Privacy-Preserving Support Vector Machines with Flexible Deployment and Error Correction

Weican Huang and Ning Ding(✉)

School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, 800 Dongchuan RD, Shanghai 200240, China
{hwc394519627,dingning}@sjtu.edu.cn

Abstract. Support vector machines (SVMs) are one of the most commonly used models for classification problems in machine learning. Nowadays there is an important scenario that many different parties jointly perform SVM training by integrating their individual data, while at the same time it is required that privacy of data can be preserved. At present there are three main routes to achieving privacy-preserving SVM. First, all parties jointly generate kernel matrices privately and then use them for remaining training (e.g. Yu et al. 2006). Second, based on the first route, an additional randomization is adopted to randomize kernel matrices in order to (heuristically) hide information exposed by kernel matrices (e.g. Mangasarian et al. 2008). Third, also the securest one, all parties run MPC protocols for computing whole optimization algorithms privately (not merely the generation of kernel matrices as the first two routes do) (e.g. Liu et al. 2018 and Wang et al. 2020).

In this paper we propose a new efficient privacy-preserving SVM protocol in the third route that privately realizes the gradient descent method to optimize SVM and its security is proven in the semi-honest model. Our protocol admits the following advantages.

- The protocol is of flexible deployment. It supports the deployment of arbitrarily multiple servers and multiple clients.
- The protocol can tolerate dropping-out of some servers.
- The protocol admits the ability of malicious-error-message correction (which is actually beyond the semi-honest security). If a small number of messages are corrupted, it can still recover correct messages as desired.

We remark that none of the above advantages can be obtained by some known work. Moreover, when compared to the privacy-preserving SVM by Liu et al. 2018 and Wang et al. 2020, our protocol achieves higher efficiency. We implement our protocol in Python and the experiments verify its efficiency.

Keywords: Privacy-preserving SVM · Multi-party computation · Shamir's secret sharing

1 Introduction

Machine learning has been widely used in various fields, in which classification is one of the most commonly used functions and often applied in image recognition, data mining, text analysis, anomaly detection, recommendation systems and many other businesses. Usually sufficient perfect training data is always the premise to ensure the accuracy of trained models. But in reality complete data and sufficient computing power are not always held by one party, and the need for protecting privacy prevents arbitrary flow of data among different owners. Due to the increasing requirement of distributed machine learning and privacy preserving, privacy-preserving machine learning has attracted more and more attention recently.

Support vector machines (SVMs) are a widely used model in machine learning for classification problems. So there have been many achievements in the research line of privacy-preserving SVM. Basically there are three main routes to realizing it. The first route is that all parties jointly generate kernel matrices privately and then use them for remaining training [15,19,21]. When data is vertically partitioned and linearly separable, [21] asks each party computes its own kernel matrix and then lets a party/server integrate them to the whole kernel matrix. To prevent a party from obtaining the kernel matrix of someone else, this method requires data to be held by at least three parties. [19] uses homomorphic encryption to compute kernel matrices under arbitrary partition based on similar ideas. [15] also uses Paillier homomorphic encryption to calculate the kernel matrix under vertical partition.

The second route is that based on the first route an additional randomization is adopted to randomize kernel matrices in order to (heuristically) hide information exposed by kernel matrices [11,12,22]. Combining random kernel functions and matrix summation, [12] uses random linear transformations to avoid possible information leakage caused by publishing local kernel matrices. In [8] random linear transformations are also applied to one class SVM, and transformed data is used as a new input to calculate kernel matrices. [22] uses random kernel functions and integer vector encryption to encrypt data sets with horizontal or vertical partitions, allowing different parties to encrypt their data with different keys, and train them by a single server.

The third route is that all parties run MPC/2PC protocols for computing whole optimization algorithms privately (not merely the generation of kernel matrices as the first two routes do), which include SMO (Sequential Minimal Optimization), the kernel-adatron algorithm and gradient descent algorithm [7,9,20] etc. [7] proposes protocols to implement kernel-adatron and kernel perceptron learning algorithms, but without conducting experiments to verify efficiency. [9] implements a secure SMO protocol with the distributed two-trapdoor public-key cryptosystem (DT-PKC). [20] also designs GD (gradient descent) based secure SVM training using DT-PKC. These methods can protect all data throughout the whole training, but also at a cost of great time.

Besides the above works specializing in SVM, there are some works aiming at realizing privacy-preserving training for a variety of models. SecureML [14] applies the ABY framework with a new fixed-point multiplication protocol to linear regression, logistic regression and neural network. Chameleon [16] and ABY3 [13] change the setting from two servers to three servers, managing to simplify the protocols. FLASH [1] expands to four servers, allowing at most one malicious, proposing a framework with high robustness only by using symmetric-key primitives. These works have made great achievements in improving efficiency, but there are still some shortcomings and, for instance, the number of servers have to be fixed to constants, which is inflexible in deployment. (In practice each data owner is usually willing to participate joint training instead of just providing data to others. So an owner is not only a data provider, but also a server. The present works require the number of servers small (≤ 4), which thus limits their applications. Besides flexibility, another motivation to increase the number of servers is to enhance the resistance to collusive servers. Assuming the protocol has the (n, t)-threshold property, i.e. at least t corrupted servers of n ones together can recover data, which makes malicious recovery more difficult as n, t increases.)

Summary. We provide a summary to the current state of the art in privacy-preserving SVM. The first route is essentially private-preserving matrix summation which gains high efficiency, but kernel matrices are exposed and may leak information. The second route, adopting an additional randomization to kernel matrices, only provides a heuristic strategy to hide kernel matrices without a security proof. The third route is the securest, protecting all data throughout the training but at the cost of a large loss of efficiency. Moreover, all the works, including general privacy preserving machine learning for a variety of models, cannot be applied to scenarios of multiple servers and cannot handle the case that some messages are corrupted.

1.1 Our Contribution

In this paper we propose a new efficient privacy-preserving SVM protocol in the third route that privately realizes the SGD (stochastic gradient descent) method to optimize SVM. Our protocol admits the following advantages.

- The protocol is of flexible deployment. It supports the deployment of arbitrarily multiple servers and multiple clients.
- The protocol can tolerate dropping-out of some servers (up to some threshold value).
- The protocol admits the ability of malicious-error-message correction. Error messages caused by the adversary's malicious behaviors are not randomly distributed and will hinder secret reconstruction, or even lead to wrong results. If a small number of messages (up to some threshold value) are corrupted, it can still recover correct messages as desired.

We note that none of the above advantages can be obtained by some known work. The security of privacy preserving is established in the semi-honest adversarial model. Our protocol has the ability to deal with the collusion of servers less than the secret sharing threshold. We remark that the third advantage above shows that our protocol can resist some malicious-message attack, which is actually beyond the semi-honest adversarial model. For simplicity we just claim that our protocol is secure in the semi-honest adversarial model.

Thus our protocol can be flexibly used between n servers (calculators) and m clients (data holders), and tolerates some servers dropping out halfway. In the running of the protocol, all clients submit the sharing of their sample data to all servers, which then perform MPC to execute the SGD algorithm to optimize the parameters of SVM, and finally output the shares of the optimized parameters. We note that there is no significant difference in accuracy between the models trained by our protocol and those trained with plain data directly.

