RESEARCH

Logistic regression model training based on the approximate homomorphic encryption

Andrey Kim[1†], Yongsoo Song[2*†], Miran Kim[3], Keewoo Lee[1] and Jung Hee Cheon[1]

Abstract

Background: Security concerns have been raised since big data became a prominent tool in data analysis. For instance, many machine learning algorithms aim to generate prediction models using training data which contain sensitive information about individuals. Cryptography community is considering secure computation as a solution for privacy protection. In particular, practical requirements have triggered research on the efficiency of cryptographic primitives.

Methods: This paper presents a method to train a logistic regression model without information leakage. We apply the homomorphic encryption scheme of Cheon et al. (ASIACRYPT 2017) for an efficient arithmetic over real numbers, and devise a new encoding method to reduce storage of encrypted database. In addition, we adapt Nesterov's accelerated gradient method to reduce the number of iterations as well as the computational cost while maintaining the quality of an output classifier.

Results: Our method shows a state-of-the-art performance of homomorphic encryption system in a real-world application. The submission based on this work was selected as the best solution of Track 3 at iDASH privacy and security competition 2017. For example, it took about six minutes to obtain a logistic regression model given the dataset consisting of 1579 samples, each of which has 18 features with a binary outcome variable.

Conclusions: We present a practical solution for outsourcing analysis tools such as logistic regression analysis while preserving the data confidentiality.

Keywords: homomorphic encryption; machine learning; logistic regression

Background

Machine learning (ML) is a class of methods in artificial intelligence, the characteristic feature of which is that they do not give the solution of a particular problem but they learn the process of finding solutions to a set of similar problems. The theory of ML appeared in the early 60's on the basis of the achievements of cybernetics [1] and gave the impetus to the development of theory and practice of technically complex learning systems [2]. The goal of ML is to partially or fully automate the solution of complicated tasks in various fields of human activity.

The scope of ML applications is constantly expanding; however, with the rise of ML, the security problem has become an important issue. For example, many medical decisions rely on logistic regression model, and biomedical data usually contain confidential information about individuals [3] which should be treated carefully. Therefore, privacy and security of data are the major concerns, especially when deploying the outsource analysis tools.

There have been several researches on secure computation based on cryptographic primitives. Nikolaenko et al. [4] presented a privacy preserving linear regression protocol on horizontally partitioned data using Yao's garbled circuits [5]. Multi-party computation technique was also applied to privacy-preserving logistic regression [6, 7, 8]. However, this approach is vulnerable when a party behaves dishonestly, and the assumption for secret sharing is quite different from that of outsourcing computation.

Homomorphic encryption (HE) is a cryptosystem that allows us to perform certain arithmetic operations on encrypted data and receive an encrypted result that corresponds to the result of operations performed in plaintext. Several papers already discussed ML with HE techniques. Wu et al. [9] used Paillier cryptosys-

*Correspondence: yongsoosong@ucsd.edu
[2]Department of Computer Science and Engineering, University of California-San Diego, 9500 Gillman Drive, 92093-0404 San Diego, California, USA
Full list of author information is available at the end of the article
†Equal contributor

tem [10] and approximated the logistic function using polynomials, but it required an exponentially growing computational cost in the degree of the approximation polynomial. Aono et al. [11] and Xie et al. [12] used an additive HE scheme to aggregate some intermediate statistics. However, the scenario of Aono et al. relies on the client to decrypt these intermediary statistics and the method of Xie et al. requires expensive computational cost to calculate the intermediate information. The most related research of this paper is the work of Kim et al. [13] which also used HE based ML. However, the size of encrypted data and learning time were highly dependent on the number of features, so the performance for a large dataset was not practical in terms of storage and computational cost.

Since 2011, the iDASH Privacy and Security Workshop has assembled specialists in privacy technology to discuss issues that apply to biomedical data sharing, as well as main stakeholders who provided an overview of the main uses of the data, different laws and regulations, and their own views on privacy. In addition, it has began to hold annual competitions on the basis of the workshop from 2014. The goal of this challenge is to evaluate the performance of state-of-the-arts methods that ensures rigorous data confidentiality during data analysis in a cloud environment.

In this paper, we provide a solution to the third track of iDASH 2017 competition, which aims to develop HE based secure solutions for building a ML model (i.e., logistic regression) on encrypted data. We propose a general practical solution for HE based ML that demonstrates good performance and low storage costs. In practice, our output quality is comparable to the one of an unencrypted learning case. As a basis, we use the HE scheme for approximate arithmetic [14]. To improve the performance, we apply several additional techniques including a packing method, which reduce the required storage space and optimize the computational time. We also adapt Nesterov's accelerated gradient [15] to increase the speed of convergence. As a result, we could obtain a high-accuracy classifier using only a small number of iterations.

We give an open-source implementation [16] to demonstrate the performance of our HE based ML method. With our packing method we can encrypt the dataset with 1579 samples and 18 features using 39MB of memory. The encrypted learning time is about six minutes. We also demonstrate our implementation on the datasets used in [13] to compare the results. For example, the training of a logistic regression model took about 3.6 minutes with the storage about 0.02GB compared to 114 minutes and 0.69GB of Kim et al. [13] when a dataset consists of 1253 samples, each of which has 9 features.

Methods

Logistic Regression

Logistic regression or logit model is a ML model used to predict the probability of occurrence of an event by fitting data to a logistic curve [17]. It is widely used in various fields including machine learning, biomedicine [18], genetics [19], and social sciences [20].

Throughout this paper, we treat the case of a binary dependent variable, represented by ± 1. Learning data consists of pairs $(\mathbf{x}_i, y_i)$ of a vector of co-variates $\mathbf{x}_i = (x_{i1}, ..., x_{if}) \in \mathbb{R}^f$ and a dependent variable $y_i \in \{\pm 1\}$. Logistic regression aims to find an optimal $\boldsymbol{\beta} \in \mathbb{R}^{f+1}$ which maximizes the likelihood estimator

$$\prod_{i=1}^{n} \Pr(y_i|\mathbf{x}_i) = \prod_{i=1}^{n} \frac{1}{1+\exp(-y_i(1,\mathbf{x}_i)^T\boldsymbol{\beta})},$$

or equivalently minimizes the loss function, defined as the negative log-likelihood:

$$J(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-\mathbf{z}_i^T\boldsymbol{\beta}))$$

where $\mathbf{z}_i = y_i \cdot (1, \mathbf{x}_i)$ for $i = 1, \ldots, n$.

