Non-interactive privacy-preserving naïve Bayes classifier from leveled fully homomorphic encryption

Jingwei Chen  Yong Feng  Yang Liu  Wenyuan Wu  Guanci Yang

March 23, 2023

Abstract

In this paper, we propose a non-interactive privacy-preserving naïve Bayes classifier from leveled fully homomorphic encryption schemes. The classifier runs on a server that is also the model’s owner (modeler), whose input is the encrypted data from a client. The classifier produces encrypted classification results, which can only be decrypted by the client, while the modeler’s model is only accessible to the server. Therefore, the classifier does not leak any privacy on either the server’s model or the client’s data and results. More importantly, the classifier does not require any interactions between the server and the client during the classification phase. The main technical ingredient is an algorithm that computes the maximum index of an encrypted array homomorphically without any interactions. The proposed classifier is implemented using HElib. Experiments show the accuracy and efficiency of our classifier. For instance, the average cost can achieve about 34ms per sample for a real data set in UCI Machine Learning Repository with the security parameter about 100 and accuracy about 97%.

Keywords: privacy-preserving machine learning, naïve Bayes classifier, fully homomorphic encryption, leveled fully homomorphic encryption, BGV, HElib.

1 Introduction

Over the past ten years, 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 collects a large amount of data to train the model and then uses the trained model to infer/predict some results based on clients’ new data. However, as security incidents continue to occur (such as data breaches), the demand for privacy-preserving MLaaS is rapidly increasing. On the one hand, the model owner does not want to leak information about the model. On the other hand, the data owner is also reluctant to leak any information about the data. To resolve this contradiction, the community studies privacy-preserving machine learning.

![Figure 1: Framework of privacy-preserving classifiers](image)

In this paper, we consider the framework presented in [BPTG15] 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 sample data and classification result to the client. The framework in Fig. 1 happens quite often in practice. For instance, the server might be a big-data service provider with a predictive model for a specific disease, and the client...
might be a hospital that needs to diagnose many potential cases every day. Following the framework, the hospital first sends the encrypted samples to the service provider for a diagnosis. The service provider runs the classifier with input as its model and the received encrypted samples to obtain an encrypted diagnosis result, and sends it to the hospital. The hospital decrypts it to disclose the diagnosis result.

In particular, we present a privacy-preserving naive Bayes classifier (Protocol 1) based on leveled fully homomorphic encryption schemes. The leveled fully homomorphic encryption schemes used in Protocol 1 allow us to evaluate functions with a bounded multiplicative depth on encrypted data, such as BGV [BGV14] and BFV [Bra12, FV12]. These schemes are based on the Learning With Errors over Rings (RLWE) assumption [LPR13] and hence thought to be post-quantum safe. Based on the RLWE assumption and the underlying leveled fully homomorphic encryption schemes, we prove that Protocol 1 is correct and secure in the honest-but-curious (semi-honest or passive) model.

Protocol 1 is a minimally interactive protocol. The sample 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 input as the encrypted data $c$. The server sends the resulting ciphertext to the client, and the client decrypts the ciphertext using its secret key in which class $x$ lies. Thanks to an algorithm that computes a ciphertext of the maximum index (i.e., 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 in Protocol 1).

We implement Protocol 1 using the C++ homomorphic encryption library HElib [HEl21] and test for the Iris and Wisconsin Breast Cancer (WBC) data set in UCI Machine Learning Repository [DG17]. Experiments show that Protocol 1 is comparable to existing privacy-preserving naive Bayes classifiers in literature. For example, with security parameter 100, the average cost of our classifier is 214ms and 34ms per sample for the Iris and WBC data sets, respectively.

**Related work** It seems challenging to list all literature on privacy-preserving protocols for classifiers. We refer to [BPTG15, SYZ’20, WNK20] for good surveys. Here we only focus on those privacy-preserving naive Bayes classifiers based on homomorphic encryption.

Naïve Bayes classifiers is a simple but powerful algorithm to predict the category label of unclassified samples; see, e.g., [DP97]. Bost et al. proposed in [BPTG15, Sec. VI] the first efficient privacy-preserving protocols for naive Bayes classifier based on the Quadratic Residuosity (QR) [GM82] and Paillier [Pai99] cryptosystems, which are known to be broken by quantum computers. Li et al. proposed in [LZW16] a secure naive Bayes classifier for four parties, without experimental results reported. Later on, Kim et al. [KOH*18] adapted Li et al.’s framework using the homomorphic encryption scheme presented by Brakerski, Gentry, and Vaikuntanathan (BGV) [BGV14]. Yasumura et al. [YIT19] and Sun et al. [SZL’20] also gave privacy-preserving protocols for naive Bayes classification based on BGV. In addition, Wood et al. presented in [WSNK19] a private naive Bayes classifier based on a private fully homomorphic encryption scheme proposed by Gribov, Kahrobaei, and Shpilrain [GKS18]. While all of these privacy-preserving naive Bayes classifiers require interactions among participants during the classification phase (the classifier in Fig. 1), Protocol 1 presented in this paper does not require any interactions at all. For most of them, interactions are needed to compute the maximum index of an encrypted array. Instead, we present a non-interactive algorithm (Algorithm 3), which makes our protocol non-interactive. Furthermore, being different from those protocols based on non-quantum-resistant assumptions, our used leveled fully homomorphic cryptosystem is BGV or BFV, which are based on the RLWE assumption [LPR13] and hence thought to be post-quantum safe.