Compared with [9,20], our protocol achieves higher efficiency. [9,20] use a public key encryption system based on modular exponentiation, so 100 rounds of their training takes nearly 10 h on a 236×13 training set. Our protocol runs MPC based on shares, which only consists of addition and multiplication over finite fields, so 1000 rounds of our training takes about 48 s on the breast-cancer training set of 500×10 and 1000 rounds of training takes about 10 min on the diabetes training set of 500×10, and 30 min on the german.number training set of 800×24. The programming language used in our experiment code is Python.

Compared with [1,13,14] as well as [9,20], which should fix the number of servers to a value among 2 to 4, our protocol can be deployed among any number of servers (no less than 3). Also, by using Shamir's (n, t)-threshold secret sharing scheme, we can arbitrarily deploy n servers and tolerate dropping out of at most $n - 2t$ ones. Also, less than t colluding servers learn nothing in our protocol more than what they deserve. Moreover, by introducing the Berlekamp-Welch algorithm as an optional recovery algorithm, correct messages can be recovered even if less than t messages/shares are corrupted. The security parameter of the protocol is the bit length of the random number in the protocol.

Finally, we note that the SGD optimization algorithm we use (see Algorithm 1) is for linear kernel functions originally, it can be extended for nonlinear SVM training. To train nonlinear kernel function SVMs, the input feature $\mathbf{x}$ should be replaced by its mapping result $\phi(\mathbf{x})$ corresponding to nonlinear kernel functions, or apply the quadratic form (see [17]).

1.2 Our Techniques

Now we present a high-level description of our protocol and sketch main techniques. Assume there are m clients trying to cooperate on privacy-preserving SVM training, while n servers provide secure computation services. Basically, the protocol runs as follows. First the clients submit the shares of the sample data to the servers. Then the servers jointly compute the parameters of the SVM model using the SGD strategy. That is, the protocol consists of many repetitions, each of which computes an iteration of the SGD method. In each repetition, the servers have as input the shares of current values of the parameters and then perform some MPC protocols to compute gradient iteration and finally obtain the shares of the new values of the parameters. When the protocol halts, all servers output the shares of the optimized parameters of the model.

More concretely, in the SGD for SVM, the gradient and iteration of the parameters are given in the following formulas.

$$grad = \alpha \cdot \mathbf{w} - C \cdot \mathtt{I}(\mathbf{y}_i(\mathbf{x}_i \cdot \mathbf{w}^T) - 1)(\mathbf{y}_i \mathbf{x}_i) \tag{1}$$

$$\mathbf{w} = \mathbf{w} - l \cdot grad, \tag{2}$$

where $\mathbf{w}$ are the parameters to be optimized, $\mathbf{y}_i$ and $\mathbf{x}_i$ are the label and features of the ith sample, C is the penalty coefficient of the relaxation variable, and l is the learning rate and C, l are constants, and $\mathtt{I}(x)$ is the function such that if $x > 0, \mathtt{I}(x) = 0$, and if $x < 0, \mathtt{I}(x) = 1$.

According to the above formulas, each iteration of SGD only consists of addition, multiplication and comparison operations. Thus to realize the MPC for SGD, it suffices to show how to realize these operations privately.

Notice that the SGD algorithm is not over integers, while Shamir's secret sharing is built over finite fields like $\mathbb{Z}_p$. Recall that [3] presents secure fixed-point addition, multiplication and truncation with respect to Shamir's scheme. Thus we adopt [3] to realize the secure addition and multiplication.

We introduce the Berlekamp-Welch algorithm as an optional error correction algorithm. The algorithm takes the received codewords (i.e. shares of Shamir's secret sharing) as input, and recovers the correct values. In our protocol, all communication takes place within the reveal protocol. By replacing the calculation in the reveal protocol with the Berlekamp-Welch algorithm, we can get the correct result when there are a few errors in the received messages.

1.3 Organization

The rest of the paper is arranged as follows. In Sect. 2 we present a part of preliminaries and relegate the rest to Appendix A due to lack of space. In Sect. 3 we present the security model. In Sect. 4 we show the details of building blocks (i.e. secure addition, multiplication and comparison etc.). In Sect. 5, we present our privacy-preserving SVM protocol. In Sect. 6 we give performance evaluation of our protocol via theoretical analysis and experiments.

2 Preliminaries

We recall the notion of support vector machines and Shamir's secret sharing here, and relegate the Berlekamp-Welch algorithm to Appendix A.

2.1 Support Vector Machines

Support vector machines (SVM) are a classical machine learning model. The concept of SVM was proposed by Vladimir N. Vapnik and Alexey Ya Chervonenkis in 1963. The current version was proposed by Corinna Cortes and Vapnik in 1993 and published in 1995 [5]. As a supervised learning model, SVM is mainly used in regression and classification problems. The principle of SVM is to find a hyperplane to maximize the minimum distance between the hyperplane and the two kinds of sample points.

SVM can be optimized by the gradient descent method. We consider linear SVM which has a loss function, called hinge loss function, $\max(0, 1 - \mathbf{y}_i(\mathbf{x}_i\mathbf{w}^T))$, where the subscript i represents the ith sample, and $\mathbf{x}$ and $\mathbf{y}$ represent the feature data and labels. Considering slack variable ξ_i and regular term $\alpha \|\mathbf{w}\|$, the gradient of the objective function is $grad = \alpha \cdot \mathbf{w} - C(\mathbf{y}_i(\mathbf{x}_i \cdot \mathbf{w}^T) < 1)(\mathbf{y}_i\mathbf{x}_i)$. The gradient descent method of SVM uses the following algorithm to optimize $\mathbf{w}$. In the algorithm, the bias b can be optimized by directly adding an all 1 column to $\mathbf{x}$.

Algorithm 1. SVM – GD

```
Input: features $\mathbf{x} \in R^{t \times d}$, labels $\mathbf{y} \in \{\pm 1\}^t$, batch-size $k$, learning rate $l$, $\alpha$, $C$, $T$
Output: parameters $\mathbf{w} \in R^d$

1: Randomly initialize $\mathbf{w}$, set $t = 0$
2: while $t < T$ do
3:     Random sample batch $B$
4:     $grad = \alpha \cdot \mathbf{w}$
5:     $grad- = \frac{C}{k} \sum_{i \in B} (\mathbf{y}_i(\mathbf{x}_i \cdot \mathbf{w}^T) < 1)(\mathbf{y}_i\mathbf{x}_i)$
6:     $\mathbf{w}- = l \cdot grad$
7:     $t+ = 1$
8: return $w$
```

2.2 Shamir's Secret Sharing

The secret sharing decomposes a secret into several shares. It is required that no information about the secret can be extracted from the sets of shares that do not meet specific requirements, while the sets that do can restore the secret. Shamir's secret sharing [18] is a classic (n, t)-threshold secret sharing, that is, it decomposes a secret value into n shares, and only when the number of shares is greater than t can it be recovered the secret value. In Shamir's secret sharing scheme, the secret owner generates a $t-1$ degree polynomial $f(x) = s + a_1 x + \cdots + a_{t-1}x^{t-1}$ over a finite field, where $s = f(0)$ is set as the secret value and the other parameters are random values. Then $(f(i), i), i \in [1, n]$ are distributed to party i as the secret shares. When parties need to reconstruct the secret, first collect enough secret shares (at least t), and then calculate the secret value $f(0) = \sum_{i \in A, |A| = t}(f(i) \prod_{j \in A, j \neq i} \frac{-j}{i-j})$ according to the Lagrange interpolation formula. We use $[\![x]\!]$ to denote the Shamir's secret shares of x, and $x \leftarrow \mathsf{Reveal}([\![x]\!])$ is the reconstruct protocol as described above.

