Privacy preserving distributed machine learning with federated learning

Chamikara, Pathum; Bertok, Peter; Khalil, Ibrahim; Liu, D.; Camtepe, Seyit

Privacy Preserving Distributed Machine Learning with Federated Learning

M.A.P. Chamikara[a,b,*], P. Bertok[a], I. Khalil[a], D. Liu[b], S. Camtepe[b]

[a]*RMIT University, Australia*
[b]*CSIRO Data61, Australia*

Abstract

The published article can be found at `https://doi.org/10.1016/j.comcom.2021.02.014`

Edge computing and distributed machine learning have advanced to a level that can revolutionize a particular organization. Distributed devices such as the Internet of Things (IoT) often produce a large amount of data, eventually resulting in big data that can be vital in uncovering hidden patterns, and other insights in numerous fields such as healthcare, banking, and policing. Data related to areas such as healthcare and banking can contain potentially sensitive data that can become public if they are not appropriately sanitized. Federated learning (FedML) is a recently developed distributed machine learning (DML) approach that tries to preserve privacy by bringing the learning of an ML model to data owners'. However, literature shows different attack methods such as membership inference that exploit the vulnerabilities of ML models as well as the coordinating servers to retrieve private data. Hence, FedML needs additional measures to guarantee data privacy. Furthermore, big data often requires more resources than available in a standard computer. This paper addresses these issues by proposing a distributed perturbation algorithm named as DISTPAB, for privacy preservation of horizontally partitioned data. DISTPAB alleviates computational bottlenecks by distributing the task of privacy preservation utilizing the asymmetry of resources of a distributed environment, which can have resource-constrained devices as well as high-performance computers. Experiments show that DISTPAB provides high accuracy, high efficiency, high scalability, and high attack resistance. Further experiments on privacy-preserving FedML show that DISTPAB is an excellent solution to stop privacy leaks in DML while preserving high data utility.

*Corresponding author
Email address: `pathumchamikara.mahawagaarachchige@rmit.edu.au` (M.A.P. Chamikara)

Keywords: data privacy, distributed data privacy, privacy preserving machine learning,distributed machine learning, federated learning

1. Introduction

The amalgamation of different technologies such as edge computing, IoT, cloud computing, and machine learning has contributed to a rapid proliferation of technological development in many areas such as healthcare and banking [1, 2, 3]. The increase of cheap pervasive sensing devices has contributed to the rapid growth of IoT, becoming one of the main sources of big data [4]. In the broader spectrum of sensor systems, cyber-physical systems and advanced analysis tools are converged together to provide consolidated services. As a result, a particular system (e.g. healthcare, banking) can now be benefited from multiple sources of data, additionally to what is accumulated by conventional means [5]. This growing availability of different sources of data has been able to revolutionize leading fields such as healthcare technologies to achieve excellent achievements in many areas such as drug discovery, early outbreak detection, epidemic control analysis, which were once considered to be complicated [2, 3]. However, data related to the fields such as healthcare, banking and policing are massively convoluted with sensitive private data [6, 7, 8]. It is essential to go for extreme measures to protect sensitive data while analyzing them to generate meaningful insights [9, 10]. However, it is an extremely challenging task as systems related to fields such as healthcare and banking are often densely distributed. This paper examines the issues related to distributed data sharing and analysis in order to devise an optimal privacy preservation solution towards distributed machine learning [11] in environments such as presented in Figure 1, which represents a typical distributed industry setup (e.g. smart healthcare, open banking) that runs on IoT, edge, fog, and cloud computing.

Privacy violations in fields such as healthcare and banking can be catastrophic due to the availability of highly sensitive person-specific data [7]. Among different definitions, privacy for data sharing and analysis can be defined as “Controlled Information Release” [12]. It has been shown before that it is easy to identify patients in a database by combining several quasi-identifiers such as age, postcode, and sex [13]. Removing just the identifiers from the dataset before releasing is not enough to protect the individuals’ privacy, and leaking personal information to untrusted third parties can be catastrophic [14, 15, 16, 17]. Privacy-preserving data mining (PPDM) is the area that applies privacy-preserving approaches to data mining methods to protect the private information of the users of the underlying input data during the data mining processes [14]. In this paper, we investigate the PPDM solutions that can be applied to limit privacy leaks in distributed machine learning (DML) under big data settings. The area of homomorphic encryption is widely explored for PPDM. However, in terms of big data and DML, homomorphic encryption cannot address the three challenges, (i) efficiency, (ii) high volume, and (iii) massive distribution of data. Furthermore, homomorphic encryption increases the data size during the encryption (e.g. single bit can be multiplied to 16 bits), which is unreliable for big data and increases the data storage burdens [18]. Compared to encryption, data perturbation (data modification) can provide efficient solutions towards privacy preservation with a predetermined error that can result due to the data modification [14, 19].

Federated Learning (FedML) is a distributed machine approach that is developed to provide efficient privacy-preserving machine learning in a distributed environment [20, 21]. In FedML, the machine learning model generation is done at the data owners' computers, and a coordinating server (e.g. a cloud server) is used to generate a global model and share the ML knowledge among the distributed entities (e.g. edge devices). Since the original data never leave the data owners' devices, FedML is assumed to provide privacy to the raw data. However, ML models show vulnerability to privacy inference attacks such as model memorizing attacks and membership inference, which focus on retrieving sensitive data from trained ML models even under black-box settings [22, 23]. Model inversion attacks that recover images from a facial recognition system [24] is another example that shows the vulnerability of ML to advanced adversarial attacks. If adversaries gain access to the central server/coordinating server, they can deploy attacks such as model memorizing attacks, which can memorize and extract raw data from the trained models [22]. Hence, DML approaches, such as FedML, need additional measures to guarantee that there will not be any unanticipated privacy leaks. Differential privacy is a privacy definition (privacy model) that offers a strong notion of privacy compared to previous models [6]. Due to the application of heavy noise, the previous attempt to enforce differential privacy on big data has resulted in low utility in terms of advanced analytics, which can be catastrophic for applications such as healthcare [25]. A major disadvantage of other techniques such as random rotation and geometric perturbation is their incapability to process high dimensional data (big data) efficiently. These perturbation approaches spend an excessive amount of time to generate better results with good utility and privacy [26, 27]. In terms of efficiency, additive perturbation provides good performance. However, the perturbed data end up with a low privacy guarantee [28]. Another issue that is often ignored when developing privacy-preserving mechanisms for big data is data capacity issues. The application of privacy preservation (the specific algorithms based on encryption on perturbation) on a large database can be extensive and impossible if the exact resource allocation scenarios are not implemented correctly. A recently developed algorithm named PABIDOT for big data privacy preservation promises to provide high efficiency towards big data [29]. The proposed method shows high classification accuracy for big data while providing high privacy. However, PABIDOT cannot be used for DML as it does not address the data distribution and data perturbation using resource-constrained devices (as depicted in Figure 1).

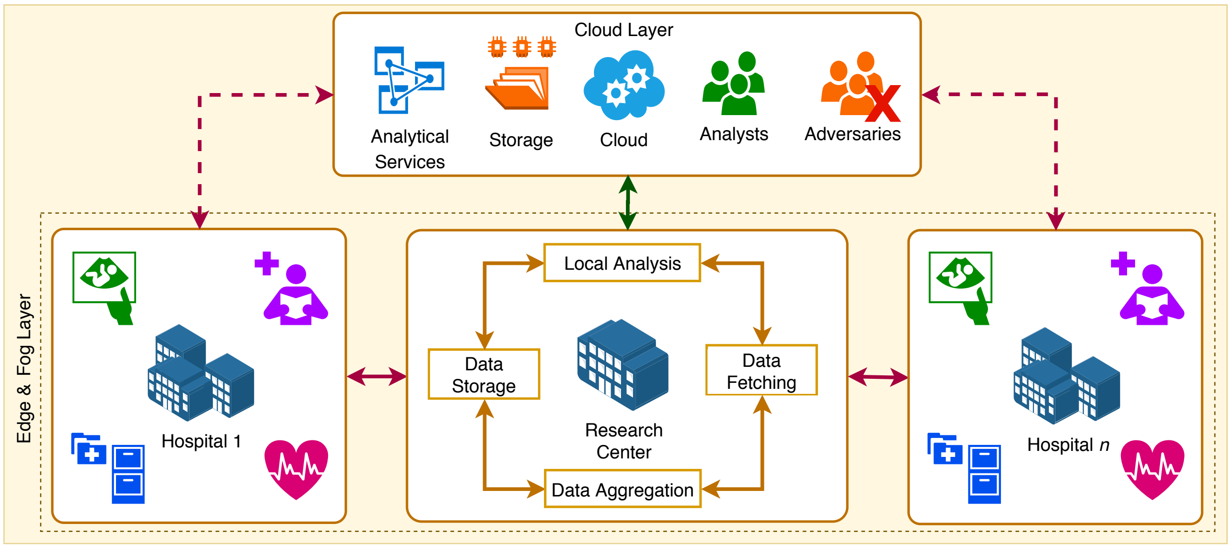


Figure 1: An example for a distributed organizational setting: a healthcare ecosystem which is geographically distributed among multiple locations. A healthcare system may have many distributed branches to it, facilitating and collecting many healthcare data including IoT sensor data. The central body coordinates the distributed hospitals in terms of maintaining data integrity to support a wide range of analytics. The central authority/research centre is also responsible for sharing data to cloud-based third parties for enhanced intelligence and quality of service towards their patients.

We propose a new DISTributed Privacy-preserving Approach for distriButed machine learning in geographically distributed data and systems (DISTPAB). DISTPAB is a distributed privacy-preserving algorithm that employs a data perturbation mechanism. The distributed scenario of data perturbation of DISTPAB allows the perturbation of extensive datasets that need to be shared among distributed entities without leaking privacy. The actual data perturbation is conducted in the distributed entities (in edge/fog devices) locally using the global perturbation parameters generated in a central coordinating node before conducting FedML. This way, DISTPAB restricts original data to be communicated (before perturbation) via the network, which can be attacked by adversaries. The global perturbation parameter generation of DISTPAB ensures that there is no degradation of accuracy or attack resistance of the perturbed data. DISTPAB was first tested using six datasets obtained from the data repository named "UCI Machine Learning Repository"[1]. The results show that DISTPAB provides excellent classification accuracy, attack resistance, and excellent efficiency towards distributed machine learning under big data settings.

The following sections of the paper are set out as follows. Section 2 includes a summary of related work. Section 3 provides a summary of the fundamentals used in developing DISTPAB. Section 4 provides background information related to DISTPAB. Section 5 presents the experimental evaluations of the the performance and the attack resistance of DISTPAB. Section 6 provides a discussion on the results provided in Section 5. The paper is concluded in Section 7.