We note that this paper is an extended version of [CFL*22]. In this extended version, a rigorous proof for the security of Protocol 1 and an extensive experimental study on the performance are supplemented.

**Road-map** In Section 2, we give a brief introduction to the naïve Bayes classifier, homomorphic encryption, and adversarial model. We present several building blocks in Section 3 for our classifier, including the main technical ingredient, Algorithm 3. In Section 4, we propose a privacy-preserving naïve Bayes classifier and prove its correctness and its security in the passive (or honest-but-curious [Go04]) model. In Section 5, we report extensive experimental results on our implementation of Protocol 1.

### 2 Preliminaries

In this section, we give some backgrounds useful for the rest of this paper.

#### 2.1 Naïve Bayes classifier

Naïve Bayes classifier is based on the assumption that all features are conditional independent. Consider a data set with $s$ categories $1, \ldots, s$ and $n$ features $X_1, \ldots, X_n$, where each feature $X_k$ has at most $t$ different values $1, 2, \ldots, t$. 
We restrict to those RLWE-based schemes. In such schemes, the plaintext space is $\mathbb{Z}_q$ where $q$ is typically a ciphertext of $b$ and $c$. Let $c_2$. The security of many efficient FHE schemes, including BGV [BGV14], BFV [Bra12, FV12], CKKS [CKKS17], depends on the RLWE assumption or its variants. For more details, we refer the reader to [LPR13].

### 2.2 The RLWE assumption

The security of many efficient FHE schemes, including BGV [BGV14], BFV [Bra12, FV12], CKKS [CKKS17], FHEW [DM15], TFHE [CGGI20]. Each of them has its features. For instance, BGV is weakly circular under a new secret key $\lambda$ and let $\chi$ be a distribution over $R$. The R-LWE$_{m,q,\chi}$ problem is to distinguish between the following two distributions: In the first distribution, one samples $(a_i, b_i)$ uniformly from $R_q$. In the second distribution, one first draw $s \leftarrow R_q$ uniformly and then sample $(a_i, b_i) \in R_q^2$ by sampling $a_i \leftarrow R_q$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = a_i \cdot s + e_i$. The R-LWE$_{m,q,\chi}$ assumption is that the R-LWE$_{m,q,\chi}$ problem is infeasible.

### 2.3 Leveled fully homomorphic encryption

Fully homomorphic encryption schemes allow arithmetic circuits to be evaluated directly on ciphertexts [RAD78, Gen09]. Since Gentry’s seminal work [Gen09], multiple HE schemes have been designed, such as BGV [BGV14], BFV [Bra12, FV12], CKKS [CKKS17], FHEW [DM15], TFHE [CGGI20]. 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. As naïve Bayes classifiers require many integer arithmetic operations, we choose BGV or BFV as the leveled fully homomorphic encryption schemes, with parameters supporting integer arithmetic circuits of a certain bounded depth.

For completeness, we briefly describe the leveled fully homomorphic encryption schemes used in this paper. We restrict to those RLWE-based schemes. In such schemes, the plaintext space is $R_p = \mathbb{Z}_p[x]/\langle \Phi_m(x) \rangle$ and the ciphertext space is $R_q = \mathbb{Z}_q[x]/\langle \Phi_m(x) \rangle$, where $\Phi_m(x)$ is the $m$-th cyclotomic polynomial, $p$ is a prime number, and $q \gg p$ is an integer. Both of the BGV [BGV14] and the BFV [Bra12, FV12] schemes have this structure.

Typically, a leveled fully homomorphic encryption scheme FHE can be described by the following randomized algorithms:

- **FHE.Setup($1^\lambda$)**. Given a security parameter $\lambda$ as input, outputs $\text{parms}$.
- **FHE.KeyGen($\text{parms}$)**. Output a secret key $sk = s$ and the corresponding public key $pk$. (For convenience, we let $pk$ also include one or more evaluation keys $ek$.)
- **FHE.Enc$_{pk}(b)$**. Given a message $b \in R_p$, outputs a ciphertext $c \in R_q$.
- **FHE.Dec$_{sk}(c)$**. Given a ciphertext $c \in R_q$, outputs a message $b \in R_p$.
- **FHE.Eval$_{pk}(C, (c_1, \ldots, c_k))$**. Given an arithmetic circuit $C$ of a function $f$ with $k$ input wires, and input $c_1, \ldots, c_k$ with $c_i \leftarrow \text{FHE.Enc}_{pk}(b_i)$, outputs a ciphertext $c$ such that $\text{Pr}[\text{FHE.Dec}_{sk}(c) \neq f(b_1, \ldots, b_k)] = \text{negl}(\lambda)$.

FHE is said to be **compact** if the size of the output of FHE.Eval is not more than polynomial in $\lambda$ and is independent of $f$. FHE is said to be **secure** if it is IND-CPA secure and weakly **circular** secure, which means that the scheme remains secure even if the adversary is given encryptions of the bits of the secret key. We say that FHE achieves **circuit privacy** if the distribution of the outputs of any fixed homomorphic evaluation is indistinguishable from the distribution of fresh encryptions of the plaintext outputs.