Like some other secret sharing schemes, Shamir's secret-sharing also has the homomorphic property. The secret value can be calculated by calculating the shared value. When the shares corresponding to the two polynomials f_1, f_2 are added accordingly, the result is exactly the share corresponding to $f_3 = f_1 + f_2$. Therefore, Shamir's secret sharing has the property of additive homomorphism. When we want to calculate the addition of the secret values of two corresponding polynomials on the same finite field, we can simply add their shared values of the same independent variable without any communication. Similarly, Shamir's secret sharing has similar properties to multiplication, but it should be noted that the multiplication of two $t-1$ degree polynomials will produce a $2(t-1)$ degree polynomial, which will change the threshold structure, so the degree of the new polynomial needs to be reduced in time, which brings additional communication.

3 Security Definition for Privacy-Preserving SVM

In this section, we specify the security definition for privacy-preserving computing Algorithm 1. Assume there is a group of clients $\mathcal{C}_1, \cdots, \mathcal{C}_m$, which want to train a SVM model. Each client holds some data. Assume there are n servers $\mathcal{S}_1, \cdots, \mathcal{S}_n$ that provide secure computation services. Assume there is a semi-honest adversary $\mathcal{A}$ that can corrupt $t-1$ servers and any proper subset of clients. Very informally, we say a protocol involving these roles secure, if the semi-honest adversary above cannot obtain more knowledge than what can be retrieved from outputs of the protocol.

Let $\mathbf{x}$ be the feature data of the samples, which is arbitrarily divided into m pieces $\mathbf{x}_1, \cdots, \mathbf{x}_m$ and held by m clients (i.e. data holders). Let $\mathbf{y}$ be the labels of the samples, and its partition also does not affect training. Let $\mathcal{S}_\mathcal{A}$ denote the set of corrupted servers, $\mathcal{C}_\mathcal{A}$ the set of corrupted clients. The view of the adversary, denoted $\mathsf{view}_\mathcal{A}(\mathbf{x}_\mathcal{A})$, is defined as $(\mathbf{x}_\mathcal{A}, r_\mathcal{A}, M, \mathsf{output}_\mathcal{A})$, where $\mathbf{x}_\mathcal{A}$ are the input held by the clients in $\mathcal{C}_\mathcal{A}$, $r_\mathcal{A}$ are the random numbers used by the servers in $\mathcal{S}_\mathcal{A}$ in the protocols, M is the set of messages sent by honest participants in protocols, and $\mathsf{output}_\mathcal{A}$ is the output of the protocol that $\mathcal{A}$ can get. We hope that our protocol can realize the function of Algorithm 1, denoted by f, which takes the sample data $\mathbf{x}_1, \cdots, \mathbf{x}_m$ of m clients and the labels $\mathbf{y}$ as inputs, and outputs the optimized model parameters $\mathbf{w}$. Let $f_\mathcal{A}$ denote the subset of f's output that can be obtained by $\mathcal{A}$. We have security definition as follows.

Definition 1. *Let $f : (\mathbf{x}_1, \cdots, \mathbf{x}_m, \mathbf{y}) \rightarrow \mathbf{w}$ be the ideal function of Algorithm 1 where $\mathbf{x}_1, \cdots, \mathbf{x}_m$ denote the sample data of m clients, $\mathbf{y}$ denotes the labels of the samples, $\mathbf{w}$ denote the optimized parameters. We use $f_\mathcal{A}$ to denote the subset of f's output that $\mathcal{A}$ can get. We say a protocol π privately–computes f against the semi-honest adversary $\mathcal{A}$ described above, if for any $(\mathbf{x}_1, \cdots, \mathbf{x}_m, \mathbf{y})$, the output of the protocol π $\mathsf{output}^\pi(\mathbf{x}_1, \cdots, \mathbf{x}_m, \mathbf{y}) = f(\mathbf{x}_1, \cdots, \mathbf{x}_m, \mathbf{y})$, and there exists a probabilistic poly-nomial-time algorithm S:*

$$\{S(x_\mathcal{A}, f_\mathcal{A}(\mathbf{x}, \mathbf{y})), f(\mathbf{x}, \mathbf{y})\} \equiv \{\mathsf{view}_\mathcal{A}(\mathbf{x}_\mathcal{A}), \mathsf{output}^\pi(\mathbf{x}_1, \cdots, \mathbf{x}_m, \mathbf{y})\}$$

where $\equiv$ denotes that the distributions on both sides are statistical indistinguishable.

4 Building Blocks of Our Protocol

As sketched previously, our protocol consists of many repetitions, each of which computes an iteration of the SGD method. In each repetition, the servers have as input the shares of current values of the parameters and then perform some MPC protocol to compute gradient iteration and finally obtain the shares of the new values of the parameters. According to Formula 1 and 2, each iteration of SGD only consists of addition, multiplication and comparison operations. Thus to realize the MPC for SGD, it suffices to show how to realize these operations privately.

Notice that the SGD algorithm is not over integers, while Shamir's secret sharing is built over finite fields like $\mathbb{Z}_p$. Recall that [2] and [3] presents secure fixed-point addition, multiplication, truncation and less-then-zero (LTZ) protocol with respect to Shamir's scheme. Thus we adopt [2] and [3] to realize the secure addition and multiplication, which details are recalled in Sect. 4.1. Lastly in Sect. 4.2 we show the details of how to use the Berlekamp-Welch algorithm to correct wrong messages.

4.1 Secure Fixed-Point Calculation [2] and [3]

We now present a detailed overview of fixed-point calculations in [2] and [3], which also explains the notations to make it easier to read the following subsections. We will first introduce the representation of fixed-point numbers in Shamir's sharing and then recall the addition and multiplication with truncation, and finally introduce the less-than-zero protocol. More details of the protocols can be referred to [2] and [3] or Appendix B.

Data Type and Encoding. In this paper the target data is signed fixed-point numbers, denoted as $\mathbb{Q}_{\langle k,f\rangle} = \{\tilde{x} \in \mathbb{Q} | \tilde{x} = \bar{x} \cdot 2^{-f}, \bar{x} \in \mathbb{Z}_{\langle k\rangle}\}$, where $\mathbb{Z}_{\langle k\rangle} = \{\bar{x} \in \mathbb{Z} | -2^{k-1} \leq \bar{x} \leq 2^{k-1}-1\}$ denotes the signed integers. f is the number of decimal places, k is the number of significant digits, and $\tilde{x}$ and $\bar{x}$ indicate their types $\mathbb{Q}_{\langle k,f\rangle}$ and $\mathbb{Z}_{\langle k\rangle}$. We use the integer function $\mathsf{int}_f : \mathbb{Q}_{\langle k,f\rangle} \mapsto \mathbb{Z}_{\langle k\rangle}, \mathsf{int}_f(\tilde{x}) = \tilde{x} \cdot 2^f$ to realize the conversion of elements between these two types. We use the p's complement encoding system to encode elements on $\mathbb{Z}_{\langle k\rangle}$ onto $\mathbb{Z}_p$.

