Privacy-Preservation for Gradient Descent Methods[*]

Li Wan
School of Computer Engineering
Nanyang Technological University
Singapore
wanl0001@ntu.edu.sg

Wee Keong Ng
School of Computer Engineering
Nanyang Technological University
Singapore
awkng@ntu.edu.sg

Shuguo Han
School of Computer Engineering
Nanyang Technological University
Singapore
hans0004@ntu.edu.sg

Vincent C. S. Lee
School of Business Systems
Monash University
Australia
vincent.lee@infotech.monash.edu.au

ABSTRACT

Gradient descent is a widely used paradigm for solving many optimization problems. Stochastic gradient descent performs a series of iterations to minimize a target function in order to reach a local minimum. In machine learning or data mining, this function corresponds to a decision model that is to be discovered. The gradient descent paradigm underlies many commonly used techniques in data mining and machine learning, such as neural networks, Bayesian networks, genetic algorithms, and simulated annealing. To the best of our knowledge, there has not been any work that extends the notion of privacy preservation or secure multi-party computation to gradient-descent-based techniques. In this paper, we propose a preliminary approach to enable privacy preservation in gradient descent methods in general and demonstrate its feasibility in specific gradient descent methods.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*; H.2.7 [**Database Management**]: Database Administration—*Security, integrity, protection*

General Terms

Theory, Algorithms, Security

Keywords

Privacy Preservation, Gradient Descent Method, Secure Multi-party Computation, Regression

[*]This work is supported in part by grant P0520095 from the Agency for Science, Technology and Research (A*STAR), Singapore.

1. INTRODUCTION

Many techniques in data mining and machine learning follow a gradient descent paradigm in the iterative process of discovering a target function or decision model. For instance, neural networks generally perform a series of iterations to converge the weight coefficients of edges in the network; thus, settling into a decision model. Linear regression is a basic statistical method that finds a function to correlate two or more attributes. Linear regression can also be resolved through a gradient descent method that iteratively minimizes the error of the target function. Other gradient-descent-based methods include Bayesian networks induction, genetic algorithms, and simulated annealing.

Secure multi-party computation and privacy preservation have attracted much attention recently in incorporating security into data mining and machine learning algorithms. A key issue in multi-party secure methods is to allow individual parties to preserve the privacy of its data, while contributing to the computation of a global result together with other parties. Many methods have been proposed to perform Secure Multi-party Computation (SMC) on various basic operations required in data mining. For instance, the scalar product is a basic operation in inducing decision trees and association rule mining that can now be performed securely involving two or more parties [1, 3]. Basic matrix operations such as matrix multiplication and matrix inversion have also been extended in a secure manner for preserving privacy in various statistical methods [2].

To the best of our knowledge, there has not been any work that extends privacy preservation or secure multi-party computation to gradient descent methods. Our contributions in this paper are as follows:

1. We propose a generic formulation of gradient descent methods for secure computation by defining the target function f as a composition $g \circ h(a_1, a_2, \ldots, a_m)$, where g is any differentiable function and $h = \sum_{j=1}^{m} h_j(a_j, \omega_j)$ is linearly separable.
2. With this formulation, we propose a secure two-party protocol for performing gradient descent. We show that the protocol is correct and privacy preserving. We then extend the protocol to perform secure multi-party gradient descent.

We demonstrate how the generic secure gradient descent formulation we propose can be used by specific iteration-based algorithms such as linear regression and neural networks. We believe this work is significant as the gradient-descent paradigm is widely used.

The organization of this paper is as follows: In Section 2, we discuss related work. As there is no work on secure gradient descent, the related work is on secure computations in general and privacy-preserving data mining techniques. In Section 3, we briefly introduce the general method of gradient descent and identify the convergence of weight vectors as the core problem. We propose a formulation of gradient descent that enables us to perform the weight vector convergence securely. Section 4 describes a secure two-party protocol to perform stochastic gradient descent. The correctness, security, computational complexity and communication cost are analyzed in Section 5. In Section 6, we briefly describe how the protocol can be extended to multiple parties. The last section concludes the paper with a summary.

2. RELATED WORK

In this section, we review work that is related to secure computation, privacy preservation and data mining. Much of the work focused on conventional data mining techniques (Apriori algorithm, k-means, Naïve Bayes, etc.) that work on horizontally, vertically partitioned and/or arbitrarily partitioned data involving two or more parties.

Du and Zhan [3] addressed the secure decision tree construction problem for vertically partitioned data involving only two parties. They used one semi-trusted party to securely compute scalar products—an operation that is central to decision tree induction. Vaidya and Clifton [12] extended the secure decision tree induction problem to vertically partitioned data involving multiple parties based on the secure set intersection cardinality.

Vaidya and Clifton [9] proposed an algorithm that securely mines association rules in vertically partitioned data held by two parties. The algorithm makes use of the secure scalar product operation. An algorithm for association rule mining in horizontally partitioned data has been proposed by Kantarcioglu and Clifton [6] that incorporates cryptographic techniques to minimize the information shared with little overhead to the mining task.

Vaidya and Clifton proposed a secure k-means clustering algorithm [10] for vertically partitioned data held by multiple parties. The algorithm makes use of secure sum computation to compute distance between two points and compare the difference among point-centroid distances. The authors also proposed an algorithm for the Naïve Bayes classifier [11] that makes use of the scalar product operation to determine the probability estimate of each class label.

Yu, Jiang, and Vaidya [14] addressed privacy-preserving SVM for horizontally partitioned data involving multiple parties. The algorithm uses a secure set intersection cardinality protocol. Vertically partitioned data have been addressed by Yu, Vaidya and Jiang [15] using generic circuit evaluation technique [13] developed for secure multiparty computation.

Du, Han and Chen [2] presented two protocols for performing two-party privacy-preserving linear regression and classification on vertically partitioned data. They defined basic matrix operations that are used in the computation in linear regression and classification and showed how they can be performed securely. Thus, these secure matrix operations form basic secure building blocks for linear regression and classification. Their protocols also require a semi-trusted third party. In this paper, we adopt a secure iteration-based approach that allows the solution to be evolved securely over a series of iterations. Hence, our method can be applied to linear regression where the target function is securely evolved, rather than computed based on matrix operations. In addition, our method does not require any third party.

Du and Atallah [1] presented two protocols for secure scalar production. The general idea of the first protocol is to hide data by partitioning one vector into many partitions and mixing them with random vectors. The scalar product operation is then performed on these mixed vectors and privacy is preserved. The second protocol is based on the idea that the same mutations of two vectors do not alter the scalar product value. The authors showed how the protocol can be applied to perform secure statistical computations such as mean and correlation coefficient. Du and Atallah's protocol is among the various protocols proposed to date that perform secure scalar product. Although our proposed protocol is for secure gradient descent, we make use of one of these secure scalar product protocols as a basic operation.

