Secure Multi-Label Tumor Classification Using Homomorphic Encryption

Seungwan Hong (swanhong@snu.ac.kr)
Seoul National University
Jai Hyun Park
Seoul National University
Wonhee Cho
Seoul National University
Hyeongmin Choe
Seoul National University
Jung Hee Cheon
Seoul National University

Research Article

Keywords: Homomorphic encryption, Multi-label classification, Privacy, Neural Network, Softmax

DOI: https://doi.org/10.21203/rs.3.rs-584746/v1

RESEARCH

Secure Multi-label Tumor Classification Using Homomorphic Encryption

Seungwan Hong*, Jai Hyun Park, Wonhee Cho, Hyeongmin Choe and Jung Hee Cheon

Abstract

Background: In a secure genome analysis competition called iDASH 2020, the homomorphic encryption task was to develop a multi-label tumor classification method for predicting the classes of samples based on genetic information. The scenario is that a data holder encrypts a genetic variant dataset from tumor samples and provides the encrypted data to an untrusted server. Then, the server evaluates homomorphically encrypted data in its model which is trained in plaintext using the published data or own genetic data and outputs the result in an encrypted state so that there is no leakage of genetic information.

Methods: We develop a secure multi-label tumor classification method using the CKKS scheme, the approximate homomorphic encryption scheme. We first propose a new data preprocessing method to reduce the size of large-scale genetic data of tumor samples. Our method aims to analyze the dataset from iDASH 2020 competition track I, which originated from The Cancer Genome Atlas (TCGA) dataset, which consists of 2,713 samples from 11 types of cancers, genetic features from more than 25,000 genes. Secondly, we propose the new data packing method for CKKS ciphertext to provide a trade-off between the number of ciphertexts and the number of rotations in matrix multiplication. Lastly, we suggest the approximation method for softmax activation of a neural network model.

Results: Our preprocessing method reduces the number of genes from more than 25,000 to 2048 or less and achieves a microAUC value of 0.9865 with a 1-layer shallow neural network. Using our model, we successfully compute the tumor classification inference steps on the encrypted test data in 4.5 minutes. Despite using the approximate softmax function, the difference in microAUC value from our implementation results in the encrypted state is less than 10^{-3} compared to the plain result.

Conclusions: We present preprocessing and evaluation methods for secure multi-label tumor classification based on approximate homomorphic encryption using a shallow neural network model with the softmax activation function.

Keywords: Homomorphic encryption; Multi-label classification; Privacy; Neural Network; Softmax

Background

Cancer is a disease caused by the unlimited proliferation of certain cells in the human body, and the exact cause of cancer is not yet known. In 2020 alone, 19.3 million new cases were reported worldwide, and 10 million people died because of cancer [1]. For this reason, the prediction of cancer through genetic data analysis has been regarded as one of the most important tasks since early treatment can reduce the lethal effects of cancer on the human body.

Machine learning (ML) is one of the fields of artificial intelligence that learns the process of finding solutions on its own without human assistance to a given problem. Due to the difficulty in the process of diagnosing tumors through genes, tumor classification using ML-based on large amounts of genetic data contributes to decision making to diagnose and treat cancer. Some cancer genome studies using ML techniques on large-scale data have shown the relationship between genetic modification and specific cancer types [2, 3, 4, 5].

Motivation

Since genetic data contains a lot of personal information and cannot be discarded or changed even if it is leaked, it is essential to protect the privacy of information about genetic data in the data analysis using ML. Various techniques, including differential pri-

*Correspondence: swanhong@snu.ac.kr
Department of Mathematical Sciences, Seoul National University, 1, Gwanak-ro, Gwanak-gu, Seoul, Republic of Korea
Full list of author information is available at the end of the article

vacy [6, 7] or multi-party computation [8], have been used to ensure privacy in the data analysis process. However, each of these approaches has the disadvantage of losing accuracy in the process of anonymizing the data or requiring multiple phases in the process of data sharing.

Homomorphic Encryption (HE) is a cryptographic scheme that enables us to perform arithmetic operations between encrypted data without decryption. HE has been considered one of the useful applications for privacy-preserving ML, as it allows the computation of desired operations without disclosing information about the data [9, 10]. However, most HE libraries [11, 12, 13] mainly support only addition and multiplication of arithmetic operations, so many popular activation functions for neural networks, such as sigmoid, ReLU, or softmax functions, cannot be directly operated for encrypted data. Most of the applications using HE overcomes such problems by approximating non-arithmetic operations with a polynomial that minimizes the error in a certain interval. The main problem with applying HE is to minimize computational increases occurring during the polynomial approximation process while limiting errors.