The p's complement encoding system uses a sufficiently large p to generate $\mathbb{Z}_p$, where $p > 2^{2k+\kappa}$, κ is the security parameter, and uses the function $\mathsf{fld}(\bar{x}) = \bar{x} \pmod{p}$ mapping the element $\bar{x}$ over $\mathbb{Z}_{\langle k\rangle}$ to the element over the finite field $\mathbb{Z}_p$. This mapping allows addition and multiplication of elements on $\mathbb{Z}_{\langle k\rangle}$ to be directly implemented through the corresponding calculations on $\mathbb{Z}_p$, and realizing of the calculation of fixed-point numbers by simpler conversion. In addition, choosing a large enough p can also ensure that the signed multiplication does not cross the bounds, and retains many related properties.

Fixed-Point Calculation. As we have already said in the previous paragraphs, we realize the calculation of the signed integer elements over $\mathbb{Z}_{\langle k\rangle}$ by directly calculating the elements over $\mathbb{Z}_p$, and further realizing the calculation of the signed fixed-point numbers over $\mathbb{Q}_{\langle k,f\rangle}$. Since f, the number of decimal places we set, is fixed and public, we can directly use the calculation over $\mathbb{Z}_{\langle k\rangle}$ of the signed integer element $\bar{x}$ to implement the addition and subtraction of $\tilde{x}$ and determine the sign, for $\mathsf{int}_f(\tilde{x}_1) + \mathsf{int}_f(\tilde{x}_2) = \mathsf{int}_f(\tilde{x}_1 + \tilde{x}_2)$. But the multiplication between $\tilde{x}$ will cause the expansion of digits. The key difference between fixed-point calculations and integer calculations is that in order to maintain the number of decimal places, we need to truncate the results after performing multiplication calculations. Next, we will introduce the truncation protocol in [3] that we will use in this paper.

Truncation. Div2mP [3] is the truncation protocol that we will use in this paper. It takes a secret integer value $\bar{a} \in \mathbb{Z}_{\langle k\rangle}$ and a public integer $m \in [1, k-1]$ as inputs, calculates the shares of $\bar{c} = \lfloor \bar{a}/2^m \rceil$ and rounds up or down with some probability. Details of the protocol are shown as Protocol 5.

Using the above truncation protocol, we get the multiplication of fixed-point numbers. As for $\tilde{x}_3 = \tilde{x}_1\tilde{x}_2 = \bar{x}_1\bar{x}_2 \cdot 2^{-2f} \in \mathbb{Q}_{\langle 2k,2f\rangle}$, using $\mathsf{Div2mP}([\![\tilde{x}_3]\!], k+f, f)$ to do the truncation, $\tilde{x}_3$ will be turned to $\tilde{x'}_3 = \bar{x}_1\bar{x}_2 \cdot 2^{-f} \in \mathbb{Q}_{\langle k,f\rangle}$. And this is how FXMul works.

The Less-Than-Zero Protocol. In this subsection we introduce the LTZ protocol from [2] based on bit comparison and precise truncation Div2m for secure comparison. According to Formula 1, we need to decide whether a number is greater than 0. The LTZ protocol obtains the sign of the secret value by truncating to only one bit remaining. Because Shamir's secret sharing works on $\mathbb{Z}_p$, we will map signed integers to $\mathbb{Z}_p$ by modulo p, and positive numbers will be mapped to $[0, p/2]$, while negative numbers will be mapped to $(p/2, p)$. Therefore, when guaranteed to be rounded down, the truncated result of a positive number is 0, and the result of a negative number is -1 (i.e. $p-1$ in $\mathbb{Z}_p$), so that the two can be distinguished. $\mathsf{LTZ}([\![a]\!], k)$ outputs $s = (\bar{a} < 0)?1 : 0$ as $[\![s]\!] = -\mathsf{Div2m}([\![a]\!], k, k-1)$.

4.2 Error-Message Recovery via the Berlekamp-Welch Algorithm

Now we show the details of how to use the Berlekamp-Welch algorithm to correct wrong messages. Assume that there are wrong messages in communication and these error messages are corrupted shares. Notice that all the communications in our secure computation framework are in Reveal (see Sect. 2.2), except the initial share distribution of clients, and all situations only include the Reveal of random values or the Reveal hidden in the degree reduction protocol. Moreover, error correction algorithms such as Berlekamp–Welch algorithm can fully assume the role of recovering secret values from shares in the Reveal protocol, and have the ability of error correction (see Appendix A.1). Therefore, in our secure computation, participants can directly replace the Reveal with the Reveal_{BW} of Berlekamp–Welch algorithm version, when they reveal the secrets after each

Protocol 2. $\mathsf{Div2mP}_{BW}([\![a]\!]_p, k, m)$

Input: Secret share $[\![a]\!]_p$, digits length k, divisor length m, random number r
Output: Secret sharing modulus result $[\![c]\!]_p$, where $\bar{c} = \lfloor \bar{a}/2^m \rfloor + u$, and $u \leftarrow \{0, 1\}$

```
1: $[\![r'']\!] \leftarrow \mathsf{PRandInt}(k - m - 1), [\![r']\!] \leftarrow \mathsf{PRandInt}(m)$
2: $[\![r]\!] \leftarrow 2^m [\![r'']\!] + [\![r']\!]$
3: $[\![a']\!] \leftarrow 2^{k-1} + [\![a]\!] + [\![r]\!]$
4: $b \leftarrow \mathsf{Reveal}_{BW}(r[\![a']\!])$
5: $b' \leftarrow (r^{-1}b) \pmod{2^m}$
6: $[\![c]\!] \leftarrow ([\![a]\!] - (b' - [\![r']\!]))2^{-m}$
7: return $[\![c]\!]$
```

round of communication. In order to ensure correctness, we also need some other operations.

Since the Berlekamp-Welch algorithm is proposed as a decoding algorithm, the errors it deals with are considered as random noise. Similarly, there are some heuristic algorithms among the following multi-polynomial reconstruction algorithm, which clearly requires that the errors should be random. In secure computing, error messages may be actively tampered with by malicious adversaries to prevent the reconstruction of secrets or even lead to the wrong results. Such errors are obviously not random. To use these algorithms in our secure computation, we need to randomize the errors in different distributions.

Let the received shares be $[\![x]\!]$, some of which are corrupted to $[\![x]\!]_i + \Delta x_i$, where Δx_i is the malicious error. The participant can multiply all shares $[\![x]\!]$ by a same random number r, i.e. $[\![x']\!] = r[\![x]\!]$. So the error Δx_i is turned to $r\Delta x_i$, which is a uniformly distributed random number. Then the participant can apply Berlekamp-Welch algorithm to recover the secret $x' = rx \leftarrow \mathsf{Reveal}_{BW}([\![x']\!])$. In the end he multiplies the result x' by the inverse of the random number r^{-1} to obtain the secret value $x \leftarrow r^{-1}x'$. Taking Div2mP for example, (see Protocol 5 in Appendix B.1) after introducing BW algorithm, the protocol is modified as Protocol 2.

