Non-interactive Privacy-preserving Naïve Bayes Classifier Using Homomorphic Encryption*

Jingwei Chen[1,2], Yong Feng[1,2], Yang Liu[3](✉),
Wenyuan Wu[1,2], and Guanci Yang[4]

[1] Chongqing Key Laboratory of Automated Reasoning and Cognition, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences
[2] Chongqing College, University of Chinese Academy of Sciences, Chongqing, China
{chenjingwei, yongfeng, wuwenyuan}@cigit.ac.cn
[3] Information Science and Engineering, Chongqing Jiaotong University, China
liuyang13@cqjtu.edu.cn
[4] Key Laboratory of Advanced Manufacturing Technology of Ministry of Education, Guizhou University, Guiyang, China
guanci_yang@163.com

Abstract. In this paper, we propose a privacy-preserving naive Bayes classifier based on a leveled homomorphic encryption scheme due to Brakerski-Gentry-Vaikuntanuthan (BGV). The classifier runs on a server that is also the owner of the model, with input as BGV encrypted data from a client. The classifier produces encrypted classification results which can only be decrypted by the client, whereas the model is only accessible to the server itself. This ensures that the classifier does not leak any private information on either the model of the server or the data and results of the client. More importantly, the classifier does not require any interaction between the server and the client during the classification phase. The main technical ingredient is an algorithm to compute the index of the maximum of an encrypted array homomorphically, which does not require any interaction. The proposed classifier is implemented using a homomorphic encryption library HElib. Preliminary experiments demonstrate the efficiency and accuracy of the proposed privacy-preserving naive Bayes classifier.

Keywords: Privacy-preserving data mining · Homomorphic encryption · Naïve Bayes Classifier.

* This research was supported in part by National Key R&D Program of China (2020YFA0712303), NSFC (61903053), Guizhou Science and Technology Program [2020]4Y056, Chongqing Science and Technology Program (cstc2021jcyj-msxmX0821, cstc2019yszx-jcyjX0003, cstc2020yszx-jcyjX0005, cstc2021yszx-jcyjX0004, KJQN201900702), and Youth Innovation Promotion Association of CAS (2018419).

1 Introduction

Over the past decade, Machine Learning as a Service (MLaaS) has been involved in various fields, from academia to industry. A typical application scenario is that the model vendor uses a large amount of user data to train the model and then uses the trained model to infer/predict some results based on data supplied by clients. However, as security incidents such as data breaches continue to occur, the demand for privacy-preserving MLaaS is rapidly increasing. On the one hand, the model owner is unwilling to leak information about the model. On the other hand, the data owner is reluctant to leak information about the data as well. To resolve this contradiction, privacy-preserving machine learning is proposed.

In this paper, we consider the framework presented in [2] for privacy-preserving classifiers. As shown in Fig. 1, each shaded box indicates private data that should be accessible to only one party: the model to the server, and the data and prediction result to the client. In particular, we present a privacy-preserving naïve Bayes classifier (Protocol 1) based on a leveled homomorphic encryption (LHE) scheme BGV [4], which is based on R-LWE [20] and hence thought to be post-quantum safe, and which allows us to evaluate functions with a bounded multiplicative depth on encrypted data.

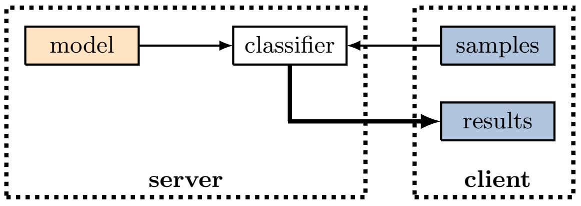

Fig. 1. Framework of privacy-preserving classifiers

In Protocol 1, the data owner (client) encrypts its data x to be predicted as a ciphertext c and sends c to the model owner (server). After receiving the ciphertext c, the server evaluates the model using the client's public key, with the input as the encrypted data c. The server sends the resulting ciphertext to the client, and the client decrypts the ciphertext using itself's secret key to specify in which class x lies. Thanks to an algorithm to compute a ciphertext of the index of the maximum of an encrypted array (Algorithm 3), no interaction between the client and the server happens during the classification phase (Step 3–6 in Protocol 1). This is one of the most important feature of our classifier.

In section 2, we give a brief introduction to homomorphic encryption and present several building blocks for our classifier, including the main technical ingredient, Algorithm 3. In section 3, we propose a privacy-preserving naïve Bayes classifier and prove its correctness and its security in the passive (or honest-but-curious [13]) model. In section 4, we report some preliminary experimental results based on our implementation of Protocol 1.