#### 2.3.1 Homomorphic evaluation

Let $c_1$ and $c_2$ be two ciphertexts of two plaintexts $b_1$ and $b_2$ under the same secret key $sk$. Suppose that the noise of $c_1$ and $c_2$ is bounded from above by $B$. The addition (FHE.Add) of the two ciphertexts is typically $c_3 = c_1 + c_2$, which is a ciphertext of $b_1 + b_2$ under the secret key $sk$. The noise of $c_3$ is at most $2B$. For multiplication (FHE.Mul), $c_3 = c_1 \otimes c_2$ is typically a ciphertext of $b_1 \cdot b_2$ under a new secret key $sk \otimes sk$ with larger dimension, where $\otimes$ is the usual tensor product.
product. The noise of \( c_k \) can only be bounded from above by \( B^2 \). To keep the dimension of the secret key and to decrease the noise of evaluated ciphertext, a refresh procedure \( \text{FHE.Refresh} \) (consisting of key switching and modulus switching) follows every homomorphic addition and multiplication. Of course, one can call \( \text{FHE.Refresh} \) only if necessary for efficiency. Note that the public key \( pk \) of \( \text{FHE} \) also includes all keys for \( \text{FHE.Refresh} \). 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. Besides, \( \text{FHE} \) also supports plaintext-ciphertext addition (\( \text{FHE.AddConst} \)) and plaintext-ciphertext multiplication (\( \text{FHE.MulConst} \)).

2.3.2 Batching

Recall the plaintext space \( R_p = \mathbb{Z}_p[x]/(\Phi_m(X)) \). 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 \( R_p \cong \mathbb{F}_p^\ell \) with \( \ell = \phi(m)/d \). Therefore each plaintext can be seen as a packed message with \( \ell \) 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 [GHS12, SV14] significantly decreases the amortized cost (i.e., the total cost divided by \( \ell \)) of homomorphic encryption schemes based on RLWE.

To keep the dimension of the secret key and to decrease the noise of evaluated ciphertext, a refresh procedure \( \text{FHE.Refresh} \) (consisting of key switching and modulus switching) follows every homomorphic addition and multiplication. Of course, one can call \( \text{FHE.Refresh} \) only if necessary for efficiency. Note that the public key \( pk \) of \( \text{FHE} \) also includes all keys for \( \text{FHE.Refresh} \). 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. Besides, \( \text{FHE} \) also supports plaintext-ciphertext addition (\( \text{FHE.AddConst} \)) and plaintext-ciphertext multiplication (\( \text{FHE.MulConst} \)).

2.4 Adversarial model

Our protocol only involves the client and the server, labeled as parties \( C \) and \( S \), respectively. To show Protocol 1 preserves the privacy of both parties, we work in the honest-but-curious (semi-honest or passive) model as described in [Gol04, Sec. 7.2]. The materials presented here are mainly taken from the full version of [BPTG15].

Let \( f = (f_C, f_S) \) be a (probabilistic) polynomial function and \( \Pi \) a protocol computing \( f \). \( C \) and \( S \) want to compute \( f(a, b) \) where \( a \) is \( C \)'s input and \( b \) is \( S \)'s input, using \( \Pi \) and with the security parameter \( \lambda \). The view of party \( C \) during the execution of \( \Pi \) is the tuple \( V_C(\lambda, a, b) = (1^k; a, r_C^\ell; m_{i,1}^C, \cdots, m_{i,\ell}^C) \) where \( r_C \) is \( C \)'s random tape and \( m_{i,1}^C, \cdots, m_{i,\ell}^C \) are the messages received by \( C \). We define the view of \( S \) similarly. The outputs of parties \( C \) and \( S \) for the execution of \( \Pi \) on input \( (a, b) \) as \( \text{Output}^C(\lambda, a, b) \) and \( \text{Output}^S(\lambda, a, b) \), which is implicit in the party’s own view of the execution, and the global output as \( \text{Output}(\lambda, a, b) = (\text{Output}^C(\lambda, a, b), \text{Output}^S(\lambda, a, b)) \).

To ensure security, we have to show that whatever \( C \) can compute from its interactions with \( S \) can be computed from its own input and output, which leads us to the following security definition.

**Definition 2** ([Gol04, Def. 7.2.1]). The two-party protocol \( \Pi \) securely computes the function \( f \) if there exist two probabilistic polynomial-time algorithms \( S_C \) and \( S_S \) such that for every possible input \( a, b \) of \( f \),

\[
S_C(1^k, a, f_C(a,b)), f(a,b) \equiv_c V_C(\lambda, a, b), \text{Output}^C(\lambda, a, b)
\]

and

\[
S_S(1^k, b, f_S(a,b)), f(a,b) \equiv_c V_S(\lambda, a, b), \text{Output}^S(\lambda, a, b)
\]

where \( \equiv_c \) means computational indistinguishability against probabilistic polynomial time adversaries with negligible advantage in the security parameter \( \lambda \).