2. Literature Review

Distributed systems such as available in healthcare have become vastly complex due to the amalgamation of different technologies such as IoT, edge computing, and cloud computing. Due to these advanced capabilities, a modern system can utilize a myriad of data sources to facilitate improved capabilities towards essential services. The nature of distributed data platforms introduces a plethora of complexities towards preserving user privacy of users without compromising data utility. Extremely high dimensions and massive distribution of data sources are two of the main complexities that need to be addressed when designing robust privacy-preserving approaches for DML systems. Due to these reasons, privacy-preserving data mining (PPDM) approaches should be efficient and should be able to work in distributed settings. Approaches based on secure multi-party computation [30], attribute access control [4], lightweight cryptographic procedures [18], homomorphic encryption [18] are few examples for some encryption approach which can provide good privacy. However, the high computational complexity of such encryption scenarios can seriously jeopardize the performance of a distributed ML setup. The inherent high computational complexity of the cryptographic approaches requires excessive amounts of computational resources, which include large storage facilities such as cloud computing that are often controlled by third-parties. Furthermore, the encryption approaches such as homomorphic encryption for PPDM increase the size of the data after the application of encryption, which is not suitable for the domain of big data as the data sizes can already be extensive. Hence, the implementation of cryptographic scenarios for distributed databases can be unrealistic and unaffordable [25]. Compared to encryption, data perturbation/modification provides efficient and lightweight solutions for big data

[1]http://archive.ics.uci.edu/ml/index.php

privacy [28], and hence, data perturbation is a better fitting solution for PPDM of distributed data.

Previous data perturbation approaches include swapping [31], additive perturbation [32], condensation [33], randomized response [34], microaggregation [35], random rotation [26], random projection [36], geometric perturbation [27], and hybrid perturbation [37]. Due to the modifications, the data end up with reduced utility. The relationship between utility and privacy granted by a particular perturbation algorithm is defined via a privacy model [38]. More than a few privacy models have been introduced where one model tries to overcome the defects of another. $k - anonymity$ [39, 40], $l - diversity$ [41], $t - closeness$ [42], $(\alpha, k) - anonymity$ [43], $k_\theta - affinity$ [44] are some examples for privacy models. However, these models exhibit vulnerabilities to attacks such as composition attacks [45], minimality attacks [46], and foreground knowledge [47] attacks. Moreover, the perturbation approaches that use these models do not scale for big data with high dimensions. When the dimensions of the input dataset grow, the computational cost necessary to conduct data perturbation grows exponentially. The literature refers to this phenomenon as "The Dimensionality Curse" [48]. Another issue in high dimensional data is the leak of extra information that can be effectively used by attackers to misuse sensitive private information [49]. Differential privacy (DP) is another privacy model, which was developed more recently to render a strong privacy guarantee compared to prior approaches. Nevertheless, due to the high noise levels imposed by DP algorithms, DP might not be the optimal choice for big data privacy [25].

The distribution of data sources and infrastructures also introduces a massive complexity towards the development of robust privacy-preserving approaches for distributed databases. Modern distributed privacy-preserving approaches include federated (FedNN) machine learning and split learning (SplitNN), which can provide a certain level of privacy to distributed databases. In FedML, the distributed clients use their local data to train local ML models and communicate the locally trained ML model parameters with the central server to train a global representation of the local models. The server then distributes the global model parameters with the clients so that the clients can now generalize their local models based on the model parameters federated by the server. SplitML has a similar distributed setup. However, instead of communicating the model parameters, SplitML splits the ML model between the clients and the server. During the training process, a client will hold a portion of the ML model, whereas the server will hold the other portion of the ML model. As a result, a client will transfer activations from its split layer to the split layer of the server during both forward and backward passes of the training process of the ML model. When many clients need to connect to the server for ML model training, the clients will either have a peer to peer communication or a client-server communication for the model parameter communications in order for each client to have a synchronous ML model training process. However, both FedML and SplitML have the same problem of the central point of failure as the server in both cases has too much control over the model learning process. If the server is attacked, the whole framework becomes vulnerable, and as a result, user privacy cannot be guaranteed. Especially the attack methods such as membership inference and model memorization can exploit the vulnerabilities of the central point of failure to retrieve raw data used for the model training process. The other distributed approaches which work on horizontally and vertically partitioned data also tend to produce issues towards efficiency or privacy [50]. As most of these methods use the approaches based on homomorphic encryption to provide privacy, such methods become inefficient when working with big data, and the need to share too much information with the server makes most of them untrusted and yields the same issue of the central point of failure [50].

Among data perturbation approaches, matrix multiplicative approaches proved to provide high utility towards data clustering and classification [14, 29]. Examples for matrix multiplicative perturbation approaches include random rotation, geometric, and projection perturbation [28] methods. For example, random rotation perturbation repeatedly multiplies the input data by a randomly generated matrix with the properties of an orthogonal matrix, until the perturbed data satisfy a pre-anticipated level of privacy [26]. An additional matrix of translation and distance perturbation are combined with random rotation to produce geometric data perturbation. The added randomness of the random translation matrix improves the privacy of the perturbed data. However, geometric data perturbation also follows the same repeated approach until the perturbed data satisfy an expected level of privacy [27]. Random projection perturbation follows a different approach by projecting the high dimensional input data to a low dimensional space [36]. One of the important properties of matrix multiplicative data perturbation approaches is their ability to preserve the distance between tuples of the input dataset [26, 27, 36]. As a result of this property, matrix multiplicative approaches provide high utility for data classification and clustering, which are based on distance calculations. However, due to the inefficient approaches utilized for the optimal perturbation parameter generation, rotation perturbation, geometric perturbation, and projection fail to provide enough scalability. PABIDOT is a recently developed perturbation approach for the privacy preservation of big data classification, using the properties of geometric data perturbations. PABIDOT provides high efficiency towards big data while maintaining high privacy and classification accuracy. However, for distributed healthcare scenarios such as shown in Figure 1, PABIDOT cannot be applied as it is not a distributed algorithm. As shown in Figure 1, a distributed system can be composed of branches that are geographically dispersed. The data coming from each branch should be perturbed before they leave the local network. A distributed data perturbation algorithm which can provide high utility while maintaining high privacy for distributed healthcare is essential.

3. Fundamentals

The proposed method uses multidimensional transformations and *Randomized Expansion* [29], which improves the randomness of the data. DISTPAB considers the input dataset as a data matrix in which each tuple is regarded as a column vector for applying the transformations. This section explains how the data transformation is for perturbation and how the randomization of the final output is improved using *Randomized Expansion*.

3.1. Data matrix (D)

We consider the input dataset as a data matrix. The proposed perturbation mechanism only addresses numeric data perturbation. Hence a data matrix will contain only numeric data. The columns of the data matrix represent the attributes/features, whereas the rows (row vectors) represent the member (data owner) records. Each row vector will be subjected to multidimensional transformations on an n-dimensional Cartesian coordinate (n = number of attributes) system.

3.2. Multidimensional isometric transformations

If a multidimensional transformation, $T : R^n \rightarrow R^n$ holds Equation 1. Reflection, translation and rotation are examples for multidimensional isometric transformation [51].

$$|T(A) - T(B)| = |A - B|, \quad \forall\ A, B \in R^n \tag{1}$$

3.3. Homogeneous coordinate form and composite operations

In $n - dimensional$ space, we can write a homogeneous coordinate point as an $(n+1)$ dimensional position vector $(x_1, x_2, \ldots, x_n, h)$, where $h \neq 0$ is an additional term.

An $(n+1)$ dimensional position vector $(x_1, x_2, \ldots, x_n, h)$, represents a homogeneous coordinate point in the $n - dimensional$ space when $h \neq 0$ is an additional term. This coordinate form allows representing all transformations in matrix multiplication form of size of $(n+1) \times (n+1)$ [52].

A transformation operation is called a composite transformation when it involves the operations between several matrices. Equation 2 shows the composite transformation of the sequential application of $M_1, M_2, M_3, \ldots$ to a homogeneous matrix X.

$$X' = \ldots (M_3(M_2(M_1X))) = \ldots M_3 \times M_2 \times M_1 \times X \tag{2}$$

We can use a new column of ones, where $h = 1$, to convert a data matrix into its homogeneous matrix form [29].

3.4. z-score normalization of the input dataset, D

The attributes of an input dataset can have different scales, which requires different levels of perturbation to each attribute. Consequently, applying the same levels of perturbation would not provide equal protection to all the attributes. As a result, the final version of the perturbed dataset will have attributes with insufficient levels of perturbation. We apply z-score normalization (also referred to as standardization) to impose equal weights to all the attributes during the transformations. The z-score normalization scales the input dataset with a standard deviation equals to 1 and a mean equals to 0 [53].

3.5. Generating transformation matrices for perturbation

For the perturbation of the input data matrix, we need to generate the n-dimensional homogeneous translation, reflection, and rotation matrices. We can generate the translational matrix according to [52] (refer Appendix .1). Since the input data matrix is z-score normalized, the translational coefficients of the translational matrix are sampled from a uniform random distribution that lies in the interval $(0, 1)$ [29]. We can generate the (n+1) axis reflection matrix utilizing n-dimensional reflection matrix (refer to Appendix .2) [29]. The rotational matrix can be generated using the concept of concatenated subplane rotation (refer Appendix .3) [29].

3.6. Randomized expansion

Randomized expansion is a noise addition approach which improves the randomness of the perturbed data [29]. Equation 3 shows the randomized expansion noise generation where $S_{\pm}$ is the sign matrix generated based on the values of D^{p2} (intermediate perturbed data matrix). As the equation shows, the positiveness or the negativeness is improved by randomized expansion to provide high utility with improved randomization [29].

$$D^{p2} = (\left\|D^{p2}\right\| + \|\mathcal{N}(0,\sigma)\|) \bullet S_{\pm} \tag{3}$$

3.7. Quantification of privacy

$Var(D - D^p)$ is one approach to measure the level of privacy of the perturbed data, where D is original data and D^p is perturbed data [32]. The higher the $Var(P)$, the higher the difficulty, hence the higher the privacy. Take X^p to be a perturbed data series of attribute X. Now, $Var(P)$ can be written as in Equation 4, where $P = (X^p - X)$.

$$Var(P) = Var(p_1, p_2, \ldots, p_n) = \frac{1}{n}\sum_{i=1}^{n}(p_i - \bar{p})^2 \tag{4}$$

3.8. $\Phi - separation$

$\Phi - separation$ is a privacy model that allows the selection of optimal perturbation parameters for a perturbation algorithm in a given instance of data. $\Phi - separation$ allows the determination of the best perturbation parameters while providing an optimal empirical privacy guarantee [29].

Definition 1 ($\Phi-$*separation*)**.** *Apply a perturbation algorithm M to the dataset $D = [d_1, d_2, \ldots, d_n]_{m\times n}$ to generate the perturbed instances of D as $D^p_j = [d^p_{j1}, d^p_{j2}, \ldots, d^p_{jn}]_{m\times n}$ for $j = 1, 2, \ldots, k$. If k represents the number of all feasible ways to apply M to D to generation D^p. Then,*

$$\phi_j = min\{[var(d_i - d^p_{ji})]\}\forall j = 1, 2, \ldots, k \tag{5}$$

where ϕ_j is the minimum privacy guarantee.

We can maximize all the possible minimum privacy guarantees (ϕ_j) to obtain the optimal privacy guarantee Φ as

$$\Phi = max\{\phi_j\}_{j=1}^{k} \tag{6}$$

A perturbed dataset provides $\Phi - separation$ if it satisfies Equation 6.