To the best of our knowledge, there has not been any work on privacy-preserving gradient descent methods. We note that most of the current approaches to privacy-preservation isolated some common and basic operations in the data mining/knowledge discovery algorithm so that if these basic operations can be performed securely, then the algorithm on the whole is also secure. Two of the basic operations are the scalar product and sum computation. There exist known approaches to perform these two operations securely, without each participating party compromising its data privacy [1, 4, 5, 13]. Our protocol makes use of the secure scalar product operation.

3. GRADIENT DESCENT METHODS

In this section, we introduce general gradient descent methods and identify linear regression and neural networks as two instances of stochastic gradient descent which we show how privacy preservation can be introduced. Gradient descent [7] is a general paradigm that underlies many algorithm in machine learning and knowledge discovery. In neural networks, updating the weight value of the output and hidden nodes is a form of gradient descent. An important advantage of the gradient descent method is that it may process the input data set one row at a time, so that memory requirement is low as only the row and weight vectors need be stored during the computation.

3.1 Standard versus Stochastic Gradient Descent

Given a set of data samples $D = \{(\mathbf{x}_i, y_i) \mid i = 1, 2, \ldots, N\}$ where each $\mathbf{x}_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,m}\}$ represents a set of m attributes of the i-th sample and y_i corresponds to the target attribute, we wish to determine a target function $f(\mathbf{x}_i)$ that yields the lowest error when predicting an unknown sample's y value. This is equivalent to finding a set of weight values $\{\omega_1, \omega_2, \ldots, \omega_m\}$ such that the overall error of prediction is minimized.

Gradient descent approaches search the space of error functions for the optimal function that minimizes the prediction error with respect to the given set of data samples. If the set of data samples is linearly separable, gradient descent approaches are able to determine an optimal set of weight values for the weight vector. In general, a gradient descent approach performs a series of iterations to converge to the optimal weight vector. It uses an initial weight vector to determine the prediction error. If the error is not satisfactorily low, the weight vector is modified to reduce the error. At each iteration, the weight vector guides the error function search in the direction of steepest descent. This process continues until the global minimum error point is reached.

In standard gradient descent, all data samples are processed at each iteration to determine the steepest descent. If the data samples are linearly separable, the global minimum point is reachable. However, if the data samples are not linearly separable, then standard method will never converge as the optimum point does not exist. The standard gradient descent method has another weakness. In terms of computational complexity, it requires all data samples to be processed at each iteration. This has practical limitations when the data set is very large.

In practice, the stochastic version of gradient descent is used. At each iteration, the stochastic version processes only one data sample to determine the steepest descent and update the weight vector. The significant advantage of the stochastic approach is that the method will converge (and at a much faster rate) even when the data samples are not linearly separable. However, only a local minimum is reached.

In this paper, we use the stochastic version of gradient descent. The convergence speed does not depend on the fraction of partitions held by each party as stochastic gradient descent processes data in a row-wise manner. In our case, data is vertically partitioned. This has no effect on the convergence speed.

3.2 Generalizing Stochastic Gradient Descent Methods

We note that the error function in gradient descent methods provides the means to perform descent and the weight vector update function determines the way to perform the steepest descent. In general, as long as the error function is any form of partially differentiable function, a weight update function can be derived to effectively perform gradient descent. This includes machine learning algorithms that involve multiple error functions (and correspondingly multiple update functions), such as multi-layer neural networks.

In this section, we provide a formulation of gradient descent methods in general by generalizing the influence of weight values for each attribute as a function of the attribute's value and its corresponding weight parameter. This allows us to introduce a secure protocol in Section 4 to evolve a set of values for the weight parameter.

For any generic gradient descent method and a given error function $e : \mathcal{R}^2 \longrightarrow \mathcal{R}$, the total error E between the target y_i and predicted output $f(\mathbf{x}_i) = f(x_{i,1}, x_{i,2}, \ldots, x_m)$ with respect to all N samples is defined as:

$$E = \sum_{i=1}^{N} e(f(\mathbf{x}_i), y_i)$$

Define the prediction function f as a composition of two functions g and h as follows:

$$f(\mathbf{x}_i) \quad = \quad g \circ h(\mathbf{x}_i) \tag{1}$$

Function h should be a linearly separable function such that:

$$h(a_1, a_2, \ldots, a_m) = \sum_{j=1}^{m} h_j(a_j, \omega_j) \tag{2}$$

where $h_j(a, \omega)$, $1 \leqslant j \leqslant m$, and ω is a parameter that bares on the attribute a. In many gradient descent methods, function $h_i(a, \omega) = \omega \cdot a$ is a simple multiplication. Here, we express this as a function of a and ω for generality. Note that the values of the ω_j parameters constitute the actual form of prediction function f. Their optimum set of values will yield an accurate prediction function f.

In determining the set of optimum values for ω_j's, the stochastic gradient descent approach starts with an initial set of random values, and gradually refine (update) these values through a series of iterations. At each iteration, the ω_j values are updated with respect to a particular sample i from the training data set in the direction of steepest descent as follows:

$$\omega_j = \omega_j + \eta \Delta\omega_j \tag{3}$$

where η is constant parameter corresponding to the learning rate and $\Delta\omega_j = \partial E / \partial\omega_j$, which is determined as follows:

$$\begin{aligned}
\frac{\partial E}{\partial \omega_j} &= \frac{\partial e(f(\mathbf{x}_i), y_i)}{\partial \omega_j} \\
&= \frac{\partial e(f(\mathbf{x}_i), y_i)}{\partial f(\mathbf{x}_i)} \frac{\partial f(\mathbf{x}_i)}{\partial \omega_j} \\
&= \frac{\partial e(f(\mathbf{x}_i), y_i)}{\partial f(\mathbf{x}_i)} \frac{\partial g(h(\mathbf{x}_i))}{\partial \omega_j} \\
&= \frac{\partial e(f(\mathbf{x}_i), y_i)}{\partial f(\mathbf{x}_i)} \frac{\partial g(h(\mathbf{x}_i))}{\partial h(\mathbf{x}_i)} \frac{\partial h(\mathbf{x}_i)}{\partial \omega_j} \\
&= \frac{\partial e(f(\mathbf{x}_i), y_i)}{\partial f(\mathbf{x}_i)} \frac{\partial g(h(\mathbf{x}_i))}{\partial h(\mathbf{x}_i)} \frac{\partial}{\partial \omega_j} \left(\sum_{k=1}^{m} h_k(x_{i,k}, \omega_k) \right) \\
&= \frac{\partial e(f(\mathbf{x}_i), y_i)}{\partial f(\mathbf{x}_i)} \frac{\partial g(h(\mathbf{x}_i))}{\partial h(\mathbf{x}_i)} \frac{\partial h_j(x_{i,j}, \omega_j)}{\partial \omega_j}
\end{aligned}$$

Therefore, the above show how the steepest descent can be computed for any general error function that is partially differentiable and any prediction function f that satisfies the requirements expressed in Equations 1 and 2.