To simplify the notation and proof, we omit the security parameter. As we only consider deterministic functions \( f \), we can simplify the distributions we want to show being indistinguishable: when \( f \) is deterministic, to prove the security of \( \Pi \) that computes \( f \), we only have to show that

\[
S_C(a, f_C(a,b)) \equiv_c V_C(a,b), \quad S_S(b, f_S(a,b)) \equiv_c V_S(a,b).
\]

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 not the server but the client since we follow the framework given in Fig. 1. For simplicity, we use the functions without explicitly showing the name of the scheme \( \text{FHE} \) in the rest of this paper. For instance, we use \( \text{ADD} \) to replace \( \text{FHE.ADD} \).
3.1 Plaintext matrix-encrypted vector multiplication

Matrix-vector multiplication is reasonably common in practice. Here we focus on plaintext matrix-encrypted vector multiplication. Given a plaintext matrix $A \in \mathbb{Z}^{t \times t}$ and ciphertexts of a vector $z \in \mathbb{Z}^t$, our goal is to obtain ciphertexts of $Az$. We present two methods based on different ways to encode a vector.

3.1.1 Naïve encoding

To encrypt a vector $z = (z_i)_{i \leq t} \in \mathbb{Z}^t$, one can encrypt each entry $z_i$ of $z$ to a ciphertext. The encryption of $z$ is a vector $c' = (c'_i)_{i \leq t} \in R_q$ of ciphertexts, whose $i$-th entry $c'_i$ is a ciphertext of $z_i$. Algorithm 1 computes a vector in $R_q$ as the encryption of $Az$. Obviously, Algorithm 1 costs no multiplicative depth.

**Algorithm 1** Naïve plaintext matrix-encrypted vector multiplication

**Input:** $c' = (c'_i)_{i \leq t} \in R_q$ ($c'_i$ encrypts the $i$th entry of $z = (z_i)_{i \leq t}$), and public key pk; $A = (a_{i,j}) \in \mathbb{Z}^{t \times t}$.

**Output:** $(c_i)_{i \leq s}$, with $c_i = \text{Enc}_{pk}((\sum_{j=1}^{t} a_{i,j}z_j))$.

1. For $i = 1, \cdots, s$ do the following:
   
   (a) $c_i \leftarrow \text{Enc}_{pk}(0)$;
   
   (b) For $j = 1, \cdots, t$
   
   i. Update $c_i := \text{ADD}_{pk}(c_i, \text{Mul}_{pk}(a_{i,j}, c'_j))$.

2. Return $(c_i)_{i \leq s}$.

3.1.2 Packed encoding

Instead of the above element-wise method, we can pack the vector $z \in \mathbb{Z}^t$ into $t$ slots of one plaintext for batching in Sec. 2.3.2, and encrypt it to only one ciphertext $c \in R_q$, which leads to Algorithm 2. Note that $\text{FHE.TotalSum}$ costs no multiplicative depth since it uses only $\text{FHE.Rotate}$ and $\text{FHE.Add}$, and $\text{FHE.Rotate}$ costs no multiplicative depth (see, e.g., [HS20]). Thus, Algorithm 2 costs no multiplicative depth as well.

**Algorithm 2** Packed plaintext matrix-encrypted vector multiplication

**Input:** $c \in R_q$ that encrypts $u \in R_p$ with $u = \text{Encode}(z)$, and public key pk; $A = (a_{i,j}) \in \mathbb{Z}^{t \times t}$.

**Output:** $(c_i)_{i \leq s}$, with $c_i = \text{Enc}_{pk}((\sum_{j=1}^{t} a_{i,j}z_j))$.

1. For $i = 1, \cdots, s$ do the following:

   (a) $\text{Encodes the } i\text{-th row } a_i \text{ of } A \text{ as } u_i = \text{Encode}(a_i)$;

   (b) $c_i = \text{Total}_{pk}(\text{Mul}_{pk}(u_i, c))$.

2. Return $(c_i)_{i \leq s}$.

3.2 Argmax of an encrypted array

We first recall a recent comparator presented by Iliashenko and Zucca in [IZ21], which supports comparison operations for BGV and BFV, and then present our method to compute the index of the maximum of an encrypted array.

3.2.1 Comparison

Essentially, the comparison method for encrypted arrays presented in [IZ21] homomorphically evaluates the Lagrange interpolated polynomial of the $\text{less-than}$ function over $S = \{0, (p - 1)/2\}$ defined as follows:

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

It can be interpolated by the following polynomial over $\mathbb{F}_p$ of degree $p - 1$ by [IZ21, Thm. 3]:

$$\frac{p + 1}{2}(x - y)^{p - 1} + \sum_{i=1, \text{odd}}^{p-2} \left( \sum_{a=1}^{p-1} a^{p-1-i} \right) \cdot (x - y)^i.$$ 

This polynomial can be evaluated within

$$\sqrt{p} - 3 + \frac{3}{2} \log_2(p - 3) + O(1)$$

(2)
multiplicative depth.

3.2.2 Argmax

Based on the less-than function $LT$, we now present an algorithm (Algorithm 3) to compute the maximum index of an encrypted array, denoted $\text{arg max}$.

For a given array $z = (z_1, \cdots, z_s)$, Algorithm 3 firstly computes a comparison matrix $L = (\ell_{i,j})$ with

$$
\ell_{i,j} = \begin{cases} 
1 - LT(z_i, z_j) & \text{if } i < j, \\
1 & \text{if } i = j, \\
LT(z_j, z_i) & \text{if } i > j.
\end{cases}
$$

For instance, if $z = (7, 6, 2, 4, 5)$ the matrix $L \in \{0, 1\}^{5 \times 5}$ is given by

$$
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1
\end{pmatrix}.
$$

It is easy to see that there exists only one row with all entries one, and the index of that row is $\text{arg max}_i(z)$. Equivalently, we have

$$
\text{arg max}_i(z)_{1 \leq i \leq s} = \sum_{j=1}^{s} j \prod_{k=1}^{s} \ell_{j,k},
$$

which results in Algorithm 3.