3.9. Federated Learning

Federated learning provides the capability to train a machine learning model using distributed data without sharing the original data between the participating entities. Assume that $\{D_1, \ldots D_N\}$ are distributed datasets, which are distributed among N data owners $\{O_1, \ldots O_N\}$. In a federated learning setup, each of these datasets (D_i) never have to leave the corresponding owner (O_i) and each owner trains ML models locally without exposing the corresponding data to any external parties. The model parameters of the locally trained models are then collected in a server (a central entity/authority), which federate the parameters to generate a global representation of all the models (ML_{fed}). The accuracy (A_{fed}) of ML_{fed} should be very close to the accuracy (A_{ctr}) of the model trained centrally with all the data [20]. This relationship can be represented using Eq. 7, where δ is a non-negative real number.

$$|A_{fed} - A_{ctr}| < \delta \tag{7}$$

3.9.1. Horizontal and vertical federated learning

In horizontal federated learning, all the clients share the same feature space but different space in samples. Consider a feature space $X : x_1, \ldots, x_n$, a sample ID space of $I : i_1, \ldots, i_j$, and a label space of $Y : y_1, \ldots, y_k$ constitutes the complete training dataset (I, X, Y). Then a particular distributed client will have the dataset (i_l, X, Y). Several regional banks, with different user groups with similar business characteristics having the same feature spaces, can be an example of a horizontally partitioned scenario. In vertical federated learning, all the clients share the same sample ID (I) space and different feature spaces $(x_1, \ldots, x_n)$. Then a particular distributed client will have the dataset (I, x_l, Y). Different companies with different businesses having different feature spaces and the same user group, is an example of a vertically partitioned scenario. In this work, we consider only the horizontal federated learning scenario.

In the next section, we provide a detailed description of how these fundamentals are used in developing the proposed approach (DISTPAB) for distributed machine learning.

4. Our Approach: DISTPAB

This section explains the steps of developing a distributed data perturbation algorithm (named as DISTPAB) for a distributed data and machine learning. Figure 2 shows the application of privacy to a distributed healthcare ecosystem shown in Figure 1. As shown in the figure, the main goal of DISTPAB is to shift the perturbation to the distributed branches before the data leave the local edge and fog layer. However, in doing so, the algorithm should not lose global utility. To achieve that, the main branch (the research center) and the distributed branches will conduct a perturbation parameter exchange. As the perturbation is done using the globally optimal parameters, DISTPAB can maintain a proper balance between utility and privacy. During our explanations, we will be using the words node and entity alternatively, representing the same concept. However, more specifically, we consider an entity to be a fully populated institute, whereas a node to be one or more computing/processing devices within an entity.

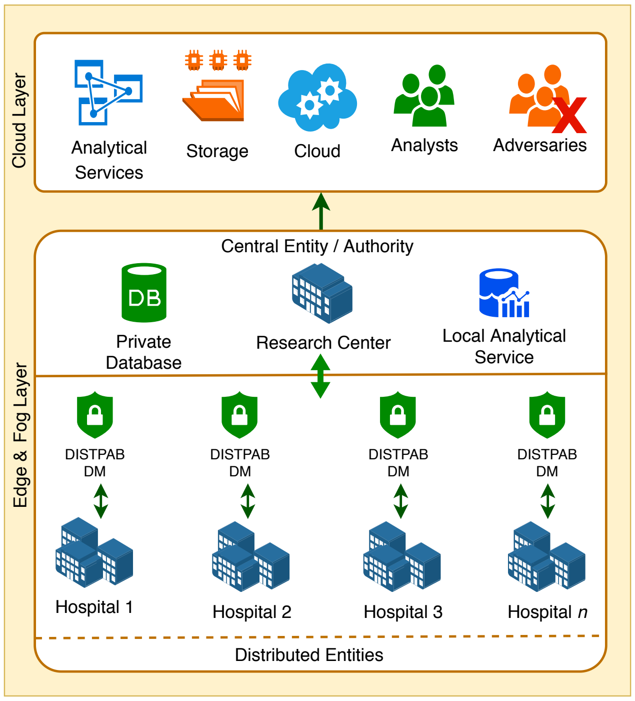


Figure 2: An example scenario where DISTPAB is integrated to a distributed healthcare system to preserve the privacy of data. The lowest layer represents the distributed set of entities (hospitals) and their interaction with DISTPAB within the fog and edge bounds. As the figure shows, the distributed components of DISTPAB are integrated into each of the distributed hospitals for performing the data perturbations. The central controlling entity (research centre) will be communicating with the cloud layer, which supports third-party analytical services. In this setup, we assume that no party saves original IoT data, and will only save the perturbed data.

The proposed algorithm delegates data maintenance and perturbation to the distributed entities, leaving only the global perturbation parameter generation to the central entity. Figure 3 shows the distributed architecture of DISTPAB, which can be used for the perturbation of healthcare data. In this way, the distributed branches do not have to share the original data with any untrusted third party. However, due to the coordination of the proposed algorithm in generating the global perturbation parameters, the perturbed data can provide high utility.

4.1. Centralized perturbation paradigm

DISTPAB first applies geometric transformations in the order of (1) reflection, (2) translation, and (3) rotation to an input dataset. Next, DISTPAB performs randomized expansion and random tuple shuffling to enhance the randomness of the perturbed data [29]. The perturbation is repeated until the perturbed dataset satisfies $\Phi - separation$. In order to devise the distributed paradigm in DISTPAB, we discuss the centralized approach, which is given in Algorithm 1. Algorithm 1 takes only two inputs: (1) the original dataset, (2) the standard deviation of the normal random noise generated for randomized expansion.

Before applying the composite geometric transformations, the algorithm derives the best perturbation parameters using the z-score normalized input data matrix. However, obtaining the best perturbation parameters at Φ for a big dataset can be extremely inefficient, as it involves running $D^{p1} = (M_i \times TN^{noise} \times RF_{ax} \times (D^N)^T)^T$ and $\phi_{i,ax} = min(Var(x_{ij} - p_{ij})_{i=1}^{m})_{j=1}^{n} \; \forall \; x_{ij} \in D^N$ for multiple perturbation instances (under the loops in Step 7 and 9 of Algorithm 1) of the original datasets. We can prove that these two steps are equal to $\phi_{i,ax} = min(\vec{1} + diag(M_i \times RF_{ax} \times CD^N \times RF_{ax} \times M_i^T) + sum((CD^N \times RF_{ax} \bullet M_i)^T)^T)$ [29] which is much simpler in time complexity as it uses the corresponding covariance matrix to determine the best perturbation parameters instead of browsing through the whole dataset in each iteration.

The algorithm maximizes the ϕ value in each iteration until it obtains Φ as given in Equation 10. The number of perturbed instances of D can be given as, $D^p_{1,1}, D^p_{1,2}, \ldots, D^p_{1,179}, D^p_{2,1}, D^p_{2,2}, \ldots, D^p_{2,179}, \ldots, D^p_{n,1}, D^p_{n,2}, \ldots, D^p_{n,179}$ where n represents the number of attributes, consequently, the axis of reflection varies from 1 to n. $0 < \theta < \pi$, where $\theta \notin \{\pi/6, \pi/4, \pi/3, \pi/2, 2\pi/3, 3\pi/4, 5\pi/6\}$. This will result in a matrix of ϕ values (local minimum privacy guarantees) as represented in Equation 8.

$$\begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \phi_{1,3} & \cdots & \phi_{1,179} \\ \phi_{2,1} & \phi_{2,2} & \phi_{2,3} & \cdots & \phi_{2,179} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_{n,1} & \phi_{n,2} & \phi_{n,3} & \cdots & \phi_{n,179} \end{bmatrix}_{n \times 179} \tag{8}$$

As given in Equation 9, we obtain the minimum value from each column of Equation 8 to identify the best ϕ_j for each instance of θ.

$$\begin{bmatrix}\phi_1 & \phi_2 & \phi_3 & \dots & \phi_{179}\end{bmatrix} \tag{9}$$

Now we choose Φ, as the largest of global minimum privacy guarantees (ϕ_j) as represented in Equation 10. This can also be explained as finding the highest privacy guarantee Φ that can be rendered by the most vulnerable attribute in the database.

$$\Phi = max([[\phi_j]_{j=1}^{179}]) \tag{10}$$

After determining the best perturbation parameters, the geometric transformations are conducted on the normalized input dataset following the order of reflection, noise translation, and rotation. We follow this order of transformations to avoid the points (in the n-dimensional space) closer to the origin, getting lower levels of perturbation. Rotation has a larger effect on the points that are far away from the origin compared to those which are close to the origin. As a result, the points which are close to the origin can get easily attacked. Equation 11 shows the order of the application of the three transformations on the normalized input data matrix, where rotation is conducted as the last transformation for the highest perturbation possible. As the next step, we add noise using randomized expansion. Reverse z-score normalization on the perturbed data is used to generate an output in the attribute ranges similar to the original dataset. Next, the tuples are randomly swapped in order to limit vulnerability to data linkage attacks.

$$D^p = (M \times T_{ND} \times RF_{\overline{ND}} \times (D^N)^T)^T \tag{11}$$

Algorithm 1: Centralized perturbation algorithm

```
Input:   $D$ ← original dataset
         $\sigma$ ← input noise standard deviation
             (default value=0.3)
Output:  $D^p$ ← perturbed dataset
1  $\Phi = 0$;
2  $\theta_{optimal} = 0$ ;
3  $Rif_{optimal} = 0$ ;
4  generate $D^N$;
5  generate $Cov(D^N)$;
6  generate $TN^{noise}$;
7  for each $ax$ in $\{1, 2, \ldots, n\}$ do
8      generate $RF_{ax}$ ;
9      for each $\theta_i$ do
10         generate $M_i$ using Algorithm 4;
11         $\phi_{i,ax} = min(\vec{1} + diag(M_i \times RF_{ax} \times Cov(D^N) \times RF_{ax} \times M_i^T) + sum((Cov(D^N) \times RF_{ax} \bullet M_i)^T)^T)$
12 for each $\theta_i$ do
13     $\phi_i = min(\phi_{i,ax})$ where, $ax \in \{1, 2, \ldots, n\}$;
14 $\Phi = max(\phi_i)$ where, $i \in \{1, 2, \ldots, \theta\}$;
15 $\theta_{optimal} = \theta_i$ at $\Phi$;
16 $Rif_{optimal} = ax$ at $\Phi$;
17 generate $M\theta$ ;
18 generate $RF_{optimal}$ ;
19 $D^{pt} = (M\theta \times TN^{noise} \times RF_{optimal} \times (D^N)^T)^T$ ;
20 $D^{pt} = (\|D^{pt}\| + \|\mathcal{N}(0, \sigma)\|) \bullet S_{\pm}$ ;
21 $D^p = D^{pt} \bullet STDVEC + MEANVEC$;
22 randomly swap the tuples of $D^p$ ;
```

4.2. Distributing the perturbation

In order to distribute the perturbation among distributed entities, we break the steps of Algorithm 1 into two main phases, as shown in Figure 3. (1) Generate the global perturbation parameters using the local properties forwarded to the central entity (research centre). (2) Conduct perturbation of data at the distributed entities using the global parameters. (3) Conduct machine learning on perturbed data. In order to achieve these three goals, first, the variance-covariance matrix of each partition of the dataset is passed to the central node, which will run Algorithm 3. Assume that there are k distributed branches which have the data partitions $D_1, D_2, \ldots, D_k$. Take D to be the dataset created by merging all the k datasets, as given in Equation 12.