Related Work. It seems to be impossible to list all literature on privacy-preserving protocols for classifiers. We refer to [2, Sec. II] for a good survey. Here we focus mainly on those privacy-preserving naïve Bayes classifiers based on homomorphic encryption (HE).

Naïve Bayes classifiers is a simple but powerful algorithm to predict the category label of unclassified samples; see, e.g., [7]. Bost *et al.* proposed in [2, Sec. VI] the first efficient privacy-preserving protocols for naïve Bayes classifier based on the Paillier encryption scheme [21], which would be broken by quantum attackers. Li *et al.* proposed in [18] a secure naïve Bayes classifier for four parties. Later on, Kim *et al.* [17] adapted Li *et al.*'s framework using the HE scheme BGV. Yasumura *et al.* [25] and Sun *et al.* [23] also give privacy-preserving protocols for naïve Bayes classification based on BGV. All of these privacy-preserving naïve Bayes classifiers require interaction between participants. During the classification phase, Protocol 1 presented in this paper does not require interaction at all. Furthermore, being different from those protocols based on the Paillier scheme, our protocol is post-quantum safe because of the post-quantum safe BGV.

2 Preliminaries

In this section, we give some basics on homomorphic encryption schemes, which will be helpful for the rest of the paper. We refer the reader to [11,4,14] for more details. We also introduce a few building blocks for our protocol.

2.1 Homomorphic Encryption

A *homomorphic encryption* (HE) scheme is composed of four algorithms: KeyGen, Enc, Dec and Eval. Given a security parameter λ as input, KeyGen outputs a secret key sk, a public key pk, and an evaluation key ek. The encryption algorithm encrypts a plaintext b to a ciphertext $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(b)$, and the decryption algorithm decrypts a ciphertext c to a plaintext $b := \mathsf{Dec}_{\mathsf{sk}}(c)$. The evaluation algorithm evaluates a function of f on a ciphertext $c = \mathsf{Enc}_{\mathsf{pk}}(b)$ to a new ciphertext $c' \leftarrow \mathsf{Eval}_{\mathsf{ek}}(f, c)$ satisfying $\mathsf{Dec}_{\mathsf{sk}}(c') = f(b)$. For convenience, we call f an *HE function.*

For almost all known HE schemes, every ciphertext has noise, and every operation on ciphertext introduces noise in the resulting ciphertext. A *Leveled* HE (LHE) scheme allows to evaluate any HE function f with bounded number of arithmetic/boolean operations. Beyond this bound, such an LHE scheme cannot correctly decrypt the ciphertext since the accumulated noise is too large. Contrarily, if an HE scheme supports to evaluate an arbitrary computable function, then it is called a *fully homomorphic encryption* (FHE) scheme, which depends on the so-called *bootstrapping* technique [12] to reduce noise.

Since Gentry's seminal work [12], multiple HE schemes have been designed, such as BGV [4], BFV [3,10], CKKS [5], FHEW [9], TFHE [6]. Each of them has its features. For instance, BGV and BFV are good at performing large vectorial arithmetic operations, CKKS supports floating-point computations and FHEW and TFHE run bootstrapping for one bit extremely fast but slow for arithmetic operations. Since naïve Bayes classifiers require many integer arithmetic operations, and as reported in [1,19] that BGV is faster than BFV, especially for cases with large modulus, we choose an LHE variant of the BGV scheme, which will be introduced in the next subsection.

2.2 The BGV Scheme

For completeness, we give a brief description of a variant of BGV that is implemented in HElib [15].

- $\mathsf{Setup}(1^\lambda)$. Given a security parameter λ as input, set an integer m (that defines the m-th cyclotomic polynomial $\Phi_m(x)$), a prime number p (the plaintext space is $R_p = \mathbb{Z}_p[x]/\langle \Phi_m(x) \rangle$), an odd modulus q (the ciphertext space is $R_q = \mathbb{Z}_q[x]/\langle \Phi_m(x) \rangle$), and a noise distribution χ over R_q. Output $params = (m, p, q, \chi)$.
- $\mathsf{KeyGen}(params)$. Sample $s \leftarrow \chi$. Let $\boldsymbol{s} = (1, s) \in R_q^2$. Set $\mathsf{sk} = \boldsymbol{s}$. Generate $a \leftarrow R_q$ uniformly at random and a column vector with "small" coefficients $e \leftarrow \chi$. Set $b = a \cdot s + p \cdot e$. Output $\mathsf{sk} = \boldsymbol{s}$ and the public key $\boldsymbol{a} = (b, -a)$.
- $\mathsf{Enc}(params, \mathsf{pk}, b)$. To encrypt a message $b \in R_p$, set $\boldsymbol{m} = (b, 0) \in R_p^2$, sample a polynomimal $r \leftarrow R$ with small coefficients and output the ciphertext $\boldsymbol{c} = \boldsymbol{m} + r\boldsymbol{a} \in R_q^2$.
- $\mathsf{Dec}(params, \mathsf{sk}, \boldsymbol{c})$. Output the message $b = [[\langle \boldsymbol{c}, \boldsymbol{s} \rangle]_q]_p$.