Gradient Descent

Gradient Descent (GD) is a method for finding a local extremum (minimum or maximum) of a function by moving along gradients. To minimize the function in the direction of the gradient, one-dimensional optimization methods are used.

For logistic regression, the gradient of the cost function with respect to $\boldsymbol{\beta}$ is computed by

$$\nabla J(\boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=1}^{n} \sigma(-\mathbf{z}_i^T\boldsymbol{\beta}) \cdot \mathbf{z}_i$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$. Starting from an initial $\boldsymbol{\beta}_0$, the gradient descent method at each step t updates the regression parameters using the equation

$$\boldsymbol{\beta}^{(t+1)} \leftarrow \boldsymbol{\beta}^{(t)} + \frac{\alpha_t}{n}\sum_{i=1}^{n} \sigma(-\mathbf{z}_i^T\boldsymbol{\beta}^{(t)}) \cdot \mathbf{z}_i$$

where α_t is a learning rate at step t.

Nesterov's Accelerated Gradient

The method of GD can face a problem of zig-zagging along a local optima and this behavior of the method becomes typical if it increases the number of variables of an objective function. Many GD optimization algorithms are widely used to overcome this phenomenon.

Momentum method, for example, dampens oscillation using the accumulated exponential moving average for the gradient of the loss function.

Nesterov's accelerated gradient [15] is a slightly different variant of the momentum update. It uses moving average on the update vector and evaluates the gradient at this "looked-ahead" position. It guarantees a better rate of convergence $O(1/t^2)$ (vs. $O(1/t)$ of standard GD algorithm) after t steps theoretically, and consistently works slightly better in practice. Starting with a random initial $\mathbf{v}_0 = \boldsymbol{\beta}_0$, the updated equations for Nesterov's Accelerated GD are as follows:

$$\begin{cases} \boldsymbol{\beta}^{(t+1)} &= \mathbf{v}^{(t)} - \alpha_t \cdot \bigtriangledown J(\mathbf{v}^{(t)}), \\ \mathbf{v}^{(t+1)} &= (1-\gamma_t)\cdot \boldsymbol{\beta}^{(t+1)} + \gamma_t \cdot \boldsymbol{\beta}^{(t)}, \end{cases} \tag{1}$$

where $0 < \gamma_t < 1$ is a moving average smoothing parameter.

Approximate Homomorphic Encryption

HE is a cryptographic scheme that allows us to carry out operations on encrypted data without decryption. Cheon et al. [14] presented a method to construct a HE scheme for arithmetic of approximate numbers (called `HEAAN` in what follows). The main idea is to treat an encryption noise as part of error occurring during approximate computations. That is, an encryption ct of message $m \in \mathcal{R}$ by a secret key sk for a ciphertext modulus q will have a decryption structure of the form $\langle \mathsf{ct}, \mathsf{sk} \rangle = m + e \pmod q$ for some small e.

The following is a simple description of `HEAAN` based on the ring learning with errors problem. For a power-of-two integer N, the cyclotomic polynomial ring of dimension N is denoted by $\mathcal{R} = \mathbb{Z}[X]/(X^N+1)$. For a positive integer ℓ, we denote $\mathcal{R}_\ell = \mathcal{R}/2^\ell\mathcal{R} = \mathbb{Z}_{2^\ell}[X]/(X^N+1)$ the residue ring of $\mathcal{R}$ modulo 2^ℓ.

- $\underline{\mathtt{KeyGen}(1^\lambda)}$.
  - For an integer L that corresponds to the largest ciphertext modulus level, given the security parameter λ, output the ring dimension N which is a power of two.
  - Set the small distributions $\chi_{key}, \chi_{err}, \chi_{enc}$ over $\mathcal{R}$ for secret, error, and encryption, respectively.
  - Sample a secret $s \leftarrow \chi_{key}$, a random $a \leftarrow \mathcal{R}_L$ and an error $e \leftarrow \chi_{err}$. Set the secret key as $\mathsf{sk} \leftarrow (1, s)$ and the public key as $\mathsf{pk} \leftarrow (b, a) \in \mathcal{R}_L^2$ where $b \leftarrow -as + e \pmod{2^L}$.
- $\underline{\mathtt{KSGen}_{\mathsf{sk}}(s')}$. For $s' \in \mathcal{R}$, sample a random $a' \leftarrow \mathcal{R}_{2\cdot L}$ and an error $e' \leftarrow \chi_{err}$. Output the switching key as $\mathsf{swk} \leftarrow (b', a') \in \mathcal{R}_{2\cdot L}^2$ where $b' \leftarrow -a's + e' + 2^L s' \pmod{2^{2\cdot L}}$.
  - Set the evaluation key as $\mathsf{evk} \leftarrow \mathtt{KSGen}_{\mathsf{sk}}(s^2)$.
- $\underline{\mathtt{Enc}_{\mathsf{pk}}(m)}$. For $m \in \mathcal{R}$, sample $v \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi_{err}$. Output $v \cdot \mathsf{pk} + (m + e_0, e_1) \pmod{2^L}$.
- $\underline{\mathtt{Dec}_{\mathsf{sk}}(\mathsf{ct})}$. For $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_\ell^2$, output $c_0 + c_1 \cdot s \pmod{2^\ell}$.
- $\underline{\mathtt{Add}(\mathsf{ct}_1, \mathsf{ct}_2)}$. For $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{R}_\ell^2$, output $\mathsf{ct}_{\mathsf{add}} \leftarrow \mathsf{ct}_1 + \mathsf{ct}_2 \pmod{2^\ell}$.
- $\underline{\mathtt{CMult}_{\mathsf{evk}}(\mathsf{ct}; c)}$. For $\mathsf{ct} \in \mathcal{R}_\ell^2$ and $a \in \mathcal{R}$, output $\mathsf{ct}' \leftarrow c \cdot \mathsf{ct} \pmod{2^\ell}$.
- $\underline{\mathtt{Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)}$. For $\mathsf{ct}_1 = (b_1, a_1), \mathsf{ct}_2 = (b_2, a_2) \in \mathcal{R}_\ell^2$, let $(d_0, d_1, d_2) = (b_1 b_2,\ a_1 b_2 + a_2 b_1, a_1 a_2) \pmod{2^\ell}$. Output $\mathsf{ct}_{\mathsf{mult}} \leftarrow (d_0, d_1) + \lfloor 2^{-L} \cdot d_2 \cdot \mathsf{evk} \rceil \pmod{2^\ell}$.
- $\underline{\mathtt{ReScale}(\mathsf{ct}; p)}$. For a ciphertext $\mathsf{ct} \in \mathcal{R}_\ell^2$ and an integer p, output $\mathsf{ct}' \leftarrow \lfloor 2^{-p} \cdot \mathsf{ct} \rceil \pmod{2^{\ell-p}}$.