3.3 Specific Applications

In this section, we demonstrate the generality of the formulation in the previous section by showing the form of prediction function f in neural networks and linear regression—two specific gradient descent methods.

Linear Regression. For a set of data samples $D = \{(\mathbf{x}_i, y_i) \mid i = 1, 2, \ldots, N\}$, linear regression is a method to learn a target function f that best maps each attribute set $\mathbf{x}$ into its target output y with minimum error. A commonly used error function is the squared error: $E = \frac{1}{2} \sum_{1 \leqslant i \leqslant N} (y_i - f(\mathbf{x}_i))^2$. For an arbitrary sample $\mathbf{x}_i$, the prediction function f is defined as:

$$\begin{aligned} f(\mathbf{x}_i) &= g \circ h(\mathbf{x}_i) \\ &= g\left(\sum_{j=1}^{m} h_j(x_{i,j}, \omega_j)\right) \\ &= g\left(\sum_{j=1}^{m} \omega_j x_{i,j}\right) \\ &= \sum_{j=1}^{m} \omega_j x_{i,j} \end{aligned}$$

Steepest descent is achieved by taking the partial derivative of E with respect to each weight element ω_j, $1 \leqslant j \leqslant m$:

$$\frac{\partial E}{\partial \omega_j} = \frac{\partial}{\partial \omega_j}\left(\frac{1}{2}\sum_{k=1}^{N}(y_k - f(\mathbf{x}_k))^2\right)$$

It has been shown that $\partial E/\partial \omega_j = -\mathbf{x}_{i,j}(y_i - f(\mathbf{x}_i))$ in stochastic gradient descent methods [7]. The weight vector is updated until the error function E reaches a reasonably low value or no further improvements can be made.

Neural Networks. For neural networks, the two component functions for the prediction function f are: $h_j(x_j, \omega_j) = \omega_j x_j$ and $g(z) = 1/(1 + e^{-\alpha z})$. We show below how the update function can be computed with respect to the formulation of the general case described earlier:

$$\begin{aligned} \frac{\partial E}{\partial \omega_j} &= \frac{\partial}{\partial \omega_j}\left(\frac{1}{2}\sum_{k=1}^{N}(y_k - f(\mathbf{x}_k))^2\right) \\ &= \frac{\partial e(f(\mathbf{x}_i), y_i)}{\partial f(\mathbf{x}_i)} \frac{\partial g(h(\mathbf{x}_i))}{\partial h(\mathbf{x}_i)} \frac{\partial h_j(x_{i,j}, \omega_j)}{\partial \omega_j} \\ &= -(y_i - f(\mathbf{x}_i))\left(\frac{ae^{-ah(\mathbf{x}_i)}}{(1 + e^{-ah(\mathbf{x}_i)})^2}\right) x_{i,j} \\ &= -(y_i - f(\mathbf{x}_i))\left(\frac{1}{(1 + e^{-ah(\mathbf{x}_i)})} \frac{ae^{-ah(\mathbf{x}_i)}}{(1 + e^{-ah(\mathbf{x}_i)})}\right) x_{i,j} \\ &= -a(y_i - f(\mathbf{x}_i)) f(\mathbf{x}_i)(1 - f(\mathbf{x}_i)) x_{i,j} \end{aligned}$$

4. TWO-PARTY PROTOCOL

In this section, we propose a protocol to perform two-party privacy preserving gradient descent. This protocol will be extended to multi-party gradient descent in the next section.

4.1 Problem Statement

Given a data set $\mathbf{M}$ of size $N \times m$ and target vector $\mathbf{Y}$ as shown below:

$$\mathbf{M} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,m} \end{bmatrix}$$

$$\mathbf{Y} = [y_1, y_2, \ldots, y_N]^T$$

such that each row $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,m})$ $(1 \leqslant i \leqslant N)$ of $\mathbf{M}$ is associated with element y_i of $\mathbf{Y}$. We say that y_i is $\mathbf{x}_i$'s observed target value.

Suppose Alice and Bob each holds partition $\mathbf{M}_1$ and $\mathbf{M}_2$ respectively of $\mathbf{M}$ as follows $(m = m_1 + m_2)$, in addition to also holding the target vector $\mathbf{Y}$:

$$\mathbf{M}_1 = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m_1} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m_1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,m_1} \end{bmatrix}$$

$$\mathbf{M}_2 = \begin{bmatrix} x_{1,m_1+1} & x_{1,m_1+2} & \cdots & x_{1,m_1+m_2} \\ x_{2,m_1+1} & x_{2,m_1+2} & \cdots & x_{2,m_1+m_2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,m_1+1} & x_{N,m_1+2} & \cdots & x_{N,m_1+m_2} \end{bmatrix}$$

With ($\mathbf{M}_1$,$\mathbf{Y}$) and ($\mathbf{M}_2$,$\mathbf{Y}$) forming Alice and Bob's respective training data sets, the task now is for Alice and Bob to jointly discover a target function $f(\mathbf{x}, y)$ that most accurately predicts an unknown sample $\mathbf{x}$'s target value y using a gradient descent method, while preserving the privacy of their respective data partitions $\mathbf{M}_1$ and $\mathbf{M}_2$.

Given the general form of target function f (Eq. 1) as described in Section 3.2, where function $h(a_1, a_2, \ldots, a_m)$ is any linearly separable function and $g(z)$ is any differentiable function, discovering function f becomes a problem of finding the values for parameters ω_j in $h_j(a_j, \omega_j)$, $1 \leqslant j \leqslant m$.

In order to achieve the task, Alice and Bob need to follow a protocol that will allow them to jointly compute $f(\mathbf{x}, y)$ without comprising the privacy of their respective $\mathbf{M}_1$ and $\mathbf{M}_2$. As we assume that Alice and Bob are semi-honest parties [5], they will follow the protocol and perform the correct computations; however, they may retain records of the intermediate computation results which they may use later to derive the other party's data.

4.2 Protocol 1

We propose the following protocol to perform secure two-party gradient descent. We define the component function of h as follows: $h_j(a, \omega) = \omega a$ (Eq. 2). As described in Section 3.3, this definition suffices for conventional gradient descent methods such as linear regression and neural networks. The definitions of component functions of target function f, functions g and h_j $(1 \leqslant j \leqslant m)$ are known to Alice and Bob.

Input. Alice holds a $N \times m_1$ matrix $\mathbf{M}_1 = \{a_{i,j}\}$. Bob holds a $N \times m_2$ matrix $\mathbf{M}_2 = \{b_{i,j}\}$. Both $\mathbf{M}_1$ and $\mathbf{M}_2$ are the same as that defined in Section 4.1. The target vector $\mathbf{Y}$ is known to both Alice and Bob.