Integrating Data for Analysis, Anonymization and SHaring (iDASH) has held an annual secure genome analysis competition since 2014. Each year, important topics in the field of genetic analysis are selected to compete for the most effective solution. The problem of multi-label tumor classification using HE was one of the three tasks for the 2020 iDASH competition. Given the dataset with a total of 2,713 patients and their 25,128 genes, Participants had to pre-process the given data, train the ML model with plain data, and obtain the highest microAUC score [14] within 5 minutes when the inference step was performed with encrypted test data.

Summary of Results

In this work, we propose the privacy-preserving multi-label classification method using a shallow neural network with softmax activation based on HE, which is a solution of the first track of the iDASH 2020 competition. Our work aims to build the model and run it over encrypted data to get the highest microAUC score while the total round trip time (including encryption, computation, and decryption) for the inference step costs less than 5 minutes.

To compute the inference step within a limited amount of time with encrypted data, the following challenges must be solved: first, since given data is very large, a large number of ciphertexts are required to encrypt the data, and a lot of time cost is also needed for operations between them. Second, the softmax function cannot be directly computed using HE because it consists of exponential and inverse functions.

To overcome such challenges, we propose a data preprocessing method suitable for a given dataset, so that the number of genes for both training and inference steps is reduced to less than 2^B for some B (we choose $B = 9, 10$, or 11). The data consist of two types: Copy Number (CN) data and Variants data. For CN data, we divide CN data into groups through the Hamming distance function and select a representative gene for each group, called CN data filtering. Since adjacent genes mutate similarly, our technique can be thought to have effectively selected representatives of genes in similar positions. For Variants data, only specific genes data exist for each person, so we select the representative genes for each tumor based on mutation effect and frequency, called Variants data filtering.

Also, we find that the elementary exponential approximation method retains the properties for calculating the microAUC value. For calculating microAUC, the difference in comparison between different samples is important for each feature, so an approximation method that reduces errors, such as minimax approximation, does not preserve the order and results in a large loss of microAUC through the approximation. On the other hand, even if the error is not minimized in a given interval, our approximation similarly satisfies the property of exponential function: 1) reducing the space between small inputs and 2) increasing the space between large inputs, so the difference between the microAUC value through the approximation softmax is small. Using these properties, we confirmed through experiments that the microAUC score through our softmax approximation can obtain a value that is almost similar to that computed with the exact softmax function.

Additionally, we modify the data packing method for matrix-vector multiplication [15] by duplicating the data in encryption to make a trade-off between the number of ciphertexts for packing and the number of rotations. Since the number of rotations occupies the most time cost in the multiplication between the encrypted matrix and the plain vector, it can be reduced by our method and the total time can be minimized by choosing an appropriate number of copies in the encryption step.

Finally, we exploit an approximate homomorphic encryption scheme HEaaN [16] to implement our inference method over encrypted data. As one of our results, by reducing the number of genes by less than 2^{11}, we got an inference result that outputs a microAUC value of 0.9865 in about 4.5 minutes.

Related Works

Various ML techniques are using to classify the cancer type of a sample from its genetic data. Chen et al. [17] classified across 17 cancer types using Support Vector Machine (SVM), a kind of supervised learning. They used Catalogue Of Somatic Mutations In Cancer (COSMIC) data, an online database of somatically acquired mutations found in human cancer. The dataset consists of gene symbols, mutations, chromosomes, and pathways. They have pre-processed to remove all samples without information about some features and achieved 62% accuracy.

Yuan et al. [4] introduced a multi-label classifier named DeepGene, using a Deep Neural Network (DNN) with 4 hidden layers, achieving maximum accuracy of 65.5% across 12 cancer types. The data was from The Cancer Genome Atlas (TCGA), a project to list genetic mutations subject to cancer, using genome sequencing and bioinformatics. The data was the presence of mutations for each of the 22k genes in 3k samples. They introduced two filtering techniques named Clustered Gene Filtering (CGF) and Indexed Sparsity Reduction (ISR) to filter out irrelevant genes and reduce the data sparsity. Especially, CGF clusters the genes with Jaccard distance and extracts the representative genes. The filtering techniques made it easier and more accurate for the DNN model to learn the data.

Yuan et al. [18] also introduced DeepCNA, a cancer classification scheme based on Convolution Neural Network (CNN) from copy number aberrations (CNAs) and chromatin 3D structure data. The data was from the COSMIC CNA dataset and scaled into an interval and padded with zero to fit into a CNN input image size. A 2D CNN model with 7 hidden layers was used in their work, which led to 57.4% accuracy across 25 cancer types.