For a power-of-two integer $k \le N/2$, `HEAAN` provides a technique to pack k complex numbers in a single polynomial using a variant of the complex canonical embedding map $\phi : \mathbb{C}^k \to \mathcal{R}$. We restrict the plaintext space as a vector of real numbers throughout this paper. Moreover, we multiply a scale factor of 2^p to plaintexts before the rounding operation to maintain their precision.

- $\underline{\mathtt{Encode}(\mathbf{w}; p)}$. For $\mathbf{w} \in \mathbb{R}^k$, output the polynomial $m \leftarrow \phi(2^p \cdot \mathbf{w}) \in \mathcal{R}$.
- $\underline{\mathtt{Decode}(m; p)}$. For a plaintext $m \in \mathcal{R}$, the encoding of an array consisting of a power of two $k \le N/2$ messages, output the vector $\mathbf{w} \leftarrow \phi^{-1}(m/2^p) \in \mathbb{R}^k$.

The encoding/decoding techniques support the parallel computation over encryption, yielding a better amortized timing. In addition, the `HEAAN` scheme provides the rotation operation on plaintext slots, i.e., it enables us to securely obtain an encryption of the shifted plaintext vector $(w_r, \ldots, w_{k-1}, w_0, \ldots, w_{r-1})$ from an encryption of $(w_0, \ldots, w_{k-1})$. It is necessary to generate an additional public information rk, called the rotation key. We denote the rotation operation as follows.

- $\underline{\mathtt{Rotate}_{\mathsf{rk}}(\mathsf{ct}; r)}$. For the rotation keys rk, output a ciphertext ct' encrypting the rotated plaintext vector of ct by r positions.

Refer [14] for the technical details and noise analysis.

Database Encoding

For an efficient computation, it is crucial to find a good encoding method for the given database. The `HEAAN` scheme supports the encryption of a plaintext vector and the slot-wise operations over encryption. However, our learning data is represented by a matrix $(z_{ij})_{1\le i\le n, 0\le j\le f}$. A recent work [13] used the column-wise approach, i.e., a vector of specific feature data

$(z_{ij})_{1\le i\le n}$ is encrypted in a single ciphertext. Consequently, this method required $(f+1)$ number of ciphertexts to encrypt the whole dataset.

In this subsection, we suggest a more efficient encoding method to encrypt a *matrix* in a single ciphertext. A training dataset consists of n samples $\mathbf{z}_i \in \mathbb{R}^{f+1}$ for $1 \le i \le n$, which can be represented as a matrix Z as follows:

$$Z = \begin{bmatrix} z_{10} & z_{11} & \cdots & z_{1f} \\ z_{20} & z_{21} & \cdots & z_{1f} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n0} & z_{n1} & \cdots & z_{nf} \end{bmatrix}.$$

For simplicity, we assume that n and $(f+1)$ are power-of-two integers satisfying $\log n + \log(f+1) \le \log(N/2)$. Then we can pack the whole matrix in a single ciphertext in a row-by-row manner. Specifically, we will identify this matrix with the k-dimensional vector by $(z_{ij})_{1\le i\le n, 0\le j\le f} \mapsto \mathbf{w} = (w_\ell)_{0\le \ell < n\cdot(f+1)}$ where $w_\ell = z_{ij}$ such that $\ell = (f+1)(i-1)+j$, that is,

$$Z \mapsto \mathbf{w} = (z_{10}, \ldots, z_{1f}, z_{20}, \ldots, z_{2f}, \ldots, z_{n0}, \ldots, z_{nf}).$$

In a general case, we can pad zeros to set the number of samples and the dimension of a weight vector as powers of two.

It is necessary to perform shifting operations of row and column vectors for the evaluation of the GD algorithm. In the rest of this subsection, we explain how to perform these operations using the rotation algorithm provided in the `HEAAN` scheme. As described above, the algorithm $\mathtt{Rotate}(\mathtt{ct}; r)$ can shift the encrypted vector by r positions. In particular, this operation is useful in our implementation when $r = f+1$ or $r = 1$. For the first case, a given matrix $Z = (z_{ij})_{1\le i\le n, 0\le j\le f}$ is converted into the matrix

$$Z' = \begin{bmatrix} z_{20} & z_{21} & \cdots & z_{2f} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n0} & z_{n1} & \cdots & z_{nf} \\ z_{10} & z_{11} & \cdots & z_{1f} \end{bmatrix},$$

while the latter case outputs the matrix

$$Z'' = \begin{bmatrix} z_{11} & \cdots & z_{1f} & z_{20} \\ z_{21} & \cdots & z_{2f} & z_{30} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n1} & \cdots & z_{nf} & z_{10} \end{bmatrix}$$

over encryption. The matrix Z' is obtained from Z by shifting its row vectors and Z'' can be viewed as an *incomplete* column shifting because of its last column.

Polynomial Approximation of the Sigmoid Function

One limitation of the existing HE cryptosystems is that they only support polynomial arithmetic operations. The evaluation of the sigmoid function is an obstacle for the implementation of the logistic regression since it cannot be expressed as a polynomial.