$$D = merge(D_1, D_2, \ldots, D_k) \tag{12}$$

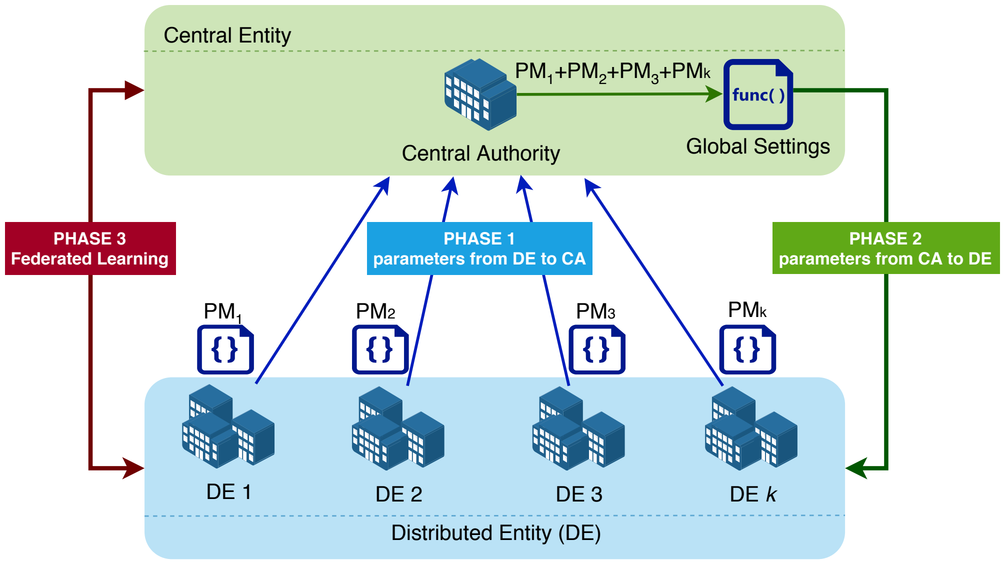


Figure 3: Distributed architecture of the perturbation algorithm. The figure shows the abstract view of the distributed perturbation scenario over distributed machine learning. There can be k number of distributed branches that are communicating with the central entity (coordinating server). There are two phases of parameter communication. Phase 1 involves the distributed branches sending local parameters sending to the central entity, whereas Phase 2 involves the central node sending the optimal perturbation parameters (which are calculated based on the local parameters which were sent by the distributed branches) to the distributed branches for perturbation.

To merge the covariance matrices, the pairwise covariance update formula introduced in [54] is adapted. The pairwise covariance update formula for the two merged two column (u and v) data partitions, A and B, can be written as shown in Equation 13 where the merged dataset is denoted as X.

$$Cov(X) = \frac{\frac{C_A}{(m_A-1)} + \frac{C_B}{(m_B-1)} + (\mu_{u,A} - \mu_{u,B})(\mu_{v,A} - \mu_{v,B}).\frac{m_A.m_B}{m_X}}{(m_X - 1)} \tag{13}$$

where $\mu_{u,A}, \mu_{u,A}, \mu_{v,A}, \mu_{v,B}$ are means of u and v of the two data partitions A and B respectively. C_A and C_B are the co-moments of the two data partitions A and B where the co-moment of a two column (u and v) dataset D is represented as

$$C_D = \sum_{(u,v)\in D} (u - \mu_u)(v - \mu_v) \tag{14}$$

Therefore, the variance-covariance matrix update formula of the two data partitions D_1 and D_2 can be written as shown in Equation 15.

$$Cov(D_{new}) = \frac{\frac{Cov(D_1)}{(m_{D_1}-1)} + \frac{Cov(D_2)}{(m_{D_2}-1)} + (\mu_{D_1}(MI_1) - \mu_{D_2}(MI_1))(\mu_{D_1}(MI_2) - \mu_{D_2}(MI_2)).\frac{m_{D_1}.m_{D_2}}{m_{D_{new}}}}{(m_{D_{new}}-1)} \tag{15}$$

where $Cov(D_{new}), Cov(D_1), Cov(D_2)$ are the covariance matrices of the merged dataset D_{new} and the data partitions D_1 and D_2 respectively. μ_{D_1} and μ_{D_2} are mean vectors of D_1 and D_2 respectively and

$$MI_1 = \begin{bmatrix} [1]_n \\ [2]_n \\ [3]_n \\ \vdots \\ [n]_n \end{bmatrix}_{n \times n} \tag{16}$$

$$MI_2 = (MI_1)^T \tag{17}$$

After generating the covariance matrix $(Cov(D))$ based on the global parameters (refer Equation 18), we can produce the vector of standard deviations ($STDVEC = \sqrt{diag(Cov(D))}$) based on the diagonal vector of $Cov(D)$. We can generate the vector of means ($MEANVEC$) using Equation 19 where $\bar{y_i} = Mean(D_i)$.

$$Cov(D) = \begin{bmatrix} Var_1 & Cov_{1,2} & \dots & Cov_{1,n} \\ Cov_{2,1} & Var_2 & \dots & Cov_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ Cov_{n,1} & Cov_{n,2} & \dots & Var_n \end{bmatrix}_{n \times n} \tag{18}$$

$$MEANVEC = \frac{m_1 \times \overline{y_1} + m_2 \times \overline{y_2} + \ldots + m_k \times \overline{y_k}}{m_1 + m_2 + \ldots + m_k} \tag{19}$$

Each distributed branch needs to generate the covariance matrix ($Cov(partition_i)$) and the mean vector ($Mean(partition_i)$) of the corresponding data partition and pass it to the central entity for the execution of Algorithm 3. After completing the execution of Algorithm 3, the central node will pass the required parameters to the distributed entities according to Algorithm 2, as shown in Figure 3. Table 1 shows the parameter exchange between the central entity and the distributed entities during the perturbation process.

Table 1: Data (parameter values) passed between the central entity and the distributed entities in the two phases of communications. As shown in Figure 3, in phase 1, the distributed entities send the local perturbation parameters to the central entity. In phase2, the central entity sends the global parameters (which are calculated using the local parameters) to the distributed entities.

Phase 1: from distributed nodes to central node **(Local parameters related to each data partition)**	**Phase 2: from centralized node to distributed nodes** **Global parameters related to all the partitions**
1. covariance matrix 2. vector of means 3. Number of attributes	1. random reflection matrix 2. random translation matrix 3. random rotation matrix 4. standard deviation for randomized expansion

4.3. Workload of a distributed entity

Algorithm 2 shows the workload of a distributed node (hospital) where the actual data perturbation is conducted. As the algorithm shows, the data will not be transmitted to the central entity as part of the perturbation process. First, the local parameters which are determined based on the local dataset will be sent to the central node. Next, the global parameters which are calculated and sent to the distributed entities will be used for the perturbation of the local dataset. However, the step of randomized expansion (refer step 7 of Algorithm 2) doesn't involve any interactions with the central node as randomized expansion needs to evaluate the noise based on each individual instance. Consequently, in the distributed setting, randomized expansion increases the randomization of the perturbed data over the centralized approach.

Algorithm 2: Task of a distributed entity

```
Input: $D_i$, $M\theta$, $TN^{noise}$, $RF_{optimal}$, $STDVEC$,
       $MEANVEC$
Output: $D_i^p$ ← perturbed data partition
1 generate $D_i^N$;
2 generate $Cov(D_i^N)$;
3 generate $Mean(D_i^N)$;
4 send Phase 1 data to the central entity;
5 receive Phase 2 data from the central entity;
6 $D_i^{pt} = (M\theta \times TN^{noise} \times RF_{optimal} \times (D_i^N)^T)^T$ ;
7 $D_i^{pt} = D_i^{pt} + \mathcal{N}(0, \min(\sigma_{D_i^{pt}}))$;
8 $D_i^p = D_i^{pt} \bullet STDVEC + MEANVEC$;
9 randomly swap the tuples of $D_i^p$;
```

4.4. Workload of the central node

Algorithm 3 shows the workload of the central entity. As shown in the algorithm, the central entity calculates the global perturbation parameters that need to be transmitted to the distributed entities. We assume that the distributed nodes are selected through a prior handshake mechanism that confirms the validity and reliability of the distributed nodes. However, since the proposed algorithm does not share the original data with any party before the perturbation, there will not be any privacy leak during the communication of the parameters.

Algorithm 3: Task of the central entity

```
Input:  $Cov(D_1), Cov(D_2), ..., Cov(D_k) \leftarrow$ Covariance matrices of the data partitions
        $Mean(D_1), Mean(D_2), ..., Mean(D_k) \leftarrow$ Mean vectors of the data partitions
Output: $M\theta$, $TN^{noise}$, $RF_{optimal}$, $STDVEC$, $MEANVEC$
receive Phase 1 data from the distributed entiteis;
$\Phi = 0$;
$\theta_{optimal} = 0$;
generate $TN^{noise}$;
$Cov(D_t) = Cov(D_1)$;
for each covariance matrix, $Cov(D_i)$ do
    if $i < k$ then
        $Cov(D_o) = Cov(D_{i+1})$;
        $Cov(D_{new}) = merge(Cov(D_t), Cov(D_o))$, according to Equation 15;
        $Cov(D_t) = Cov(D_{new})$;
$Cov(D) = Cov(D_{new})$;
$STDVEC = \sqrt{diag(Cov(D))}$;
$MEANVEC = combine(Mean(D_1), Mean(D_2), ..., Mean(D_k))$, using,
  $MEANVEC = \frac{m_1 \times \overline{y_1} + m_2 \times \overline{y_2} + ... + m_k \times \overline{y_k}}{m_1 +,_2 + ... + m_k}$,
where, $y_i = Mean(D_i)$;
for each $ax$ in $\{1, 2, \ldots, n\}$ do
    generate $RF_{ax}$ ;
    for each $\theta_i$ do
        generate $M_i$ using Algorithm 4;
        $\phi_{i,ax} = min(\vec{1} + diag(M_i \times RF_{ax} \times Cov(D^N) \times RF_{ax} \times M_i^T) + sum((Cov(D^N) \times RF_{ax} \bullet M_i)^T)^T)$
for each $\theta_i$ do
    $\phi_i = min(\phi_{i,ax})$ where, $ax \in \{1, 2, \ldots, n\}$ ;
$\Phi = max(\phi_i)$ where, $i \in \{1, 2, \ldots, \theta\}$;
$\theta_{optimal} = \theta_i$ at $\Phi$;
$Rif_{optimal} = ax$ at $\Phi$;
generate $M\theta$ ;
generate $RF_{optimal}$ ;
send Phase 2 data to the distributed entities;
```

4.5. Federated learning over perturbed data

As shown in Figure 3, the federated module comes into action after the perturbation is completed (in the first two phases). Each of the distributed entities will use the local data (perturbed) to train a local ML model (e.g. deep neural network) for a certain number of local epochs. After finishing the local epochs, the distributed entities will send the model parameters to the central repository to generate a global representation of the model by aggregating the model parameters (refer to section 3.9). The server (central authority) then passes the aggregated parameters back to the distributed entities to update the local models to generalize the models. This is called one round of federation. The federated learning setup will conduct a sufficient number of local epochs and federation rounds to train the ML models based on the requirements of the organization (e.g. production of data streams or big data), ML model architecture, and the properties of the input dataset.

5. Results

This section provides the experimental results of the performance of DISTPAB. We tested DISTPAB on six datasets to compare and evaluate its performance against three algorithms: RP (rotation perturbation), GP (geometric perturbation), and PABIDOT. We considered 0.3 as the default value of σ for the experiments unless specified otherwise. Next, we measured the performance of DISTPAB on FedML to examine the utility loss due to the perturbation. More details on the FedML setup is provided in section 5.4.1. For the experiments, we used a Windows 7 (Enterprise 64-bit, Build 7601) computer with an Intel(R) i7-4790 (4^{th} generation) CPU (8 cores, 3.60 GHz) and 8GB RAM. We declared a virtual distributed environment (refer Section 5.1) to run DISTPAB under the computer settings mentioned above. Table 2 has a summary of the datasets used for the experiments. We selected the datasets by considering a diverse range of domains. We first perturbed the data using RP, GP, PABIDOT, and DISTPAB. Next, we used the perturbed data to evaluate and compare the attack resistance and the classification accuracy of RP, GP, PABIDOT, and DISTPAB. For classification accuracy analysis, we used Weka 3.6 [55], which is a data mining tool that packages a collection of data mining algorithms. We used the following classification algorithms: Naive Bayes, k-nearest neighbor (kNN)[2], Support vector machine (SVM)[3], Multilayer perceptron (MLP), and J48 [55] to investigate the utility of the perturbed data.