Sun et al. [19] reached 70.1% classification accuracy across 12 cancer types using 5-layer DNN model. They used Whole Exon Sequencing (WES) mutations data from the TCGA and of healthy samples from the 1000 Genomes Project (1KGP), a project to list detailed human genetic variation. The genes with a low mutation occurrence rate were considered irrelevant and filtered out.

Lee et al. [20] classified across 31 cancer types with their CPEM scheme, an ensemble of two ML techniques: random forest and DNN. The data was from the COSMIC database consists of gene-level mutation profiles, mutation rates, mutation spectra, gene-level SCNAs, and mutation signatures. They compared the three most commonly used supervised feature selection approaches in ML: the extra tree-based, Least Absolute Shrinkage and Selection Operator (LASSO), and Linear Support Vector Classifier (Linear SVC). With the LSVC feature selection method, they reached 84% classification accuracy.

As one solution to privacy-preserving ML on genetic data, many researches are proposed that use ML on encrypted data with HE. Kim et al. introduced a novel method to train a logistic regression model without information leakage with HE [21]. The model was trained to classify samples with 18 genetic features into two: disease or healthy. The authors adapted Nesterov's accelerated gradient descent method to reduce the number of iterations, since the depth of the homomorphic circuit highly affects to the computational cost. The classifier was selected as the best solution of Track 3 at iDASH competition 2017, where similar approaches were also be done [9, 22, 23]. In the second track of iDASH 2018 competition, which aimed to develop a method for outsourcing computation of Genome Wide Association Studies (GWAS) on homomorphically encrypted data, various solutions based on logistic regression were introduced [10, 24, 25, 26, 27]. Parallel logistic regression models on encrypted input data were used to identify genetic variants correlated with specific phenotypes. In 2019, the HE task of iDASH competition was a secure genotype imputation using HE. The solution showed "ultra-fast" HE models to verify genotype imputation that took less than 10 seconds for evaluation with only a 2–3% decrease in accuracy [28]. The authors have confirmed that similar results can be obtained for various HE libraries, including BFV [29], CKKS, and TFHE [30].

Methods

Notations

We denote Hadamard multipcation between two vectors by $\vec{a} \odot \vec{b}$. Also, for simplicity, we denote the $[x]_n$ be the number in $\{1, \cdots, n\}$ satisfying $[x]_n = x \mod n$. Using this notation, we can denote $a^{(i)} = (a_{[i+j]_n})_{1 \le j \le n}$.

For a matrix A, $A(i, j)$ means the i th row and j th column element of A. Also, we denote the submatrix of A with $i \in I$ th rows and $j \in J$ th columns from A by $A[I, J]$. In this case, ':' indicates the whole index set.

Approximate Homomorphic Encryption

Homomorphic Encryption (HE) is a cryptographic scheme that enables computing operations between encrypted data without decryption. For applications that require computations between real data, such as machine learning [31, 32, 33, 34] or cyber physics system [35], the approximate HE scheme, namely as CKKS scheme, proposed by Cheon et al. [16] that

supports arithmetic operations between encrypted real data is mainly used.

In this section, we formally describe CKKS scheme, but we omit details of scheme construction. Let L be a level parameter that a fresh ciphertext is equipped with a modulus $q_L = 2^L$, and $q_\ell := 2^\ell$ for $1 \le \ell \le L$. Let $R := \mathbb{Z}[X]/(X^N + 1)$ be a cyclotomic ring for a power-of-two N and R_q be a modulo-q quotient ring of R, i.e., $R_q = R/qR$. The distribution χ_{enc} and χ_{err} denote the discrete Gaussian distribution with some fixed standard deviation. The distribution χ_{key} outputs a polynomial of $\{-1, 0, 1\}$-coefficient. We denote the rounding function $\lfloor \cdot \rceil$ and modulo-q operation $[\cdot]_q$.

CKKS scheme uses a plaintext vector $\vec{m} \in \mathbb{C}^{N/2}$ and provides enrty-wise operation, called Single-Instrument-Multiple-Data (SIMD) operation, such as addition, substitution, and Hadamard multiplication between vectors. Before encrypting a plaintext vector, we have to convert a complex vector into a integer polynomial in R, called encoding message. To encrypt complex value, CKKS uses a field isomorphism $\tau : \mathbb{R}[X]/(X^N + 1) \to \mathbb{C}^{N/2}$ called canonical embedding.