Output. Weight vector $\mathbf{W} = [\omega_1, \omega_2, \ldots, \omega_{m_1}, \omega_{m_1+1}, \omega_{m_1+2}, \ldots, \omega_{m_1+m_2}]$. Define $\mathbf{W}_1 = [\omega_1, \omega_2, \ldots, \omega_{m_1}]$ and $\mathbf{W}_2 = [\omega_{m_1+1}, \omega_{m_1+2}, \ldots, \omega_{m_1+m_2}]$ as two partitions of $\mathbf{W}$.

1. Alice randomly generates a $(1 \times m_1)$ vector $\mathbf{W}_1$ and a $(1 \times m_2)$ vector $\mathbf{R}_1$. Bob randomly generates a $(1 \times m_2)$ vector $\mathbf{W}_2$ and a $(1 \times m_1)$ vector $\mathbf{R}_2$. Each element of vector $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{R}_1$, and $\mathbf{R}_2$ follows a uniform distribution over some user-defined field.

2. For each row i $(1 \leqslant i \leqslant N)$ of matrix $\mathbf{M}$ (equivalently, the same for matrices $\mathbf{M}_1$ and $\mathbf{M}_2$), the following is performed:

   (a) As Alice holds $\mathbf{M}_1$ and $\mathbf{W}_1$, she computes a value α_i as follows: $\alpha_i = \sum_{j=1}^{m_1} \omega_j a_{i,j}$. Similarly, as Bob holds $\mathbf{M}_2$ and $\mathbf{W}_2$, he computes $\beta_i = \sum_{j=1}^{m_2} \omega_j b_{i,j}$. Alice discloses α_i to Bob. Bob also discloses β_i to Alice.

	Alice	Bob
Step 1:	Generate $\mathbf{W}_1$ and $\mathbf{R}_1$	Generate $\mathbf{W}_2$ and $\mathbf{R}_2$
Step 2(a):	Compute α_i and send to Bob	Compute β_i and send to Alice
Step 2(b):	Jointly compute φ_i with Bob	Jointly compute φ_i with Alice
Step 2(c):	Compute $g(\alpha_i + \beta_i + \varphi_i)$	Compute $g(\alpha_i + \beta_i + \varphi_i)$
Step 2(d):	Update $\mathbf{W}_1$	Update $\mathbf{W}_2$
Step 2(e):	Jointly compute overall error with Bob	Jointly compute overall error with Alice

Figure 1: Steps in the secure two-party protocol.

(b) Using Protocol 2 (Section 4.3), Alice and Bob jointly and securely compute the sum of two scalar products $\varphi_i = [a_{i,1}, a_{i,2}, \ldots, a_{i,m_1}] \cdot \mathbf{R}_2 + [b_{i,1}, b_{i,2}, \ldots, b_{i,m_2}] \cdot \mathbf{R}_1$. Using Protocol 2, Alice and Bob only know φ_i, but not any portion of it.

(c) Given that α_i, β_i, and φ_i are known to Alice and Bob, both parties may now individually compute $o_i = g(\alpha_i + \beta_i + \varphi_i)$, which is the predicted output value for sample i of the data matrix $\mathbf{M}$.

(d) Now that the predicted output value o_i is known, Alice and Bob individually compute the decrements for the ω parameters as follows: $\Delta\omega_j = \partial E/\partial\omega_j = \partial e(f(\mathbf{x}_i), y_i)/\partial\omega_j$ (Section 3.2) and update the ω parameters.

(e) To terminate the iteration, Alice and Bob jointly compute the sum of error $E = \sum_{i=1}^{N} e(o_i, y_i)$. If E stays within a reasonable range or does not change after many iterations, the protocol stops.

The protocol is summarized in Figure 1. When the protocol terminates, Alice and Bob have two options on how they can use the securely computed weight vector to perform prediction.

Without loss of generality, we denote Alice's final weight vector as: $\mathbf{W}_1 = [\omega_1^a, \omega_2^a, \ldots, \omega_{m_1}^a]$ (for data partition $\mathbf{M}_1$) and her random vector as: $\mathbf{R}_1 = [r_{m_1+1}^a, r_{m_1+2}^a, \ldots, r_{m_1+m_2}^a]$. Likewise, we denote Bob's final weight vector $\mathbf{W}_2 = [\omega_{m_1+1}^b, \omega_{m_1+2}^b, \ldots, \omega_{m_1+m_2}^b]$ (for data partition $\mathbf{M}_2$) and his random vector as: $\mathbf{R}_2 = [r_1^b, r_2^b, \ldots, r_{m_1}^b]$.

Option 1. Using $\mathbf{W}_1$, $\mathbf{R}_1$, $\mathbf{W}_2$, and $\mathbf{R}_2$, Alice and Bob jointly compute the final weight vector $\mathbf{W}$ for prediction function f with respect to original data set $\mathbf{M}$ as follows:

$$\begin{aligned}\mathbf{W} &= [(\omega_1^a + r_1^b), (\omega_2^a + r_2^b), \ldots, (\omega_{m_1}^a + r_{m_1}^b),\\ &\quad (\omega_{m_1+1}^b + r_{m_1+1}^a), (\omega_{m_1+2}^b + r_{m_1+2}^a), \ldots,\\ &\quad (\omega_{m_1+m_2}^b + r_{m_1+m_2}^a)]\\ &= [\omega_1, \omega_2, \ldots, \omega_{m_1}, \omega_{m_1+1}, \omega_{m_1+2}, \ldots, \omega_{m_1+m_2}]\end{aligned}$$

This option is useful to scenarios whereby two parties come together to securely compute a result (weight vector) which they can use individually later; they can part ways after the joint computation.

Option 2. In this option, Alice and Bob do not compute the final weight vector representing the prediction function. Instead, they use information ($\mathbf{W}_1$, $\mathbf{R}_1$, $\mathbf{W}_2$, and $\mathbf{R}_2$) that they each hold to jointly perform on-demand prediction whenever an unknown sample's target value is required. Hence, each party's private information (i.e., Alice's $K_a = \{\mathbf{W}_1, \mathbf{R}_1\}$ and Bob's $K_b = \{\mathbf{W}_2, \mathbf{R}_2\}$) functions like a secure key, and an additional level of security is created whereby Alice and Bob must come together whenever some information/task needs to be unlocked/predicted, using the key-pair (K_a, K_b). This joint-unlocking process is described as follows:

Let each new unknown sample be $\mathbf{P} = [p_1, p_2, \ldots, p_{m_1}, p_{m_1+1}, p_{m_2+2}, \ldots, p_{m_1+m_2}]$. For convenience, we define $\mathbf{P}_1 = [p_1, p_2, \ldots, p_{m_1}]$ and $\mathbf{P}_2 = [p_{m_1+1}, p_{m_2+2}, \ldots, p_{m_1+m_2}]$. Alice and Bob jointly predict P's target value as follows:

1. Alice computes $c = \mathbf{P}_1 \cdot \mathbf{W}_1 + \mathbf{P}_2 \cdot \mathbf{R}_1$
2. Bob computes $d = \mathbf{P}_2 \cdot \mathbf{W}_2 + \mathbf{P}_1 \cdot \mathbf{R}_2$
3. Alice and Bob exchange c and d. Each of them now computes $c + d$ individually and thus, hold the predicted target value of P.