[2]In Weka, kNN is named as IBk (Instance Based Learner)
[3]In Weka, Sequential Minimal Optimization (SMO) algorithm is used to train an SVM

Table 2: Information about the generic datasets used for the experiments. We selected the data based on varying dimensions from small to large in order to test the behavior of DISTPAB on different dynamics of the input data dimensions.

Dataset	**Abbreviation**	**Number of Records**	**Number of Attributes**	**Number of Classes**
Wholesale customers[4]	WCDS	440	8	2
Wine Quality[5]	WQDS	4898	12	7
Page Blocks Classification [6]	PBDS	5473	11	5
Letter Recognition[7]	LRDS	20000	17	26
Statlog (Shuttle)[8]	SSDS	58000	9	7
HEPMASS[9]	HPDS	3310816	28	2

5.1. Distributed computing setup

To test DISTPAB in a distributed setting, we used the "parpool" function in MATLAB [56]. "parpool(N)" creates N number of parallel processors (named as parallel workers) which can perform distributed computing. For performance testing and comparison, we distributed an input dataset among four workers (unless specified otherwise) by equally dividing the input dataset to make sure each distributed entity entails the same computational workload during the experiments.

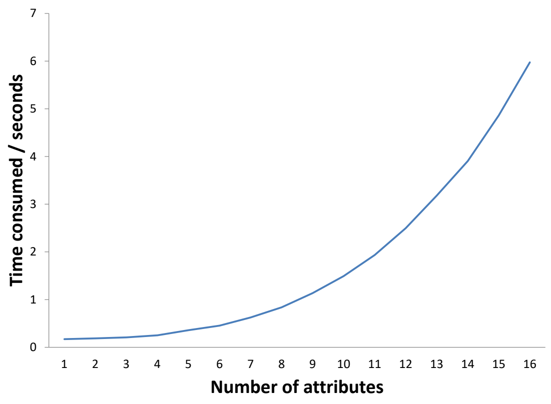
5.2. Time Complexity

We used both theoretical and runtime analysis to evaluate the computational complexity of DISTPAB. First, we conducted theoretical evaluations of the computational complexity of DISTPAB. Next, we conducted runtime analyses to provide empirical evidence on the estimated time complexities. The section from line 15 to line 19 (in Algorithm 1) has two loops that have a significant effect on the computational complexity of Algorithm 1 as one is constrained by the number of attributes whereas the other is controlled by the range of rotation angle. We can estimate that the runtime complexity of the corresponding loops is $O(k \times n) = O(n)$ (n = the number of attributes, m = the number of tuples, and k is a constant for the constant range of angle). We can estimate that step 11 (of Algorithm 1) is $O(n^3)$. The loop segment (starts from step 7 to 11) contributes a $O(n^4)$ complexity. In can also be estimated that step 19 introduces the highest computational complexity of $O(n^3 \times m)$ as $m >>> n$. In terms of worst-case computational complexity, we can state that Algorithm 1 has a computational complexity $O(n^3 \times m)$ when $m >>> n$. The steps 7 to 11 of Algorithm 1 are moved to Algorithm 3

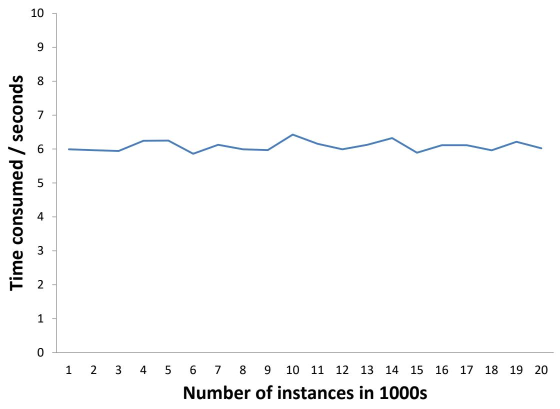
[4]https://archive.ics.uci.edu/ml/datasets/Wholesale+customers
[5]https://archive.ics.uci.edu/ml/datasets/Wine+Quality
[6]https://archive.ics.uci.edu/ml/datasets/Page+Blocks+Classification
[7]https://archive.ics.uci.edu/ml/datasets/Letter+Recognition
[8]https://archive.ics.uci.edu/ml/datasets/Statlog+%28Shuttle%29
[9]https://archive.ics.uci.edu/ml/datasets/HEPMASS#

which runs on the central node, and step 19 of Algorithm 1 is moved Algorithm 2 which runs on the distributed nodes. Consequently, the central node will have a computational complexity of $O(n^4)$. The distributed nodes will have a computational complexity of $O(n^3 \times m)$. However, the communication delay can increase the data perturbation time in a distributed setting.

Figures 4 and 5 show the empirical trends of time consumption by the central node and the distributed nodes in perturbing the LRDS dataset. During the time calculations, we considered only the time consumed for the perturbations by the corresponding processing unit (central or distributed) to get an absolute understanding of time consumption. We ignored all the other factors, such as communication delays and I/O operation delays. For this experiment, we considered four distributed nodes communicating with the central node (refer Section 5.1). We used the LRDS dataset for the analysis. During the study on how the time consumption is affected by the number of attributes, we distributed the LRDS dataset equally among the distributed nodes (each processing 5000 tuples). Next, we evaluated the effect of the number of distributed nodes on the classification accuracy and attack resistance by gradually increasing the number of distributed nodes from 1 to 20. For this experiment also, we used the LRDS dataset, and for each case, the dataset was distributed equally among the distributed nodes.

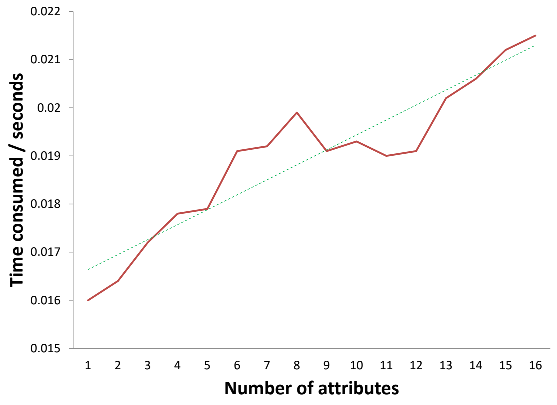


(a) The time consumption by the central entity for perturbation against the number of attributes. The plot confirms the theoretical evaluation of the central entity's time complexity, which is $O(n^4)$, where n is the number of attributes.

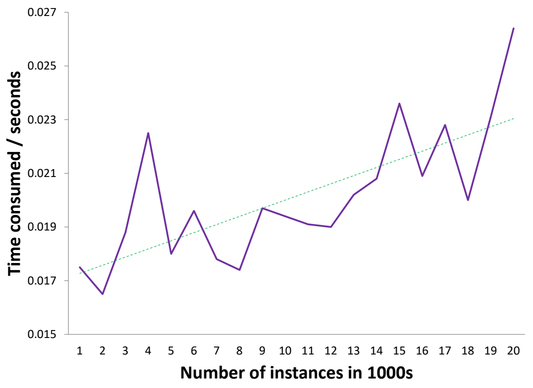


(b) The time consumption by the central entity for perturbation against the number of tuples. The plot confirms the theoretical evaluation of the central entity's time complexity, which is $O(n^4) = O(1)$ as n is constant. The central node consumes a constant amount of time when n (the number of attributes) is constant.

Figure 4: Time consumption by the central entity. The plots do not include communication delays, and they show the time consumption for the perturbation of the LRDS dataset.



(a) The average time consumption of the distributed nodes for the perturbation against the number of attributes. According to the time complexity of $O(n^3 \times m)$, as m remains constant, the plot should show a time complexity of $O(n^3)$ where m and n represent the number of tuples and the number of attributes respectively.



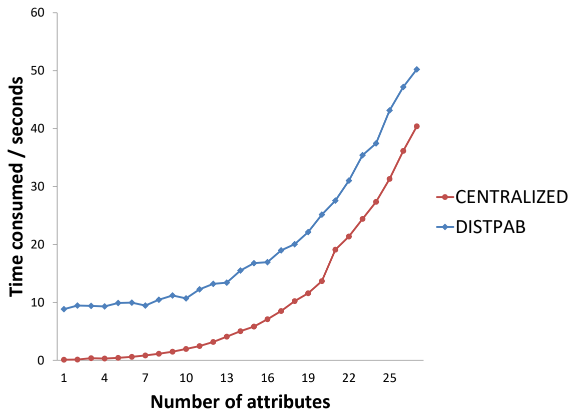
(b) The average runtime consumption of the distributed nodes for the perturbation against the number of instances. The plot confirms the theoretical evaluation of the time complexity, which is $O(m)$, where m is the number of tuples.

Figure 5: Average time consumption by the central entity (for 4 nodes, refer Section 5.1 for the specifications of the distributed setup). The plots do not include communication delays, and they show the time consumption for the perturbation of the LRDS dataset.

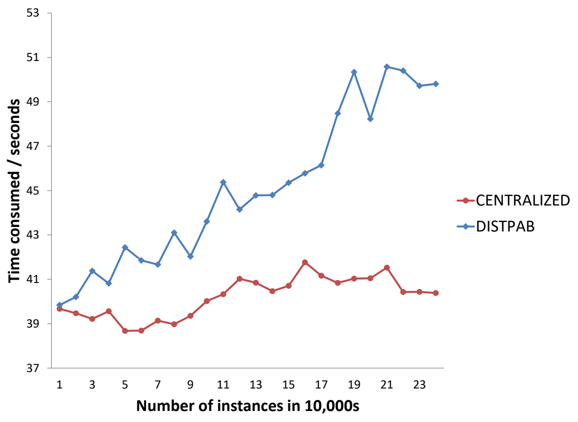
5.3. Complexity of distributed data perturbation vs. centralized data perturbation

In this section, we first compare the computational complexities of DISTPAB and PABIDOT (the centralized approach). Next, we investigate the impact of communication costs on data perturbation. As explained in Section 5.2, in DISTPAB, the central node has a worst-case computational complexity of $O(n^4)$, while a distributed node has a worst-case computational complexity of $O(n^3 \times m)$ (where n is the number of attributes and m is the number of instances). Since the number of attributes of a given setup is fixed and $m >>> n$, the computational complexity of DISTPAB is governed by $O(n^3 \times m)$. The worst-case computational complexity of PABIDOT is also $O(n^3 \times m)$ [29]. Consequently, for a given setup with a fixed number of attributes (a fixed number of IoT sensors), both DISTPAB and PABIDOT will provide linear complexity ($O(m)$). The only factor that would increase the time consumption of DISTPAB compared to PABIDOT is the data communication cost.