The quantity $\langle \boldsymbol{c}, \boldsymbol{s} \rangle$ is called the *noise* of the ciphertext $\boldsymbol{c}$ under the secret key $\boldsymbol{s}$. Decryption works correctly only if the noise does not wrap around modulo q. If it is the case, the correctness follows from $[[\langle \boldsymbol{c}, \boldsymbol{s} \rangle]_q]_p = [[\langle \boldsymbol{m} + r\boldsymbol{a}, \boldsymbol{s} \rangle]_q]_p = [b + p \cdot re]_p = b$. The security is based on the Ring-LWE assumption [20], which is thought to resist attacks from quantum computers.

Homomorphic Evaluation. The BGV scheme supports homomorphic addition and multiplication. Let $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ be two ciphertexts of two plaintexts b_1 and b_2 under the same secret key $\boldsymbol{s}$. Suppose that the noise of $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ is bounded from above by B. The addition ($\mathsf{BGV.Add}$) of the two ciphertexts is simply $\boldsymbol{c}_+ = \boldsymbol{c}_1 + \boldsymbol{c}_2$, which is a ciphertext of $b_1 + b_2$ under the secret key $\boldsymbol{s}$. The noise of $\boldsymbol{c}_+$ is at most $2B$. For multiplication ($\mathsf{BGV.Mul}$), $\boldsymbol{c}_\times = \boldsymbol{c}_1 \otimes \boldsymbol{c}_2$ is a ciphertext of $b_1 \cdot b_2$ under a new secret key $\boldsymbol{s} \otimes \boldsymbol{s}$, where $\otimes$ is the usual tensor product. The noise of $\boldsymbol{c}_\times$ can only be bounded from above by B^2. To keep the secret key with small size and to decrease the noise of evaluated ciphertext, a refresh procedure $\mathsf{BGV.Refresh}$ (consisting of key switching and modulus switching) follows every homomorphic addition and multiplication. Of course, one can call $\mathsf{BGV.Refresh}$ only if necessary for efficiency. Note that the public key pk of

BGV also includes all keys for BGV.Refresh. Theoretically, the cost of each homomorphic addition or multiplication increases fast as L grows, where L is the circuit depth of the function f to be evaluated; see, e.g., [4] for more details. Besides, BGV also supports plaintext-ciphertext addition (BGV.AddConst) and plaintext-ciphertext multiplication (BGV.MulConst).

Batching. Recall the plaintext space $R_p = \mathbb{Z}_p[x]/\langle \Phi_m(X) \rangle$. Let d be the multiplicative order of p modulo m, and $\phi(m)$ be the Euler's totient function. Then d divides $\phi(m)$ and $R_p \cong \mathbb{F}_{p^d}^{\ell}$ with $\ell = \phi(m)/d$. Therefore each plaintext can be seen as a packed message with ℓ slots. From this view, each homomorphic operation on a ciphertext is equivalent to the same operation on all slots independently and simultaneously. This batching technique was introduced by Smart and Vercauteren in [22] and significantly decreased the amortized cost of Ring-LWE based HE schemes. For batching, BGV supports data packing (BGV.Encode), data rotating (BGV.Rotate), and data shifting (BGV.Shift). Based on these operations, one can build some advanced functions. For instance, BGV.TotalSum converts a ciphertext that encrypts $(z_1, \cdots, z_t)$ into a ciphertext that encrypts $(y, \cdots, y)$ with $y = \sum_{i=1}^{t} z_i$.

2.3 Building Blocks

We now describe a few necessary building blocks that will be used to build our classifier. Note that all the following algorithms will be executed on the server and that the owner of pk (the public key) in these algorithms is the client, not the server since we follow the framework given in Fig. 1.

Plaintext Matrix-Encrypted Vector Multiplication. Matrix-vector multiplication is fairly common in practice. Here we focus on plaintext matrix-encrypted vector multiplication. Given a matrix $\boldsymbol{A} \in \mathbb{Z}^{s \times t}$ and an encryption of a vector $\boldsymbol{z} \in \mathbb{Z}^t$, our goal is to obtain an encryption of $\boldsymbol{Az}$. We give two methods based on different method to encrypt a vector.