Depending on the domain of interest, the true target value of P may be known at a later point in time. In this case, Alice and Bob may use the true target value to update their own weight vector $\mathbf{W}_1$ and $\mathbf{W}_2$ as in Steps 2(c) and 2(d) of Protocol 1. For instance, Alice and Bob may be predicting the atmospheric humidity 7 days from now. The true atmospheric humidity will be known 7 days later.

4.3 Protocol 2

Protocol 2 securely computes the sum of two scalar products between two parties. It is used in Step 2(b) of Protocol 1. We denote the two parties by Party A and Party B to avoid any confusion with Alice and Bob in Protocol 1.

Input. Party A has a $(1 \times n)$ vector $\mathbf{X}_1$ and a $(1 \times m)$ vector $\mathbf{R}_1$. Party B has a $(1 \times m)$ vector $\mathbf{X}_2$ and a $(1 \times n)$ vector $\mathbf{R}_2$.

Output. Sum of two scalar products $\mathbf{X}_1 \cdot \mathbf{R}_2 + \mathbf{X}_2 \cdot \mathbf{R}_1 = v_1 + v_2$ where v_1 is held by Party A and v_2 is held by Party B.

1. Party A and Party B compute $\mathbf{X}_1 \cdot \mathbf{R}_2$ using existing secure scalar product protocols [1, 4, 9]. Using these protocols, Party A and Party B hold private values v_a and v_b respectively after the computation where $v_a + v_b = \mathbf{X}_1 \cdot \mathbf{R}_2$.
2. Party A and Party B securely perform a similar computation for $\mathbf{X}_2 \cdot \mathbf{R}_1$ as in the previous step. Let the

two values held by them after the secure computation be v'_a and v'_b respectively where $v'_a + v'_b = \mathbf{X}_2 \cdot \mathbf{R}_1$.

3. Party A sends the sum $v_1 = v_a + v'_a$ to Party B while Party B sends the sum $v_2 = v_b + v'_b$ to Party A.

4. Party A and Party B compute $v_1 + v_2$ individually.

Throughout this protocol, each party does not know the information held by the other party; neither are they able to speculate the value of $\mathbf{X}_1 \cdot \mathbf{R}_2$ in Step 2 and $\mathbf{X}_2 \cdot \mathbf{R}_1$ in Step 3. There exist many protocols [1, 4, 9] for the secure computation of scalar product $\mathbf{X} \cdot \mathbf{Y}$.

5. PROTOCOL ANALYSIS

In this section, we show that Protocol 1 is correct and privacy-preserving.

5.1 Correctness

We show (i) if Alice and Bob follow Protocol 1, they will jointly derive a prediction function f in terms of its weight vectors through an iterative gradient descent process; and (ii) if Alice and Bob did not share their respective weight vector information $(\mathbf{W}_1, \mathbf{R}_1, \mathbf{W}_2, \mathbf{R}_2)$, then function f can be used to predict the target value of an unknown sample.

Part (i): We show that Steps 2(a)–(c) computes $f(\mathbf{x}_i) = g(\alpha_i + \beta_i + \varphi_i)$. This is equivalent to showing that $h(\mathbf{x}_i) = \alpha_i + \beta_i + \varphi_i$.

$$\begin{aligned}
h(\mathbf{x}_i) &= \sum_{j=1}^{m_1+m_2} h_j(x_{i,j}, \omega_j) = \sum_{j=1}^{m_1+m_2} (\omega_j x_{i,j}) \\
&= \sum_{j=1}^{m_1} (\omega_j x_{i,j}) + \sum_{j=m_1}^{m_1+m_2} (\omega_j x_{i,j}) \\
&= \sum_{j=1}^{m_1} \left((\omega_j^a + r_j^b) x_{i,j} \right) + \sum_{j=m_1}^{m_1+m_2} \left((\omega_j^b + r_j^a) x_{i,j} \right) \\
&= \sum_{j=1}^{m_1} \left(\omega_j^a x_{i,j} \right) + \sum_{j=1}^{m_1} \left(r_j^b x_{i,j} \right) + \\
&\quad \sum_{j=m_1}^{m_1+m_2} \left(\omega_j^b x_{i,j} \right) + \sum_{j=m_1}^{m_1+m_2} (r_j^a x_{i,j}) \\
&= \sum_{j=1}^{m_1} \left(\omega_j^a x_{i,j} \right) + \sum_{j=m_1}^{m_1+m_2} \left(\omega_j^b x_{i,j} \right) + \\
&\quad \left(\sum_{j=1}^{m_1} \left(r_j^b x_{i,j} \right) + \sum_{j=m_1}^{m_1+m_2} (r_j^a x_{i,j}) \right) \\
&= \alpha_i + \beta_i + \varphi_i
\end{aligned}$$

Thus, $f(\mathbf{x}_i) = g \circ h(\mathbf{x}_i) = g(\alpha_i + \beta_i + \varphi_i)$. In Step 2(d), the weight vector is updated using $\Delta\omega_j = \partial E / \partial \omega_j = \partial e(f(\mathbf{x}_i), y_i) / \partial \omega_j$. Step 2(e) determines whether the protocol terminates in the way conventional gradient descent methods do. Therefore, Protocol 1 performs gradient descent correctly.

Part (ii): We show how the prediction function determines the target value of an unknown sample $\mathbf{P}$; i.e., $f(\mathbf{P}) = g(\mathbf{P} \cdot\ \mathbf{W}) = g(c + d)$.

$$\begin{aligned}
\mathbf{P} \cdot \mathbf{W} &= \sum_{k=1}^{m_1+m_2} (p_k \omega_k) \\
&= \sum_{k=1}^{m_1} (p_k \omega_k) + \sum_{k=m_1+1}^{m_1+m_2} (p_k \omega_k) \\
&= \sum_{k=1}^{m_1} (p_k(\omega_k^a + r_k^b)) + \sum_{k=m_1+1}^{m_1+m_2} (p_k(\omega_k^b + r_k^a)) \\
&= \sum_{k=1}^{m_1} (p_k \omega_k^a) + \sum_{k=1}^{m_1} \left(p_k r_k^b \right) + \\
&\quad \sum_{k=m_1+1}^{m_1+m_2} \left(p_k \omega_k^b \right) + \sum_{k=m_1+1}^{m_1+m_2} (p_k r_k^a) \\
&= \mathbf{P}_1 \cdot \mathbf{W}_1 + \mathbf{P}_1 \cdot \mathbf{R}_2 + \mathbf{P}_2 \cdot \mathbf{W}_2 + \mathbf{P}_2 \cdot \mathbf{R}_1 \\
&= (\mathbf{P}_1 \cdot \mathbf{W}_1 + \mathbf{P}_2 \cdot \mathbf{R}_1) + (\mathbf{P}_2 \cdot \mathbf{W}_2 + \mathbf{P}_1 \cdot \mathbf{R}_2) \\
&= c + d
\end{aligned}$$

Thus, $f(\mathbf{P}) = g(\mathbf{P} \cdot \mathbf{W}) = g(c + d)$.