Kim et al. [13] used the least squares approach to find a global polynomial approximation of the sigmoid function. We adapt this approximation method and consider the degree 3, 5, and 7 least squares polynomials of the sigmoid function over the domain $[-8, 8]$. We observed that the inner product values $\mathbf{z}_i^T\boldsymbol{\beta}^{(t)}$ in our experimentations belong to this interval. For simplicity, a least squares polynomial of $\sigma(-x)$ will be denoted by $g(x)$ so that we have $g(\mathbf{z}_i^T\boldsymbol{\beta}^{(t)}) \approx \sigma(-\mathbf{z}_i^T\boldsymbol{\beta}^{(t)})$ when $|\mathbf{z}_i^T\boldsymbol{\beta}^{(t)}| \le 8$. The approximate polynomials $g(x)$ of degree 3, 5, and 7 are computed as follows:

$$\begin{cases} g_3(x) &= 0.5 - 1.20096\cdot(x/8) + 0.81562\cdot(x/8)^3, \\ g_5(x) &= 0.5 - 1.53048\cdot(x/8) + 2.3533056\cdot(x/8)^3 \\ &\quad -1.3511295\cdot(x/8)^5, \\ g_7(x) &= 0.5 - 1.73496\cdot(x/8) + 4.19407\cdot(x/8)^3 \\ &\quad -5.43402\cdot(x/8)^5 + 2.50739\cdot(x/8)^7. \end{cases}$$

A low-degree polynomial requires a smaller evaluation depth while a high-degree polynomial has a better precision. The maximum errors between $\sigma(-x)$ and the least squares $g_3(x)$, $g_5(x)$, and $g_7(x)$ are approximately 0.114, 0.061 and 0.032, respectively.

Homomorphic Evaluation of the Gradient Descent

This section explains how to securely train the logistic regression model using the `HEAAN` scheme. To be precise, we explicitly describe a full pipeline of the evaluation of the GD algorithm. We adapt the same assumptions as in the previous section so that the whole database can be encrypted in a single ciphertext.

First of all, a client encrypts the dataset and the initial (random) weight vector $\boldsymbol{\beta}^{(0)}$ and sends them to the public cloud. The dataset is encoded to a matrix Z of size $n \times (f+1)$ and the weight vector is copied n times to fill the plaintext slots. The plaintext matrices of the resulting ciphertexts are described as follows:

$$\mathtt{ct}_z = \mathtt{Enc}\begin{bmatrix} z_{10} & z_{11} & \cdots & z_{1f} \\ z_{20} & z_{21} & \cdots & z_{1f} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n0} & z_{n1} & \cdots & z_{nf} \end{bmatrix},$$

$$\mathtt{ct}_\beta^{(0)} = \mathtt{Enc}\begin{bmatrix} \beta_0^{(0)} & \beta_1^{(0)} & \cdots & \beta_f^{(0)} \\ \beta_0^{(0)} & \beta_1^{(0)} & \cdots & \beta_f^{(0)} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_0^{(0)} & \beta_1^{(0)} & \cdots & \beta_f^{(0)} \end{bmatrix}.$$

As mentioned before, both Z and $\boldsymbol{\beta}^{(0)}$ are scaled by a factor of 2^p before encryption to maintain the precision of plaintexts. We skip to mention the scaling factor in the rest of this section since every step will return a ciphertext with the scaling factor of 2^p.

The public server takes two ciphertexts $\mathtt{ct}_z$ and $\mathtt{ct}_\beta^{(t)}$ and evaluates the GD algorithm to find an optimal modeling vector. The goal of each iteration is to update the modeling vector $\boldsymbol{\beta}^{(t)}$ using the gradient of loss function:

$$\boldsymbol{\beta}^{(t+1)} \leftarrow \boldsymbol{\beta}^{(t)} + \frac{\alpha_t}{n} \sum_{i=1}^{n} \sigma(-\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}) \cdot \mathbf{z}_i$$

where α_t denotes the learning rate at the t-th iteration. Each iteration consists of the following eight steps.

Step 1: For given two ciphertexts $\mathtt{ct}_z$ and $\mathtt{ct}_\beta^{(t)}$, compute their multiplication and rescale it by p bits:

$$\mathtt{ct}_1 \leftarrow \mathtt{ReScale}(\mathtt{Mult}(\mathtt{ct}_\beta^{(t)}, \mathtt{ct}_z); p).$$

The output ciphertext contains the values $z_{ij} \cdot \beta_j^{(t)}$ in its plaintext slots, i.e.,

$$\mathtt{ct}_1 = \mathtt{Enc} \begin{bmatrix} z_{10} \cdot \beta_0^{(t)} & z_{11} \cdot \beta_1^{(t)} & \cdots & z_{1f} \cdot \beta_f^{(t)} \\ z_{20} \cdot \beta_0^{(t)} & z_{21} \cdot \beta_1^{(t)} & \cdots & z_{1f} \cdot \beta_f^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n0} \cdot \beta_0^{(t)} & z_{n1} \cdot \beta_1^{(t)} & \cdots & z_{nf} \cdot \beta_f^{(t)} \end{bmatrix}.$$

Step 2: To obtain the inner product $\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}$, the public cloud aggregates the values of $z_{ij}\beta_j^{(t)}$ in the same row. This step can be done by adapting the incomplete column shifting operation.

One simple way is to repeat this operation $(f+1)$ times, but the computational cost can be reduced down to $\log(f+1)$ by adding $\mathtt{ct}_1$ to its rotations recursively:

$$\mathtt{ct}_1 \leftarrow \mathtt{Add}(\mathtt{ct}_1, \mathtt{Rotate}(\mathtt{ct}_1; 2^j)),$$

for $j = 0, 1, \ldots, \log(f+1) - 1$. Then the output ciphertext $\mathtt{ct}_2$ encrypts the inner product values $\mathbf{z}_i^T \boldsymbol{\beta}^{(t)}$ in the first column and some "garbage" values in the other columns, denoted by $\star$, i.e.,

$$\mathtt{ct}_2 = \mathtt{Enc} \begin{bmatrix} \mathbf{z}_1^T \boldsymbol{\beta}^{(t)} & \star & \cdots & \star \\ \mathbf{z}_2^T \boldsymbol{\beta}^{(t)} & \star & \cdots & \star \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{z}_n^T \boldsymbol{\beta}^{(t)} & \star & \cdots & \star \end{bmatrix}.$$

Step 3: This step performs a constant multiplication in order to annihilate the garbage values. It can be obtained by computing the encoding polynomial $c \leftarrow \mathtt{Encode}(C; p_c)$ of the matrix

$$C = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{bmatrix},$$

using the scaling factor of 2^{p_c} for some integer p_c. The parameter p_c is chosen as the bit precision of plaintexts so it can be smaller than the parameter p.