Naïve Encoding. To encrypt a vector $\boldsymbol{z} = (z_i)_{i \le t} \in \mathbb{Z}^t$, one can encrypt each entry z_i of $\boldsymbol{z}$ as a ciphertext. The resulting encryption of $\boldsymbol{z}$ is a vector $\boldsymbol{c} = (c_i)_{i \le t} \in R_q^t$ of ciphertexts, whose i-th entry c_i is a ciphertext of z_i. Then we have Algorithm 1.

```
Input: $\boldsymbol{c}' = (c'_i)_{i \le t} \in R_q^t$ ($c'_i$ encrypts the $i$th entry of $\boldsymbol{z} = (z_i)_{i \le t}$), and public key pk;
    $\boldsymbol{A} = (a_{i,j}) \in \mathbb{Z}^{s \times t}$.
Output: $(c_i)_{i \le s}$ with $c_i = \mathsf{Enc}_{\mathsf{pk}}(\sum_{j=1}^{t} a_{i,j} z_j)$.
1: For $i = 1, \cdots, s$ do the following:
2:     $c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$;
3:     For $j = 1, \cdots, t$, update $c_i := \mathsf{Add}_{\mathsf{pk}}(c_i, \mathsf{MulConst}_{\mathsf{pk}}(a_{i,j}, c'_j))$.
4: return $(c_i)_{i \le s}$.
```

Algorithm 1: Naïve plaintext matrix-encrypted vector multiplication

Packed Encoding. Instead of the above elements-wise method, we can pack the vector $\boldsymbol{z} \in \mathbb{Z}^t$ into t slots of one plaintext, and encrypt it to only one ciphertext $c \in R_q$, which leads to Algorithm 2.

```
Input: $c \in R_q$ that encrypts $u \in R_p$ with $u = \mathsf{Encode}(\boldsymbol{z})$, and public key $\mathsf{pk}$; $\boldsymbol{A} = (a_{i,j}) \in \mathbb{Z}^{s \times t}$.
Output: $(c_i)_{i \le s}$ with $c_i = \mathsf{Enc}_{\mathsf{pk}}(\sum_{j=1}^{t} a_{i,j} z_j)$.
1: For $i = 1, \cdots, s$ do the following:
2:     Encodes the $i$-th row $\boldsymbol{a}_i$ of $\boldsymbol{A}$ as $v_i = \mathsf{Encode}(\boldsymbol{a}_i)$;
3:     Computes $c_i = \mathsf{TotalSum}_{\mathsf{pk}}(\mathsf{MulConst}_{\mathsf{pk}}(v_i, c)))$.
4: return $(c_i)_{i \le s}$.
```

Algorithm 2: Pakced plaintext matrix-encrypted vector multiplication

Comparison among Encrypted Integers. Comparison is a commonly used function in many applications. In this work, we use a very recent comparator presented by Iliashenko and Zucca in [16], which supports comparison operations for BGV ($\mathsf{BGV.Comparator}$). Essentially, this method homomorphically evaluates the Lagrange interpolated polynomial of the less-than function over $S = [0, (p-1)/2]$ defined as follows:

$$\mathsf{LT}_S(x, y) = \begin{cases} 1, & \text{if } 0 \le x < y \le (p-1)/2, \\ 0, & \text{if } 0 \le y \le x \le (p-1)/2. \end{cases}$$

```
Input: $\boldsymbol{c} = (c_0, \cdots, c_{t-1}) \in R_q^t$ ($c_i$ encrypts the $i$th entry of $\boldsymbol{z} = (z_0, \cdots, z_{t-1})$) and public key $\mathsf{pk}$.
Output: A ciphertext $c \in R_q$ that encrypts the index of the maximal value of $\boldsymbol{z}$.
1: Set $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$. For $i = 0, \cdots, t-1$ do the following:
2:     Compute $c' := \prod_{k=0, k \ne j}^{t-1} \mathsf{AddConst}_{\mathsf{pk}}(1, \mathsf{MulConst}_{\mathsf{pk}}(-1, \mathsf{Comparator}_{\mathsf{pk}}(c_j, c_k)))$.
3:     Update $c := \mathsf{Add}_{\mathsf{pk}}(c, \mathsf{MulConst}_{\mathsf{pk}}(j, c'))$.
4: return $c$.
```

Algorithm 3: Encrypted index of maximum of an encrypted array

Furthermore, one can apply the less-than function LT to compute the arg max function of an encrypted array, where the arg max function returns an encryption of the index of the maximum of an array. As a matter of fact, for a given array $\boldsymbol{z} = (z_0, \cdots, z_{t-1})$, we have

$$\arg\max_i (z_i)_{0 \le i \le t-1} = \sum_{j=0}^{t-1} j \cdot \prod_{k=0, k \ne j}^{t-1} (1 - \mathsf{LT}(z_j, z_k)),$$

which results in Algorithm 3. Note that one would better use some recursive methods in practice to compute the encrypted product in Step 2 of Algorithm 3 for saving multiplicative depth.