5.2 Security

In this section, we show that Protocol 1 and Protocol 2 are privacy preserving.

Protocol 2. We show that Protocol 2 securely computes the sum of two scalar products $\mathbf{X}_1 \cdot \mathbf{R}_2 + \mathbf{X}_2 \cdot \mathbf{R}_1 = v_1 + v_2$ without either party knowing $\mathbf{X}_1 \cdot \mathbf{R}_2$ and $\mathbf{X}_2 \cdot \mathbf{R}_1$. We note that Protocol 2 requires the secure scalar product operation, where there already exist many protocols that are correct and secure [1, 4, 9].

In Step 1, Party A and B securely perform scalar product $\mathbf{X}_1 \cdot \mathbf{R}_2 = v_a + v_b$. As neither party share their private values, they do not know the value of $\mathbf{X}_1 \cdot \mathbf{R}_2$. Step 2 is similar, Party A and B perform $\mathbf{X}_2 \cdot \mathbf{R}_1$ and obtain values v'_a and v'_b respectively. As they do not share these values, neither of them know the value of $\mathbf{X}_2 \cdot \mathbf{R}_1$.

In Step 3, each party discloses only the partial sum of the final result to the other party. Neither of them knows the other party's private values (v_a, v_b, v'_a, v'_b). So both parties only have the sum of $v_1 + v_2 = \mathbf{X}_1 \cdot \mathbf{R}_2 + \mathbf{X}_2 \cdot \mathbf{R}_1$ without knowing other useful information of the other party.

Protocol 2 can also be proved using a simulation method [5]. The basic idea is to show that the *view* (value received) of each party during the execution of the protocol can be effectively simulated given the input and output of that party. Here, we show that both parties are not able to learn anything other than the final result. We assume that the secure scalar product protocol used in Protocol 2 is secure and correct. By this assumption, we have the following:

1. Party A is able to effectively simulate the value received from Party B (which has probability density function $f_{v_b}(x)$) and Party B is also able to effectively simulate the value received from Party A (which has probability density function $f_{v_a}(x)$).

2. Party A is able to effectively simulate the value received from Party B (which has probability density function $f_{v'_b}(x)$) and Party B is also able to effectively simulate the value received from Party A (which has probability density function $f_{v'_a}(x)$).

We now show that Party A learns nothing from the computation other than $\mathbf{X}_1 \cdot \mathbf{R}_2 + \mathbf{X}_2 \cdot \mathbf{R}_1$. The proof for Party B is similar. Party A generates random value r as follows:

1. Generate a random number r_1 with probability density function $f_{v_b}(x)$.

2. Generate a random number r_2 with probability density function $f_{v'_b}(x)$.

3. $r = r_1 + r_2$.

The probability density function of random value r is:

$$\begin{aligned} Pr\{Simulator = x\} &= Pr\{r = x\} \\ &= Pr\{r_1 + r_2 = x\} \\ &= \int_{-\infty}^{+\infty} f_{v_b}(x-y) f_{v'_b}(y) dy \end{aligned}$$

The probability density function of the message received in $view = x$ is:

$$\begin{aligned} Pr\{View = x\} &= Pr\{v_b + v'_b = x\} \\ &= Pr\{v'_b = y\} \times Pr\{v_b = x - y\} \\ &= \int_{-\infty}^{+\infty} f_{v_b}(x-y) f_{v'_b}(y) dy \\ &= f_r(x) \\ &= Pr\{Simulator = x\} \end{aligned}$$

For party A, the probability function that simulates $r = x$ is the same as the probability distribution of the received message. We can now conclude that Party A learn nothing from the computation other than the computation result.

In the case when dot products are known to any party at *every* iteration, the party may discover the other party's random vector value. This is not fair to the other party when gradient descent terminates and both parties choose the second option. We discuss this minor result in Appendix A.

Protocol 1. We show (i) the secureness of each step in the protocol and then (ii) show that the overall iteration is also secure.

Part (i): In Step 2(a), Alice and Bob exchange the values of α and β. As these value are scalar products (singular values) computed from each party's private weight and data vectors, each party would not be able to discover the other party's private weight and data vector using only a single value. In Step 2(b), Alice and Bob follow Protocol 2, which we have shown earlier to be secure. In Step 2(c), Alice and Bob individually compute o_i. Hence, there is no privacy issue in this step. In Step 2(d), there is no exchange of information between Alice and Bob as they each update their own weight vector. Hence, there is no privacy issue.

In Step 2(e), Alice and Bob jointly compute the overall error of the joint data set. This requires the computation of $f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_N)$ using the latest weight vector $\mathbf{W}$. Hence, Steps 2(a)–(c) are repeated for each sample i of the original data set. We have shown above that the Steps 2(a)–(c) are secure, so overall, Step 2(e) is secure.

Part (ii): We show that the overall set of iterations of gradient descent is secure. Let $\mathbf{W}_2(t)$ and $\beta(t)$ denote the value of Bob's weight vector and β value at the t-th iteration. The initial iteration (t_0) consists of randomly initialized weight vector. The final iteration occurs at t_T. We show that if Alice were to keep records of all computation results and $\beta(t_0), \beta(t_1), \ldots, \beta(t_T)$ values received from Bob over the iterations, she would not be able to derive any information of Bob's data set $(\mathbf{M}_2)$, with an extreme exception.

In part (i) above, we showed that the β value received by Alice from Bob will not compromise Bob's data privacy. We now show that the sequence of $\beta(t_0), \beta(t_1), \ldots, \beta(t_T)$ values received from Bob will likewise not compromise Bob's data privacy. The argument for Bob receiving $\alpha(t_0), \alpha(t_1), \ldots, \alpha(t_T)$ values from Alice is identical and is omitted.

Alice knows that each β is the dot product of Bob's weight vector $\mathbf{W}_2$ and one row of its data matrix $\mathbf{M}_2$. We enumerate all such dot products corresponding to all iterations of the gradient descent convergence process as follows:

$$\begin{aligned} \beta(t_0) &= \mathbf{W}_2(t_0) \cdot [b_{1,1}, b_{1,2}, \ldots, b_{1,m_2}] \\ \beta(t_1) &= \mathbf{W}_2(t_1) \cdot [b_{2,1}, b_{2,2}, \ldots, b_{2,m_2}] \\ \ldots &= \ldots \\ \beta(t_k) &= \mathbf{W}_2(t_k) \cdot [b_{u,1}, b_{u,2}, \ldots, b_{u,m_2}] \\ \ldots &= \ldots \\ \beta(t_T) &= \mathbf{W}_2(t_T) \cdot [b_{s,1}, b_{s,2}, \ldots, b_{s,m_2}] \end{aligned}$$

where $u = t_k \mod N$ and $s = t_T \mod N$.