Note that it requires at most $s(s - 1)/2$ comparisons to construct the comparison matrix $L$. Furthermore, one would better use some recursive methods in practice to compute the encrypted product in Step 2a of Algorithm 3 for saving multiplicative depth. From Eq. (2), Algorithm 3 costs at most

$$\begin{align*}
\log_2 s + \sqrt{p-3} + \frac{3}{2} \log_2 (p-3) + O(1)
\end{align*}$$

multiplicative depth to compute the arg max of an encrypted array.

---

**Algorithm 3** Encrypted maximum index of an encrypted array

Input: $c = (c_1, \cdots, c_s) \in \mathbb{R}_q^s$ ($c_i$ encrypts the $i$-th entry of $z = (z_1, \cdots, z_s)$) and public key $pk$.

Output: A ciphertext $c \in \mathbb{R}_q$ that encrypts $\text{arg max}(z)$.

1. Set $c \leftarrow \text{Enc}_{pk}(0)$.

2. For $i = 1, \cdots, s$ do the following:

   (a) Set $c'$ to be

   $$
   \prod_{k=1,k \neq j}^{s} \text{AddConst}_{pk}(1, \text{MulConst}_{pk}(-1, LT_{pk}(c_j, c_k))).
   $$

   (b) Update $c := \text{ADD}_{pk}(c, \text{MulConst}_{pk}(j, c'))$.

3. Return $c$.

---

4 Privacy-preserving naïve Bayes classification

In this section, we present our privacy-preserving naïve Bayes classifier and prove its correctness and security.

4.1 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 $k$th 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 limit the values of features \( x_1, \ldots, x_n \) in \( \{1, 2, \ldots, t\} \). Furthermore, for numerical stability, we work with the logarithm of the probability:

\[
\mathbf{s}^* = \arg \max_{\mathbf{s} \in \{1, \ldots, t\}^n} \left\{ \log \Pr[Y = i] + \sum_{k=1}^n \log \Pr[X_k = x_k|Y = i] \right\},
\]

where \( x_k \in \{1, \ldots, t\} \). Another convenient simplification is to take the numbering of the \( s \) classes as contiguous integers from 1 to \( s \). Then \( \mathbf{s}^* \) is precisely the index of the maximum over the \( s \) values in (3).

Additionally, since the BGV encryption scheme works with integers, one needs to convert each logarithm of probability in (3) 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., [TRMP12, BPTG13].

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 \( \mathbf{b} = (b_1, \ldots, b_s) \in \mathbb{Z}^s \), 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 \( A_k \in \mathbb{Z}^{n \times t} \) for \( k = 1, \ldots, n \), where the \((i,j)\)-entry of \( A_k \) is obtained by rounding \( K \cdot \log \Pr[X_k = j|Y = i] \) with the same integer \( K \).

### 4.2 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 \( x \).

The client has data \( x = (x_1, \ldots, x_n) \) with \( x_k \in \{1, \ldots, t\} \) and wants the server to predict which class \( x \) is in using a naïve Bayes classifier without learning any information about \( x \). One choice of the client is to encrypt \( x \) using himself’s public key. However, since \( x \) is encrypted, the server cannot decide which entry of \( A_k \) should be chosen. For instance, the first feature of \( x \) is \( x_1 \), i.e., \( X_1 = x_1 \). To access the information about \( \Pr[X_1 = x_1|Y = i] \) in \( A_1 \), we need to select the \((1, x_1)\) entry of \( A_1 \). However, as the first entry of \( x, x_1 \) is only available in encrypted form on the server-side. To get around this obstacle, one can encode the sample \( x \) as a 0-1 matrix

\[
X = (e_{x_1}, \ldots, e_{x_n}) \in \{0, 1\}^{t \times n},
\]

where \( e_j \) is the \( t \)-dimensional vector whose \( j \)-th entry is one and all others are zero. Now, to select the \( x_k \)-th row of a matrix \( A_k \in \mathbb{Z}^{n \times t} \) is just to compute \( A_k \cdot e_{x_k} \). If \( e_{x_k} \) is in encrypted form, this is a plaintext matrix-encrypted vector multiplication discussed in Section 3.

Now we are ready to present our privacy-preserving naïve Bayes classifier as Protocol 1, assuming that FHE achieves circuit privacy. Clearly, the classification phase (Step 3) of Protocol 1 does not require any interactions between the server and the client.

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 as possible from the other party; however, this adversary faithfully follows the prescribed protocol.

**Proposition 1.** Protocol 1 is correct and secure in the honest-but-curious model.

**Proof.** The correctness follows directly from that what the server does is to evaluate the following procedure homomorphically:

1. Set \( y := b \), the information of the prior probability.
2. For \( k = 1, \ldots, n \), set \( y := y + A_k \cdot e_{x_k} \).
3. Return \( y \) as the index of the maximum entry of \( y = (y_0, y_1, \ldots, y_{t-1}) \).