As shown in Table 1, during the data perturbation process, DISTPAB transfers 3 parameters (a matrix of size $n \times n$, a vector of size n, and a number) from a distributed node to the central entity, while 4 parameters (three matrices of size $n \times n$, and a standard deviation value) from the central node to a distributed node, where n is the number of attributes. Consequently, the total worst-case communication cost per node can be approximated as $O(4n^2 + n + 1)$, and for p number of nodes, the worst-case computational complexity can be approximated as $O(p(4n^2 + n + 1))$, which is independent of the number of data instances (tuples). For a given setup, the values of p and n are constants. Consequently, the communication cost is constant, no matter how many data tuples is introduced to DISTPAB. For empirical analysis of data communication costs, we use a large enough portion (240550×28) of the HPDS dataset to obtain a significant enough trend while avoiding extensive time consumption. Figure 6a shows the time consumption of the DISTPAB and the centralized version for an increasing number of attributes (the number of instances is kept constant). Figure 6b shows the time consumption of DISTPAB for an increasing number of instances (while the number of attributes is kept constant). In both cases, DISTPAB consumes more time than the centralized version (PABIDOT) due to communication delays. However, according to the plots, the effect of communication delays on DISTPAB can be considered reasonable.



(a) Time consumption of the centralized version and DISTPAB against the number of attributes. Both algorithms show similar trends (exponential) for the number of attributes.



(b) Time consumption of the centralized algorithm and DISTPAB against the number of tuples.The plots confirm the linear complexity of both versions for the number of tuples.

Figure 6: Impact of communication delay over distributed perturbation in DISTPAB. In a real-world scenario, network communication can be influenced by many factors, such as network latency and network congestion. It is complex to get an exact estimation over the communication delays. As the figures show, DISTPAB follows the expected time complexity (as available in the plot for the centralized algorithm) while adding the communication delays to its patterns.

5.4. Classification Accuracy

Although the perturbation is distributed, DISTPAB always generates optimal global perturbation parameters similar to the perturbation parameter generation of the centralized algorithm (Algorithm 1). However, the *randomized expansion* step is carried out by each distributed entity separately to improve the randomness of data. Consequently, the distributed version produces data with slightly higher randomization than its centralized version. This feature also reduces the utility of the data produced by the distributed version compared to the centralized version. Figure 7 shows the box plots of average accuracies produced by each of the algorithms against five classification algorithms. The classification accuracy was generated using 10-fold cross-validation. For the experiments on classification accuracy of a particular perturbed dataset, both classification model training and model testing were done using the same perturbed dataset. This is to check the model performance under strict privacy conditions where both the trained model and testing data do not leak any privacy to a third party. As shown in the figure, DISTPAB and PABIDOT produce similar classification accuracies. The figure also shows that both DISTPAB and PABIDOT outperform RP and GP.

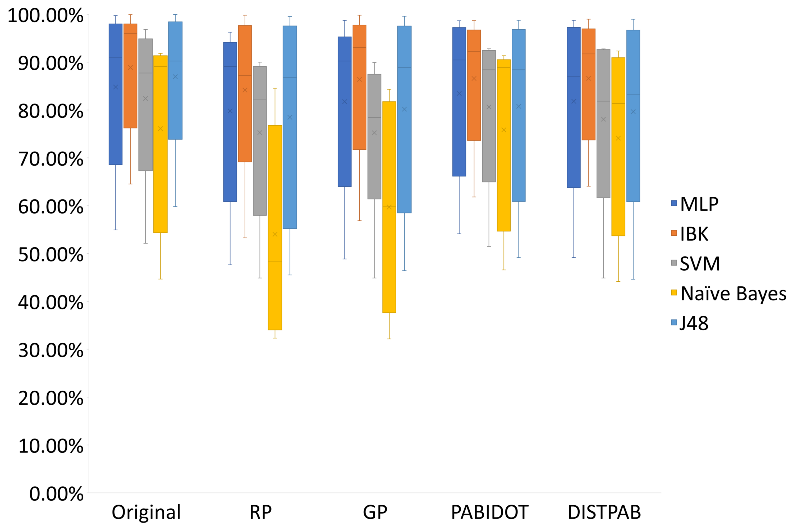


Figure 7: Classification Accuracy. The figure shows the variation of the classification accuracy produced by the original dataset and the datasets perturbed by the four perturbation approaches. In Weka, kNN is referred to as IBk (Instance Based Learner).

5.4.1. Federated learning setup

We simulated the federated learning setup using the Python socket programming interface and the _thread interface. In the default configuration, we considered 4 distributed clients. For this experiment, we used the SSDS dataset that has a sufficient number of tuples to see a noticeable impact on the performance while making a low burden on the computer resources (mentioned at the beginning of section 5). We used a train/test split of 75%/25% during the experiments. We chose a simpler ANN model architecture for local models to avoid overfitting and achieve high efficiency with better training and testing accuracy. We estimated the number of hidden neurons to have a good model but at the same time to avoid overfitting by having too many neurons; the values were chosen after considering the data at hand. Each client trained a fully connected neural network; activation='relu', batch size=64, hidden layer sizes=(10, 200, 200), final layer size = num of classes, final layer activation = 'softmax', shuffle=True, optimizer=‘SGD’, learning rate=‘constant’, learning rate init=0.0001, momentum=0.5, verbose=True. The models were trained for 20 federation rounds, while each model was locally trained for 3 epochs. However, it is essential to note that, for a different dataset, a different ANN architecture might need to be selected, as the performance of a particular ANN architecture is dependent upon the properties of the input dataset.

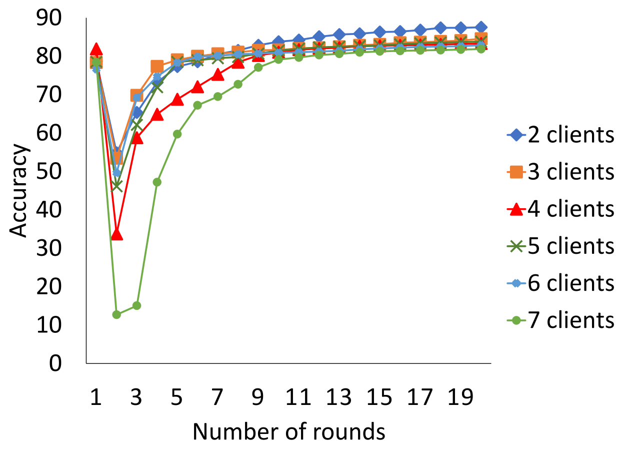


Figure 8: Classification accuracy during federated learning. The plots show the performance of federated learning on DISTPAB perturbed data under different numbers of distributed clients. The higher the number of clients, the higher the amount of time taken for ML model convergence.

Fig 8 shows the performance of the federated model using data perturbed by DISTPAB. The figure shows the performance against different number of distributed branches (clients). During the experiment, the input dataset was equally divided between distributed entities. Consequently, the higher the number of clients, the lower the number of tuples in each client. Thus, the model needs more federation rounds for convergence when the number of clients is higher. We can also notice that there is a sudden drop in accuracy in the second round, and the accuracy increases as the federation continues. This is due to the extensive parameter modification that takes place during the second round. The higher the number of clients, the more significant the effect that results in a more substantial drop in the accuracy.

In the proposed setting, DISTPAB works as a stand-alone data perturbation mechanism (a local data perturbation approach), which needs to be applied to the raw input data before subjected to federated learning. Hence, a model that would not converge on non-perturbed (raw/original) data could not be considered for convergence under perturbed data. From an empirical standpoint, according to Figure 7, we can notice that the classification accuracy generated by the perturbed data is very close to that of the original data. This implies that the perturbation employed by DISTPAB does not adversely affect the data representation. Consequently, we can assume that a model that can converge on the original raw data can converge on the perturbed data, subject to suitable hyperparameter tuning. However, more theoretical analysis is necessary to identify the exact effect of the perturbation on model convergence. The theoretical proof of convergence is beyond the scope of this paper [57, 58]. We consider this as a future direction of the proposed work.

5.5. Scalability

The scalability of the proposed approach is based on three factors: (1) the increase in the number of instances (number of tuples), (2) the increase in the number of attributes, and (3) the increase in the number of distributed entities. According to the computational complexity analysis (refer Section 5.2), DISTPAB provides a linear complexity towards an increasing number of instances (for a given setup where the number of sensors (attributes) is fixed). Hence, for a given setup where the number of attributes and the number of distributed entities are fixed, DISTPAB will provide high scalability. When the number of sensors of the environment (which corresponds to the number of attributes of the dataset) is increased, the worst-case computational complexity changes to $O(n^4)$, where n is the number of attributes. However, the number of attributes would not increase as frequently as the number of instances, and this increase is much smaller compared to the instances. When it comes to the distributed entities, the number of parameter communications will increase with the number of distributed entities. Compared to the number of instances, and the number of attributes, the increase in the number of distributed entities is extremely small. Hence, the increase in the number of distributed entities will not have a drastic impact on the scalability of DISTPAB. Hence, we can consider DISTPAB to provide high scalability towards distributed data perturbation and machine learning.

5.6. Effect of the distribution of data instances (tuples) among distributed entities on the performance of perturbation

A distributed entity can hold a certain number of data instances that match the corresponding computing node's capacity. In a conventional distributed setup, the distributed entities can have different computational resources and hold different numbers of data instances. It is important to investigate how this asymmetry can play a role in the performance of data perturbation employed by DISTPAB. Different distributed entities with different numbers of instances will consume different amounts of perturbation time. For a given distributed setup, a distributed entity has a data perturbation complexity of $O(m)$ (refer to Section 5.2). Besides, as discussed in Section 5.3, when the number of attributes is fixed, DISTPAB exhibits an overall complexity (computation + communication) of $O(m)$. Hence, the differences in the numbers of instances in distributed entities will not have an adverse effect on the overall complexity of DISTPAB. As explained in Section 4, before the application of perturbation, DISTPAB generates the global perturbation parameters based on the local perturbation parameters retrieved from the distributed entities. As a result of this step, the perturbation effect applied by DISTPAB is almost the same as the effect of perturbation applied by the centralized approach (PABIDOT). As shown in Figure 7, DISTPAB ends up producing classification accuracy similar to that of PABIDOT. Hence, there is no effect of the number of instances in each distributed entity on the classification accuracy.

5.7. Attack Resistance

We investigated the attack resistance of DISTPAB against Known I/O (IO) attacks, naive inference (NI), and Independent Component Analysis (ICA) [28] which are considered to be three potential attacks on matrix multiplicative approaches. We considered the default values of ten iterations and a sigma (noise factor) of 0.3 for the experiments on GP and RP. We obtained $std(D - D^p))$ (std represents the standard deviation, D the input data matrix, D^p represents the perturbed data of D), which provides an estimate to the resistance against NI. Next, we applied ICA and IO on the perturbed data to generate reconstructed data. We assumed that 10% of the original data as the background knowledge of an adversary during the IO attack investigation. We obtained the minimum values (NImin, ICAmin, and IOmin) of the $std(D - D^p)$ under each attack type and plotted them in Figure 9. As shown in the figure, both PABIDOT and DISTPAB provide better attack resistance compared to RP and GP. We can see a slight increment in the minimum attack resistance of DISTPAB compared to PABIDOT due to the enhanced randomization as the *randomized expansion* step is carried out by each distributed entity separately to improve the randomness of data.