As the value of $\mathbf{W}_2$ changes at each iteration, it is impossible for Alice to know $\mathbf{W}_2$. Hence, Alice will also not know any information regarding each row of Bob's data matrix. Hence, the gradient descent iteration of Steps 2(a) to 2(e) is secure.

When the protocol terminates, Alice and Bob have two options: (i) Share the weight and random vectors in order to obtain the final weight vector, or (ii) Use their private weight and random vectors to perform on-demand prediction. If Alice and Bob choose Option 1, then Alice knows Bob's $\mathbf{W}_2$. At the last iteration t_T, in the extreme case when Bob's data matrix $\mathbf{M}_2$ has only one column; i.e., $m_2 = 1$, then Alice will know the s-th row of $\mathbf{M}_2$, as $\beta(t_T) = \mathbf{W}_2(t_T) \cdot [b_{s,1}, b_{s,2}, \ldots, b_{s,m_2}] = [\omega_{m_1+1}] \cdot [b_{s,1}]$; so $b_{s,1} = \beta(t_T)/\omega_{m_1+1}$. If Bob's $\mathbf{M}_2$ has more than one columns, then no data privacy is compromised. Likewise for all other iterations before the final one, no data privacy is compromised.

5.3 Complexity Analysis

In this section, we derive the computational complexity and communication cost of Protocol 1.

Computational Complexity. We determine the computational complexity of one iteration of Protocol 1 as follows: In Step 1, Alice and Bob generate the weight and random vectors of lengths m_1 and m_2. So the computational complexity is $O(m_1 + m_2)$. In Step 2(a), Alice and Bob perform the scalar product of data and weight vectors. The computational complexity for Alice and Bob is $O(m_1)$ and $O(m_2)$ respectively. In Step 2(b), Alice and Bob executes the secure scalar product protocol (Protocol 2) for two vectors of length of m_1 and two vectors of length of m_2. If we define the computational complexity of secure scalar product as $O(\tau(z))$ where z is the number of elements in the multiplicand vectors and $\tau(z)$ is an expression for computational complexity of z with respect to some secure scalar product protocol used. The computational complexities for Alice and Bob in this step are $O(\tau(m_1) + \tau(m_2))$ and $O(\tau(m_1) + \tau(m_2))$ respectively.

In Step 2(c), the computational complexity is $O(g)$ as it involves the computation of function g. In Step 2(d), the weight vectors of Alice and Bob are updated once. Hence, the computational complexity is $O(\max\{m_1, m_2\})$. In Step 2(e), the overall prediction error is computed using all the rows of data matrix $\mathbf{M}$. Hence, the complexity is $O(N \times (m_1 + m_2 + O(\tau(m_1) + \tau(m_2) + O(g)))$. In practice, it is not necessary to compute the overall prediction error at every iteration. It can be performed only with a probability.

The overall complexity for the entire iteration is: $O(N \times (\tau(m_1) + \tau(m_2) + O(g)))$.

Communication Cost. In Step 1, Alice and Bob generate the weight and random vectors of lengths m_1 and m_2. No communication is needed. Thus, there is no communication cost.

In Step 2(a), Alice and Bob exchange α and β values. The communication cost is $O(1)$.

In Step 2(b), Alice and Bob executes the secure scalar product protocol (Protocol 2) for two vectors of length of m_1 and two vectors of length of m_2. If the communication cost of secure scalar product is $O(\tau'(z))$, where z is the number of elements in the multiplicand vectors and $\tau'(z)$ is an expression for the communication cost of z with respect to some secure scalar product protocol used. The computational complexities for Alice and Bob in this step are $O(\tau'(m_1) + \tau'(m_2))$ and $O(\tau'(m_1) + \tau'(m_2))$ respectively.

In Step 2(c), there is no communication cost as Alice and Bob compute function g locally. In Step 2(d), there is no communication cost as Alice and Bob update their weight vector locally. In Step 2(e), the overall prediction error is computed using all the rows of data matrix $\mathbf{M}$. Hence, the complexity is $O(N \times (O(\tau'(m_1) + \tau'(m_2)))$.

The overall complexity for the entire iteration is: $O(N \times (\tau'(m_1) + \tau'(m_2)))$.

6. EXTENSION TO MULTIPLE PARTIES

The secure two-party gradient descent protocol presented in Section 4 can be extended to multi-parties as follows: Let there be P_0, P_1, ..., P_{K-1} parties cooperating to perform stochastic gradient descent with respect to a $(N \times m)$ data matrix $\mathbf{M}$. Each party P_i holds a $(N \times m_i)$ partition $\mathbf{M}_i$ of the matrix so that $\sum_{j=0}^{K-1} m_j = m$. The secure multi-party protocol executes as follows:

1. Each party P_i generates a $(1 \times m_i)$ weight vector $\mathbf{W}_i$.
2. Each party P_i also generates a $(1 \times m_{i+1 \mod K})$ random vector $\mathbf{R}_i$. That is, the number of columns in the random vector is the number of columns of its next neighboring party's weight vector. For the last party, the number of columns of the random vector is the number of columns of the first party's weight vector.
3. The remaining steps: Computing secure scalar products with its neighbors, computing prediction function output, updating weight vector, and computing the overall error follow the same steps of the two-party protocol described in Section 4.

As the extension is straightforward, it can be easily shown that the multi-party version of Protocol 1 is also correct and privacy-preserving.

7. CONCLUSIONS

Gradient descent is a widely used method for solving many optimization and learning problems. So far, there has not been any work that extends privacy preservation to gradient descent methods. In this paper, we presented a generic formulation of secure gradient descent methods by defining the target function f as $g \circ h(a_1, a_2, \ldots, a_m)$, where g is any differentiable function and $h = \sum_{j=1}^{m} h_j(a_j, \omega_j)$ is linearly separable. The latter property of function h allows the ω_j's to be updated locally.

With this formulation, we propose a secure two-party protocol for performing gradient descent. We showed that the protocol is correct and privacy preserving. We also analyzed its computational and communication cost. We extended the protocol to perform secure multi-party gradient descent. We believe the work presented in this paper is significant as gradient descent is a widely used.

For future work, we intend to apply the secure gradient descent protocol to two- or multi-party Bayesian network induction. Arising from our investigations in this paper, we are also interested in looking into the following problem: We have formulated $h_j(a_j, \omega_j)$ as a product of two values $\omega_j \cdot a_j$. This definition suffices for many gradient descent-based algorithms such as neural networks, linear regression, Bayesian networks, and so on. In the case when function h_j is not a multiplication and the secure protocol requires random values to be used to hide the weight vectors, how to define a general protocol that will work for this scenario?

8. REFERENCES

[1] W. Du and M. Atallah. Privacy-Preserving Cooperative Statistical Analysis. *Proceedings of the 17th Annual Computer Security Applications Conference*, pp. 103–110, Louisiana, USA, 2001.

