IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 12, 2019, accepted March 13, 2020, date of publication March 18, 2020, date of current version April 1, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2981818

HE-Friendly Algorithm for Privacy-Preserving SVM Training

SAEROM PARK[1], JUNYOUNG BYUN[2], JOOHEE LEE[3], JUNG HEE CHEON[3], AND JAEWOOK LEE[2]

[1]Department of Convergence Security Engineering, Sungshin Women's University, Seoul 02844, South Korea
[2]Department of Industrial Engineering, Seoul National University, Seoul 08826, South Korea
[3]Department of Mathematical Sciences, Seoul National University Seoul 08826, South Korea

Corresponding author: Jaewook Lee (jaewook@snu.ac.kr)

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1D1A1A02085851), and in part by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIT) under Grant NRF-2019R1A2C2002358, and Grant NRF-2017R1A5A1015626.

ABSTRACT Support vector machine (SVM) is one of the most popular machine learning algorithms. It predicts a pre-defined output variable in real-world applications. Machine learning on encrypted data is becoming more and more important to protect both model information and data against various adversaries. While some studies have been proposed on inference or prediction phases, few have been reported on the training phase. Homomorphic encryption (HE) for the arithmetic of approximate numbers scheme enables efficient arithmetic evaluations of encrypted data of real numbers, which encourages to develop privacy-preserving machine learning training algorithm. In this study, we propose an HE-friendly algorithm for the SVM training phase which avoids inefficient operations and numerical instability on an encrypted domain. The inference phase is also implemented on the encrypted domain with fully-homomorphic encryption which enables real-time prediction. Our experiment showed that our HE-friendly algorithm outperformed the state-of-the-art logistic regression classifier with fully homomorphic encryption on toy and real-world datasets. To the best of our knowledge, this study is the first practical algorithm for training an SVM model with fully homomorphic encryption. Therefore, our result supports the development of practical applications of the privacy-preserving SVM model.

INDEX TERMS Cryptography, data privacy, fully homomorphic encryption, support vector machine, privacy-preserving training.

I. INTRODUCTION

Machine learning has gained considerable attention recently, because of its usefulness in many big data analytic tasks involving artificial intelligence such as marketing, healthcare, and financial services. Data privacy has become more and more crucial in machine learning since the GDPR (General Data Protection Regulation) will make the unauthorized use of data owners' personal information not only immoral but also illegal. As a result, there are increasing demands for services using machine learning algorithms. This is generating a definite need for new and increasingly effective privacy-preserving technologies.

Among machine learning algorithms, Support Vector Machine (SVM) is one of the most popular methods of classifying data. Despite the explosive popularity of deep learning, the SVM model is still crucial because deep learning algorithms require a lot of data, and the kernel methods work well on medium-sized data. The SVM works by learning complicated nonlinear data patterns using a kernel trick [1]. To find optimal parameters, the training phase of the SVM model must solve the convex optimization problem, while for deep learning models it must solve the non-convex optimization problem. In the test phase of the SVM, the decision function, including some of the training data, called support vectors (SVs), is calculated to obtain a label for the new data. Accordingly, to apply SVM models in real-world scenarios, the model parameters and training data need to be protected to preserve secrecy.

The associate editor coordinating the review of this manuscript and approving it for publication was Berdakh Abibullaev.

Fully Homomorphic Encryption (FHE) enables homomorphic computations of ciphertexts, which mainly supports addition and multiplication of encrypted data. Using FHE, multiple institutes can share their data in an encrypted form and evaluate machine learning algorithms on them. Several works have been conducted on secure computations in machine learning algorithms with fully homomorphic encryptions: prediction phases [2]–[4] and training phases [5], [6] of the Logistic Regression Model (LRM), decision trees inference phases [7], and deep neural network inference phases [8], [9]. Notably, few works have been reported that propose private training for support vector machine (SVM) algorithms using FHE because the SVM training model has many constraints and non-polynomial functions.

Recently, an FHE scheme called HEAAN (Homomorphic Encryption for Arithmetic of Approximate Numbers) [10] has been developed for approximate arithmetic among real number data. HEAAN is much more suitable than other FHE schemes for many machine learning tasks since it serves homomorphic computations on real number data, although it requires careful manipulation to employ HEAAN in such tasks efficiently. In this paper, we present a new privacy-preserving SVM training algorithm using HEAAN. In earlier works [11], [12], private training and prediction for SVM were suggested using additive HE and several secure multi-party computation protocols. However, their protocols still reveal some information, such as the number of rounds of SVM, and they did not present any evidence that their solutions ccould be used in practice. Several papers have suggested privately computing the SVM kernel [13], [14] using additive HE and secure multi-party computations, but they did not implement all of the training phases in an encrypted form, and implied the use of the computation in the inference phase. Subsequent works [15]–[21] have only focused on the private SVM prediction phases. Recently, Barnett *et al.* [22] suggested a way to compute a polynomial kernel and to carry out a prediction for SVM in a privacy-preserving manner with FHE.

In this study, we propose a secure training algorithm for the SVM model which protects both model information and training data without having access to them in a two-party computation scenario. We can summarize our contributions as follows.

- First, we introduced the least square SVM algorithm to replace the constrained optimization problem that contains the inefficient operations in the encrypted domain.
- We propose a privacy-preserving training algorithm for the SVM model with FHE for the first time, which is based on a gradient descent for the least squares problem.
- We devised two different packing strategies with parallelizable calculations (comlumn-wise packing and sub-matrix packing). While column-wise packing consumes one less ciphertext level per iteration, submatrix packing is scalable for a large matrix and extendible to the multi-class classification problem.
- We implemented a secure SVM training algorithm and compared it with the state-of-the-art secure training algorithm. The training algorithms were evaluated using various real-world datasets.

II. DESIGN COMPONENTS

A. HOMOMORPHIC ENCRYPTION

FHE is a cryptosystem which enables homomorphic operations such as additions and multiplications on encrypted data. After Gentry's blueprint [23], there have been many following works [24]–[40], and many applications of FHE have also emerged for various tasks in medical, genomic, or financial fields of studies [2], [3], [41]–[43]. Remarkable improvements in the area of FHE have occured, and HE libraries such as HElib [39], [44] or YASHE [36] have shown some good implementation results in their applications [3], [41], [43], [45]. However, there have been some difficulties adapting the real datasets in general cases since they only support operations over a fixed integer modulus space. In fact, the encoding strategy for real data affects the real time needed for implementation, and especially for huge dataset with much entropy, any hired encoding strategy can cause a blow-up in parameters. They can also be very costly to implement. Recently, a homomorphic encryption scheme which focuses on the arithmetic of approximate numbers, namely HEAAN [10], appeared to mitigate this problem. The main idea of HEAAN is to treat a noise from the hardness assumption of a scheme as part of an error that occurred in the approximate computations. Indeed, they suggest a kind of converting technique to turn a FHE scheme with certain hardness assumptions into a FHE scheme that carries out approximate computations efficiently. The efficient computations enables the development of training algorithms for machine learning models with FHE.

We describe the algorithms in the (leveled) FHE scheme HEAAN of depth L here. First, we set the parameters such as a power of two M', integers h, P, p, q_0, and $q_L = p^L \cdot q_0$, and a real number σ, where the underlying hard problem with regard to these parameters and certain secret distribution achieves λ-bit security for targeted λ. Let $\Phi_{M'}(x)$ be an M'-th cyclotomic polynomial of degree $\phi(M')$ so that $\Phi_{M'}(x) = x^M + 1$ where $\phi(M') = M$.

Let $\mathcal{R} = \mathbb{Z}[x]/(\Phi_{M'}(X))$ and $\mathcal{R}_q := \mathcal{R}/q\mathcal{R}$ for $q \in \mathbb{Z}$. The sequence of moduli $\{q_i\}_{i=0}^{L}$ for the ciphertext space in each depth is set by $q_i = p^i \cdot q_0$ for $0 < i \leq L$. Then, the ciphertext space of the level ℓ is $\mathcal{R}_{q_\ell}^2$. One of the characteristics of HEAAN is that a plaintext can be an arbitrary complex vector, and it is encoded into the space $\mathcal{R}$. Let S be a subgroup of $\mathbb{Z}_{M'}^*$ satisfying $\mathbb{Z}_{M'}^*/S = \{\pm 1\}$, and $\mathbb{H} = \{z = (z_j) \in \mathbb{Z}_{M'}^* : z_j = \overline{z_{-j}}, \forall j \in \mathbb{Z}_{M'}^*\}$. For an efficiently computable field isomorphism $\phi : \mathbb{R}[X]/(\Phi_{M'}(X)) \to C_{N/2}$, the specification of their algorithms is as follows.