Finally, we summarize the error correction algorithm. We use noisy polynomial reconstruction algorithms such as Berlekamp-Welch to replace the original reveal algorithm in Shamir's secret sharing. Since there is no change in the interaction, the introduced algorithm will not reduce security. On the contrary, due to its error correction function, it can resist the dropping out and tampering of some messages, thereby obtaining stronger security than before. As these correction algorithms increase the computational cost compared to the original Reveal, it is sufficient to use the original Reveal under semi-honest security, so as in our experiments.

5 Privacy-Preserving SVM

After presenting the building blocks, this section will formally introduce our privacy-preserving SVM protocol. Section 5.1 will give an overview of the protocol. Section 5.2 will explain the details of the protocol and prove its security.

Protocol 3. $\mathsf{PPSVM-GD}$

Input: Features $\mathbf{x} \in R^{t\times d}$, labels $\mathbf{y} \in \{\pm 1\}^t$, batch-size bs, learning rate r, PRG, α, k, f, C, T

Output: shares $[\![\mathbf{w}]\!]$ of parameters $\mathbf{w} \in R^d$

```
 1: Clients generate the shares $[\![\mathbf{x}]\!]$, $[\![\mathbf{y}]\!]$, and send them to the servers
 2: Servers randomly initialize $[\![\mathbf{w}]\!]$, set $t = 0$
 3: calculate $\lambda = [\frac{\mathsf{C}\cdot\mathsf{r}}{\mathsf{bs}} \times 2^{\mathsf{f}}]$
 4: while $t < T$ do
 5:     Get batch index according to PRG
 6:     $grad = \alpha \cdot [\![\mathbf{w}]\!]$
 7:     $a_1 = [\![\mathbf{y}_i]\!][\![\mathbf{x}_i]\!]$     //need degree reduction
 8:     $a_2 = a_1 \cdot [\![\mathbf{w}]\!]$     //need degree reduction
 9:     $a_3 = \mathsf{LTZ}(a_2 - 2^f)$
10:     $a_4 = \lambda \sum_i a_3 a_1$     //need degree reduction
11:     $grad = \mathsf{Div2mP}(grad - a_4, k+f, f)$
12:     $[\![\mathbf{w}]\!] -= \mathsf{Div2mP}(r \cdot grad, k+f, f)$
13:     $t += 1$
14: Servers return $[\![\mathbf{w}]\!]$
```

5.1 Protocol Overview

We use the above protocols based on Shamir's secret sharing to realize the secure training of SVM. In the training protocol, the clients submit the shares of the sample data $[\![\mathbf{x}]\!]$ and labels $[\![\mathbf{y}]\!]$ to the servers. The servers use them to calculate the homomorphism of gradient descent according to Formula (1) and (2), so as to optimize the model parameters $\mathbf{w}$ in privacy, and finally output the shares of the optimized model parameters $[\![\mathbf{w}]\!]$.

Our secure training has no special requirements for the distribution of data, whether it is horizontal or vertical. As long as the data from different clients can form a complete training set, and this combination is public, then the servers can do the same operation to the shares of these data. However, in order to normalize the data before training, the horizontal distributed data may need other additional operations to get the maximum value per column, while the vertical distributed data can be calculated directly and locally.

5.2 Protocol Details

The following is the specific process of the protocol. Before the formal training, the data holders (the clients) and the calculators (the servers) need to agree on the number of fixed-point decimal places f, as well as the parameters related to the secure protocols, such as the modulus p of the finite field. The data holders owning vertical partitioned data need to align the data. During the training, the data holders submit the secret shares to the calculators. After coordination and sorting, the calculators hold the shares of the same and complete sample data matrix. The calculators use the shares to perform the secure computation of the iterations in the SGD method according to Formula (1) and (2), as shown in the Protocol 3, and finally get the shares of the optimized parameter $\mathbf{w}$. The shares can be given to the data holders to reveal $\mathbf{w}$, or can be left to the calculators for secure classification.

While compared with the other secure machine learning protocols like SecureML, our protocol works in the scenario of n servers and m clients, while theirs need two to four fixed servers. And because of the (n, t) threshold property of Shamir's scheme, our SGD protocol can tolerate dropping out of at most $n - 2t$ servers.

Security: The protocol can maintain privacy when facing semi-honest adversaries that corrupt at most $t - 1$ servers. Therefore, we have the ability to deal with the collusion of up to $t - 1$ servers. We believe that the protocol will not leakage any additional information except normal output. Since our protocol is implemented by the secure computing framework from [3] and [2], the security of the protocol can also be reduced to their security. More specifically, we make the following claim and proof.

Theorem 1. *Protocol 3 privately computes SVM training with respect to Definition 1.*

Proof. Our model should be able to deal with such an adversary $\mathcal{A}$: it can corrupt at most $t - 1$ out of the total number of n servers and a subset of clients, and executes the protocol semi-honestly. We believe that it cannot obtain any information other than its own input and output. We set the scenario where $\mathcal{A}$ corrupts $t-1$ servers $\mathcal{S}_1, \mathcal{S}_2, \cdots \mathcal{S}_{t-1}$ and $m-1$ clients $\mathcal{C}_1, \mathcal{C}_2, \cdots \mathcal{C}_{m-1}$. The above two sets of servers and clients are denoted by $\mathcal{S}_\mathcal{A}$ and $\mathcal{C}_\mathcal{A}$.

Next we start to construct a simulator $\mathcal{S}$ that runs algorithm S in Definition 1. The input of $\mathcal{S}$ is the sample data of $\mathcal{C}_\mathcal{A}$ and the output of both $\mathcal{C}_\mathcal{A}$ and $\mathcal{S}_\mathcal{A}$, which is denoted by $x_\mathcal{A}$ and $f_\mathcal{A}$.

Now we analyze the messages $\mathcal{A}$ gets in its view. Since all communications take place in Reveal, all the messages $\mathcal{A}$ receive are shares. All the results of Reveals contain two types of secret values. One is the random elements over the finite field, which $\mathcal{S}$ can directly simulate directly with random elements over the field, while the other is the random value generated by additive hiding. Given a shared variable $[\![x]\!]$ and an unknown shared random secret value $[\![r]\!]$ jointly generated by participants, calculate $[\![y]\!] = [\![x]\!] + [\![r]\!] \mod p$ and reveal $y = x + r \mod p$. For $x \in [0, 2^k - 1]$, $r \in [0, 2^{k+\kappa} - 1]$, $p > 2^{k+\kappa+1}$, the statistical distance between y and r $\Delta(y, r) = \frac{1}{2}\sum_{v \in [0, 2^{k+\kappa}+2^k-1]} |Pr(x = v) - Pr(r = v)| < 2^{-\kappa}$, leading to statistical privacy with security parameter κ. Therefore, as long as we ensure that the bit length involved in addition hiding in the protocol is κ bits longer than the actual digital range, we can also maintain the above statistical indistinguishability. Thus, the simulator can sample random number shares of the same bit length for simulation.