We prove the security by Eq. (1). The client’s view is

\[
V_C = (pk, sk, x; c, y).
\]

The simulator \( S_C \), on input \((pk, sk, x, y')\) with

\[
y' = \arg \max_{i=1,\ldots,t} \left( \sum_{k=1}^n A_k e_{x_k} + b \right),
\]

generates a ciphertext \( c' = Enc_{pk}(z) \), where \( z \) is a random integer and outputs \((pk, sk, x; c', y')\). As the integer \( y \) that the client receives is its output, and as the given FHE scheme is semantically secure and achieves circuit privacy, the distributions \( S_C = (pk, sk, x; c', y') \) and \( V_C = (pk, sk, x; c, y) \) are computationally indistinguishable.

The view of the server is

\[
V_S = ((A_k)_k, b, pk; X', c),
\]

where \( X' \) is ciphertexts that encrypt \( X \). The simulator \( S_S \), on input \((A_k)_k, b, pk\),
generates a random 0-1 matrix $Y$ of size $t \times n$ and computes the ciphertexts $Y'$ that encrypt $Y$,
- generates a ciphertext $c' = \text{Enc}_{pk}(z)$, where $z$ is a random integer,
- outputs $((A_k)_k, b, pk; Y', c')$.

The distributions $V_S$ and $S_S$ are computationally indistinguishable, because of the same reason for $V_C$ and $S_C$. This completes the proof. 

**Protocol 1** Privacy-preserving naïve Bayes classifier

**Input of the client:** A sample $x = (x_1, \cdots, x_n)$ to be classified, the secret and public key $sk$ and $pk$.

**Input of the server:** The model consisting of the likelihood information $(A_k)_{k \leq n}$ and the prior information $b = (b_i)_{i \leq s}$, and the client’s $pk$.

1. The client encodes $x$ to a matrix $X$ as in (4).
2. The client encrypts the column vectors $e_{x_k}$ of $X$ for $k = 1, \cdots, n$ and sends the ciphertexts to the server.
3. The server do the following:
   - (a) For $i = 1, \cdots, s$, set $c_i \leftarrow \text{Enc}_{sk}(0)$ and update $c_i := \text{AddConst}_{pk}(c_i, b_i)$.
   - (b) For $k = 1, \cdots, n$, calling Algorithm 1 or 2 with input as $A_k$, $pk$, and the ciphertexts of $e_{x_k}$ received from the client, outputs $(c'_i)_{i \leq s}$.
   - (c) Update $c_i := \text{Add}_{pk}(c_i, c'_i)$ for $i = 1, \cdots, s$.
   - (d) Calling Algorithm 3 with input as $c = (c_i)_{i \leq s}$ and $pk$ returns $c$.
4. The server sends $c$ to the client.
5. The client decrypts $c$ to $y = \text{Dec}_{sk}(c)$ and outputs $y$.

Protocol 1 assumes that FHE is with circuit privacy. For fully homomorphic encryption schemes, circuit privacy can be achieved using, e.g., the techniques of [DS16]. In practice, the slightly weaker notion of (statistical) function privacy [GHV10] suffices, and is easier to achieve in the leveled fully homomorphic encryption setting using re-randomization and noise flooding, where the server re-randomizes the output ciphertexts by homomorphically adding a ciphertext of zero with a large noise [Gen09, DS16].

## 5 Implementation and experiments

We have implemented three variants of Protocol 1 in C++ using HElib (v2.1.0) [HEI21]. The first two variants come from the different choices in Step 3b of Protocol 1. These two variants deal with only one sample at a time. The third variant comes from the batching technique (see Section 2.3.2), which we call the batching variant. It is based on the same encoding scheme as Algorithm 1. Assume that the selected parameters support $t$ plaintext slots. In the batching variant, we pack the information of a sample into one slot to deal with at most $t$ samples at a time.

In this section, we will report the prediction accuracy, communication cost and calculation time of our implementations of Protocol 1. All experiments run serially (using only one thread) on a laptop with a Ubuntu 20.04 OS as Windows Subsystem for Linux, 2.60 GHz Intel Core i7-10750H CPU (64 bit) with 16 GB RAM.

### 5.1 Data set

The Iris Wisconsin Breast Cancer (WBC), and Lymphography data sets in UCI Machine Learning Repository [DG17] were used in this experiment.

The Iris data set has 150 samples classified into three categories (i.e., $s = 3$). Each sample has four features (i.e., $n = 4$), and each feature takes at most five different values (i.e., $t = 5$). Our experiment used 120 samples (80%) for training the model and the remaining 30 samples for prediction.

For WBC, 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.

For Lymphography, the dataset has 148 samples. The number of categories $s = 4$. There are 18 features for each sample, and each feature may take at most eight different values, i.e., $n = 18$ and $t = 8$. We used 104 samples (72.2%) for training and 44 for testing.
5.2 Parameter setting

For Iris, the scaling factor in Section 4.1 is set to be one, i.e., $K = 1$, which leads that the entries of the rounded logarithm of likelihood $A_k$ for $k = 1, \cdots, 4$ are integers between $-4$ and $0$, and the entries of the rounded logarithm of the prior probability are bounded by $1$. Hence the resulting integers to be compared must be at most $4 \cdot 4 + 1 = 17$, which implies that $p = 37$ is enough for our purpose. In addition, $m$ is fixed to $14539$. In this setting, each plaintext in $R_p$ has $\ell = 1980$ slots.