3 Privacy-preserving Naïve Bayes Classification

In this section, we first introduce the plaintext Naïve Bayes Classifier, and then adapt the classifier to the privacy-preserving setup and prove its correctness and security.

3.1 Naïve Bayes Classifier

Naïve Bayes classifier is based on the assumption that all features are conditional independent. Consider a data set with s categories $1, \cdots, s$ and n features $X_1, \cdots, X_n$, where each feature X_k has at most t different values $1, 2, \cdots, t$. Under the conditional independence assumption, the classification of a sample $\boldsymbol{x} = (x_1, \cdots, x_n)$ is

$$s^* = \arg\max_{i=1,\dots,s} \Pr[Y = i] \prod_{k=1}^{n} \Pr[X_k = x_k | Y = i],$$

where $\Pr[Y = i]$ is the probability that each class i occurs, i.e., the *prior probability*, and $\Pr[X_k = x_k|Y = i]$ is the probability of the kth feature X_k to be $x_k \in \{1, 2, \cdots, t\}$ when $\boldsymbol{x}$ belongs to category i, i.e., the *likelihood*. As in [2], we only deal with the case that the domain of the feature values (the x_i's) is discrete and finite, so the $\Pr[X_k = x_k|Y = i]$'s are probability masses.

3.2 Preparing the model

If the domain of the feature values is continuous, we first find a bound B on the values and then discretize them by splitting $[-B, B]$ into several equal intervals. For example, if the domain of the kth feature X_k is continuous on $[-1, 1]$, then one can discretize X_k as $X_k = 0$ if $X_k \in [-1, 0)$ and $X_k = 1$ if $X_k \in [0, 1]$. This discretization technique enables our classifier to deal with continuous features as well, possibly at the cost of decreasing the prediction accuracy.

For convenience, we can further simplify the feature values $x_1, \cdots, x_n$ to $1, \cdots, t$. Furthermore, for numerical stability, we work with the logarithm of the probability:

$$s^* = \arg\max_{i=1,\dots,s} \left\{ \log \Pr[Y = i] + \sum_{k=1}^{n} \log \Pr[X_k = x_k|Y = i] \right\}, \tag{1}$$

where $x_k \in \{1, \cdots, t\}$. Another convenient simplification is to take the numbering of the s classes as contiguous integers from 1 to s. Then s^* is precisely the index of the maximum over the s values in (1).

Additionally, since the BGV encryption scheme works with integers, one needs to convert each logarithm of probability in (1) to an integer by multiplying it with a certain number $K > 0$ and rounding it to the closest integer. A similar *shifting* technique is already used and analyzed in, e.g., [24,2].

In summary, for a data set with s categories and n features (each feature has at most t different values), the prior probability in the model will be converted into a vector $\boldsymbol{b} = (b_1, \cdots, b_s) \in \mathbb{Z}^s$, where b_i is obtained by rounding $K \cdot \log(\Pr[Y = i])$ for an appropriate scaling integer K. The likelihoods will be converted into n matrices $\boldsymbol{A}_k \in \mathbb{Z}^{s \times t}$ for $k = 1, \cdots, n$, where the (i, j)-entry of $\boldsymbol{A}_k$ is derived by rounding $K \cdot \log \Pr[X_k = j | Y = i]$ with the same integer K.

3.3 Privacy-preserving Naïve Bayes Classifier

To resolve the privacy concerns, the client should only obtain the classification result s^* without learning any information about the prior probability and likelihood, and the server should learn nothing about the client's data $\boldsymbol{x}$.