[2] W. Du, Y. S. Han, and S. Chen. Privacy-preserving Multivariate Statistical Analysis: Linear Regression and Classification. *Proceedings of the SIAM International Conference on Data Mining*, Florida, USA, 2004.

[3] W. Du and Z. Zhan. Building Decision Tree Classifier on Private Data. *Workshop on Privacy, Security, and Data Mining*, held in conjunction with the IEEE International Conference on Data Mining, pp. 1–8, Maebashi City, Japan, 2002.

[4] B. Goethals, S. Laur, H. Lipmaa, and T. Mielik. On Private Scalar Product Computation for Privacy-Preserving Data Mining. *Proceedings of the 7th Annual International Conference on Information Security and Cryptology*, pp. 104–120, Seoul, Korea, 2004.

[5] O. Goldreich. *The Foundations of Cryptography.* Cambridge University Press, 2004.

[6] M. Kantarcioglu and C. Clifton. Privacy-Preserving Distributed Mining of Association Rules on Horizontally Partitioned Data. *ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pp. 24–31, 2002.

[7] T. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[8] S. Russell, J. Binder, D. Koller, and K. Kanazawa. Local Learning in Probabilistic Networks with Hidden Variables. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

[9] J. Vaidya and C. Clifton. Privacy Preserving Association Rule Mining in Vertically Partitioned

Data. *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining*, pp. 639–644, Edmonton, Canada, 2002.

[10] J. Vaidya and C. Clifton. Privacy Preserving K-means Clustering over Vertically Partitioned Data. *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining*, pp. 206–215, Washington D.C., USA, 2003.

[11] J. Vaidya and C. Clifton. Privacy Preserving Naïve Bayes Classifier for Vertically Partitioned Data. *Proceedings of the SIAM International Conference on Data Mining*, Florida, USA, 2004.

[12] J. Vaidya and C. Clifton. Privacy-Preserving Decision Trees over Vertically Partitioned Data. *Proceedings of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Storrs, USA, 2005.

[13] A. Yao. How to Generate and Exchange Secrtes. *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pp. 162–167, 1986.

[14] H. Yu, X. Jiang, and J. Vaidya. Privacy-Preserving SVM using Nonlinear Kernels on Horizontally Partitioned Data. *Proceedings of the 21st Annual ACM Symposium on Applied Computing*, Dijon, France, 2006.

[15] H. Yu, X. Jiang, and J. Vaidya. Privacy Preserving SVM Classification on Vertically Partitioned Data. *Proceedings of the 10th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Singapore, 2006.

APPENDIX

A. FAIRNESS

In Section 5 when we analyzed Protocol 2, we mentioned that in the case when the dot products are known to any of the party at *every* iteration, the party might discover the other party's random vector value. We show how this is possible in this section.

Alice and Bob uses Protocol 2 in Step 2(b) to jointly compute the sum of two dot products $\varphi(t_i) = \varphi^a(t_i) + \varphi^b(t_i)$ where $\varphi^b = \mathbf{X}_2 \cdot \mathbf{R}_1$ and $\varphi^a = \mathbf{X}_1 \cdot \mathbf{R}_2$ at each iteration t_i of the gradient descent convergence process.

If Alice knows the φ^a, φ^b values for *all* iterations, she could form the following set of expressions corresponding to all the iterations to infer information about Bob's $\mathbf{R}_2$:

$$\begin{aligned}
\varphi^a(t_0) &= \mathbf{R}_2 \cdot [a_{1,1}, a_{1,2}, \ldots, a_{1,m_1}] \\
\varphi^a(t_1) &= \mathbf{R}_2 \cdot [a_{2,1}, a_{2,2}, \ldots, a_{2,m_1}] \\
\ldots &= \ldots \\
\varphi^a(t_k) &= \mathbf{R}_2 \cdot [a_{u,1}, a_{u,2}, \ldots, a_{u,m_1}] \\
\ldots &= \ldots \\
\varphi^a(t_T) &= \mathbf{R}_2 \cdot [a_{s,1}, a_{s,2}, \ldots, a_{s,m_1}]
\end{aligned}$$

where $u = t_k \mod N$ and $s = t_T \mod N$. There are N equivalence classes for the above expressions as the gradient descent convergence process repeatedly cycles through every row of the data matrix. Note that for each dot product in the above expressions, the left vector $\mathbf{R}_2$ is identical and does not change. The right vector repeatedly cycles through each row of Alice's matrix $\mathbf{M}_1$. Since φ^a and the right vector is known to Alice, an expression can be formed to deduce the elements of $\mathbf{R}_2$. As there are m_1 elements in $\mathbf{R}_2$, as long as there are at least m_1 iterations, the m_1 expressions will be able to completely reveal $\mathbf{R}_2$. That is, if $N \geqslant m_1$, then R_2 will be found out by Alice. The same argument holds when Bob knows the φ^a, φ^b values for *all* iterations; he will likewise be able to infer Alice's $\mathbf{R}_1$.

Despite of the above consequence, Alice (Bob) will not know Bob (Alice)'s data. Consider the following set of expressions arising from the iterations:

$$\begin{aligned}
\varphi^b(t_0) &= \mathbf{R}_1 \cdot [b_{1,1}, b_{1,2}, \ldots, b_{1,m_2}] \\
\varphi^b(t_1) &= \mathbf{R}_1 \cdot [b_{2,1}, b_{2,2}, \ldots, b_{2,m_2}] \\
\ldots &= \ldots \\
\varphi^b(t_k) &= \mathbf{R}_1 \cdot [b_{u,1}, b_{u,2}, \ldots, b_{u,m_2}] \\
\ldots &= \ldots \\
\varphi^b(t_T) &= \mathbf{R}_1 \cdot [b_{s,1}, b_{s,2}, \ldots, b_{s,m_2}]
\end{aligned}$$

where $u = t_k \mod N$ and $s = t_T \mod N$. Note that the right vector in each dot product above is different. For those iterations t_i and t_j where $i \mod N = j \mod N$, the right vector of the dot product comes from the same row of Bob's $\mathbf{M}_2$ matrix. Hence, $\varphi^b(t_i) = \varphi^b(t_j)$; so Alice would not be able to find out the elements of this row. So although Alice knows φ^b and $\mathbf{R}_1$, she would not be able to infer Bob's data matrix. Hence, Bob's data privacy is not compromised.

Instead of knowing each φ^a and φ^b values, consider the case when Alice (Bob) knows the φ^i value for all iterations. Then each iteration yields an expression:

$$\varphi(t_k) = \mathbf{R}_2 \cdot [a_{u,1}, a_{u,2}, \ldots, a_{u,m_1}] + \mathbf{R}_1 \cdot [b_{u,1}, b_{u,2}, \ldots, b_{u,m_2}]$$

It is quite obvious that Alice would neither know $\mathbf{R}_2$ nor Bob's data.