- $(pk, sk, evk) \leftarrow \mathsf{KeyGen}(1^\lambda)$:
  - Sample random $a \leftarrow \mathcal{R}_{q_L}$, sparse signed binary polynomial s in $\mathcal{R}$ of h non-zero coefficients, and $e \in \mathcal{R}_{P\cdot q_L}$ of small sizes of which sizes depend on σ^2. Calculate $b \leftarrow -as + e \mod q_L$, and set the secret key $sk \leftarrow (1, s)$ and the public key $pk \leftarrow (b, a) \in \mathcal{R}^2_{q_L}$.
  - Sample random $a' \leftarrow \mathcal{R}_{P\cdot q_L}$ and $e' \in \mathcal{R}_{P\cdot q_L}$ of small sizes of which sizes depend on σ^2. Calculate $b' \leftarrow -a's + e' + Ps' \pmod{P\cdot q_L}$, where $s' \leftarrow s^2$. Set the evaluation key $evk \leftarrow (b', a') \in \mathcal{R}^2_{P\cdot q_L}$.
- $m \in \mathcal{R} \leftarrow \mathsf{Encode}(\vec{v} \in \mathbb{R}^M, \Delta)$: For a plaintext vector of real numbers $\vec{v} \in \mathbb{R}^M$, output $m \in \mathcal{R} \leftarrow \lfloor \phi^{-1}(\Delta \cdot \vec{v}) \rceil$.
- $\vec{v} \in \mathbb{R}^M \leftarrow \mathsf{Decode}(m \in \mathcal{R}, \Delta)$: Output $\vec{v} \in \mathbb{R}^M \leftarrow \Delta^{-1} \cdot \phi(m)$.
- $\vec{c} \leftarrow \mathsf{Encrypt}(pk, m)$: Sample $\vec{r} \in \mathcal{R}^2$ for which coefficients of each component would be zero with probability 1/2, and ± 1 with probability 1/4, respectively, and $e_0, e_1 \leftarrow \mathcal{DG}_{q_L}(\sigma^2)$.
  Output $\vec{c} \leftarrow \vec{r} \cdot pk + (m + e_0,\ e_1) \in \mathcal{R}^2_{q_L}$.
- $m \leftarrow \mathsf{Decrypt}(sk, \vec{c} = (c_0, c_1))$: Output $m \leftarrow c_0 + c_1 \cdot s \mod q_\ell$.
- $\vec{c}_{\mathsf{add}} \leftarrow \mathsf{Add}(\vec{c}_1, \vec{c}_2)$: Output $\vec{c}_{\mathsf{add}} \leftarrow \vec{c}_1 + \vec{c}_2 \mod q_\ell$.
- $\vec{c}_{\mathsf{cmult}} \leftarrow \mathsf{CMult}(evk, a \in \mathcal{R}, \vec{c})$: For $\vec{c} \in \mathcal{R}^2_{q_\ell}$, output $\vec{c}_{\mathsf{cmult}} = a \cdot \vec{c} \pmod{q_\ell}$.
- $\vec{c}_{\mathsf{mult}} \leftarrow \mathsf{Mult}(evk, \vec{c}_1, \vec{c}_2)$: Set $c_1 = (b_1, a_1)$ and $c_2 = (b_2, a_2)$. Let $(d_0, d_1, d_2) \leftarrow (b_1 \cdot b_2, a_1 \cdot b_2 + a_2 \cdot b_1, a_1 \cdot a_2) \in \mathcal{R}^3_{q_\ell}$. Output $\vec{c}_{\mathsf{mult}} \leftarrow (d_0, d_1) + \lfloor \frac{1}{P} \cdot (d_2 \cdot evk \mod P \cdot q_\ell) \rceil \in \mathcal{R}^2_{q_\ell}$.
- $\vec{c}' \leftarrow \mathsf{Rescaling}_{\ell \to \ell'}(\vec{c})$: For a level ℓ ciphertext $\vec{c}$, output $\vec{c}' \leftarrow \lfloor \frac{q_{\ell'}}{q_\ell} \cdot \vec{c} \rceil \in \mathcal{R}^2_{q_{\ell'}}$ at level ℓ'.

HEAAN also provides a rotation function called **Rotate** on plaintext slots. The **Rotate** function requires a rotation key rk which is an additionally generated public information, a ciphertext, and the number of slots to rotate, and produces a ciphertext of plaintext of rotated slots.

- $\vec{c}_{\mathsf{Rotate}} \leftarrow \mathsf{Rotate}(rk, \vec{c}, R)$: Given rotation key rk, encrypt rotated plaintext vector of $\vec{c}$ by R position and output $\vec{c}_{\mathsf{Rotate}}$, where $R > 0$ represents right-shift, and $R < 0$ represents left-shift.

For more details, we recommend [10]. Despite its usefulness, using FHE to privately evaluate machine learning algorithms is not straightforward and needs careful manipulation for the following reasons.

1) The homomorphic evaluation of non-polynomial operations is still heavy. For example, divisions, comparisons, and sortings are very important in many machine learning algorithms, but they are expensive when evaluated on ciphertexts.
2) Even though all the operations can be effectively represented as a polynomial evaluation, the homomorphic evaluation would be infeasible if the depth of the circuit is too high. When the circuit depth is higher than the limit which is determined by the prefixed parameter set, bootstrapping becomes necessary which causes overwhelming costs to the whole process.
3) In our context the FHE only supports component-wise addition, multiplication, and scalar multiplication although a vector of multiple plaintexts is packed in a single ciphertext. Multiplications between the matrix and vector, or the inner products of vectors, require redundant rotations which is costly, and therefore a lot of wasted plaintext slots occurs
4) Accessing a plaintext of an arbitrary index cannot be done efficiently, since it requires heavy rotations and multiplications in order to pad 0's instead of dummy values. Hence, it is hard to employ some useful tools such as Gaussian elimination when encrypted.
5) HEAAN supports the approximate computations. Because of this property, it is required that all evaluations to be homomorphically calculated should be numerically stable to achieve the correctness property. Hence, arithmetic operations like inversion (for arbitrary numbers) are not adequate for FHE.

These properties are fundamental in homomorphic encryption, and they are crucial to implement the machine learning algorithm with FHE. Therefore, we aim to implement the SVM algorithm suggesting a HE-friendly algorithm to avoid all of them.

B. LEAST SQUARE SUPPORT VECTOR MACHINE

In this study, we aim to develop a scalable secure SVM training algorithm based on the HEAAN scheme. SVM is one of the most popular classification algorithms which predict class labels from training examples in a non-parametric way. SVM can find the maximum margin nonlinear classifier in the high-dimensional space with kernel tricks. We assume that there are some training examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \{-1, 1\}$. The optimization problem of the linear SVM model is as follows:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \ & \frac{1}{n}\sum_{i=1}^{n} \xi_i + \lambda \|\mathbf{w}\|^2 \\ \text{s.t. } & y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \forall i = 1, \ldots, n, \end{aligned} \tag{1}$$

where $b \in \mathbb{R}$, $\xi_i \in \mathbb{R}$, $i = 1, \ldots, n$, and $\mathbf{w} \in \mathbb{R}^d$. Using hinge loss $l(\hat{y}) = \max\{0, 1 - y\cdot\hat{y}\}$, the above problem (1) is equivalent to minimizing $\frac{1}{n}\sum_{i=1}^{n} l(\mathbf{w}\cdot\mathbf{x}_i + b) + \lambda\|\mathbf{w}\|^2$. However, this problem contains homomorphic encryption unfriendly functions such as $\max\{0, x\}$. Although the SVM problem is usually transformed into a dual problem which represents quadratic programming, solving the problem is still difficult to implement on the encrypted domain because of linear and box constraints. Thus, a secure SVM algorithm can be formulated based on the least square SVM algorithm to train the SVM model over encrypted data where all operations should be represented as **Add**, **Mult** and **Rotate**. In addition, if the nonlinear basis function $\phi(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^l$ is introduced, we can obtain the nonlinear SVM model. The nonlinear least

square SVM problem can be formulated as follows [46]:

$$\min_{\mathbf{w},b,e_i} \frac{1}{n}\sum_{i=1}^{n} e_i + \lambda\|\mathbf{w}\|^2$$
$$\text{s.t. } y_i\,[\mathbf{w}\cdot\phi(\mathbf{x}_i)+b] = 1-e_i, \quad \forall i=1,\ldots,n, \tag{2}$$

where $\mathbf{w}\in\mathbb{R}^l$. From this optimization problem (2), we can construct the Lagrangian function:

$$L(\mathbf{w},b,\mathbf{e},\boldsymbol{\alpha}) = \lambda\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} e_i^2 - \sum_{i=1}^{n}\alpha_i\{[\mathbf{w}\cdot\phi(\mathbf{x}_i)+b]+e_i-1\} \tag{3}$$

where dual variables are $\boldsymbol{\alpha} = [\alpha_1,\ldots,\alpha_n]\in\mathbb{R}^n$. With the optimality conditions of the Lagrangian function (3), we can the following equations:

$$\frac{\partial L}{\partial w} = \mathbf{0} \rightarrow \mathbf{w} = \sum_{i=1}^{n}\alpha_i y_i \phi(\mathbf{x}_i) \tag{4}$$

$$\frac{\partial L}{\partial b} = 0 \rightarrow \sum_{i=1}^{n}\alpha_i y_i = 0 \tag{5}$$

$$\frac{\partial L}{\partial e_i} = 0 \rightarrow e_i - \lambda\alpha_i = 0, \quad \forall i \tag{6}$$

$$\frac{\partial L}{\partial \alpha_i} = 0 \rightarrow y_i\{\mathbf{w}\cdot\phi(x_i)\} - 1 + e_i = 0, \quad \forall i \tag{7}$$

We can obtain the following linear system by removing $\mathbf{w}$ and $\mathbf{e}$ using (4) and (6):

$$\mathbf{Ab} = \begin{bmatrix} 0 & \mathbf{y}^T \\ \mathbf{y} & \Omega + \lambda\mathbf{I}_n \end{bmatrix}\begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{1}_n \end{bmatrix} = \tilde{\mathbf{1}} \tag{8}$$

where $\mathbf{I}_n\in\mathbb{R}^{n\times n}$ is an identity matrix, $\mathbf{1}_n = [1,1,\ldots,1]^T \in \mathbb{R}^n$, and $\Omega\in\mathbb{R}^{(n+1)\times(n+1)}$ s.t. $\Omega_{i,j} = k(\mathbf{x}_i,\mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$.

Finally, training the SVM classifier can be reduced to solving the linear system (8). In this study, we aim to develop an HE-friendly SVM training algorithm based on the linear system (8). After obtaining the solution $\boldsymbol{\alpha}$ and b, we can acquire the following SVM classifier.

$$y(\mathbf{x}) = \left(\sum_{m\in\mathcal{I}_{sv}} \alpha_m y_m k(\mathbf{x},\mathbf{x}_m) + b\right) \tag{9}$$

where $\mathcal{I}_{sv} = \{m|\alpha_m > 0\}$, and $k(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^T\cdot\phi(\mathbf{x}')$.

There are three popular approaches for solving a linear system: 1. obtain the inverse of the matrix; 2. Gaussian elimination (GE); 3. iterative method. First, obtaining the inverse matrix is not appropriate because it is numerically unstable and costly on an encrypted domain ($O(n^3)$ on plaintext domain). GE implemented with FHE is also highly inefficient because GE requires many random accesses and reciprocal operations. With iterative methods, it is important to examine the convergence conditions.

Fortunately, our linear system is positive semi-definite because we usually use Mercer's kernels such as linear, polynomial, and radial basis function (rbf) kernels. In addition, we can obtain positive definite linear system by multiplying A^T on both sides in (8). Therefore, both the Gauss-Seidel method and iterative least-square method can be converged to the solution of the linear system (8). However, we finally selected the least-square iterative algorithm as our training algorithm because the Gauss-Seidel method consumes ciphertext levels in every coordinate update. In the following section, we will introduce the HE-friendly least square SVM algorithm to minimize the expensive operations, and to avoid numerical instability.

III. TRAINING ALGORITHM FOR SECURE SUPPORT VECTOR MACHINE

In this study, we propose an HE-friendly support vector machine algorithm to secure data and model information during the training and inference phases. Our algorithm enables training of the SVM model with FHE, in contranst to most researches on secure SVM algorithm, which have focused on either an inference phase with FHE, or training phases with multi-party computation protocols. We assume that all inputs are encrypted using the same key instead of using multiple keys. Therefore, in this section, we explain how to construct and pack the training data for encryption and how to implement training and prediction phases using the HE-friendly operations.

A. DATA PREPARATION

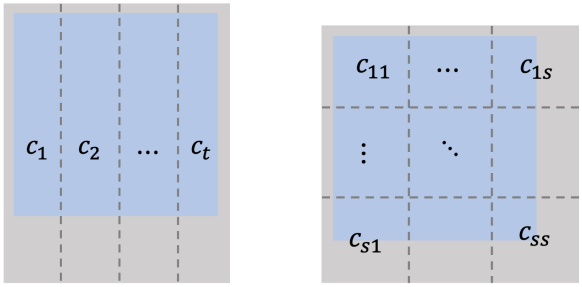
The HEAAN supports encrypting a vector of multiple plaintexts into one ciphertext with slot-wise operations. For efficient computation, it is essential to design the packing of plaintexts from a given database. Thus, we propose two different encoding methods depending on the size of the given data. Suppose that we have n instances $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_n \in \mathbb{R}^d$ which have d features and their labels $y_1,y_2,\ldots,y_n \in \{-1,1\}$. First, we constructed the input design matrix $\mathbf{X}\in\mathbb{R}^{d\times(n+1)}$ with a zero column as follows:

$$\mathbf{X} = \begin{bmatrix}\mathbf{0} & \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n\end{bmatrix} \tag{10}$$

This matrix is used to build the matrix $\mathbf{A}$ in (8) because most of the kernel matrices can be obtained from $\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{(n+1)\times(n+1)}$. Let the number of slots be m. Then, we can encode a matrix column-wise if $\max\{d, n+1\} \le m$; otherwise encode to submatrices.

Moreover, when applying previous researches on secure computations of nonlinear kernels, we can start our algorithm with the matrix $\mathbf{A}$ Other encryption methods can be used for efficient calculations even if re-encryption is required. As a result, like the input design matrix, the matrix $\mathbf{A}$ will be encoded column-wise if $n+1 \le m$; otherwise they can be encoded to submatrices. If possible, we pack multiple columns into one ciphertext to make the computations more efficient. Fig.1 illustrates two types of packing for encrypting a matrix with ciphertexts $c_1,\ldots,c_t$.

Depending on the packing method, our training procedure with FHE uses different algorithms for elementary operations



(a) Column-wise packing (b) Submatrices packing

FIGURE 1. Two types of packing methods for encrypting a matrix with ciphertexts.

and different parallelization strategies for efficient computations. Therefore, in the next section, we introduce the SVM training procedure with HE-friendly operations and parallelizable components.

B. TRAINING PHASE

In the training phase, we select the least-square approaches to solve the linear system (8). The least-square problem has a unique solution if $\mathbf{A}^T\mathbf{A}$ is positive-definite (PD). However, obtaining the exact least square solution is still expensive and numerically unstable because it requires computing $(\mathbf{A}^T\mathbf{A})^{-1}$. Thus, to train a secure SVM model we use the gradient descent method for the least square problem, because the iterative solution can approach to the optimal solution even when computations on the HEAAN scheme involve approximation errors. A secure iterative least square algorithm requires matrix multiplications and gradient descent steps with fully homomorphic encryption.

1) MATRIX MULTIPLICATIONS

Matrix multiplications on an encrypted domain should be carefully designed because the product of two square $n \times n$ matrices has n^3 multiplications and $(n-1)n^2$ additions in the worst case. Moreover, for the ciphertexts, the cost of multiplication is much more expensive than the cost of addition. After packing the multiple slots into one ciphertext, Rotate enables the sum of slots and column-unit or row-unit operations.