The client has data $\boldsymbol{x} = (x_1, \cdots, x_n)$ with $x_k \in \{1, \cdots, t\}$ and wants the server to predict which class $\boldsymbol{x}$ is in by using a naïve Bayes classifier without leaking any information about $\boldsymbol{x}$. One choice of the client is to encrypt $\boldsymbol{x}$ using himself's public key. However, since $\boldsymbol{x}$ is encrypted, the server cannot decide which entry of $\boldsymbol{A}_k$ should be chosen. For instance, the first feature of $\boldsymbol{x}$ is x_1, i.e., $X_1 = x_1$. To access the information about $\Pr[X_1 = x_1 | Y = i]$ in $\boldsymbol{A}_1$, we need to select the (i, x_1) entry of $\boldsymbol{A}_1$. However, as the first entry of $\boldsymbol{x}$, x_1 is only available in encrypted form on the server-side. To get around this obstacle, one can encode the sample $\boldsymbol{x}$ as a 0-1 matrix

$$\boldsymbol{X} = (\boldsymbol{e}_{x_1}, \cdots, \boldsymbol{e}_{x_n}) \in \{0, 1\}^{t \times n}, \tag{2}$$

where $\boldsymbol{e}_j$ is the t-dimensional vector whose jth entry is 1 and all others are 0. Now, to select the x_k-th row of a matrix $\boldsymbol{A}_k \in \mathbb{Z}^{s \times t}$ is just to compute $\boldsymbol{A}_k \boldsymbol{e}_{x_k}$. If $\boldsymbol{e}_{x_k}$ is in encrypted form, then this is a plaintext matrix-encrypted vector multiplication discussed in Section 2.3.

Now we are ready to present our privacy-preserving naïve Bayes classifier as Protocol 1, which does not require any interaction between the server and the client during the classification phase (Step 3–6).

We prove the security of our protocol using the secure two-party computation framework for passive adversaries. Roughly speaking, a passive adversary tries to learn as much private information from the other party; however, this adversary follows the prescribed protocol faithfully.

Proposition 1. *Protocol 1 is correct. It is secure in the honest-but-curious model, assuming that the used LHE scheme is secure.*

```
Input of the client: A sample $\boldsymbol{x} = (x_1, \cdots, x_n)$ to be predicted, the secret and public key $\mathsf{sk}$ and $\mathsf{pk}$.
Input of the server: The model consisting of the likelihood information $(\boldsymbol{A}_k)_{k \leq n}$ and the prior information $(b_i)_{i \leq s}$, and the client's public key $\mathsf{pk}$.
1: The client encode $\boldsymbol{x}$ to a matrix $\boldsymbol{X}$ as in (2).
2: The client encrypts the column vectors $\boldsymbol{e}_{x_k}$ of $\boldsymbol{X}$ for $k = 1, \cdots, n$ and sends these ciphertexts to the server.
3: The server do the following:
4:     For $i = 1, \cdots, s$, set $c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$ and update $c_i := \mathsf{AddConst}_{\mathsf{pk}}(c_i, b_i)$.
5:     For $k = 1, \cdots, n$, calling Algorithm 1 or 2 with input as the ciphertexts of $\boldsymbol{e}_{x_k}$, $\boldsymbol{A}_k$ and $\mathsf{pk}$ outputs $(c'_i)_{i \leq s}$. Update $c_i := \mathsf{Add}_{\mathsf{pk}}(c_i, c'_i)$ for $i = 1, \cdots, s$.
6:     Calling Algorithm 3 with input as $\boldsymbol{c} = (c_i)_{i \leq s}$ and $\mathsf{pk}$ returns $c$.
7: The server sends $c$ to the client.
8: The client decrypt $c$ to $y = \mathsf{Dec}_{\mathsf{sk}}(c)$.
```

Protocol 1: Privacy-preserving naïve Bayes classifier

Proof (Sketch). The correctness follows from that what the server does is to evaluate the following procedure homomorphically:

```
1: Set $\boldsymbol{y} := \boldsymbol{b}$, the information of the prior probability.
2: For $k = 1, \cdots, n$, set $\boldsymbol{y} := \boldsymbol{y} + \boldsymbol{A}_k \cdot \boldsymbol{e}_{x_k}$.
3: return $y$ as the index of the maximum of $\boldsymbol{y} = (y_i)_{0 \leq i \leq s-1}$.
```

From the server-side, the only message received is an encryption of the client's data $\boldsymbol{x}$, and the only message sent to the client is a ciphertext of the prediction result. From the client-side, $\boldsymbol{x}$ is encrypted, the classification result is also encrypted, and the model is not acccessible. So the security of Protocol 1 follows from that of the used LHE scheme. □

4 Experiments

We have implemented Protocol 1 in C++ using HElib (v2.1.0) [15]. In this section, we will report the prediction accuracy and calculation time of our implementation. All experiments run serially (using only one thread) on a laptop with a Ubuntu 20.04 OS as Windows Subsystem for Linux, Intel i7-10750H CPU, and 16 GB RAM.