These two are statistically indistinguishable due to the security of Shamir's secret sharing that a group of less than t servers cannot obtain any information about the secret. Finally, also from the security of Shamir's secret sharing for homomorphic computation, the view of $\mathcal{A}$ and the overall output of f are also independent of each other. Therefore, we construct such $\mathcal{S}$, which outputs the same number of random field elements corresponding to the view of $\mathcal{A}$, so that

$$\{S(x_{\mathcal{A}}, f_{\mathcal{A}}(\mathbf{x}, \mathbf{y})), f(\mathbf{x}, \mathbf{y})\} \equiv \{\mathsf{view}_{\mathcal{A}}(\mathbf{x}_{\mathcal{A}}), \mathsf{output}^{\pi}(\mathbf{x}_1, \cdots, \mathbf{x}_m, \mathbf{y})\}$$

In summary, Protocol 3 privately computes SVM training for Definition 1.

6 Evaluation

In this section, we will evaluate the proposed privacy preserving support vector machine protocol. We will evaluate the efficiency and accuracy through theoretical analysis and experimental verification. Finally, we will compare our results with the existing works.

6.1 Theoretical Analysis

We first analyze the theoretical communication complexity of PPSVM. In each iteration, the participants perform $\mathsf{Div2mP}$ protocol twice, degree reduction protocol twice, and LTZ protocol once. They need 2 rounds, 3 rounds, 3 rounds of communication respectively, for a total of 8 rounds of communication. In these communications, they need to make $k+6$ calls of Reveal, where the revealed original matrix size is one of $s \times d$ and $k+2$ of $s \times 1$ size, 3 of $1 \times d$, so the total communication volume of each participant in one round of SVM training is $(s \times d + s \times (k+2) + d \times 3)(n-1)\log p$ bits. In the above, n represents the number of servers, s is the number of samples, d is the feature dimension of samples, and k represents the bit length of secrets.

Then we discuss the computational complexity. Each calculator's calculation includes four element-wise multiplications, one matrix multiplication and one comparison in each round of the original SVM training algorithm, and their extra computational complexity comes from the security protocols. Each degree reduction protocol needs at least 2 matrix multiplications, and for the truncation protocol, 1 additional matrix multiplication and 2 element-wise multiplications, while the LTZ protocol requires $k-1$ matrix multiplications and $8k-2$ element-wise multiplications. Therefore, each cycle introduces an additional $k+7$ matrix multiplications and $8k+4$ element-wise multiplications.

6.2 Experimental Analysis

Because the calculation of long integer matrix is involved, we use Python to program, and simulate the scene of secure multi-party SVM training on a single machine. The communication related time will not be included, which can be calculated by the previous section. We use native Python for programming, so the running speed of the experiment will be lower than the theoretical value, but our results are still much faster than the results of [9,20]. Another significance of the experiment is to verify that the errors brought by fixed-point number in SVM training can be ignored.

Settings. The parameters we use are as follows: the number of calculators $n = 3$, the threshold $t = 2$, the number of fixed-point digits $f = 20$, and the finite field prime p's length $\lceil \log p \rceil = 120$. Other parameters such as C are selected by grid search method. Before the experiment, the sample feature data is expanded and rounded according f, and then they are secret shared, which are used as inputs.

Table 1. Data set details

Dataset	Feature	Trainset	Testset
Breast-cancer	10	500	183
Diabetes	8	500	268
German.number	24	800	200

Dataset. The data sets used in the experiment are three binary-class data sets from libsvm: breast cancer, german.number and diabetes. All three datasets have linearly scaled each attribute to $[-1, 1]$ or $[0, 1]$. See Table 1 for details.

Result. We test the secure training protocol on the above three data sets and compare it with gradient descent training using plaintext directly. We repeat each experiment 10 times and take the average of 10 results as the final result. The results are shown in the Table 2.

Table 2. Accuracy of normal SVM and PPSVM among datasets

Dataset	T	SVM	PPSVM
Breast-cancer	100	98.56% (0.0210 s)	98.98% (47.98 s)
Diabetes	1000	68.28% (0.0822 s)	68.06% (677.78 s)
German.number	1000	69.90% (0.2588 s)	68.40% (1846.67 s)

Due to the different separability of the three data sets, we use different iteration numbers. Obviously, the total training time is directly proportional to the number of iterations, whether secure computation is used or not. At the same time, the rise of feature dimension and training set size will also increase the calculation time. Comparing the time of two models on the same data set, the training time of PPSVM is about 2000–8000 times that of ordinary SVM. Considering that we use a single machine to simulate three parties, this proportion needs to be reduced to about 1/3. However, this is still quite different from the previous theoretical analysis. We believe that the magnification outside the theory comes from the native Python language and numpy library. In the secure protocol, we use the native data type of Python to increase the number of data bits, which will greatly reduce the efficiency of numpy library.

For accuracy, the results of the two models on the breast-cancer dataset are the best, where the average accuracy of ordinary SVM and privacy-preserving SVM is 98.56% and 98.98%. The results on german.number and diabetes are not ideal, which are about 68%. This may be because the kernel function type or penalty coefficient is not appropriate. It can be seen that the average accuracy difference between PPSVM and ordinary SVM on the same data set is less than 1%. Considering the randomness of gradient descent algorithm, we believe that there is no significant difference between the results whether using secure calculation or not. That is, when the number of fixed-point digits is sufficient, the calculation using the number of fixed-point digits will not affect the optimization result of gradient descent.

Table 3. Comparison among normal SVM and PPSVMs of different thresholds on breast-cancer dataset

	Time/s	Single server time/s	Accuracy
SVM	0.0109		96.89%
(3, 2)	81.30	27.10	96.61%
(5, 3)	127.95	25.59	96.39%
(7, 4)	195.13	27.88	96.34%

We also conduct experiments on different server numbers and secret sharing thresholds to explore their impact on the efficiency of the protocol. Our experiment is a single machine simulation, so it does not include communication, but only the total computing cost. We use three different thresholds for experiments, each of which conduct 10 experiments and average their results, and further divide the average results by the number of simulated servers to obtain the approximate computing time of a single server. The results are shown in Table 3. The time of a single server on the three thresholds is 27.10 s, 25.59 s and 27.88 s, which are almost the same. This is consistent with our results in theoretical analysis, that is, the number of servers only affects a small number of matrix shapes in the matrix calculations, and has little impact on the calculation cost. When the threshold and the number of servers are small, their growth has almost no impact on the calculation. But on the other hand, according to the theoretical analysis above, the communication cost is linearly related to the threshold and the number of servers, so it is more affected.

6.3 Comparison

To conclude this section, we will compare our protocol with some previous works. Our work is closer to the methods of [9,20]. Compared with them, our protocol achieves higher efficiency. They use a public key encryption system based on modular exponentiation, so their 100 rounds of training take nearly 10 h on a 236×13 training set. Our protocol runs MPC based on shares, which only contains addition and multiplication over the finite field. Therefore, our 100 rounds of training takes about 48s on the breast-cancer training set of 500×10, our 1000 rounds of training takes about 10 min on the diabetes training set of 500×8, and 30 min on the german.number training set of 800×24. For efficiency, our protocol is much faster than theirs. They need two semi-honest servers, while we need not less than three servers.