Our algorithm contains one matrix multiplication to calculate $\mathbf{A}^T\mathbf{A}$, where $\mathbf{A}$ is a symmetric matrix. We propose different efficient matrix multiplication algorithms depending on the packing method. For simplicity, we can assume that $n+1$ and m are power-of-two integers where zeros can be padded to matrix $\mathbf{A}$ to make $n+1$ be power-of-two. As a result, we can pack matrix $\mathbf{A}$ into $t = (n+1)^2/m$ ciphertexts $\vec{c}_1, \vec{c}_2, \ldots, \vec{c}_t$. Let the resulting ciphertexts denote $\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_t$. Then, matrix multiplications can be efficiently implemented by parallelizing the calculations column-wise because calculating $\vec{d}_i$ is not affected by calculating $\vec{d}_j$ for $\forall i \neq j$.

We use a dot product formulation for matrix multiplication because our encryption scheme supports slotwise operations, and the ciphertext for column i of symmetric matrix $\mathbf{A}$ is the same as for row i of symmetric matrix $\mathbf{A}$. Thus, we can implement matrix multiplication only with the column-wise paddings $\vec{c}_1, \ldots, \vec{c}_t$.

Algorithm 1 shows a detailed description of the encrypted calculation on $\mathbf{A}^T\mathbf{A}$. Algorithm 1 contains some sub-functions for the matrix multiplication for columnwise packing such as `CalculateAtAColumns`, `ProductMatVec`, and `PadColumns`. For simplicity, we skipped the mod reduction procedure to make the precision bits the same. In addition, we slightly modified some notations for ciphertext operations. For example, we used $\mathsf{Rescaling}(\vec{c}; p_c)$ instead of $\mathsf{Rescaling}_{\ell \to \ell'}(\vec{c})$ by replacing the level change with the scaling factor, and we skipped the key notations such as *evk* and *rk* in CMult, Mult and Rotate.

To some extent, our algorithm can inherit the parallelizable property of the matrix multiplication as shown in the main part of algorithm 1. The function `CalculateAtAColumns` can be executed concurrently in different threads because the function execution for each ciphertext requires the read operation without modifying the original data matrix. Also, all of the functions reduce the computational cost by recursively rotating and adding the ciphertext with $\log(m/(n+1))$ times instead of $m/(n+1)$.

However, although column-wise packing can be efficient for calculating $\mathbf{A}^t\mathbf{A}$ with FHE, it cannot support an arbitrary large matrix $\mathbf{A}$. Therefore, we propose another packing method which is scalable for a large matrix. Algorithm 2 is an alternative algorithm to compute the matrix multiplication on the encrypted domain. First, the given matrix is divided into sub-matrices. Each sub-matrix is padded column-wise as a vector, which is encrypted to a ciphertext. Finally, we obtain $s \times s$ blocks as in Fig.1b. Using the encrypted submatrix blocks, we peroformed an efficient parallel implementation of the Fox algorithm.

Our algorithm can generate the maximum s^2 threads, where `ProdMatrixIter` is executed concurrently in multiple threads. Therefore, `ProdMatrixIter` conducts only s `ProdMatrixFox`s which involves expensive operations such as Mult and Rotate. Although updating $\vec{d}_{jl}$ requires temporary memory locking, this is negligible because of the low cost of the Add operation. This multithreading approach enables efficient computations on a large matrix.

In the discussion section, we compare the parallel fox algorithm with the original fox algorithm. `ProdMatrixFox` calculates the multiplication of the ciphertexts from two sub-matrices using the Fox algorithm. `ProdMatrixFox` contains `ExtractDiagonal` and `RowUpMatrix` because stage k involves row-wise broadcasting of the k-th diagonal of matrix A, and a column-wise k upward shift of matrix B. Thus, we implemented the encrypted versions of the operations for vectorized matrices in Algorithm 2. Algorithm 2 allows our SVM training algorithm to be not only scalable for a large matrix but also to be extensible to mult-class classification, just by adding linear complexity. In the discussion section, we describe the extendible property of this algorithm for the multi-class SVM problem.

Algorithm 1 Matrix Multiplication for Columnwise Packing

input : A symmetric matrix $\mathbf{A} \in \mathbb{R}^{(n+1)\times(n+1)}$, ciphertexts $\vec{c}_1, \vec{c}_2, \ldots, \vec{c}_t$ which have m slots, and the bit precision parameter p_c

output: $\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_t$ which are the resulting ciphertexts for $\mathbf{A}^T\mathbf{A}$

Encrypt the matrix $\mathbf{A}$ column-wisely into $\vec{c}_1, \vec{c}_2, \ldots, \vec{c}_t$
Denote the set $\mathcal{C} = \{\vec{c}_1, \ldots, \vec{c}_t\}$
Define a vector $\mathbf{x}_1 = [x_1^1, \ldots, x_1^m] \in \mathbb{R}^m$ where $x_1^u = 1$ if $u\%(n+1) = 1$ and $x_1^u = 0$, otherwise.
Define a vector $\mathbf{z}_l = [z_l^1, \ldots, z_l^m] \in \mathbb{R}^m$ where $z_l^u = 1$ for $u = l(n+1), \ldots, l(n+1)-1$ and $z_l^u = 0$, otherwise.
Generate t threads
for $i = 1;\ i \le t;\ i = i+1$ **do**
    `// The below functions are executed concurrently in different threads`
    $d_i =$ `CalculateAtAColumns` $((\mathcal{C}, i))$
end
`/* Define function for calculate the i-th ciphertext for` $\mathbf{A}^T\mathbf{A}$ `*/`
Function `CalculateAtAColumns` $(\mathcal{C}, i)$ **:**
    Initialize $\vec{ct}_2$ with the ciphertext encrypting a zero matrix
    for $j = 1;\ j \le m/(n+1);\ j = j+1$ **do**
        $k = i*(n+1)+j$ `// k-th column of the matrix` $\mathbf{A}$
        $\vec{ct}_1 =$ `PadColumn` $(\vec{c}_i, k, n+1, m)$
        $\vec{ct}_1 =$ `ProductMatVec` $(\mathcal{C}, \vec{ct}_1)$
        $\vec{ct}_2 \leftarrow \mathbf{Add}\left(\vec{ct}_2, \mathbf{Rotate}(\vec{ct}_1, j(n+1))\right)$
    end
    return $\vec{ct}_2$
`/* Calculate Matrix-Vector product where` $(n+1)$`-dimensional vector is repeated` $m/(n+1)$ `times and encrypted as` $\vec{ct}$`. */`
Function `ProductMatVec` $(\mathcal{C}, \vec{ct})$ **:**
    for $i = 1;\ i \le t;\ i = i+1$ **do**
        $\vec{ct}_1 \leftarrow \mathbf{Rescaling}\left(\mathbf{Mult}(\vec{c}_i, \vec{ct}); p_c\right)$
        for $j = 0;\ j < \log(m/(n+1));\ j = j+1$ **do**
            $\vec{ct}_1 \leftarrow \mathbf{Add}\left(\vec{ct}_1, \mathbf{Rotate}(\vec{ct}_1, -2^j)\right)$
        end
        $\mathsf{a} \in \mathcal{R} \leftarrow \mathbf{Encode}(\mathbf{x}_1, p_c)$
        $\vec{ct}_1 \leftarrow \mathbf{Rescaling}(\mathsf{CMult}(\mathsf{a}, \vec{ct}_1); p_c)$
        for $j = 0;\ j < \log(m/(n+1));\ j = j+1$ **do**
            $\vec{ct}_1 \leftarrow \mathbf{Add}(\vec{ct}_1, \mathbf{Rotate}(\vec{ct}_1, 2^j - 2^{j+\frac{m}{n+1}}))$
        end
        $\mathsf{b} \in \mathcal{R} \leftarrow \mathbf{Encode}(\mathbf{z}_1, p_c)$
        $\vec{ct}_2 \leftarrow \mathbf{Rescaling}\left(\mathbf{CMult}(\mathbf{b}, \vec{ct}_1); p_c\right)$
        $\vec{ct}_2 \leftarrow \mathbf{Add}(\vec{ct}_2, \mathbf{Rotate}(\vec{ct}_2, i))$
    end
    return $\vec{ct}_2$
`/* Define function to make the padded ciphertext with the k-th column of` $\mathbf{A}$ `*/`
Function `PadColumn` $(\vec{c}_i,\ k,\ n+1,\ m)$ **:**
    $l = k\%m$ `// modulo operation`
    $\mathbf{a} \in \mathcal{R} \leftarrow \mathbf{Encode}(\mathbf{z}_l, p_c)$
    $\vec{ct} \leftarrow \mathbf{Rescaling}\left(\mathbf{CMult}(evk, \mathbf{a}, \vec{c}_i); p_c\right)$
    for $j = 0;\ j < \log(m/(n+1));\ j = j+1$ **do**
        $\vec{ct} \leftarrow \mathbf{Add}\left(\vec{ct}, \mathbf{Rotate}(\vec{ct}, 2^j)\right)$
    end
    return $\vec{ct}$