Data Set. In this experiment, Wisconsin Breast Cancer (WBC) data set in UCI Machine Learning Repository [8] was used. The dataset has 683 effective samples, classified into two categories, i.e., $s = 2$. There are nine features for each sample, and each feature may take at most ten different values, i.e., $n = 9$ and $t = 10$. Among these 683 samples, 478 samples are used for training (70%), and the remaining 205 samples are used to test.

Parameter Setting. For WBC, the scaling factor K in Section 3.2 is set to 1. This leads that the entries of the rounded logarithm of likelihood $\boldsymbol{A}_k$ for $k =$

$1, \cdots, n$ are integers between -6 and 0, and the entries of the rounded logarithm of the prior probability $\boldsymbol{b}$ are bounded by 2. Hence the resulting integers to be compared must be at most $6n + 2 = 54$, which implies that $p = 113$ is enough for our purpose. In addition, m is fixed to 12883. In this setting, each plaintext in R_p has $\ell = 3960$ slots.

Accuracy. According to our experiments, the classification accuracy of our implementation of Protocol 1 based on HElib is about 97%. Note that this accuracy is almost the same as the plaintext (unencrypted) naïve Bayes classifier.

Timing. In Step 5 of Protocol 1, there are two choices (Algorithm 1 and Algorithm 2) for plaintext matrix-encrypted vector multiplication. We test them all and record their performance in Table 1. The row named "naïve" ("packed" resp.) is the performance of Protocol 1 based on Algorithm 1 (Algorithm 2 resp.). The columns with "Ave." is the average execution time for each sample. Table 1 shows that the naïve variant outperforms the packed variant. For comparison, Kim *et al.* [17] reported that their proposed privacy-preserving naïve Bayes classifier takes 17h40m on a single core for the Iris data set ($s = 3$, $n = 4$, $t = 5$, 30 samples to be classified) of UCI Machine Learning Repository [8].

Table 1. Peformance of two variants of Protocol 1

	$\log q$	λ	Compare (s)	Total (s)	Ave. Compare (s)	Ave. Total (s)
naïve	382	101	387.90	985.31	1.89	4.75
packed	476	76	369.95	1931.60	1.75	9.42

5 Conclusion

In this paper, we attempt to design privacy-preserving classifier protocols in the client-server setting. The server owns the model, which should not be accessible to any other, and the client also needs to preserve the privacy of the data to be predicted. As a result, we propose a privacy-preserving naïve Bayes classifier (Protocol 1) based on the LHE scheme BGV. We show that the classifier is correct and secure in the honest-but-curious model. The main feature of our classifier is that it does not require any interaction between the client and the server during the classification phase. We are considering how to use SIMD to accelerate the classifier and experiment with more data sets. In addition, how to optimize the performance further using, e.g., parallel computing, is also an interesting problem.

References

1. Aguilar Melchor, C., Kilijian, M.O., Lefebvre, C., Ricosset, T.: A comparison of the homomorphic encryption libraries HElib, SEAL and FV-NFLlib. In: Lanet, J.L., Toma, C. (eds.) Innovative Security Solutions for Information Technology and Communications – Proc SecITC 2018 (November 8–9, 2018, Bucharest, Romania), Lecture Notes in Computer Science, vol. 11359, pp. 425–442. Springer, Cham (2019), https://doi.org/10.1007/978-3-030-12942-2_32
2. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. In: Proceedings of the 22nd Annual Network and Distributed System Security Symposium (February 8-11, 2015, San Diego, USA). The Internet Society (2015), https://doi.org/10.14722/ndss.2015.23241
3. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – Proc CRYPTO 2012 (August 19–23, 2012, Santa Barbara, CA, USA), Lecture Notes in Computer Science, vol. 7417, pp. 868–886. Springer, Heidelberg (2012), http://doi.org/10.1007/978-3-642-32009-5_50
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory **6**(3), 13:1–13:36 (2014), https://doi.org/10.1145/2633600
5. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Proceedings of ASIACRYPT 2017 – 23rd International Conference on the Theory and Applications of Cryptology and Information Security (December 3-7, 2017, Hong Kong, China), Part I, Lecture Notes in Computer Science, vol. 10624, pp. 409–437. Springer, Heidelberg (2017), https://doi.org/10.1007/978-3-319-70694-8_15
6. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020), https://doi.org/10.1007/s00145-019-09319-x
7. Domingos, P., Pazzani, M.: On the optimality of the simple Bayesian classifier under zero-one loss. Machine Learning **29**(2), 103–130 (1997), https://doi.org/10.1023/A:1007413511361
8. Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
9. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - Proceedings of EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Part I (April 26-30, 2015, Sofia, Bulgaria), Lecture Notes in Computer Science, vol. 9056, pp. 617–640. Springer, Heidelberg (2015), https://doi.org/10.1007/978-3-662-46800-5_24
10. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive https://eprint.iacr.org/2012/144 (2012)
11. Gentry, C.: A Fully Homomorphic Encryption Scheme. Ph.D. thesis, Stanford University, Stanford (2009), https://crypto.stanford.edu/craig/craig-thesis.pdf
12. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) Proceedings of the forty-first annual ACM symposium on Theory of computing (May 31 - June 2, 2009, Bethesda, USA), pp. 169–178. ACM, New York (2009), https://doi.org/10.1145/1536414.1536440
13. Goldreich, O.: Foundations of Cryptography – Basic Applications. Cambridge University Press, Cambridge (2004)