- $\underline{\mathsf{Ecd}(\vec{m}; \Delta).}$ For a message $\vec{m} = (m_1, \cdots, m_{N/2})$ in $\mathbb{C}^{N/2}$ and scaling factor Δ, output a polynomial $\mathsf{m} = \lfloor \Delta \cdot \tau^{-1}(\vec{m}) \rceil \in R$ where the rounding function is done coefficient-wisely.
- $\underline{\mathsf{Dcd}(\vec{m}; \Delta).}$ For a polynomial $\mathsf{m} \in R$, output a plaintext vector $\vec{m}' = \Delta^{-1} \cdot \tau(\mathsf{m}) \in \mathbb{C}^{N/2}$.

To sum up, to encrypt a plaintext vector of complex numbers $\vec{m}$, we first encode $\vec{m}$ into a polynomial $\mathsf{m} \leftarrow \mathsf{Ecd}(\vec{m}, \Delta)$ and then generate a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathsf{m})$. For convenience, let n be $N/2$. Now, the scheme description of CKKS is as following:

- $\underline{\mathsf{KeyGen}(\mathsf{params}).}$
  - Sample $s \leftarrow \chi_{key}$ and Set the secret key as $\mathsf{sk} = (1, s)$.
  - Sample $a \leftarrow U(R_{q_L})$ and $e \leftarrow \chi_{err}$. Set the public key as $\mathsf{pk} = (b, a) \in R_{q_L}^2$ where $b = [-a \cdot s + e]_{q_L}$.
- $\underline{\mathsf{Enc}_{\mathsf{pk}}(\mathsf{m}).}$ Given a message $\mathsf{m} \in R$, sample $v \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi_{err}$. Output the ciphertext $\mathsf{ct} = [v \cdot \mathsf{pk} + (\mathsf{m} + e_0, e_1)]_{q_L}$.
- $\underline{\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}).}$ Given a ciphertext $\mathsf{ct} \in R_{q_\ell}^2$, output $\mathsf{m}' = \langle \mathsf{ct}, \mathsf{sk} \rangle$.
- $\underline{\mathsf{Add/Sub}(\mathsf{ct},\ \mathsf{ct'}).}$ Given two ciphertext $\mathsf{ct}, \mathsf{ct}' \in R_{q_\ell}^2$, output the ciphertext $\mathsf{ct}_{add}/\mathsf{ct}_{sub} = [\mathsf{ct} \pm \mathsf{ct}']_{q_\ell}$ encrypting a plaintext vector $\vec{m}_1 \pm \vec{m}_2$.
- $\underline{\mathsf{CMult}(\mathsf{ct}, \vec{c}).}$ Given a ciphertext $\mathsf{ct} \in R_{q_\ell}^2$ and a constant vector $\vec{c} \in \mathbb{C}^n$, output the ciphertext $\mathsf{ct}_{cmult} = \mathsf{Ecd}(\vec{c}) \cdot \mathsf{ct}$ encrypting a plaintext vector $\vec{c} \odot \vec{m}$.
- $\underline{\mathsf{Mult}_{\mathsf{evk}}(\mathsf{ct}, \mathsf{ct}').}$ Given two ciphertexts $\mathsf{ct}, \mathsf{ct}' \in R_{q_\ell}^2$, output a ciphertext $\mathsf{ct}_{mult} \in R_{q_\ell}^2$ encrypting a plaintext vector $\vec{m}_1 \odot \vec{m}_2$.
- $\underline{\mathsf{Conj}_{\mathsf{ck}}(\mathsf{ct}).}$ For a ciphertext ct encrypting a plaintext vector $\vec{m} = (m_1, \cdots, m_n)$, output a ciphertext ct' encrypting a plaintext vector $\mathsf{Conj}(\vec{m}) = (\overline{m}_1, \cdots, \overline{m}_n)$.
- $\underline{\mathsf{Rot}_{\mathsf{rk}}(\mathsf{ct}; r).}$ For a ciphertext ct encrypting a plaintext vector $\vec{m} = (m_1, \cdots, m_n)$, output a ciphertext ct' encrypting a plaintext vector $\vec{m}' = (m_{r+1}, \cdots, m_n, m_1, \cdots, m_r)$ which is the (left) rotated plaintext vector of ct by r positions.
- $\underline{\mathsf{ReScale}_{\ell \to \ell'}(\mathsf{ct}).}$ For a cipertext $\mathsf{ct} \in R_{q_\ell}^2$, return the ciphertext $\mathsf{ct}' = [\lfloor (q_{\ell'}/q_\ell) \cdot \mathsf{ct} \rceil]_{q_{\ell'}}$. This step needs to control a scaling factor after some operations.

For the remind of the paper, we may denote the operations between ciphertexts or ciphertext and plain vector by common symbols, such as $\mathsf{Add}(\mathsf{ct}_1, \mathsf{ct}_2) = \mathsf{ct}_1 + \mathsf{ct}_2$ or $\mathsf{CMult}(\mathsf{ct}, \vec{c}) = \mathsf{ct} \cdot \vec{c}$ for simplicity.