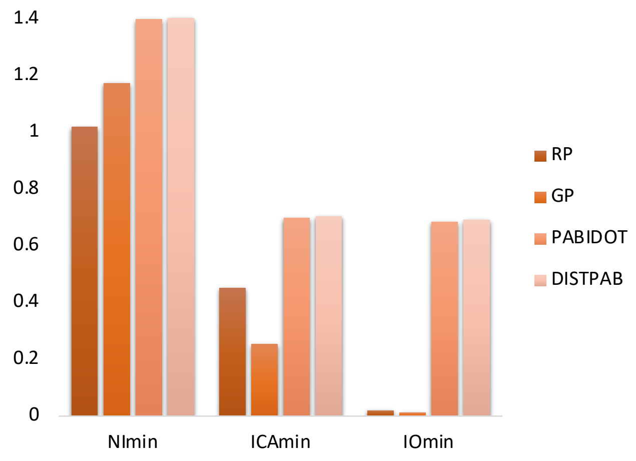


Figure 9: Attack Resistance. The figure shows the plots of the minimum values (NImin, ICAmin, and IOmin) of the $std(D - D^p)$ under NI, ICA, and IO attacks. The higher the bar, the better the resistance to attacks.

6. Discussion

This paper proposed an efficient distributed privacy preservation mechanism (named as DISTPAB) for distributed machine learning. DISTPAB applies randomized $n - dimensional$ geometric transformations followed by randomized expansion, which is a noise application mechanism that improves the positiveness and negativeness of input data while improving randomization (used to further improve the randomization without harming the utility [29]). DISTPAB uses $\Phi - separation$ [29] as the underlying privacy model to obtain the optimal perturbation parameters and generate optimal privacy for the input data. We tested and compared the performance of DISTPAB against PABIDOT, RP, and GP for the classification accuracy, time complexity, and attack resistance. Additionally, we investigated the impact of communication delay over distributed perturbation in DISTPAB, and further analyzed the impact of the number of distributed nodes on the classification during federated machine learning.

According to our time complexity analysis, DISTPAB introduces a time complexity of $O(n^4)$ to the central entity, where n represents the number of attributes. However, $O(n^4) = O(1)$, as n remains a constant for a given setting. Therefore, the central entity would consume a constant amount of time, no matter how many instances (tuples) are introduced. A distributed entity shows a time complexity of $O(n^3 \times m) = O(m)$ as n (the number of attributes) is a constant for a given dataset (m represents the number of tuples). When we consider a fixed distributed setup, new parameters/sensors are rarely added at a speed in which the data grows; and more probably it will remain constant. For a fixed distributed setting with a fixed number of distributed entities, the communication cost shows a constant complexity ($O(1)$). Consequently, for a given scenario, the amount of time consumed by a distributed node will grow linearly, which is optimal for privacy-preserving distributed machine learning.

The empirical evidence on data classification shows that DISTPAB provides similar classification accuracy to the centralized algorithm and better performance compared to RP and GP. DISTPAB provides slightly lower classification accuracy compared to the centralized approach as DISTPAB imposes an increased level of randomization as the *randomized expansion* step is carried out by each distributed entity separately to improve the randomness of data. The empirical evidence (refer to Figure 8) proved that the number of distributed clients doesn't have a noticeable impact on classification accuracy during federated learning (distributed machine learning). However, the amount of time necessary for model convergence increases with the number of clients. This feature makes DISTPAB be the perfect solution for privacy-preserving federated learning (privacy-preserving distributed machine learning) where there is a large number of distributed entities involved.

DISTPAB uses $\Phi - separation$ [29] as its underlying privacy model, which allows optimal data perturbation for a given instance. Geometric data transformations and randomized expansion noise addition followed by random shuffling allow DISTPAB to impose high privacy by reducing the probability of data reconstruction attacks. Reverse z-score normalization of the final dataset takes the data back to their original attribute value ranges, making the attackers unable to distinguish the original data from the perturbed data. This feature reduces the chance of success of an attack trying to reconstruct the original input data from a perturbed dataset. DISTPAB provides better attack resistance compared to RP and GP. We can observe that the DISTPAB provides slightly better attack resistance compared to PABIDOT because DISTPAB adds an increased level of randomization as the randomized expansion step is carried out by each distributed entity separately to improve the randomness of data. However, in a federated learning setup, we do not share the perturbed data among the distributed clients or the server. Consequently, there will not be any data reconstruction attacks on the perturbed data, and due to the strong notion of privacy of the perturbed data, the privacy of the trained models will also be high.

In essence, DISTPAB can be an optimal privacy preservation solution for data privacy and machine learning that control extensive amounts of data [59], which are deployed in geographically distributed systems.

7. Conclusions

Many modern systems, such as healthcare and open banking, are often geographically distributed and constrained with proper mechanisms for privacy-preserving data sharing for analytics. This paper proposed a distributed perturbation algorithm named DISTPAB that can enforce privacy for distributed machine learning. In the proposed setup of DISTPAB, a central/coordinating entity controls the global perturbation parameter generation, whereas the distributed entities can conduct local data perturbation. The computational complexity of the algorithm that runs in the central entity is $O(1)$ (constant time) for the number of instances. The computational complexity of the algorithm, which runs on a distributed entity, is $O(m)$ for the number of tuples as the number of attributes often remains constant in a given setting where m is the number of instances. Consequently, the operations on distributed entities have a low computational complexity resulting in excellent efficiency. DISTPAB provides high classification accuracy, which is close to that of the accuracy of classification performed with the original data. DISTPAB provides high attack resistance outperforming rotation perturbation and geometric perturbation. It was also shown that the data produced by DISTPAB is not subjected to utility degradation, against the number of distributed entities. DISTPAB can be an excellent privacy preservation algorithm for distributed machine learning.

As future work, we are interested in looking at improving the efficiency of the proposed work for the number of attributes. To achieve this, we will investigate on the vertical federated learning scheme where the distributed clients have different feature spaces. Vertical federated learning allows dividing a particular dataset into partitions by the attributes leaving a specific client to work only on a fewer number of attributes, which can improve efficiency.

Appendix .1. n-Dimensional translation matrix generation

Equation .1 represents the *n-dimensional* homogeneous translation matrix T_{ND} [52]. The translational coefficients $(t_{i(n+1)}, \ldots, t_{(n)(n+1)})$ are drawn from a uniform random distribution which is bounded within $(0, 1)$. The uniform random noise is restricted to $0 < t_{i(n+1)} < 1$ as the input dataset's attribute standard deviations and means become 1 and 0 after the z-score normalization.

$$T_{ND} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & t_{1(n+1)} \\ 0 & 1 & 0 & \dots & 0 & t_{2(n+1)} \\ 0 & 0 & 1 & \dots & 0 & t_{3(n+1)} \\ \vdots & \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & 1 & t_{(n)(n+1)} \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}_{(n+1)\times(n+1)} \tag{.1}$$

Appendix .2. n-Dimensional reflection matrix

Equation .2 shows the homogeneous reflection matrix RF_{ND}. Equation .2 represents the reflection across axis one [52]. The (n+1) axis reflection for the matrix given in Equation .2 can be written as shown in Equation .3, which provides a low level of bias in the perturbation.

$$RF_{ND} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & -1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & & \ddots & & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & -1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix}_{(n+1)\times(n+1)} \tag{.2}$$

Equation .3 represents The (n+1) axis reflection matrix matrix.

$$RF_{\overline{ND}} = \begin{bmatrix} -1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & & \ddots & & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix}_{(n+1)\times(n+1)} \tag{.3}$$

Appendix .3. Generating the rotational matrix

Algorithm 4 provides the steps in generating the rotational matrix using the concept of concatenated subplane rotation. To represent the entire $n-dimensional$ orientation, we use the concept of the concatenated sub-plane rotation. The block matrix given in Equation .4, shows the rotations on the plane represented by a pair of coordinate axes $(\hat{x_i}, \hat{x_j})$ for $1 \leq i, j \leq n$ [60]. Thus $\frac{N(N-1)}{2}$ distinct $R_{ij}(\theta_{ij})$ should be concatenated in a particular order to generate the composite *n-dimensional* orthonormal matrix as represented in Equation .5 with $\frac{N(N-1)}{2}$ degrees of freedom, parameterized by θ_{ij}. We generate M , the *n-dimensional* concatenated subplane rotation matrix for an angle, θ_i using Algorithm 4 (in Appendix .3) for each θ_i where $((0 < \theta_i < \pi), \quad \theta_i \notin \{\pi/6, \pi/4, \pi/3, \pi/2, 2\pi/3, 3\pi/4, 5\pi/6\})$ [29].

$$R_{ij}(\theta_{ij}) = \begin{bmatrix} 1 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & cos\theta_{ij} & 0 & \dots & 0 & -sin\theta_{ij} & \dots & 0 \\ 0 & \dots & 0 & 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & \dots & sin\theta_{ij} & 0 & \dots & 0 & cos\theta_{ij} & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 1 \end{bmatrix}_{(n+1)\times(n+1)} \tag{.4}$$

$$M = \prod_{i<j} R_{ij}(\theta_{ij}) \tag{.5}$$

Algorithm 4: Rotation matrix generation

```
Input:  $n$ ← number of attributes of the input dataset
        $\theta$ ← angle of rotation
Output: $M$ ← multidimensional rotation matrix of $\theta$
1  $V = \{1, 2, 3 \dots n\}$;
2  $C = \{(i, j) | i, j \in V$ and $i \neq j\}$;
3  $I_n$ = identity matrix of size $n$ ;
4  $N = \binom{n}{2}$;
5  $I3 = I_n$;
6  for $k = 1$ to $N$ do
7      $A = \{(i_k, i_k), (j_k, i_k), (i_k, j_k), (j_k, j_k)\}$ where $\{i_k, j_k\}$ is the $k^{th}$ set of $C$;
8      $I2 = I_n$;
9      $I2(i_{k1}, j_{k1}) = cos(\theta)$ where, $(i_{k1}, j_{k1})$ is the set number 1 of $A$;
10     $I2(i_{k2}, j_{k2}) = sin(\theta)$ where, $(i_{k2}, j_{k2})$ is the set number 2 of $A$;
11     $I2(i_{k3}, j_{k3}) = -sin(\theta)$ where, $(i_{k3}, j_{k3})$ is the set number 3 of $A$;
12     $I2(i_{k4}, j_{k4}) = cos(\theta)$ where, $(i_{k4}, j_{k4})$ is the set number 4 of $A$;
13     $I3 = I3 \times I2$
14 $M = I3$;
```

References

[1] T. Tegegne, T. P. T. van der Weide, Enriching queries with user preferences in healthcare, Information Processing & Management 50 (4) (2014) 599–620.

[2] K. J. Kim, D.-H. Shin, H. Yoon, Information tailoring and framing in wearable health communication, Information Processing & Management 53 (2) (2017) 351–358.

[3] O. Șerban, N. Thapen, B. Maginnis, C. Hankin, V. Foot, Real-time processing of social media with sentinel: a syndromic surveillance system incorporating deep learning for health classification, Information Processing & Management 56 (3) (2019) 1166–1184.

[4] M. A. Khan, K. Salah, Iot security: Review, blockchain solutions, and open challenges, Future Generation Computer Systems 82 (2018) 395–411.

[5] P. C. M. Arachchige, P. Bertok, I. Khalil, D. Liu, S. Camtepe, M. Atiquzzaman, A trustworthy privacy preserving framework for machine learning in industrial iot systems, IEEE Transactions on Industrial Informatics.

[6] P. C. M. Arachchige, P. Bertok, I. Khalil, D. Liu, S. Camtepe, M. Atiquzzaman, Local differential privacy for deep learning, IEEE Internet of Things Journal.

[7] M. Chamikara, P. Bertok, D. Liu, S. Camtepe, I. Khalil, An efficient and scalable privacy preserving algorithm for big data and data streams, Computers & Security 87 (2019) 101570.