14. Halevi, S., Shoup, V.: Design and implementation of HElib: a homomorphic encryption library (2020), Cryptology ePrint Archive https://eprint.iacr.org/2020/1481
15. HElib:: An implementation of homomorphic encryption (Accessed in August, 2021), https://github.com/homenc/HElib
16. Iliashenko, I., Zucca, V.: Faster homomorphic comparison operations for BGV and BFV. Proceedings on Privacy Enhancing Technologies **2021**(3), 246–264 (2021), https://doi.org/10.2478/popets-2021-0046
17. Kim, S., Omori, M., Hayashi, T., Omori, T., Wang, L., Ozawa, S.: Privacy-preserving naive Bayes classification using fully homomorphic encryption. In: Cheng, L., Leung, A.C.S., Ozawa, S. (eds.) Neural Information Processing – Proceedings of the 25th International Conference on Neural Information Processing (Siem Reap, Cambodia, December 13–16, 2018), Lecture Notes in Computer Science, vol. 11304, pp. 349–358. Springer, Cham (2018), https://doi.org/10.1007/978-3-030-04212-7_30
18. Li, X., Zhu, Y., Wang, J.: Secure naïve bayesian classification over encrypted data in cloud. In: Chen, L., Han, J. (eds.) Proceedings of the 10th International Conference on Provable Security (Nanjing, China, November 10-11, 2016), Lecture Notes in Computer Science, vol. 10005, pp. 130–150. Springer, Cham (2016), https://doi.org/10.1007/978-3-319-47422-9_8
19. Lou, Q., Feng, B., Fox, G.C., Jiang, L.: Glyph: Fast and accurately training deep neural networks on encrypted data. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Proceedings of NeurIPS 2020 (34th Conference on Neural Information Processing Systems, Vancouver, Canada). NeurIPS (2020), https://proceedings.neurips.cc/paper/2020/hash/685ac8cadc1be5ac98da9556bc1c8d9e-Abstract.html
20. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of ACM **60**(6), 43:1–35 (2013), https://doi.org/10.1145/2535925
21. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology – EUROCRYPT '99 (May 2–6, 1999, Prague, Czech Republic), Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer, Heidelberg (1999), https://doi.org/10.1007/3-540-48910-X_16
22. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Designs, Codes and Cryptography **71**(1), 57–81 (2014), https://doi.org/10.1007/s10623-012-9720-4
23. Sun, X., Zhang, P., Liu, J.K., Yu, J., Xie, W.: Private machine learning classification based on fully homomorphic encryption. IEEE Transactions on Emerging Topics in Computing **8**(2), 352–364 (2020), https://doi.org/10.1109/TETC.2018.2794611
24. Tschiatschek, S., Reinprecht, P., Mücke, M., Pernkopf, F.: Bayesian network classifiers with reduced precision parameters. In: Flach, P.A., De Bie, T., Cristianini, N. (eds.) Proceedings of ECML PKDD 2012: Joint European Conference on Machine Learning and Knowledge Discovery in Databases (Bristol, UK, September 24-28, 2012). Lecture Notes in Computer Science, vol. 7523, pp. 74–89. Springer, Heidelberg (2012), https://doi.org/10.1007/978-3-642-33460-3_10
25. Yasumura, Y., Ishimaki, Y., Yamana, H.: Secure naïve Bayes classification protocol over encrypted data using fully homomorphic encryption. In: Indrawan-Santiago, M., Pardede, E., Salvadori, I.L., Steinbauer, M., Khalil, I., Anderst-Kotsis, G. (eds.) Proceedings of the 21st International Conference on Information Integration and Web-Based Applications & Services (Munich, Germany, December 2–4, 2019), pp. 45–54. ACM, New York (2019), https://doi.org/10.1145/3366030.3366056