Finally we multiply the polynomial c to the ciphertext $\mathtt{ct}_2$ and rescale it by p_c bits:

$$\mathtt{ct}_3 \leftarrow \mathtt{ReScale}(\mathtt{CMult}(\mathtt{ct}_2; c); p_c).$$

The garbage values are multiplied with zero while one can maintain the inner products in the plaintext slots. Hence the output ciphertext $\mathtt{ct}_3$ encrypts the inner product values in the first column and zeros in the others:

$$\mathtt{ct}_3 = \mathtt{Enc} \begin{bmatrix} \mathbf{z}_1^T \boldsymbol{\beta}^{(t)} & 0 & \cdots & 0 \\ \mathbf{z}_2^T \boldsymbol{\beta}^{(t)} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{z}_n^T \boldsymbol{\beta}^{(t)} & 0 & \cdots & 0 \end{bmatrix}.$$

Step 4: The goal of this step is to replicate the inner product values to other columns. Similar to Step 2, it can be done by adding the input ciphertext to its column shifting recursively, but in the opposite direction

$$\mathtt{ct}_3 \leftarrow \mathtt{Add}(\mathtt{ct}_3, \mathtt{Rotate}(\mathtt{ct}_3; -2^j))$$

for $j = 0, 1, \ldots, \log(f+1) - 1$. The output ciphertext $\mathtt{ct}_4$ has the same inner product value in each row:

$$\mathtt{ct}_4 = \mathtt{Enc} \begin{bmatrix} \mathbf{z}_1^T \boldsymbol{\beta}^{(t)} & \mathbf{z}_1^T \boldsymbol{\beta}^{(t)} & \cdots & \mathbf{z}_1^T \boldsymbol{\beta}^{(t)} \\ \mathbf{z}_2^T \boldsymbol{\beta}^{(t)} & \mathbf{z}_2^T \boldsymbol{\beta}^{(t)} & \cdots & \mathbf{z}_2^T \boldsymbol{\beta}^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{z}_n^T \boldsymbol{\beta}^{(t)} & \mathbf{z}_n^T \boldsymbol{\beta}^{(t)} & \cdots & \mathbf{z}_n^T \boldsymbol{\beta}^{(t)} \end{bmatrix}.$$

Step 5: This step simply evaluates an approximating polynomial of the sigmoid function, i.e., $\mathtt{ct}_5 \leftarrow g(\mathtt{ct}_4)$ for some $g \in \{g_3, g_5, g_7\}$. The output ciphertext encrypts the values of $g(\mathbf{z}_i^T \boldsymbol{\beta}^{(t)})$ in its plaintext slots:

$$\mathtt{ct}_5 = \mathtt{Enc} \begin{bmatrix} g(\mathbf{z}_1^T \boldsymbol{\beta}^{(t)}) & \cdots & g(\mathbf{z}_1^T \boldsymbol{\beta}^{(t)}) \\ g(\mathbf{z}_2^T \boldsymbol{\beta}^{(t)}) & \cdots & g(\mathbf{z}_2^T \boldsymbol{\beta}^{(t)}) \\ \vdots & \ddots & \vdots \\ g(\mathbf{z}_n^T \boldsymbol{\beta}^{(t)}) & \cdots & g(\mathbf{z}_n^T \boldsymbol{\beta}^{(t)}) \end{bmatrix}.$$

Step 6: The public cloud multiplies the ciphertext ct_5 with the encrypted dataset ct_z and rescales the resulting ciphertext by p bits:

$$\mathsf{ct}_6 \leftarrow \mathtt{ReScale}(\mathtt{Mult}(\mathsf{ct}_5, \mathsf{ct}_z); p).$$

The output ciphertext encrypts the n vectors $g(\mathbf{z}_i^T\boldsymbol{\beta}^{(t)}) \cdot \mathbf{z}_i$ in each row:

$$\mathsf{ct}_6 = \mathtt{Enc}\begin{bmatrix} g(\mathbf{z}_1^T\boldsymbol{\beta}^{(t)}) \cdot z_{10} & \cdots & g(\mathbf{z}_1^T\boldsymbol{\beta}^{(t)}) \cdot z_{1f} \\ g(\mathbf{z}_2^T\boldsymbol{\beta}^{(t)}) \cdot z_{20} & \cdots & g(\mathbf{z}_2^T\boldsymbol{\beta}^{(t)}) \cdot z_{2f} \\ \vdots & \ddots & \vdots \\ g(\mathbf{z}_n^T\boldsymbol{\beta}^{(t)}) \cdot z_{n0} & \cdots & g(\mathbf{z}_n^T\boldsymbol{\beta}^{(t)}) \cdot z_{nf} \end{bmatrix}.$$

Step 7: This step aggregates the vectors $g(\mathbf{z}_i^T\boldsymbol{\beta}^{(t)})$ to compute the gradient of the loss function. It is obtained by recursively adding ct_6 to its row shifting:

$$\mathsf{ct}_6 \leftarrow \mathtt{Add}(\mathsf{ct}_6, \mathtt{Rotate}(\mathsf{ct}_6; 2^j))$$

for $j = \log(f+1), \ldots, \log(f+1)+\log n-1$. The output ciphertext is

$$\mathsf{ct}_7 = \mathtt{Enc}\begin{bmatrix} \sum_i g(\mathbf{z}_i^T\boldsymbol{\beta}^{(t)}) \cdot z_{i0} & \cdots & \sum_i g(\mathbf{z}_i^T\boldsymbol{\beta}^{(t)}) \cdot z_{if} \\ \sum_i g(\mathbf{z}_i^T\boldsymbol{\beta}^{(t)}) \cdot z_{i0} & \cdots & \sum_i g(\mathbf{z}_i^T\boldsymbol{\beta}^{(t)}) \cdot z_{if} \\ \vdots & \ddots & \vdots \\ \sum_i g(\mathbf{z}_i^T\boldsymbol{\beta}^{(t)}) \cdot z_{i0} & \cdots & \sum_i g(\mathbf{z}_i^T\boldsymbol{\beta}^{(t)}) \cdot z_{if} \end{bmatrix},$$

as desired.