Data Preprocessing

Given more than 25,000 genes, we need to select meaningful genes for efficient machine learning using HE. Therefore, we introduce new data preprocessing technique to obtain significant genes that are useful for predicting tumors. Our data preprocessing method consists of two filtering algorithms that extract specific features from two different types of data. The resulting filtered data matrices are concatenated and input to our neural network as in Figure 1(c).

Data Structure

Our scenario focuses on the dataset originated from The Cancer Genome Atlas (TCGA) database, which is widely used in genomic research. The dataset consists of somatic mutations and copy number data for 11 sites: Bladder, Breast, Bronchus and Lung, Cervix uteri, Colon, Corpus uteri, Kidney, Liver and Intrahepatic bile ducts, Ovary, Skin, and Stomach. The dataset consists of total 2,713 samples and 25,128 genes, and each sample has one cancer type out of 11 types of cancer, consists of two types of data: Copy Number (CN) data and Variants data. The dataset is available from iDASH 2020 competition track I and the details of dataset including the number of samples and the number of mutation's effects for each feature are disclosed in Table 1.

CN data consists of the copy number of the genes. The copy number data show the copy number variations (CNVs) of the genes as numbers, representing the state of the gene on the corresponding sample, whether the gene has duplication or deletion of a considerable number of base pairs. The data consist of 5 different levels 0, ±1, ±2 where the negative and positive values represent deletion and duplication of the corresponding gene from the sample, respectively. Copy number

with 0 means that the gene has no considerable variation on the corresponding gene from the sample.

Variants data consists of mutation data of selected pairs of samples and genes for each tumor type with various features: gene's location on chromosomes, mutation type, whether the mutation is Single Nucleotide Polymorphism (SNP), and mutation's effects separately predicted by two different methods.

CN Data Filtering

Our main approach for CN data filtering is to filter out the irrelevant genes based on the similarity of the neighboring genes. Let $C \in \{0, \pm 1, \pm 2\}^{s \times g_{\mathsf{cn}}}$ be the matrix of copy number data, where the s rows correspond to the s samples, and the g columns correspond to g genes; $C(i, j)$ is a copy number that corresponds to gene j of sample i. We initially sort the raw CN data according to the order of the gene positions on the chromosome.

Then, we group the genes by their copy number similarity (line 3 in Algorithm 1). For two gene columns $\vec{p}, \vec{q} \in \{0, \pm 1, \pm 2\}^{1 \times g_{\mathsf{cn}}}$, we use the Hamming distance as the similarity measure:

$$\mathsf{dist}_{\mathsf{H}}(\vec{p}, \vec{q}) = \text{the number of } i \text{ such that } p_i \neq q_i,$$

where p_i and q_i are the i th entry of vectors $\vec{p}$ and $\vec{q}$, which represents the copy numbers of each gene for ith sample. Starting from the first gene in C, say a representative of the first group, we calculate the hamming distance with the following genes in order. Until the distance is less than the predetermined threshold d_{cn}, we merge each corresponding gene to the first group. If the distance first reached the threshold d_{cn}, the first group ends and the corresponding gene becomes a representative of the second group. Then start from the new representative, calculate the hamming distance with the following genes.

Repeating this algorithm until the genes are all grouped, then each gene is in a unique group with neighboring genes. Finally, we filter the CN data with the representative gene, the first gene in each group, resulting in a matrix $\tilde{C} \in \{0, \pm 1, \pm 2\}^{s \times \tilde{g}_{\mathsf{cn}}}$ where $\tilde{g}_{\mathsf{cn}}$ is the number of groups as the result of our filtering. Note that $\tilde{g}_{\mathsf{cn}}$ is much smaller than the number of original genes. We formally state our CN data filtering algorithm in Algorithm 1 and shown in Figure 1(a).

Variants Data Filtering

Variants data filtering mechanism uses only a mutation's effect data classified in 4 different levels: LOW, MODERATE, MODIFIER, and HIGH. The Variants data filtering works based on the effectiveness of the mutations. It filters out the genes with less effective mutations.