For WBC, $K$ is also set to be $1$. Hence the entries of the rounded logarithm of likelihood $A_k$ for $k = 1, \cdots, 9$ are integers between $-6$ and $0$, and the entries of the rounded logarithm of the prior probability are bounded by $2$. Hence the resulting integers to be compared must be at most $6 \cdot 9 + 2 = 56$, 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.

For Lymphography, $K = 1$. Similarly, we choose $p = 113, m = 19351$ so that $\ell = 522$ and the security parameter achieves $136$.

5.3 Accuracy

Our experiment shows that the classification accuracy of our implementation of Protocol 1 based on HElib is about $97\%$ for both Iris and WBC and for all three variants. Note that this accuracy is almost the same as the plaintext (unencrypted) naïve Bayes classifier. The accuracy for the Lymphography data set is about $84\%$, which is concordant with the report in [CN87].

5.4 Communication

Although Protocol 1 does not require interactions between the server and client during the classification phase, it does require one interaction, which consists of that the client sends the encrypted data to the server and that the server sends the encrypted result to the client.

<table>
<thead>
<tr>
<th>Table 1: Communication cost (KB) of Protocol 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Iris</td>
</tr>
<tr>
<td>WBC</td>
</tr>
</tbody>
</table>

Table 1 counts the communication cost of the batching variant. The column labeled "Data" gives the size of the ciphertext file for all samples to be classified (i.e., 30 samples for Iris and 205 samples for WBC), and the "Result" column gives the size of the resulting ciphertext for all samples. For Iris and WBC, the amortized communication costs of the batching variant are 22.15K and 46.09KB, respectively. Table 2 compares the transferred data sizes among the batching variant of Protocol 1 and some other existing privacy-preserving naïve Bayes classifiers. The star mark (*) means that the data is taken directly from the corresponding reference. For communication cost, Protocol 1 is not as good as the privacy-preserving naïve Bayes classifier presented in [BPTG15] (based on the QR and Paillier cryptosystems) but is better than that in [YIY19] (based on BGV).

<table>
<thead>
<tr>
<th>Table 2: Total communication cost for each sample of WBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transferred data size (KB)</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>74*</td>
</tr>
</tbody>
</table>

5.5 Timing

In Step 3b of Protocol 1, there are two choices (Algorithm 1 and Algorithm 2) for plaintext matrix-encrypted vector multiplication. We test them all, together with the batching variant, and record their performance. The row named "naïve" ("packed" resp.) is the performance of Protocol 1 based on Algorithm 1 (Algorithm 2 resp.), and the row named "batching" is the performance of the batching variant of Protocol 1 using SIMD. The columns with "Each sample" give the average execution time for each sample. All timings with the star mark (*) are directly taken from the literature, in which the taken timings are the best ones reported. Note that the framework used in the literature may not be the same as in Fig. 1.

It can be observed from Tables 3 and 4 that the naïve variant is more efficient than the packed one, although the timing for encryption is worse. Overall, the batching variant is the most efficient one among the three variants of Protocol 1. The average cost of the batching variant for each sample of Iris and WBC is about 214ms and 34ms,
Table 3: Timing for Iris (s)

<table>
<thead>
<tr>
<th>log q</th>
<th>λ</th>
<th>Enc</th>
<th>Argmax</th>
<th>Dec</th>
<th>Total</th>
<th>Each sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>naïve</td>
<td>387</td>
<td>100</td>
<td>16.641</td>
<td>36.457</td>
<td>3.147</td>
<td>93.778</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.126</td>
</tr>
<tr>
<td>packed</td>
<td>488</td>
<td>74</td>
<td>4.191</td>
<td>32.299</td>
<td>6.09</td>
<td>405.181</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13.506</td>
</tr>
<tr>
<td>batching</td>
<td>387</td>
<td>100</td>
<td>0.556</td>
<td>4.571</td>
<td>0.024</td>
<td>6.423</td>
</tr>
<tr>
<td>[KOH+18]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

respectively. Since the “batching” variant can classify $\ell = 1980$ (3960 resp.) samples simultaneously, the amortized cost in our setting can be less than 4ms (2ms resp.) per sample for Iris (WBC resp.).

Table 4: Timing for WBC (s)

<table>
<thead>
<tr>
<th>log q</th>
<th>λ</th>
<th>Enc</th>
<th>Argmax</th>
<th>Dec</th>
<th>Total</th>
<th>Each sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>naïve</td>
<td>382</td>
<td>100</td>
<td>464.654</td>
<td>338.236</td>
<td>12.555</td>
<td>1332.812</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.502</td>
</tr>
<tr>
<td>packed</td>
<td>476</td>
<td>76</td>
<td>54.299</td>
<td>321.292</td>
<td>23.046</td>
<td>1814.817</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8.853</td>
</tr>
<tr>
<td>batching</td>
<td>382</td>
<td>100</td>
<td>2.376</td>
<td>1.58</td>
<td>0.397</td>
<td>7.033</td>
</tr>
<tr>
<td>[BPTG15]</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[YIY19]</td>
<td>-</td>
<td>119</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>- 0.732</td>
</tr>
<tr>
<td>[WSNK19]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>- 0.402</td>
</tr>
<tr>
<td>[SZL*20]</td>
<td>-</td>
<td>55</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>- 0.141</td>
</tr>
</tbody>
</table>