Step 8: For the learning rate α_t, it uses the parameter p_c to compute the scaled learning rate $\Delta^{(t)} = \lfloor 2^{p_c} \cdot \alpha_t \rceil$. The public cloud updates $\boldsymbol{\beta}^{(t)}$ using the ciphertext ct_7 and the constant $\Delta^{(t)}$:

$$\mathsf{ct}_8 \leftarrow \mathtt{ReScale}(\Delta^{(t)} \cdot \mathsf{ct}_7; p_c),$$
$$\mathsf{ct}_\beta^{(t+1)} \leftarrow \mathtt{Add}(\mathsf{ct}_\beta^{(t)}, \mathsf{ct}_8).$$

Finally it returns a ciphertext encrypting the updated modeling vector

$$\mathsf{ct}_\beta^{(t+1)} = \mathtt{Enc}\begin{bmatrix} \beta_0^{(t+1)} & \beta_1^{(t+1)} & \cdots & \beta_f^{(t+1)} \\ \beta_0^{(t+1)} & \beta_1^{(t+1)} & \cdots & \beta_f^{(t+1)} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_0^{(t+1)} & \beta_1^{(t+1)} & \cdots & \beta_f^{(t+1)} \end{bmatrix}.$$

where $\beta_j^{(t+1)} = \beta_j^{(t)} + \frac{\alpha_t}{n}\sum_i g(\mathbf{z}_i^T\boldsymbol{\beta}^{(t)}) \cdot z_{ij}$.

Homomorphic Evaluation of Nesterov's Accelerated Gradient

The performance of leveled HE schemes highly depends on the depth of a circuit to be evaluated. The bottleneck of homomorphic evaluation of the GD algorithm is that we need to repeat the update of weight vector $\boldsymbol{\beta}^{(t)}$ iteratively. Consequently, the total depth grows linearly on the number of iterations and it should be minimized for practical implementation.

For the homomorphic evaluation of Nesterov's accelerated gradient, a clients sends one more ciphertext $\mathsf{ct}_v^{(0)}$ encrypting the initial vector $\mathbf{v}^{(0)}$ to the public cloud. Then the server uses an encryption ct_z of dataset Z to update two ciphertexts $\mathbf{v}^{(t)}$ and $\mathsf{ct}_\beta^{(t)}$ at each iteration. One can securely compute $\boldsymbol{\beta}^{(t+1)}$ in the same way as the previous section. Nesterov's accelerated gradient requires one more step to compute the second equation of (1) and obtain an encryption of $\mathbf{v}^{(t+1)}$ from $\mathsf{ct}_\beta^{(t)}$ and $\mathsf{ct}_\beta^{(t+1)}$.

Step 9: Let $\Delta_1^{(t)} = \lfloor 2^{p_c} \cdot \gamma_t \rceil$ and let $\Delta_2^{(t)} = 2^{p_c} - \Delta_1^{(t)}$. It obtains the ciphertext $\mathsf{ct}_v^{(t+1)}$ by computing

$$\mathsf{ct}_v^{(t+1)} \leftarrow \mathtt{Add}(\Delta_2^{(t)} \cdot \mathsf{ct}_\beta^{(t+1)}, \Delta_1^{(t)} \cdot \mathsf{ct}_\beta^{(t)}),$$
$$\mathsf{ct}_v^{(t+1)} \leftarrow \mathtt{ReScale}(\mathsf{ct}_v^{(t+1)}; p_c).$$

Then the output ciphertext is

$$\mathsf{ct}_v^{(t+1)} = \mathtt{Enc}\begin{bmatrix} v_0^{(t+1)} & v_1^{(t+1)} & \cdots & v_f^{(t+1)} \\ v_0^{(t+1)} & v_1^{(t+1)} & \cdots & v_f^{(t+1)} \\ \vdots & \vdots & \ddots & \vdots \\ v_0^{(t+1)} & v_1^{(t+1)} & \cdots & v_f^{(t+1)} \end{bmatrix},$$

which encrypts $v_j^{(t+1)} = (1-\gamma_t) \cdot \beta_j^{(t+1)} + \gamma_t \cdot \beta_j^{(t)}$ in the plaintext slots.

Results

In this section, we present parameter sets with experimental results. Our implementation is based on the HEAAN library [21] that implements the approximate HE scheme of Cheon et al. [14]. The source code is publicly available at `github` [16].

Parameters settings

We explain how to choose the parameter sets for the homomorphic evaluation of the (Nesterov's) GD algorithm with security analysis. We start with the parameter L - the bitsize of a fresh ciphertext modulus. The modulus of a ciphertext is reduced after the `ReScale` operations and the evaluation of an approximate polynomial $g(x)$.

The `ReScale` procedures after homomorphic multiplications (step 1 and 6) reduce the ciphertext modulus by p bits while the `ReScale` procedures after constant multiplications (step 3 and 8) require p_c bits of

Table 1 Implementation results for iDASH dataset with 10-fold CV

Sample Num	Feature Num	$\deg g$	Iter Num	Enc Time	Learn Time	Storage	Accuracy	AUC
1579	18	3	9	4s	7.94min	0.04GB	61.72%	0.677
		5	7	4s	6.07min	0.04GB	62.87%	0.689
		7	7	4s	7.01min	0.04GB	62.36%	0.689

Table 2 Implementation results for other datasets with 5-fold CV

Dataset	Sample Num	Feature Num	Method	$\deg g$	Iter Num	Enc Time	Learn Time	Storage	Accuracy	AUC
Edinburgh	1253	9	Ours	5	7	2s	3.6min	0.02GB	91.04%	0.958
			[13]	3	25	12s	114min	0.69GB	86.03%	0.956
			[13]	7	20	12s	114min	0.71GB	86.19%	0.954
lbw	189	9	Ours	5	7	2s	3.3min	0.02GB	69.19%	0.689
			[13]	3	25	11s	99min	0.67GB	69.30%	0.665
			[13]	7	20	11s	86min	0.70GB	69.29%	0.678
nhanes3	15649	15	Ours	5	7	14s	7.3min	0.16GB	79.22%	0.717
			[13]	3	25	21s	235min	1.15GB	79.23%	0.732
			[13]	7	20	21s	208min	1.17GB	79.23%	0.737
pcs	379	9	Ours	5	7	2s	3.5min	0.02GB	68.27%	0.740
			[13]	3	25	11s	103min	0.68GB	68.85%	0.742
			[13]	7	20	11s	97min	0.70GB	69.12%	0.750
uis	575	8	Ours	5	7	2s	3.5min	0.02GB	74.44%	0.603
			[13]	3	25	10s	104min	0.61GB	74.43%	0.585
			[13]	7	20	10s	96min	0.63GB	75.43%	0.617

modulus reduction. Note that the ciphertext modulus remains the same for the step 9 for the Nesterov's accelerated gradient if we compute step 8 and 9 together using some precomputed constants. We use a similar method with a previous work for the evaluation of the sigmoid function (see [13] for details); the ciphertext modulus is reduced by $(2p+3)$ bits for the evaluation of $g_3(x)$, and $(3p+3)$ bits for that of $g_5(x)$ and $g_7(x)$. Therefore, we obtain the following lower bound on the parameter L:

$$L = \begin{cases} \textsc{IterNum} \cdot (3p + 2p_c + 3) + L_0 & g = g_3, \\ \textsc{IterNum} \cdot (4p + 2p_c + 3) + L_0 & g \in \{g_5, g_7\}, \end{cases}$$

where IterNum is the number of iterations of the GD algorithm and L_0 denotes the bit size of the output ciphertext modulus. The modulus of the output ciphertext should be larger than 2^p in order to encrypt the resulting weight vector and maintain its precision. We take $p = 30$, $p_c = 20$ and $L_0 = 35$ in our implementation.