Algorithm 1: CN data fitering

Input : CN data matrix $C \in \{0, \pm 1, \pm 2\}^{s \times g}$ sorted by gene location and threshold d_{cn}.
Output: matrix $\tilde{C} \in \{0, 1\}^{s \times \tilde{g}_{\mathsf{cn}}}$. ($\tilde{g}_{\mathsf{cn}} \ll g$)

```
1  geneList ← [1]                          // list of repre. genes
2  i ← 1
3  while i ≤ g do                          // group the genes
4  |  j ← i + 1
5  |  while (j ≤ g & dist_H(C[:, i], C[:, j]) < d_cn) do
6  |  |  j++
7  |  end
8  |  i ← j
9  |  geneList.append(i)
10 end
11 C̃ ← C[:, geneList]                      // filter C with zero padding
12 return C̃
```

In the Variants data, for each sample, only a few mutation effect data of individually selected genes are given. So the raw Variants data for whole samples and genes is almost empty, which cannot be used immediately as a neural network input. Hence our Variants data filtering is separately applied to each cancer type to reduce the empty spaces and fill in yet remaining spaces with zero.

Let V_t be the matrix of raw data with mutation's effect corresponding to the tumor type t ($1 \leq t \leq T$). V_t is a $s_t \times g$ matrix with mutation's effect as strings, where s_i rows correspond to s_t samples and g columns correspond to whole g genes for the tumor type t. $V_t(i, j)$ is the effect of the mutation in j th gene of i th sample with the tumor type t. We first encode the string data to predetermined real values between 0 and 1: LOW = 0.2, MODERATE = 0.5, MODIFIER = 0.9, HIGH = 1.0, and 0 if no information exists. The resulting encoded Variants matrix is sparse matrix in $E^{s_t \times g}$, where $E = \{0, 0.2, 0.5, 0.9, 1.0\}$ be a set of encoding values.

Secondly, we sum V_t by each gene column and if the sum is more than the predetermined threshold k_{var}, then put the gene in the list of the selected genes (line 2 in Algorithm 2). The column-wise summation $\sum_{i=1}^{s_t} V_t(i, j)$ is a mutation's effect of jth gene to the s_t samples, so the selected genes in list can be regarded as genes with considerable mutation effects. The union of the selected genes from each tumor type t is the set of filtered genes.

Finally, we filter the whole Variants data with the filtered genes for each t, resulting in a matrix $\tilde{V} \in E^{s \times \tilde{g}_{\mathsf{var}}}$ with much smaller genes to whole samples (line 11 in Algorithm 2). The workflow of the Variants data filtering technique is shown in Figure 1(b).

Modeling Shallow Neural Network

After preprocessing, we get the data matrix X with size $s \times g$ where $g = \tilde{g}_{\mathsf{cn}} + \tilde{g}_{\mathsf{var}}$. Then, with the tumor

Algorithm 2: Variants data filtering

Input : encoded Variants data matrices $V_t \in E^{s_t \times g}$ for cancer types $1 \le t \le T$, and threshold k_{var}. ($E = \{\text{encoded values}\}$)
Output: matrix $\tilde{V} \in E^{s \times \tilde{g}_{\mathsf{var}}}$. ($s = \sum s_t$, $\tilde{g}_{\mathsf{var}} \ll g$)

```
geneList ← [ ]
for t = 1 to T do                    // pick genes from each V_t
    for j = 1 to g do
        if ColSum(V_t[:, j]) > k_var then
            geneList ← geneList ∪ {j}
        end
    end
end
for t = 1 to 11 do                   // filtering V_ts
    V_t ← V_t[:, geneList]
end
Ṽ ← Concatenate(V_1, ··· , V_T, axis = 0)
return Ṽ
```

data Y, we train a neural network model from (X, Y) in plain, where T is the number of tumor types and Y is the one-hot label matrix with size $s \times T$. Note that T is 11 in our dataset, which is relatively small than s and g. Our neural network model consists of one hidden layer with 64 nodes and linear activation function and output layer with 11 nodes. In the output layer, we use softmax activation function to output their predicted value. During the training phase, we used Keras library [36] in Python with batch size of 32, number of epochs of 50 and dropout rate of 0.9. Note that we used sufficiently big dropout rate in order to avoid overfitting since the layers in our model are small.

Roadmap for Inference Step over Encrypted Data

From now on, we state the method to compute inference step from our model with encrypted data. Precisely, $s \times g$ matrix X and $g \times T$ matrix W are given (recall that s, g and T refers the number of samples, genes, and tumors, respectively). Since t is relatively small than s and f in practice, we consider this matrix multiplication $Y = X \cdot W$ as T times of matrix-vector multiplications $\vec{y}_i = X \cdot \vec{w}_i$ for $0 \le i < T$, where $\vec{w}_i$ is $i+1$th column of W. Then, for each row Y_j of Y ($0 \le j < s$), we compute softmax function to get final score of our model. Since the matrix Y is still encrypted, we need to compute approximate function of softmax. In short, our Method can be divided into three steps:

1 Data Packing : encrypt the matrix X to a number of ciphertexts $\{\mathsf{ct}_i\}$.
2 Matrix-Vector Multiplication : using $\{\mathsf{ct}_i\}$ and W, compute matrix-vector multiplication to get the ciphertext ct_Y that contains $Y = X \cdot W$.
3 Softmax Evaluation : compute approximate softmax function for ct_Y to obtain softmax output for each row of Y.