Compared with [1,13,14] as well as [9,20], which have to fix the number of servers to a value among 2 to 4, our protocol can be deployed among any number of servers (no less than 3). Increased number of servers and flexible deployment also enhance the difficulty of server collusion and enhanced the security and generality of our protocol (for example, let some clients act as the calculators). Also, by using Shamir's (n, t)-threshold secret sharing scheme, we can arbitrarily deploy among n servers and tolerate dropping out of at most $n - 2t$ ones. Moreover, by introducing the Berlekamp-Welch algorithm as an optional recovery algorithm, correct messages can be recovered even if less than t messages/shares are corrupted (Table 4).

Table 4. Comparison in functionality

Functions	[9,20]	SecureML	Ours
Efficiency	Low	High	Medium
Non fixed number of servers	✗	✗	✓
Error correction	✗	✗	✓

7 Conclusion

We propose a new privacy preserving support vector machine protocol, which enables no less than three servers to help several data holders train SVM models, where the data distribution can be can be arbitrary. We introduce Shamir's secret sharing scheme to perform secure computation and protect privacy. We verify the feasibility and effectiveness of the scheme through experiments.

Acknowledgements. This work was supported in part by the National Key Research and Development Project 2020YFA0712300.

A Preliminaries

A.1 Error-Correcting Codes and Berlekamp-Welch Algorithm

Reed Solomon code is an error correction code, which can deal with damaged and lost symbols. Like Shamir's secret sharing, RS code is based on polynomial interpolation, that is, the codewords $\{f(x_1), f(x_2), \cdots, f(x_n)\}$ can be generated by polynomial $f(x) = s + a_1x + \cdots + a_{t-1}x^{t-1}$ from source of $\{s, a_1, \cdots, a_{t-1}\}$, where n is the number of participants, t is the threshold. $\{s, a_1, \cdots, a_{t-1}\}$ is the input message for RS coding, also the secret and randomness for Shamir's secret sharing.

Berlekamp-Welch algorithm [10] is a decoding algorithm of RS code. The algorithm takes the received codewords (the share in Shamir's secret sharing) as input, and recovers the correct true values from by solving a system of equations and dividing between polynomials. It can deal with up to $v < (n - t + 1)/2$ errors in the received codewords. The principle of Berlekamp-Welch algorithm is based on error location polynomial. The error location polynomial is $E(x) = \prod_{i \in \mathbb{E}}(x - i) = e_0 + e_1x + \cdots + e_{k-1}x^{k-1} + x^k$, where $\mathbb{E}$ represents the index set of error messages that need to be found. The received codewords are $S_1, S_2, \cdots, S_n$. Note that when $f(x) \neq S_x, E(x) = 0$, so there is the equation $f(x)E(x) = S_xE(x)$. Let the left side of the equation be $Q(x)$, and we get the equation system $\{Q(x) = S_xE(x)\}_{x=1}^{n}$. As long as $2k + t + 1 \leq n$ is satisfied, there are solutions of $Q(x)$ and $E(x)$. After the two polynomials are obtained by solving the linear equations, we can get $f(x)$ by calculating $Q(x)/E(x)$. In Shamir's secret sharing, Berlekamp–Welch algorithm has the same input and output as the Reveal function, and also has the ability of error correction, so it can directly replace the Reveal function.

Protocol 4. Reveal_{BW}

Input: codewords(shares) $S_1, S_2, \cdots, S_n \in \mathbb{Z}_p$
Output: secrets $s \in \mathbb{Z}_p$

```
1: Each participant gets $S_1, S_2, \cdots, S_n$ from others
2: Determine the number of items of $Q(x)$ and $E(x)$ according to the assumed number
   of error messages
3: Solve the equation system $\{Q(x) = S_xE(x)\}_{x=1}^{n}$
4: $f(x) = Q(x)/E(x)$
5: return $s$ in $f(x)$
```

Berlekamp-Welch algorithm can only recover one secret in one calculation, while some other further algorithms [4,6] can recover multiple polynomials at the same time in one calculation. This problem is also called noisy multi-polynomial reconstruction. These different algorithms have the same application in our framework. So in our framework, we only take Berlekamp-Welch algorithm as the representative.

B Details of Protocols in [3] and [2]

Protocol 5. $\mathsf{Div2mP}([\![a]\!]_p, k, m)$ [3]

Input: Secret share $[\![a]\!]_p$, digits length k, divisor length m, security parameter κ
Output: Secret sharing modulus result $[\![c]\!]_p$, where $\bar{c} = \lfloor \bar{a}/2^m \rfloor + u$, and $u \leftarrow \{0, 1\}$

```
1: $[\![r'']\!] \leftarrow \mathsf{PRandInt}(k + \kappa), [\![r']\!] \leftarrow \mathsf{PRandInt}(m)$
2: $[\![r]\!] \leftarrow 2^m [\![r'']\!] + [\![r']\!]$
3: $b \leftarrow \mathsf{Reveal}(2^{k+\kappa-1} + [\![a]\!] + [\![r]\!])$
4: $b' \leftarrow b \pmod{2^m}$
5: $[\![c]\!] \leftarrow ([\![a]\!] - (b' - [\![r']\!]))2^{-m}$
6: return $[\![c]\!]$
```

B.1 Truncation

Div2mP [3] is the truncation protocol that we will use in this paper. It takes a secret integer value $\bar{a} \in \mathbb{Z}_{\langle k \rangle}$ and a public integer $m \in [1, k-1]$ as inputs, calculates $\bar{a}/2^m$ and rounds up or down with some probability. Details of the protocol are shown as Protocol 5.

The protocol uses a truncated random number r to mask the secret value a to the garbled value b, reveals the garbled value b for truncation, and then removes the truncated result r' of r from the result b'. The actual output of the protocol is $\bar{c} = \lfloor \bar{a}/2^m \rfloor + u$, which contain an error $u = (b' < r')?1 : 0$. This error is acceptable in the truncation of fixed-point numbers. The PRandInt in the protocol is used by each participant to generate a share of an unknown random number of a specified length without communication.

B.2 Fixed-Point Multiplication

Using the above truncation protocol, we get the multiplication of fixed-point numbers. As for $\tilde{x}_3 = \tilde{x}_1 \tilde{x}_2 = \bar{x}_1 \bar{x}_2 \cdot 2^{-2f} \in \mathbb{Q}_{\langle 2k, 2f \rangle}$, using $\mathsf{Div2mP}([\![\tilde{x}_3]\!], k+f, f)$ to do the truncation, $\tilde{x}_3$ will be turned to $\tilde{x'}_3 = \bar{x}_1 \bar{x}_2 \cdot 2^{-f} \in \mathbb{Q}_{\langle k, f \rangle}$. And this is how FXMul works.

Protocol 6. $\mathsf{FXMul}([\![a_1]\!], [\![a_2]\!], k + f, f)$ [3]

Input: Secret share $[\![a_1]\!], [\![a_2]\!]$, digits length k, decimal digits length f
Output: Secret share $[\![a_3]\!]_p$, where $\bar{a}_3 = \bar{a}_1 \cdot \bar{a}_2$

```
1: $[\![a]\!] \leftarrow [\![a_1]\!] \cdot [\![a_2]\!]$
2: $[\![a_3]\!] \leftarrow \mathsf{Div2mP}([\![a]\!], k + f, f)$
3: return $[\![a_3]\!]$
```

The communication required for each multiplication is a large overhead, and the reason for communication is that the multiplication expands the degree of polynomials in Shamir's secret sharing. In order to ensure subsequent successful recovery, the number of polynomials needs to be maintained less than n through computation.