The dimension of a cyclotomic ring $\mathcal{R}$ is chosen as $N = 2^{16}$ following the security estimator of Albrecht et al. [22] for the learning with errors problem. In this case, the bit size L of a fresh ciphertext modulus should be bounded by 1284 to ensure the security level $\lambda = 80$ against known attacks. Hence we repeat $\textsc{IterNum} = 9$ iterations of GD algorithm $g = g_3$, and $\textsc{IterNum} = 7$ iterations when $g = g_5$ or $g = g_7$.

The smoothing parameter γ_t is chosen in accordance with [15]. The choice of proper GD learning rate parameter α_t normally depends on the problem at hand. Choosing too small α_t leads to a slow convergence, and choosing too large α_t could lead to a divergence, or a fluctuation near a local optima. It is often optimized by a trial and error method, which we are not available to perform. Under these conditions harmonic progression seems to be a good candidate and we choose a learning rate $\alpha_t = \frac{10}{t+1}$ in our implementation.

Implementation

All the experimentations were performed on a machine with an Intel Xeon CPU E5-2620 v4 at 2.10 GHz processor.

Task for the iDASH challenge. In genomic data privacy and security protection competition 2017, the goal of Track 3 was to devise a weight vector to predict the disease using the genotype and phenotype data (Additional file 1: iDASH). This dataset consists of 1579 samples, each of which has 18 features and a cohort information (disease vs. healthy). Since we use the ring dimension $N = 2^{16}$, we can only pack up to $N/2 = 2^{15}$ dataset values in a single ciphertext but we

have totally $1579 \times 19 > 2^{15}$ values to be packed. We can overcome this issue by dividing the dataset into two parts of sizes 1579×16 and 1579×3 and encoding them separately into two ciphertexts. In general, this method can be applied to the datasets with any number of features: the dataset can be encrypted into $\lceil (f+1) \cdot n \cdot (N/2)^{-1} \rceil$ ciphertexts.

In order to estimate the validity of our method, we utilized 10-fold cross-validation (CV) technique: it randomly partitions the dataset into ten folds with approximately equal sizes, and uses every subset of 9 folds for training and the rest one for testing the model. The performance of our solution including the average running time per fold of 10-fold CV (encryption and evaluation) and the storage (encrypted dataset) are shown in Table 1. This table also provides the average accuracy and the AUC (Area Under the Receiver Operating Characteristic Curve) which estimate the quality of a binary classifier.

Comparison. We present some experimental results to compare the performance of implementation to [13]. For a fair comparison, we use the same 5-fold CV technique on five datasets - the Myocardial Infarction dataset from Edinburgh [23] (Additional file 2: Edinburgh), Low Birth Weight Study (Additional file 3: lbw), Nhanes III (Additional file 4: nhanes3), Prostate Cancer Study (Additional file 5: pcs), and Umaru Impact Study datasets (Additional file 6: uis) [24, 25, 26, 27]. All datasets have a single binary outcome variable.

All the experimental results are summarized in Table 2. Our new packing method could reduce the storage of ciphertexts and the use of Nesterov's accelerated gradient achieves much higher speed than the approach of [13]. For example, it took 3.6 minutes to train a logistic regression model using the encrypted Edinburgh dataset of size 0.02 GB, compared to 114 minutes and 0.69 GB of the previous work [13], while achieving the good qualities of the output models.

Discussion

The rapid growth of computing power initiated the study of more complicated ML algorithms in various fields including biomedical data analysis [28, 29]. HE system is a promising solution for the privacy issue, but its efficiency in real applications remains as an open question. It would be great if we could extend this work to other ML algorithms such as deep learning.

One constraint in our approach is that the number of iterations of GD algorithm is limited depending on the choice of HE parameter. In terms of asymptotic complexity, applying the bootstrapping method of approximate HE scheme [30] to the GD algorithm would achieve a linear computation cost on the iteration number.

Conclusion

In the paper, we presented a solution to homomorphically evaluate the learning phase of logistic regression model using the gradient descent algorithm and the approximate HE scheme. Our solution demonstrates a good performance and the quality of learning is comparable to the one of an unencrypted case. Our encoding method can be easily extended to a large-scale dataset, which shows the practical potential of our approach.

Declarations

Abbreviations

ML: Machine Learning; HE: Homomorphic Encryption; GD: Gradient Descent; CV: Cross Validation; AUC: Area Under the receiver operating characteristic Curve

Acknowledgements

The authors would like to thank the editor and reviewers for the thoughtful comments and constructive suggestions, which greatly helped us improve the quality of this manuscript. The authors also thank Jinhyuck Jeong for giving valuable comments to the technical part of the manuscript.

Funding

This work was partly supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.B0717-16-0098) and by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIP) (No.2017R1A5A1015626). MK was supported in part by NIH grants U01TR002062 and U01EB023685. Publication of this article has been funded by the NRF Grant funded by the Korean Government (MSIT) (No.2017R1A5A1015626).

Availability of data and materials

All datasets are available in the Additional files provided with the publication. The `HEAAN` library is available at https://github.com/kimandrik/HEAAN. Our implementation is available at https://github.com/kimandrik/HEML.

Authors' contributions

JHC designed and supervised the study. KL analyzed the data. AK drafted the source code and MK optimized it. AK and MK performed the experiments. AK and YS are major contributors in writing the manuscript. All authors read and approved the final manuscript.

Competing interests

All authors declare that they have no competing interests.

Consent for publication
Not applicable.

Ethics approval and consent to participate
Not applicable.