We break the method down into the first two steps and the other, and explain the main ideas in each section.

Data Packing and Matrix Multiplications

Although CKKS scheme supports SIMD operation between vectors, the SIMD operation cannot be directly applied to the matrix-vector multiplication operation. Therefore, in order to efficiently perform a matrix-vector multiplication operation using HE, a process of mapping a matrix to a number of vectors is required. In this section, focusing on the matrix multiplications in our scenario, we first state the naive approach and then suggests our optimization method. From now, we denote the rotation of a vector by superscript with parentheses, so that $\vec{v}^{(r)} = (v_{r+1}, v_{r+2}, \cdots, v_n, v_1, \cdots, v_r)$.

The main idea of mapping a matrix to vectors for this computation is suggested in [15], which is that $X \cdot \vec{w}_i$ can be understood as n times of slot-wise vector multiplication between *diagonal* part of X and w_i. Concretely, we define $\vec{x}_j$ by the j-th diagonal part of X for $0 \le j < g$ so that $\vec{x}_j = (X(k, [j+k-1]_g))_{1 \le k \le s}$. Then, the matrix-vector multiplication can be computed as $\vec{y}_i = X \cdot \vec{w}_i = \sum_{j=0}^{g-1} \vec{x}_j \odot \vec{w}_i^{(j)}$. Hence, the multiplication can be done if we encrypt $\vec{x}_j$'s in several ciphertexts and compute constant multiplication between ciphertext and plain vector $\vec{w}_i^{(j)}$.

Warm-up with Simple Parameters

Before we state our explicit method for encodings and matrix-multiplications, we start to explain with a toy example, simplifying the parameter as $s = 2, g = 4$, and $t = 2$, and assumsing that the number of slots in one ciphertext is 4 times larger than the size of vector s. Then our goal is to compute $\vec{y}_i = \sum_{j=0}^{3} \vec{x}_j \odot \vec{w}_i^{(j)}$ for $0 \le i < 4$. In the naive approach, we encode $\vec{x}_i$'s by $\mathsf{ct} = \mathsf{Enc}(\vec{x}_1 \| \vec{x}_2 \| \vec{x}_3 \| \vec{x}_4)$ and define $\tilde{w}_i = (\vec{w}_i^{(0)} \| \vec{w}_i^{(1)} \| \vec{w}_i^{(2)} \| \vec{w}_i^{(3)})$ for each i. Then we can compute Hadamard multiplication by $\mathsf{ct}_{i,\mathsf{mult}} = \mathsf{ct} \cdot \tilde{w}_i$, which contains the message vector $(\vec{x}_0 \odot \vec{w}_i^{(0)}) \| \cdots \| (\vec{x}_3 \odot \vec{w}_i^{(3)})$. Hence, the final sum is obtained by computing rotation twice as $\mathsf{ct}'_i = \mathsf{ct}_{i,\mathsf{mult}} + \mathsf{rot}(\mathsf{ct}_{i,\mathsf{mult}}; s/2)$ and $\mathsf{ct}_{i,\mathsf{sum}} = \mathsf{ct}'_i + \mathsf{rot}(\mathsf{ct}'_i; s/4)$, where the real parts of first $s/4$ slots in $\mathsf{ct}_{i,\mathsf{sum}}$ contains $\vec{y}_i$. Here, the operations we need are total 4 CMults and 4 rots.

From the naive approach, we need t number of the matrix-vector multiplication, so that the CMult and rot should be computed t times. Here, the implementation cost of rot is larger than CMult, so we suggest new packing method to reduce the number of rotation in the whole algorithm. Our main idea is to duplicate and

encode X multiple times in each ciphertexts, which reduces the number of rotations since the the number of summation between the slots in one ciphertexts is reduced.

