, we propose a collaborative NR algorithm that partitions the computation of the first-order derivative and Hessian matrix of the likelihood function, which are required to update the fixed effect coefficients, into the private and joint computation.

Above we presented the collaborative algorithms for the E- and M-step, respectively. It is straightforward to combine these into a collaborative EM algorithm, which iterates between these two steps until the estimated parameters (including the fixed effect coefficients β and the random effect parameter σ) converge. The entire process requires $C \times (M_{MH} + M_{NR})$ rounds of communications, where C represents the number of iterations of the EM algorithm.