2) EFFECT OF PARALLELIZABLE IMPLEMENTATIONS

In this section, we demonstrate the effect of our submatrix packing in terms of parallelizable implementation for matrix multiplication and the extensible implementation to multi-class classification. In the In SVM model, we need to deal with the kernel matrix to train the model. In our

Algorithm 2 Matrix Multiplication for Submatrix Packing

input : A symmetric matrix $\mathbf{A} \in \mathbb{R}^{(n+1)\times(n+1)}$, ciphertexts $\vec{c}_{ij}$ which have m slots for $i,j = 1,\ldots,s$ for encrypting the matrix $\mathbf{A}$, and the bit precision parameter p_c

output: $\vec{d}_{ij}$ for $i,j = 1,\ldots,s$ which are the resulting ciphertexts for $\mathbf{A}^T\mathbf{A}$

Encrypt the matrix $\mathbf{A}$ with sub-matrices into $\vec{c}_{ij}$ for $i,j = 1,\ldots,s$

Denote $\mathcal{C} = \{\vec{c}_{ij} : i,j = 1,\ldots,s\}$, $\mathcal{D} = \{\vec{d}_{ij} : i,j = 1,\ldots,s\}$, $\mathcal{C}_{i\cdot} = \{\vec{c}_{i1},\ldots,\vec{c}_{is}\}$ and $\mathcal{D}_{i\cdot} = \{\vec{d}_{i1},\ldots,\vec{d}_{is}\}$

Set $b = (n+1)/s$

Define $\mathbf{x}_l$ is the columnwise padded vector of the matrix that is the l right-shifted identity matrix

Define $\mathbf{y}_k(\mathbf{z}_k)$ are the columnwise padded vectors of the matrix whose rows above(below) k-th row are ones including(except) k-th row, and the other rows are zeros.

Initialize $\vec{d}_{ij}$ for $i,j = 1,\ldots,s$ with the ciphertext encrypting a zero matrix

Generate s^2 threads $Thread_{ij}$ for $i,j = 1,\ldots,s$

for $i = 1;\ i \le s;\ i = i+1$ **do**

  for $j = 1;\ j \le s;\ j = j+1$ **do**

    $k = (i+j)\%s$

    $Thread_{ij} \leftarrow$ `ProdMatrixIter(`$\mathcal{D}_{j\cdot}$, $\vec{c}_{jk}$, $\mathcal{C}_{k\cdot}$`)`

  end

end

`/* Calculate` j`-th row of the resulting matrix at` i`-th iteration */`

Function `ProdMatrixIter(`$\mathcal{D}_{j\cdot}$, $\vec{c}_{jk}$, $\mathcal{C}_{k\cdot}$`)`**:**

  for $l = 1;\ l \le s;\ l = l+1$ **do**

    $\vec{ct} \leftarrow$ `ProdMatrixFox(`$\vec{c}_{jk}$,$\vec{c}_{kl}$`)`

    `// Lock` $\vec{d}_{jl}$ `for updating`

    $\vec{d}_{jl} \leftarrow \mathbf{Add}\left(\vec{d}_{jl}, \vec{ct}\right)$

    `// Unlock` $\vec{d}_{jl}$ `after updating`

  end

Function `ProdMatrixFox(`$\vec{a}$, $\vec{b}$`)`**:**

  Initialize $\vec{ct}$ with the ciphertext encrypting a zero matrix

  for $i = 0;\ i < \log(b);\ i = i+1$ **do**

    $\vec{ct}_1 \leftarrow$ `ExtractDiagonal(`$\vec{a}$, i`)`, $\vec{ct}_2 \leftarrow$ `RowUpMatrix(`$\vec{b}$, i`)`

    $\vec{ct} \leftarrow \mathbf{Add}\left(\vec{ct}, \mathbf{ReScaling}\left(\mathbf{Mult}\left(\vec{ct}_1, \vec{ct}_2\right); p_c\right)\right)$

  end

  return $\vec{ct}$

Function `ExtractDiagonal(`$\vec{a}$, k`)`**:**

  l = k%b

  $\mathbf{a} \in \mathcal{R} \leftarrow \mathbf{Encode}(\mathbf{x}_l, p_c)$

  $\vec{ct} \leftarrow \mathbf{Rescaling}(\mathbf{CMult}(\mathbf{a}, \vec{a}); p_c)$

  for $i = 0;\ i < \log b;\ i = i+1$ **do**

    $\vec{ct} \leftarrow \mathbf{Add}(\vec{ct}, \mathbf{Rotate}(\vec{ct}, 2^i))$

  end

  return $\vec{ct}$

Function `RowUpMatrix(`$\vec{a}$, k`)`**:**

  $\mathbf{a}_1 \leftarrow \mathbf{Encode}(\mathbf{y}_k, p_c)$, $\mathbf{a}_2 \leftarrow \mathbf{Encode}(\mathbf{z}_k, p_c)$

  $\vec{ct}_1 \leftarrow \mathbf{CMult}(\mathbf{Rotate}(\vec{a}, -k), \mathbf{a}_1)$ $\vec{ct}_2 \leftarrow \mathbf{Rotate}(\mathbf{Cmult}(\mathbf{Rotate}(\vec{a}, -k), \mathbf{a}_2), b)$

  $\vec{ct} \leftarrow \mathbf{Rescaling}(\mathbf{Add}(\vec{ct}_1, \vec{ct}_2); p_c)$

  return $\vec{ct}$

experiment, we used 100 training examples and test examples, where the base matrix **A**s were 101 × 101 and the number of blocks was 16). However, when we use the larger training datasets, more submatrices are needed. In this case, our multi-threading approach can be much more efficient. We identified the effect of multithreading by increasing the number of blocks s in the sub-matrix packing. Fig.2 shows the relative time which is the ratio of original and multi-threading multiplications. We found that the relative time increases as the number of blocks increased. Therefore, if the number of submatrices is large enough, multithreading makes the matrix multiplication much more efficient.

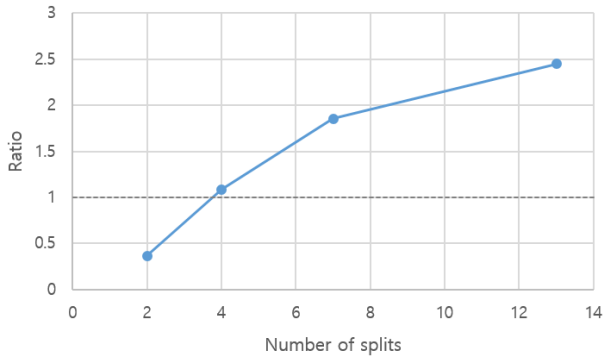


FIGURE 2. Comparison of original and multi-threading matrix multiplications. X-axis refers to s in Algorithm 2, and Y-axis refers to the relative time between the original fox algorithm and the multi-threading fox algorithm.

3) GRADIENT DESCENT FOR THE LEAST SQUARES PROBLEM

After calculating $\mathbf{A}^T\mathbf{A}$, we can use the gradient descent approach to obtain the SVM coefficient α, b in equation (8). We can designate the vector of the SVM coefficients as $\boldsymbol{\beta} = [b\ \alpha^T]^T$. The linear least square problem for equation (8) is a convex problem which satisfies the Normal equation $\mathbf{A}^T\mathbf{A}\boldsymbol{\beta} = \mathbf{A}^T\tilde{\mathbf{1}}$ and has one global minimizer $\boldsymbol{\beta}^*$. Although most algorithms for the linear least square problem require computing matrix inversions, we avoided the matrix inversion because it is highly expensive on an encrypted domain. Instead, we simply computed the descent direction of the k-th iteration as $\mathbf{p}^{(k)} = \mathbf{A}^T\mathbf{A}\boldsymbol{\beta}^{(k)} - \mathbf{A}^T\tilde{\mathbf{1}}$. The single iteration explicitly requires one matrix-vector multiplication and one vector-vector subtraction, because we can pre-compute $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\tilde{\mathbf{1}}$. We can implement the gradient descent algorithm because all operations are linear operations and, therefore, HE-friendly.