Additional Files
Additional file 1: iDASH - iDASH challenge dataset (txt 60KB)
Additional file 2: Edinburgh - the Myocardial Infarction dataset from Edinburgh (txt 26KB)
Additional file 3: lbw - Low Birth Weight Study dataset (txt 5KB)
Additional file 4: nhanes3 - Nhanes III dataset (txt 779KB)
Additional file 5: pcs - Prostate Cancer Study dataset (txt 10KB)
Additional file 6: uis - Umaru Impact Study dataset (txt 12KB)

Author details
[1]Department of Mathematical Sciences, Seoul National University, 1 Gwanak-ro, Gwanak-gu, 08826 Seoul, Republic of Korea. [2]Department of Computer Science and Engineering, University of California- San Diego, 9500 Gillman Drive, 92093-0404 San Diego, California, USA. [3]Division of Biomedical Informatics, University of California- San Diego, 9500 Gillman Drive, 92093-0728 San Diego, California, USA.

References

1. Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal of research and development **3**(3), 210–229 (1959)
2. Dietz, E.: Application of logistic regression and logistic discrimination in medical decision making. Biometrical journal **29**(6), 747–751 (1987)
3. Rousseau, D.: Biomedical Research: Changing the Common Rule by David Rousseau — Ammon & Rousseau Translations. https://www.ammon-rousseau.com/changing-the-rules-by-david-rousseau/ [Accessed 19/08/2017] Available from: http://www.webcitation.org/6spHgiYRI (2017)
4. Nikolaenko, V., Weinsberg, U., Ioannidis, S., Joye, M., Boneh, D., Taft, N.: Privacy-preserving ridge regression on hundreds of millions of records. In: Security and Privacy (SP), 2013 IEEE Symposium On, pp. 334–348 (2013). IEEE
5. Yao, A.C.-C.: How to generate and exchange secrets. In: Foundations of Computer Science, 1986., 27th Annual Symposium On, pp. 162–167 (1986). IEEE
6. El Emam, K., Samet, S., Arbuckle, L., Tamblyn, R., Earle, C., Kantarcioglu, M.: A secure distributed logistic regression protocol for the detection of rare adverse drug events. Journal of the American Medical Informatics Association **20**(3), 453–461 (2012)
7. Nardi, Y., Fienberg, S.E., Hall, R.J.: Achieving both valid and secure logistic regression analysis on aggregated data from different private sources. Journal of Privacy and Confidentiality **4**(1), 9 (2012)
8. Mohassel, P., Zhang, Y.: SecureML: A System for Scalable Privacy-Preserving Machine Learning. IEEE Symposium on Security and Privacy (2017)
9. Wu S., K.H. Teruya T. Kawamoto J. Sakuma J.: Privacy-preservation for stochastic gradient descent application to secure logistic regression. The 27th Annual Conference of the Japanese Society for Artificial Intelligence (1-4) (2013)
10. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques, pp. 223–238 (1999). Springer
11. Aono, Y., Hayashi, T., Trieu Phong, L., Wang, L.: Scalable and secure logistic regression via homomorphic encryption. In: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, pp. 142–144 (2016). ACM
12. Xie, W., Wang, Y., Boker, S.M., Brown, D.E.: Privlogit: Efficient privacy-preserving logistic regression by tailoring numerical optimizers. arXiv preprint arXiv:1611.01170 (2016)
13. Kim, M., Song, Y., Wang, S., Xia, Y., Jiang, X.: Secure logistic regression based on homomorphic encryption: Design and evaluation. JMIR medical informatics **6**(2) (2018)
14. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Application of Cryptology and Information Security, pp. 409–437 (2017). Springer
15. Nesterov, Y.: A method of solving a convex programming problem with convergence rate o (1/k2). In: Soviet Mathematics Doklady, vol. 27, pp. 372–376 (1983)
16. Cheon, J.H., Kim, A., Kim, M., Lee, K., Song, Y.: Implementation for iDASH Competition 2017. https://github.com/kimandrik/HEML [Accessed 11/07/2018] Available from: http://www.webcitation.org/70qbe6xii (2017)
17. Harrell, F.E.: Ordinal logistic regression. In: Regression Modeling Strategies, pp. 331–343 (2001). Springer
18. Lowrie, E.G., Lew, N.L.: Death risk in hemodialysis patients: the predictive value of commonly measured variables and an evaluation of death rate differences between facilities. American Journal of Kidney Diseases **15**(5), 458–482 (1990)
19. Lewis, C.M., Knight, J.: Introduction to genetic association studies. Cold Spring Harbor Protocols **2012**(3), 068163 (2012)
20. Gayle, V., Lambert, P.S.: Logistic regression models in sociological research (2009)
21. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Implementation of HEAAN. https://github.com/kimandrik/HEAAN [Accessed 19/08/2017] Available from: http://www.webcitation.org/6spMzVJ6U (2016)
22. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
23. Kennedy, R., Fraser, H., McStay, L., Harrison, R.: Early diagnosis of acute myocardial infarction using clinical and electrocardiographic data at presentation: derivation and evaluation of logistic regression models. European heart journal **17**(8), 1181–1191 (1996)
24. lbw: Low Birth Weight study data. https://rdrr.io/rforge/LogisticDx/man/lbw.html [Accessed 19/08/2017] Available from: http://www.webcitation.org/6spNFX2b5 (2017)
25. nhanes3: NHANES III data. https://rdrr.io/rforge/LogisticDx/man/nhanes3.html [Accessed 19/08/2017] Available from: http://www.webcitation.org/6spNJJFDx (2017)
26. pcs: Prostate Cancer Study data. https://rdrr.io/rforge/LogisticDx/man/pcs.html [Accessed 19/08/2017] Available from: http://www.webcitation.org/6spNLXr5a (2017)
27. uis: UMARU IMPACT Study data. https://rdrr.io/rforge/LogisticDx/man/uis.html [Accessed 19/08/2017] Available from: http://www.webcitation.org/6spNOLB9n (2017)
28. Wang, Y.: Application of deep learning to biomedical informatics. International Journal of Applied Science - Research and Review (2016)
29. Ravì, D., Wong, C., Deligianni, F., Berthelot, M., Andreu-Perez, J., Lo, B., Yang, G.-Z.: Deep learning for health informatics. IEEE journal of biomedical and health informatics **21**(1), 4–21 (2017)
30. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Advances in Cryptology–EUROCRYPT 2018: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 360–384 (2018). Springer