We also compare our implementation with several existing privacy-preserving naïve Bayes classifiers in Tables 3 and 4. For the Iris data set, Kim et al. [KOH+18] reported that their proposed privacy-preserving naïve Bayes classifier takes 17h40m on a single CPU core and 5h03m on four CPU cores. For WBC, Bost et al. [BPTG15] reported that their classifier takes about 0.479s and 14 interactions per sample. The computation time of the modified protocol presented in [YIY19] is about 0.732s per sample. Wood et al. reported that their classifier takes about 0.402s for each sample. The classifier presented by Sun et al. [SZL*20] takes about 0.141s per sample with SIMD for WBC. Note that the classifiers presented in [YIY19] and [SZL*20] are also implemented using HElib. Overall, although with a not-so-good total cost, Protocol 1 is comparable to existing protocols, especially when the client has a lot of samples to be classified at a time.

5.6 Data set with categories more than three

We also apply our implementation to the Lymphography data set, for which the number of categories $s = 4$. We only give the performance of the batching version in Table 5. Table 5 shows that the total cost increases much compared to the WBC and Iris data sets. The reasons may include the following:

- The number of features for Lymphography is 18, larger than the other two data sets (4 for Iris and 9 for WBC). The more features to be dealt with, the more noise accumulation, and hence the more multiplicative depth we may need. Thus, for correctness, we need to use the ciphertext modulus much larger (See the “log q” column in Table 3–5). However, for the security (say, achieving a security at least 100 bits), with a larger modulus $q$, we have to increase the parameter $m$ so that the polynomial degree $\phi(m)$ increases as well. Since all HE computations are essentially operations on polynomials of degree at most $\phi(m)$. The total cost increases as $\phi(m)$ increases.
- Additionally, Lymphography needs $s(s - 1)/2 = 6$ comparisons, while WBC and Iris only need 1 and 3 comparisons, respectively. From Table 3–5, we can observe that the most costly part of all computations is to homomorphically compute the argmax function. The more categories to be classified, the more time required, concordant with the analysis of Algorithm 3.

Table 5: Timing for Lymphography (s)

<table>
<thead>
<tr>
<th>log q</th>
<th>λ</th>
<th>Enc</th>
<th>Argmax</th>
<th>Dec</th>
<th>Total</th>
<th>Each sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>batching</td>
<td>466</td>
<td>136</td>
<td>6.977</td>
<td>22.845</td>
<td>0.256</td>
<td>38.707</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.880</td>
</tr>
</tbody>
</table>

5.7 The cost of noise management

The number of multiplicative depths is one of the most important parameters to measure the cost of homomorphically encrypted computation. However, HElib (v2.1.0) does not supply a function to count the consumed multiplicative
depths. Instead, the column named log \( q \) in Table 3–5 can be seen as “an analog” of the multiplicative depths since the ciphertext modulus is a product of a set of primes, each with 55–60 bits. During homomorphic computation, it may automatically remove a prime from the set of primes if modulus switching is necessary; see [HS20, Section 5.3] for details. Table 6 counts the theoretical multiplicative depths and the number of bits of ciphertext modulus consumed in practice for the batching variant (i.e., Algorithm 2 is revoked in Step 3b of Protocol 1).

Theoretically, since Algorithm 2 costs no multiplicative depth, the multiplicative depth of Protocol 1 is the same as that of Algorithm 3, i.e. \( \left[ \log_2 \frac{s}{c} \right] + \sqrt{p-3} + \frac{3}{2} \log_2 (p-3) + O(1) \), Table 6 shows that the consumed bits of ciphertext modulus in practice are not proportional to the multiplicative depths in theory. The possible reason is that the noise bound in practice is usually heuristic, while the theoretical bound is usually analyzed from the average case or even from the worst case.

<table>
<thead>
<tr>
<th>Multiplicative depth in theory</th>
<th>Iris</th>
<th>WBC</th>
<th>Lymphography</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumed bits of ciphertext modulus</td>
<td>374</td>
<td>349</td>
<td>454</td>
</tr>
</tbody>
</table>

6 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 leveled fully homomorphic encryption schemes, such as BGV and BFV. 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. According to experiments with our implementation based on HElib, the efficiency of Protocol 1 is comparable to existing ones. An intriguing direction is to extend the framework in Fig. 1 to more classifiers such as decision trees, nearest neighbor classifier, etc.

Declarations

Acknowledgment An early version of this paper was presented in part at the 4th EAI International Conference on Security and Privacy in New Computing Environments [CFL*22]. This research was supported partly by National Key R & D Project of China (2020YFA0712303), Chongqing Science and Technology Program (cstc2021-jcyj-msxmX0821, cstc2020ysxz-jcyj-X0005, cstc2021yszx-jcyjX0004, 2022YSZX-JCX0011CSTB).

Conflicts of interest The authors declare that they have no conflict of interest.

Availability of data and material The datasets generated during and/or analysed during the current study are available in the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml).

Code availability The code is available at https://github.com/velenchan/BGVNaiveBayesPredictor.

Authors contribution statement The authors contributed equally to this work.

References