C. PREDICTION PHASE

For the prediction phase, we need to construct an SVM classifier (9) from the trained SVM parameter $\boldsymbol{\beta} = [b\ \alpha^T]^T$. Calculating the kernel function $k(\mathbf{x}, \mathbf{x}_m)$ between the new test data $\mathbf{x}$ and training example $\mathbf{x}_m$ is implemented similarly to the data construction phase. If we use a polynomial kernel, this calculation can be implemented with fully homomorphic encryption without approximation. To predict the output value, we need to obtain $\boldsymbol{\beta}' = [b\ \ (\alpha \cdot \mathbf{y})^T]^T$. Without decryption, we can obtain $\boldsymbol{\beta}'$ by using matrix $\mathbf{A}$, and the inner product between the kernel value and $\boldsymbol{\beta}'$ induces the SVM function (9).

IV. EXPERIMENTS

In this section, we evalulate our secure SVM training algorithm for toy examples and various real-world datasets. We compare the performance of our model with the logistic regression model using FHE proposed in [47], which is currently the state-of-the-art model supporting fully homomorphic training. The SVM model was tested for using the polynomial kernel (**SVM-poly**) and the rbf kernel (**SVM-rbf**). Since both secure training algorithms are based on the gradient descent method, we compare the average time per iteration and test accuracy. Moreover, we compare two packing methods for secure SVM training.

A. DATA DESCRIPTION

1) TOY EXAMPLES

To demonstrate the superiority of our model over various data distributions, we conducted experiments for two non-linearly separable synthetic datasets named **linear** and **ring**, respectively. Figure 3 illustrates the two-dimensional **linear** and **ring** datasets which have a single binary class label. Both datasets contain a total 79 data points, where 63 are used for training, and the rest are used to measure test accuracy. The **linear** dataset consists of two overlapping chunks with a linear decision boundary. The **ring** dataset has a single mass wrapped in a ring with a non-linear (circular) boundary.

2) REAL-WORLD DATASETS

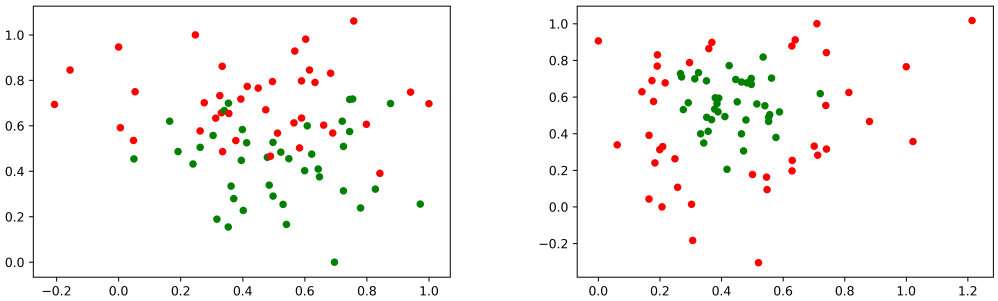
We also implemented the model for 8 widely used datasets the from UCI data repository [48], and 4 datasets used in [47]. For the UCI datasets, we excluded categorical variables (>2) as predictive variables, even if we included the binary variables which had values of 0 or 1. 100 training samples and 100 test samples were extracted from each dataset. For the SVM model, we used the pre-computed A matrix defined in 8. A detailed description of each dataset can be found in Table.1.

B. EXPERIMENTAL SETTINGS

For a fair comparison, all of the experiments were conducted on a machine with an Intel Xeon CPU E5-2660 v3 @ 2.60GHz. 10 iterations of gradient descent were done for both models. For the logistic regression model, the approximation degree of the sigmoid function was set to 7 to maximize accuracy, and the learning rate was the same as that presented in the paper [47]. The learning rate for SVM training was chosen in $\{0.00005, 0.0001, 0.0005, 0.001, 0.005\}$. In the polynomial kernel, we used degree 2 polynomial $k(\mathbf{x}, \mathbf{x}') = (\gamma\mathbf{x}^T\mathbf{x}' + c)^2$ where γ and c were chosen in $\{0.01, 0.05, 0.1, 0.5\}$ and $\{0.05, 0.1, 0.5, 1\}$, respectively. The rbf kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$ was also used, where γ was selected in $\{0.1, 0.5, 1, 5, 10\}$. Table 1 shows several parameters for encryption that should also be determined. In the experiment, the value of $\log(q_L)$ was set to 1200, which is the maximum value available. The value of $\log(p)$ was set to 25. When encrypting a real data matrix with column-wise packing, the number of sub-ciphertexts was 101, while the number of blocks for submatrices packing was 16.

C. RESULTS

Fig.4 shows the classification results for the toy examples. As shown in the first column, the logistic regression model showed relatively poor performance, especially for the **ring** dataset. This is because the logistic classifier essentially has

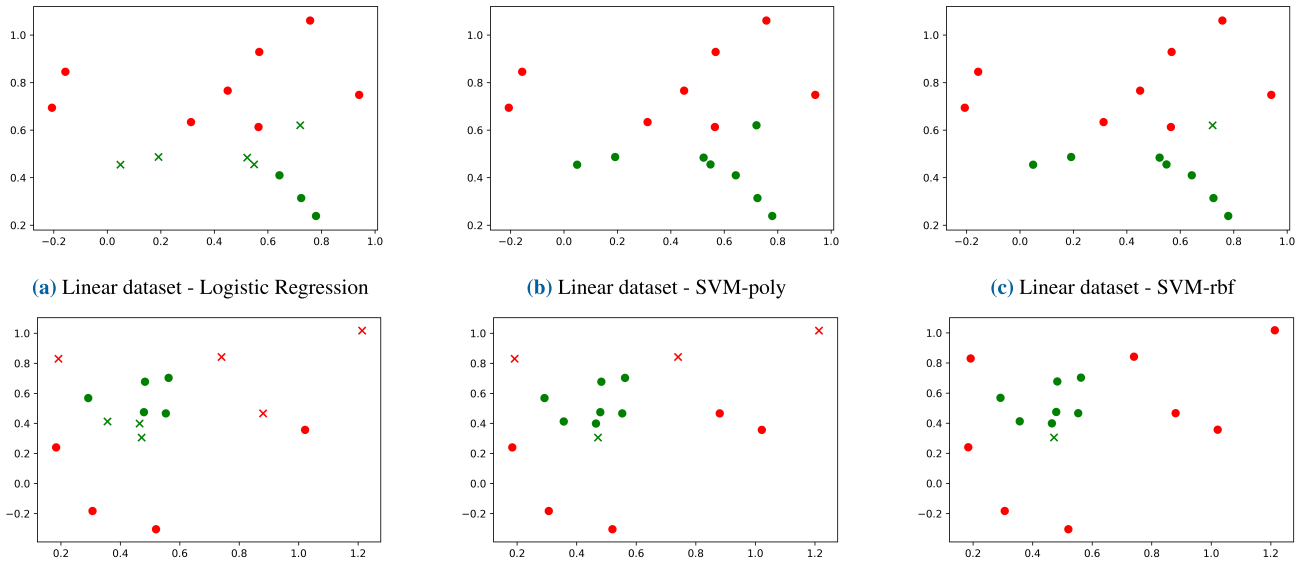


(a) Linear dataset (b) Ring dataset

FIGURE 3. Description of toy examples.

TABLE 1. Description of real datasets.

Dataset	Num. of features	Kernel hyper-parameters				Learning rate (poly)	Learning rate (rbf)
		gamma(poly)	c	d	gamma(rbf)		
Australian Credit	6	0.1	1	2	0.5	0.0001	0.001
Bupa Liver Disorders	6	0.05	0.1	2	1	0.005	0.005
German Credit	16	0.1	0.05	2	0.1	0.0001	0.005
Heart Disease	8	0.05	0.5	2	0.1	0.001	0.0005
Ionosphere	33	0.05	0.1	2	0.1	0.00005	0.005
Pima Indians Diabetes	8	0.1	0.5	2	0.1	0.001	0.001
Sonar	60	0.01	0.1	2	0.5	0.005	0.005
Wisconsin Breast Cancer	9	0.1	0.5	2	0.5	0.00005	0.001
Myocardial Infarction	9	0.1	0.5	2	0.1	0.00005	0.0005
Nhanes III	15	0.05	0.1	2	0.1	0.005	0.005
Prostate Cancer Study	9	0.1	0.5	2	0.1	0.00005	0.001
Umaru Impact Study	8	0.05	0.5	2	1	0.0005	0.005



(a) Linear dataset - Logistic Regression (b) Linear dataset - SVM-poly (c) Linear dataset - SVM-rbf

(d) Ring dataset - Logistic Regression (e) Ring dataset - SVM-poly (f) Ring dataset - SVM-rbf

FIGURE 4. Classification results for test sets of Toy Examples. True label is distinguished by color, 'o' means correctly classified, and 'x' means misclassified.

the simple decision boundary. The SVM model, on the other hand, was able to classify data very accurately even in nonlinear cases, especially with the rbf kernel, which can find more complex decision boundaries (including circular boundary). Also, Table.2 shows that the average computation time for iteration for the SVM model is about 2 times less than that of the logistic regression model.