[8] M. A. P. Chamikara, A. Galappaththi, R. D. Yapa, R. D. Nawarathna, S. R. Kodituwakku, J. Gunatilake, A. A. C. A. Jayathilake, L. Liyanage, Fuzzy based binary feature profiling for modus operandi analysis, PeerJ Computer Science 2 (2016) e65.

[9] A. Alabdulatif, I. Khalil, A. R. M. Forkan, M. Atiquzzaman, Real-time secure health surveillance for smarter health communities, IEEE Communications Magazine 57 (1) (2018) 122–129.

[10] A. Alabdulatif, I. Khalil, X. Yi, M. Guizani, Secure edge of things for smart healthcare surveillance framework, IEEE Access 7 (2019) 31010–31021.

[11] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan, et al., Towards federated learning at scale: System design, arXiv preprint arXiv:1902.01046.

[12] E. Bertino, D. Lin, W. Jiang, A survey of quantification of privacy preserving data mining algorithms, in: Privacy-preserving data mining, Springer, 2008, pp. 183–205.

[13] P. Samarati, Protecting respondents identities in microdata release, IEEE transactions on Knowledge and Data Engineering 13 (6) (2001) 1010–1027.

[14] M. Chamikara, P. Bertok, D. Liu, S. Camtepe, I. Khalil, Efficient data perturbation for privacy preserving and accurate data stream mining, Pervasive and Mobile Computing 48 (2018) 1–19.

[15] N. López, F. Sebé, Privacy preserving release of blogosphere data in the presence of search engines, Information Processing & Management 49 (4) (2013) 833–851.

[16] A. Bilge, H. Polat, A scalable privacy-preserving recommendation scheme via bisecting k-means clustering, Information Processing & Management 49 (4) (2013) 912–927.

[17] K. Li, L. Cheng, C.-I. Teng, Voluntary sharing and mandatory provision: Private information disclosure on social networking sites, Information Processing & Management 57 (1) (2020) 102128.

[18] J. Zhou, Z. Cao, X. Dong, A. V. Vasilakos, Security and privacy for cloud-based iot: Challenges, IEEE Communications Magazine 55 (1) (2017) 26–33.

[19] A. Yargic, A. Bilge, Privacy-preserving multi-criteria collaborative filtering, Information Processing & Management 56 (3) (2019) 994–1009.

[20] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning: Concept and applications, ACM Transactions on Intelligent Systems and Technology (TIST) 10 (2) (2019) 12.

[21] C. Thapa, M. A. P. Chamikara, S. Camtepe, Splitfed: When federated learning meets split learning, arXiv preprint arXiv:2004.12088.

[22] C. Song, T. Ristenpart, V. Shmatikov, Machine learning models that remember too much, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2017, pp. 587–601.

[23] R. Shokri, M. Stronati, C. Song, V. Shmatikov, Membership inference attacks against machine learning models, in: Security and Privacy (SP), 2017 IEEE Symposium on, IEEE, 2017, pp. 3–18. doi:https://doi.org/10.1109/SP.2017.41.

[24] M. Fredrikson, S. Jha, T. Ristenpart, Model inversion attacks that exploit confidence information and basic countermeasures, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, 2015, pp. 1322–1333.

[25] M. Akgün, A. O. Bayrak, B. Ozer, M. Ş. Sağıroğlu, Privacy preserving processing of genomic data: A survey, Journal of biomedical informatics 56 (2015) 103–111.

[26] K. Chen, L. Liu, A random rotation perturbation approach to privacy preserving data classification, The Ohio Center of Excellence in Knowledge-Enabled Computing.
URL https://corescholar.libraries.wright.edu/knoesis/916/

[27] K. Chen, L. Liu, Geometric data perturbation for privacy preserving outsourced data mining, Knowledge and Information Systems 29 (3) (2011) 657–695. doi:https://doi.org/10.1007/s10115-010-0362-4.

[28] B. D. Okkalioglu, M. Okkalioglu, M. Koc, H. Polat, A survey: deriving private information from perturbed data, Artificial Intelligence Review 44 (4) (2015) 547–569. doi:https://doi.org/10.1007/s10462-015-9439-5.

[29] M. A. P. Chamikara, P. Bertók, D. Liu, S. Camtepe, I. Khalil, Efficient privacy preservation of big data for accurate data mining, Information Sciences 527 (2020) 420–443.

[30] V. Oleshchuk, Internet of things and privacy preserving technologies, in: 2009 1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, IEEE, 2009, pp. 336–340.

[31] A. Hasan, Q. Jiang, J. Luo, C. Li, L. Chen, An effective value swapping method for privacy preserving data publishing, Security and Communication Networks 9 (16) (2016) 3219–3228. `doi:https://doi.org/10.1002/sec.1527`.

[32] K. Muralidhar, R. Parsa, R. Sarathy, A general additive data perturbation method for database security, management science 45 (10) (1999) 1399–1415. `doi:https://doi.org/10.1287/mnsc.45.10.1399`.

[33] C. C. Aggarwal, P. S. Yu, A condensation approach to privacy preserving data mining, in: EDBT, Vol. 4, Springer, 2004, pp. 183–199. `doi:https://doi.org/10.1007/978-3-540-24741-8_12`.

[34] J. A. Fox, Randomized response and related methods: Surveying Sensitive Data, Vol. 58, SAGE Publications, 2015.
URL `https://books.google.com.au/books?isbn=1483381056`

[35] J. Soria-Comas, J. Domingo-Ferrer, D. Sánchez, S. Martínez, t-closeness through microaggregation: Strict privacy with enhanced utility preservation, IEEE Transactions on Knowledge and Data Engineering 27 (11) (2015) 3098–3110. `doi:https://doi.org/10.1109/TKDE.2015.2435777`.

[36] K. Liu, H. Kargupta, J. Ryan, Random projection-based multiplicative data perturbation for privacy preserving distributed data mining, IEEE Transactions on knowledge and Data Engineering 18 (1) (2006) 92–106. `doi:https://doi.org/10.1109/TKDE.2006.14`.

[37] Y. A. A. S. Aldeen, M. Salleh, M. A. Razzaque, A comprehensive review on privacy preserving data mining, SpringerPlus 4 (1) (2015) 694. `doi:https://doi.org/10.1186/s40064-015-1481-x`.

[38] A. Machanavajjhala, D. Kifer, Designing statistical privacy for your data, Communications of the ACM 58 (3) (2015) 58–67. `doi:https://doi.org/10.1145/2660766`.

[39] B. Niu, Q. Li, X. Zhu, G. Cao, H. Li, Achieving k-anonymity in privacy-aware location-based services, in: INFOCOM, 2014 Proceedings IEEE, IEEE, 2014, pp. 754–762. doi:https://doi.org/10.1109/INFOCOM.2014.6848002.

[40] G. Navarro-Arribas, V. Torra, A. Erola, J. Castellà-Roca, User k-anonymity for privacy preserving data mining of query logs, Information Processing & Management 48 (3) (2012) 476–487.

[41] A. Machanavajjhala, J. Gehrke, D. Kifer, M. Venkitasubramaniam, l-diversity: Privacy beyond k-anonymity, in: Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on, IEEE, 2006, pp. 24–24. doi:https://doi.org/10.1109/ICDE.2006.1.

[42] N. Li, T. Li, S. Venkatasubramanian, t-closeness: Privacy beyond k-anonymity and l-diversity, in: Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on, IEEE, 2007, pp. 106–115. doi:https://doi.org/10.1109/ICDE.2007.367856.

[43] R. C.-W. Wong, J. Li, A. W.-C. Fu, K. Wang, (α, k)-anonymity: an enhanced k-anonymity model for privacy preserving data publishing, in: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2006, pp. 754–759. doi:https://doi.org/10.1145/1150402.1150499.

[44] C. Carpineto, G. Romano, Kθ-affinity privacy: Releasing infrequent query refinements safely, Information Processing & Management 51 (2) (2015) 74–88.

[45] S. R. Ganta, S. P. Kasiviswanathan, A. Smith, Composition attacks and auxiliary information in data privacy, in: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2008, pp. 265–273. doi:https://doi.org/10.1145/1401890.1401926.

[46] L. Zhang, S. Jajodia, A. Brodsky, Information disclosure under realistic assumptions: Privacy versus optimality, in: Proceedings of the 14th ACM conference on Computer and communications security, ACM, 2007, pp. 573–583. doi:https://doi.org/10.1145/1315245.1315316.

[47] R. C.-W. Wong, A. W.-C. Fu, K. Wang, P. S. Yu, J. Pei, Can the utility of anonymized data be used for privacy breaches?, ACM Transactions on Knowledge Discovery from Data (TKDD) 5 (3) (2011) 16. doi:https://doi.org/10.1145/1993077.1993080.

[48] C. C. Aggarwal, Privacy and the dimensionality curse, Privacy-Preserving Data Mining (2008) 433–460.doi:https://doi.org/10.1007/978-0-387-70992-5_18.

[49] C. Bettini, D. Riboni, Privacy protection in pervasive systems: State of the art and technical challenges, Pervasive and Mobile Computing 17 (2015) 159–174. doi:https://doi.org/10.1016/j.pmcj.2014.09.010.

[50] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, B. Thorne, Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption, arXiv preprint arXiv:1711.10677.

[51] J. Maruskin, Essential Linear Algebra, Solar Crest Publishing, LLC, 2012.
URL https://books.google.com.au/books?id=aOF3-hx3u1kC

[52] H. Jones, Computer Graphics through Key Mathematics, Springer London : Imprint: Springer, 2012.
URL https://books.google.com.au/books?id=f7gPBwAAQBAJ

[53] W. Kabir, M. O. Ahmad, M. Swamy, A novel normalization technique for multimodal biometric systems, in: Circuits and Systems (MWSCAS), 2015 IEEE 58th International Midwest Symposium on, IEEE, 2015, pp. 1–4. doi:https://doi.org/10.1109/MWSCAS.2015.7282214.

[54] J. Bennett, R. Grout, P. Pébay, D. Roe, D. Thompson, Numerically stable, single-pass, parallel statistics algorithms, in: Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on, IEEE, 2009, pp. 1–8.

[55] I. H. Witten, E. Frank, M. A. Hall, C. J. Pal, Data Mining: Practical machine learning tools and techniques, Morgan Kaufmann, 2016.
URL https://books.google.com.au/books?isbn=0128043571

[56] A. S. Leon, C. Goodell, Controlling hec-ras using matlab, Environmental modelling & software 84 (2016) 339–348.

[57] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, K. Chan, Adaptive federated learning in resource constrained edge computing systems, IEEE Journal on Selected Areas in Communications 37 (6) (2019) 1205–1221.

[58] C. Dinh, N. H. Tran, M. N. Nguyen, C. S. Hong, W. Bao, A. Zomaya, V. Gramoli, Federated learning over wireless networks: Convergence analysis and resource allocation, arXiv preprint arXiv:1910.13067.

[59] G. Manogaran, C. Thota, D. Lopez, V. Vijayakumar, K. M. Abbas, R. Sundarsekar, Big data knowledge system in healthcare, in: Internet of things and big data technologies for next generation healthcare, Springer, 2017, pp. 133–157. doi:https://doi.org/10.1007/978-3-319-49736-5_7.

[60] A. W. Paeth, Graphics Gems V (Macintosh Version), Academic Press, 2014.
URL https://books.google.com.au/books?isbn=1483296695