B.3 Batch Calculation

As mentioned earlier, the communication required for the degree reduction of each multiplication is a large overhead. Observing our goal, when calculating SGD, we will first use multiplication and addition to calculate the inner product. Each time the inner product is calculated, the multiplications do not interfere with each other, so we can communicate and reduce the degree after the complete inner product.

B.4 The Less-Than-Zero Protocol

The LTZ protocol in [2] is actually an application of the precise truncation protocol Div2m. The above Div2mP is a truncation protocol, but it has errors caused by random rounding. First, the protocol uses $\mathsf{PRandM}(k, m)$ to generate two shares of random numbers with specified lengths k and m, and the shares of each bit of the latter. Based on the Div2mP protocol, Div2m uses the bit comparison protocol BitLT from [2] to obtain an accurate result of truncating 2^{k-1} bits and keep rounding down, thereby revealing whether the secret is less than zero. BitLT takes a plaintext data and a set of random bit shares as input, and outputs whether this plaintext data is less than the binary random number represented by these bit shares. The BitLT requires 2 rounds and $k+1$ interactions of Reveal online, 3 rounds and $3k-1$ interactions offline.

Using BitLT, Div2m can find out whether $2^{k-1} + [\![a]\!] + [\![r]\!]$ produces carry in the least significant m bits and remove it. So that Div2mP is turned to accurate Div2m. And finally, we have $\mathsf{LTZ}([\![a]\!], k)$ outputs $s = (\bar{a} < 0)?1 : 0$ as $[\![s]\!] = -\mathsf{Div2m}([\![a]\!], k, k-1)$.

Protocol 7. $\mathsf{Div2m}([\![a]\!]_p, k, m)$ [2]

Input: Secret share $[\![a]\!]_p$, digits length k, divisor length m, security parameter κ
Output: Secret sharing modulus result $[\![c]\!]_p$, where $\bar{c} = \lfloor \bar{a}/2^m \rfloor + u$, and $u \leftarrow \{0, 1\}$

```
1: $([\![r'']\!], [\![r']\!], \{[\![r'_i]\!]\}_{i=1}^{m}) \leftarrow \mathsf{PRandM}(k + \kappa, m)$
2: $[\![r]\!] \leftarrow 2^m [\![r'']\!] + [\![r']\!]$
3: $b \leftarrow \mathsf{Reveal}(2^{k+\kappa-1} + [\![a]\!] + [\![r]\!])$
4: $b' \leftarrow b \pmod{2^m}$
5: $[\![u]\!] \leftarrow \mathsf{BitLT}(b', \{[\![r'_i]\!]\}_{i=1}^{m})$
6: $[\![c]\!] \leftarrow ([\![a]\!] - (b' - [\![r']\!] + 2^m [\![u]\!]))2^{-m}$
7: return $[\![c]\!]$
```

Security: Since all the massages exchanged in protocols above are Shamir's secret shares, and values masked by uniformity random numbers (also resulting in uniformity random values or with only negligible differences), according to the security of secret sharing, this protocol is secure.

References

1. Byali, M., Chaudhari, H., Patra, A., Suresh, A.: FLASH: fast and robust framework for privacy-preserving machine learning. Proc. Priv. Enhancing Technol. **2020**(2), 459–480 (2020)
2. Catrina, O.: Round-efficient protocols for secure multiparty fixed-point arithmetic. In: 2018 International Conference on Communications (COMM), pp. 431–436. IEEE (2018)
3. Catrina, O., de Hoogh, S.: Improved primitives for secure multiparty integer computation. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 182–199. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15317-4_13
4. Cohn, H., Heninger, N.: Approximate common divisors via lattices. In: The Open Book Series, vol. 1, no. 1, pp. 271–293 (2013)
5. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995). https://doi.org/10.1007/BF00994018
6. Guruswami, V., Rudra, A.: Explicit codes achieving list decoding capacity: error-correction with optimal redundancy. IEEE Trans. Inf. Theory **54**(1), 135–150 (2008)
7. Laur, S., Lipmaa, H., Mielikäinen, T.: Cryptographically private support vector machines. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 618–624 (2006)
8. Lin, Q., Pei, H., Wang, K., Zhong, P.: Privacy-preserving one-class support vector machine with vertically partitioned data. Int. J. Multimed. Ubiquit. Eng. **11**(5), 199–208 (2016)
9. Liu, X., Deng, R.H., Choo, K.K.R., Yang, Y.: Privacy-preserving outsourced support vector machine design for secure drug discovery. IEEE Trans. Cloud Comput. **8**(2), 610–622 (2018)
10. Lloyd, W., Elwyn, B.: Error correction for algebraic block codes, December 1986
11. Maekawa, T., Kawamura, A., Nakachi, T., Kiya, H.: Privacy-preserving support vector machine computing using random unitary transformation. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **102**(12), 1849–1855 (2019)
12. Mangasarian, O.L., Wild, E.W., Fung, G.M.: Privacy-preserving classification of vertically partitioned data via random kernels. ACM Trans. Knowl. Discov. Data (TKDD) **2**(3), 1–16 (2008)
13. Mohassel, P., Rindal, P.: ABY^3: a mixed protocol framework for machine learning. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, 15–19 October 2018, pp. 35–52. ACM (2018). https://doi.org/10.1145/3243734.3243760
14. Mohassel, P., Zhang, Y.: SecureML: a system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, 22–26 May 2017, pp. 19–38. IEEE Computer Society (2017). https://doi.org/10.1109/SP.2017.12
15. Omer, M.Z., Gao, H., Sayed, F.: Privacy preserving in distributed SVM data mining on vertical partitioned data. In: 2016 3rd International Conference on Soft Computing & Machine Intelligence (ISCMI), pp. 84–89. IEEE (2016)

16. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: a hybrid secure computation framework for machine learning applications. In: Kim, J., Ahn, G., Kim, S., Kim, Y., López, J., Kim, T. (eds.) Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, 04–08 June 2018, pp. 707–721. ACM (2018). https://doi.org/10.1145/3196494.3196522
17. Sakr, C.: Analytical guarantees for reduced precision fixed-point margin hyperplane classifiers (2017)
18. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
19. Vaidya, J., Yu, H., Jiang, X.: Privacy-preserving SVM classification. Knowl. Inf. Syst. **14**(2), 161–178 (2008). https://doi.org/10.1007/s10115-007-0073-7
20. Wang, J., Wu, L., Wang, H., Choo, K.K.R., He, D.: An efficient and privacy-preserving outsourced support vector machine training for internet of medical things. IEEE Internet Things J. **8**(1), 458–473 (2020)
21. Yu, H., Vaidya, J., Jiang, X.: Privacy-preserving SVM classification on vertically partitioned data. In: Ng, W.-K., Kitsuregawa, M., Li, J., Chang, K. (eds.) PAKDD 2006. LNCS (LNAI), vol. 3918, pp. 647–656. Springer, Heidelberg (2006). https://doi.org/10.1007/11731139_74
22. Zhang, J., Yiu, S.M., Jiang, Z.L.: Outsourced privacy-preserving reduced SVM among multiple institutions. In: Qiu, M. (ed.) ICA3PP 2020. LNCS, vol. 12453, pp. 126–141. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60239-0_9