The results for the real datasets can be found in Table.3 and Table.4. We implemented our secure SVM training for 12 real-world datasets. The tables contain the classification

TABLE 2. Description and results for Toy Datasets. The best result for each dataset is highlighted in bold.

Dataset	Kernel hyper-parameters				Learning rate(poly)	Learning rate(rbf)	Average time of iteration (sec)		
	gamma(poly)	c	d	gamma(rbf)			Logistic	SVM-poly	SVM-rbf
Linear	0.05	0.5	2	5	0.001	0.001	64.529	**31.817**	32.499
Ring	0.5	0.5	2	10	0.0005	0.001	65.548	**32.447**	34.271

TABLE 3. Results for real datasets with columnwise packing. SVM classification accuracies for plaintexts are also listed in parentheses. The best result for each dataset is highlighted in bold.

Dataset	Average time of iteration (sec)			Accuracy (%)		
	Logistic	SVM-poly	SVM-rbf	Logistic	SVM-poly	SVM-rbf
Australian Credit	74.106	**32.614**	40.102	69	72 (74)	**79 (79)**
Bupa Liver Disorders	74.548	**38.329**	38.728	57	**73 (73)**	72 (73)
German Credit	87.509	**33.520**	39.945	68	**69 (69)**	68 (68)
Heart Disease	80.210	33.233	**33.027**	**85**	**85 (85)**	84 (84)
Ionosphere	95.238	**35.407**	36.913	86	91 (91)	**92 (92)**
Pima Indians Diabetes	80.715	35.568	**33.201**	66	**73 (73)**	70 (70)
Sonar	94.339	**32.858**	35.922	83	**88 (88)**	86 (89)
Wisconsin Breast Cancer	81.253	**33.545**	38.153	98	98 (97)	98 (98)
Myocardial Infarction	79.337	39.228	**34.514**	76	**93 (93)**	86 (86)
Nhanes III	81.772	33.140	**33.094**	76	77 (77)	**78 (78)**
Prostate Cancer Study	81.825	38.453	**34.350**	59	**70 (69)**	64 (64)
Umaru Impact Study	78.710	**35.402**	35.828	78	78 (78)	78 (78)

TABLE 4. Results for real datasets with Submatrices packing.

Dataset	Average time of iteration (sec)		Accuracy (%)	
	SVM-poly	SVM-rbf	SVM-poly	SVM-rbf
Australian Credit	67.096	63.993	71	79
Bupa Liver Disorders	64.134	61.237	73	71
German Credit	67.060	63.771	68	68
Heart Disease	65.074	61.358	85	84
Ionosphere	63.788	62.583	89	93
Pima Indians Diabetes	62.658	62.705	76	69
Sonar	62.684	63.152	88	91
Wisconsin Breast Cancer	67.015	62.483	97	98
Myocardial Infarction	64.858	66.564	93	86
Nhanes III	65.585	62.433	76	78
Prostate Cancer Study	67.449	61.701	72	63
Umaru Impact Study	64.931	68.140	78	78

accuracy of plaintext. Both tables show that the SVM model had better classification accuracy than the logistic regression model for most of the real-world datasets, because nonlinear boundaries are appropriate for real-world datasets. In the SVM model, for 10 iterations, the polynomial kernel seemed to outperform the rbf kernel, but the classification accuracy of the plaintext did not, when the results in the parentheses were compared. We also compared the computation time for an iteration. Comparing Table.3 and Table.4, the column-wise packing was always faster than the submatrix packing with a fixed data size, even though the number of sub-ciphertexts was larger for column-wise packing. Therefore, column-wise packing is preferred for mid-sized datasets, but submatrix packing is scalable and can be more efficient for a large matrix, because of multithreading.

V. DISCUSSION

In this study, we proposed a secure SVM training algorithm for binary classification. However, our submatrix packing allows it to be efficiently extended to a multi-class SVM model. The SVM model was originally developed for binary classification. Thus, there are two approaches that can be used to transform a multi-class classification into multiple binary classification problems, the one-versus-one and one-versus-rest approaches. In the c-class classification problem, the one-versus-one approach requires solving the $c(c-1)/2$ binary classification problems, whereas the one-versus-rest approach requires solving the c binary classification problems. When training the SVM, calculating $\mathbf{A}^T\mathbf{A}$ is expensive. However, in the c-class classification problem, each binary classification problem can share the matrix $\mathbf{A}$, excep the first row and the first column of the matrix (label information $\mathbf{y}$).

For example, in a one-versus-rest approach, the number of multiplications to obtain the encrypted $\mathbf{A}^T\mathbf{A}$ increases by about $s \times c$ compared with one binary classification SVM problem, where s has the same meaning as in Algorithm 2.

VI. CONCLUSION

In this study, we proposed a secure least squares SVM algorithm for the training phase. To the best of our knowledge, our study is the first to propose an SVM training algorithm with FHE, because the original SVM training algorithm contains lots of HE-unfriendly operations and procedures. We addressed the problem by using a least squares SVM model which not only makes it possible to avoid many HE-unfriendly operations, such as comparisons and non-polynomial functions, but also to reduce the training procedure to the linear system.

We devised two different packing methods, column-wise packing and submatrix packing. For small datasets, column-wise packing is more efficient than submatrix packing, since column-wise packing requires one less Rescaling procedure. However, submatrix packing is applicable for an arbitrary large matrix. We implemented efficient matrix multiplication by introducing multi-threading approach. Moreover, submatrix packing makes it easy to extend the training algorithm of the binary classification to multi-class classification with a small marginal cost.

In the experimental results, we found that our algorithm achieved better classification performance for the toy dataset and most of the real datasets. In addition, the average time of iteration for our algorithm was efficient because our algorithm does not require any non-polynomial approximate operations during training iteration. Our training algorithm can be combined with the state of the art inference algorithms with FHE for the SVM model. Thus, our study enables for an entire SVM classification algorithm from training to inference to be with FHE.

REFERENCES

[1] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[2] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *J. Biomed. Informat.*, vol. 50, pp. 234–243, Aug. 2014.

[3] J. H. Cheon, J. Jeong, J. Lee, and K. Lee, "Privacy-preserving computations of predictive medical models with minimax approximation and non-adjacent form," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2017, pp. 53–74.

[4] Y. Jiang, J. Hamer, C. Wang, X. Jiang, M. Kim, Y. Song, Y. Xia, N. Mohammed, M. N. Sadat, and S. Wang, "SecureLR: Secure logistic regression model via a hybrid cryptographic protocol," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 16, no. 1, pp. 113–123, Jan. 2019.

[5] S. Wang, Y. Zhang, W. Dai, K. Lauter, M. Kim, Y. Tang, H. Xiong, and X. Jiang, "Healer: Homomorphic computation of exact logistic regression for secure rare disease variants analysis in GWAS," *Bioinformatics*, vol. 32, no. 2, pp. 211–218, 2016.

[6] J. H. Cheon, D. Kim, Y. Kim, and Y. Song, "Ensemble method for privacy-preserving logistic regression based on homomorphic encryption," *IEEE Access*, vol. 6, pp. 46938–46948, 2018.

[7] D. J. Wu, T. Feng, M. Naehrig, and K. Lauter, "Privately evaluating decision trees and random forests," *Proc. Privacy Enhancing Technol.*, vol. 2016, no. 4, p. 335, Oct. 2016.

[8] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 1209–1222.

[9] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "CHET: An optimizing compiler for fully-homomorphic neural-network inferencing," in *Proc. 40th ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*, 2019, pp. 142–156.

[10] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Cham, Switzerland: Springer, 2017, pp. 409–437.

[11] S. Laur, H. Lipmaa, and T. Mielikäinen, "Cryptographically private support vector machines," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2006, pp. 618–624.

[12] Y. Hu, L. Fang, and G. He, "Privacy-preserving SVM classification on horizontally partitioned data with secure multi-party computation," *J. Inf. Comput. Sci.*, vol. 6, no. 6, pp. 2341–2347, 2009.

[13] H. Yu, X. Jiang, and J. Vaidya, "Privacy-preserving SVM using nonlinear kernels on horizontally partitioned data," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2006, pp. 603–610.

[14] K.-P. Lin and M.-S. Chen, "On the design and analysis of the privacy-preserving SVM classifier," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 11, pp. 1704–1717, Nov. 2011.

[15] Y. Rahulamathavan, R. C.-W. Phan, S. Veluru, K. Cumanan, and M. Rajarajan, "Privacy-preserving multi-class support vector machine for outsourcing the data classification in cloud," *IEEE Trans. Depend. Sec. Comput.*, vol. 11, no. 5, pp. 467–479, Sep. 2014.

[16] S. G. Teo, S. Han, and V. C. S. Lee, "Privacy preserving support vector machine using non-linear Kernels on Hadoop mahout," in *Proc. IEEE 16th Int. Conf. Comput. Sci. Eng.*, Dec. 2013, pp. 941–948.

[17] F. Liu, W. K. Ng, and W. Zhang, "Encrypted SVM for outsourced data mining," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, Jun. 2015, pp. 1085–1092.

[18] M. Z. Omer, H. Gao, and F. Sayed, "Privacy preserving in distributed SVM data mining on vertical partitioned data," in *Proc. 3rd Int. Conf. Soft Comput. Mach. Intell. (ISCMI)*, Nov. 2016, pp. 84–89.

[19] H. Zhu, X. Liu, R. Lu, and H. Li, "Efficient and privacy-preserving online medical prediagnosis framework using nonlinear SVM," *IEEE J. Biomed. Health Inform.*, vol. 21, no. 3, pp. 838–850, May 2017.

[20] L. Wang, J. J. Shi, C. Chen, and S. Zhong, "Privacy-preserving face detection based on linear and nonlinear kernels," *Multimedia Tools Appl.*, vol. 77, no. 6, pp. 7261–7281, Mar. 2018.

[21] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. Asia Conf. Comput. Commun. Secur. (ASIACCS)*, 2018, pp. 707–721.

[22] A. Barnett, J. Santokhi, M. Simpson, N. P. Smart, C. Stainton-Bygrave, S. Vivek, and A. Waller, "Image classification using non-linear support vector machines on encrypted data," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 857, 2017. [Online]. Available: https://eprint.iacr.org/2017/857.pdf

[23] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2009. [Online]. Available: http://crypto.stanford.edu/craig

[24] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2010, pp. 24–43.

[25] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2011, pp. 487–504.

[26] J.-S. Coron, D. Naccache, and M. Tibouchi, "Public key compression and modulus switching for fully homomorphic encryption over the integers," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2012, pp. 446–464.

[27] J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun, "Batch fully homomorphic encryption over the integers," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2013, pp. 315–335.

[28] J. H. Cheon, J. Kim, M. S. Lee, and A. Yun, "CRT-based fully homomorphic encryption over the integers," *Inf. Sci.*, vol. 310, pp. 149–162, Jul. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002002551500184X

[29] Z. Brakerski and V. Vaikuntanathan, "Lattice-based FHE as secure as PKE," in *Proc. 5th Conf. Innov. Theor. Comput. Sci. (ITCS)*. New York, NY, USA: ACM, 2014, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/2554797.2554799

[30] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2011, pp. 505–524.
[31] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Proc. Annu. Cryptol. Conf., Adv. Cryptol. (CRYPTO)*. Berlin, Germany: Springer, 2012, pp. 505–524.
[32] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, Jul. 2014, [Online]. Available: http://doi.acm.org/10.1145/2633600
[33] C. Gentry, S. Halevi, and N. P. Smart, "Better bootstrapping in fully homomorphic encryption," in *Public Key Cryptography—PKC*. Berlin, Germany: Springer, 2012, pp. 1–16.
[34] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the AES circuit," in *Proc. Annu. Cryptol. Conf.* Berlin, Germany: Springer, 2012, pp. 850–867.
[35] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proc. 44th Symp. Theory Comput. (STOC)*, 2012, pp. 1219–1234.
[36] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, "Improved security for a ring-based fully homomorphic encryption scheme," in *Cryptography and Coding*, M. Stam, Ed. Berlin, Germany: Springer, 2013, pp. 45–64.
[37] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2013, pp. 75–92.
[38] J. H. Cheon and D. Stehlé "Fully homomophic encryption over the integers revisited," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2015, pp. 513–536.
[39] S. Halevi and V. Shoup, "Bootstrapping for HElib," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2015, pp. 641–670.
[40] L. Ducas and D. Micciancio, "FHEW: Bootstrapping homomorphic encryption in less than a second," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2015, pp. 617–640.
[41] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proc. 3rd ACM Workshop Cloud Comput. Secur. Workshop (CCSW)*, 2011, pp. 113–124.
[42] K. Lauter, A. López-Alt, and M. Naehrig, "Private computation on encrypted genomic data," in *Proc. Int. Conf. Cryptol. Inf. Secur. Latin Amer.* Cham, Switzerland: Springer, 2014, pp. 3–27.
[43] J. H. Cheon, M. Kim, and K. Lauter, "Homomorphic computation of edit distance," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Berlin, Germany: Springer, 2015, pp. 194–212.
[44] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proc. 3rd Innov. Theor. Comput. Sci. Conf. (ITCS)*. New York, NY, USA: ACM, 2012, pp. 309–325. [Online]. Available: http://doi.acm.org/10.1145/2090236.2090262
[45] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*. Baltimore, MD, USA: USENIX Association, 2018, pp. 1651–1669. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar
[46] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Process. Lett.*, vol. 9, no. 3, pp. 293–300, 1999.
[47] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon, "Logistic regression model training based on the approximate homomorphic encryption," *BMC Med. Genomics*, vol. 11, no. 4, p. 83, 2018.
[48] D. Dua and C. Graff. (2017). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

SAEROM PARK received the B.S. and Ph.D. degrees in industrial engineering from Seoul National University, in 2013 and 2018, respectively. She is currently an Assistant Professor with the Department of Convergence Security Engineering, Sungshin Women's University, Seoul, South Korea. Her research interests include kernel machines, representation learning, transfer learning, and secure machine learning.

JUNYOUNG BYUN received the B.S. degree in industrial engineering from Seoul National University, Seoul, South Korea, in 2017. He is currently a Graduate Student with the Department of Industrial Engineering, Seoul National University. His research interests include machine learning and neural networks.

JOOHEE LEE received the B.S. degree in mathematical education from Korea University, Seoul, South Korea, in 2013. She is currently pursuing the Ph.D. degree with the Department of Mathematical Science, Seoul National University, South Korea, advised by Prof. J. H. Cheon. Her current research interests include lattice cryptography, cryptographic protocols, and information security.

JUNG HEE CHEON received the B.S. and Ph.D. degrees in mathematics from KAIST, in 1991 and 1997, respectively.

He was with ETRI, from 1997 to 2000, Brown University, in 2000, and ICU, from 2000 to 2003. He is currently a Professor with the Department of Mathematical Sciences and the Director of the Industrial and Mathematical Data Analytics Research Center (IMDARC), Seoul National University (SNU). His research focuses on computational number theory, cryptology, and their applications to practical problems. He received the Best Paper Award in Asiacrypt 2008 and Eurocrypt 2015. He co-chaired ANTS-XI, Asiacrypt 2015/2016, and MathCrypt 2017/2018. He has served as a Program Committee for Crypto, Eurocrypt, Asiacrypt, and so on. He is also an Associate Editor of *Design, Codes and Cryptography* and the *Journal of Communication Network*, will join to the editorial board of the *Journal of Cryptology* which is the most prestigious journal in *Cryptology*.

JAEWOOK LEE received the B.S. degree in mathematics from Seoul National University, Seoul, South Korea, in 1993, and the Ph.D. degree in applied mathematics from Cornell University, in 1999. He is currently a Professor with the Department of Industrial Engineering, Seoul National University. His research interests include machine learning, neural networks, global optimization, and their applications to data mining and financial engineering.