Now, we propose a new approach for matrix-vector multiplication to reduce the number of constant multiplications and rotations. Our main idea is that we copy each $\vec{x}_j$'s several times in encoding step. For the same condition as above example, we can encode the vectors $\vec{x}_i$ twice so that we obtain two ciphertexts $\mathsf{ct}_0 = \mathsf{Enc}(\vec{x}_0\|\vec{x}_0\|\vec{x}_1\|\vec{x}_1)$ and $\mathsf{ct}_1 = \mathsf{Enc}(\vec{x}_2\|\vec{x}_2\|\vec{x}_3\|\vec{x}_3)$. In this case, we define

$$\tilde{w}_{k,l} = (\vec{w}_{2k}^{(2l)}\|\vec{w}_{2k+1}^{(2l)}\|\vec{w}_{2k}^{(2l+1)}\|\vec{w}_{2k+1}^{(2l+1)})$$

for $0 \le k, l < 2$. Then we get $\mathsf{ct}_{k,l}^{\mathsf{mult}} = \mathsf{ct}_k \cdot \tilde{w}_{k,l} = \mathsf{Enc}(\vec{x}_{2k} \odot \vec{w}_{2k}^{(2l)}\| \cdots \|\vec{x}_{2k+1} \odot \vec{w}_{2k+1}^{(2l+1)})$. Thus, by adding and computing RotSum with those ciphertexts as $\mathsf{ct}_l^{\mathsf{sum}} = \sum_{k=0}^{1} \mathsf{ct}_{k,l}^{\mathsf{mult}}$ and $\mathsf{ct}_l^{\mathsf{final}} = \mathsf{ct}_l^{\mathsf{sum}} + \mathsf{rot}(\mathsf{ct}_l^{\mathsf{sum}}; s/2)$, the result ciphertexts are

$$\mathsf{ct}_0^{\mathsf{final}} = \mathsf{Enc}(\sum_{j=0}^{3} \vec{x}_j \odot \vec{w}_0^{(j)}\| \sum_{j=0}^{3} \vec{x}_j \odot \vec{w}_1^{(j)}\| \cdots),$$
$$\mathsf{ct}_1^{\mathsf{final}} = \mathsf{Enc}(\sum_{j=0}^{3} \vec{x}_j \odot \vec{w}_2^{(j)}\| \sum_{j=0}^{3} \vec{x}_j \odot \vec{w}_3^{(j)}\| \cdots),$$

which are $\vec{y}_i$'s that we desired. In this method, the operations we need are 4 CMults and 2 rots, while 2 ciphertexts are required for encryption instead of 1.

Using Imaginary Part of Message Space

The CKKS scheme supports operations between complex numbers, but in applications using real numbers only such as neural networks, the imaginary part is never used. Here, we can make the computation in the encrypted state more efficient by using the imaginary part, which is not used in the plain operation.

The main idea which is suggested in [28] is that for four real numbers a, b, c, and d, the sum of the two products $ab + cd$ is equal to the real part of one complex product $(a + \mathsf{i}b)(c - \mathsf{i}d)$, where i denotes $\sqrt{-1}$. Since the matrix-vector multiplication we need is also composed of the sum of Hadamard multiplications between vectors, we can reduce the number of ciphertext and the number of constant multiplications by half by combining each vector by two and encoding it into one complex number.

Rotate and Sum Algorithm

For the matrix-vector multiplication, the addition between data packed in one ciphertext is needed. We use RotSum algorithm, which repeatedly computes rotation and addition to get summation of desired slots in a ciphertext.

Precisely, we define the algorithm $\mathsf{RotSum}(\mathsf{ct}, s, t, d)$ outputs the ciphertext ct_{out} that is obtained by computing $\mathsf{ct}_{out} \leftarrow \mathsf{ct}$ and $\mathsf{ct}_{out} \leftarrow \mathsf{ct}_{out} + \mathsf{Rot}_{\mathsf{rk}}(\mathsf{ct}_{out}; st^j)$ for $j = 0, 1, \cdots, d-1$ in order. As a result, if ct was an encryption of the message vector $\vec{m} = (m_i)_{1 \le i \le n}$, then ct_{out} contains the message vector $\vec{m}_{out} = (\sum_{j=0}^{t^d-1} m_{[i+js]_n})_{1 \le i \le n}$.

Putting It All Together

Combining all of the above methods, we explain the explicit method used for our implementation. For the rest of this section, we denote the concatenation of vectors by $\|_{i=s}^{t} \vec{a}_i = (\vec{a}_s\|\vec{a}_{s+1}\| \cdots \|\vec{a}_t)$ and the repeated concatenation of the same vector by $(\vec{a})^r = (\vec{a}\|\vec{a}\| \cdots \|\vec{a})$ (r times). Also, we can embed matrices X and W into large matrices with row and column lengths of power-of-2, respectively, so we assume that s, g are power-of-2's. Recall that our goal is to compute $\vec{y}_j = \sum_{i=1}^{g} \vec{x}_i \odot \vec{w}_j^{(i)}$ for $0 \le j < t$ while $\vec{x}_i$'s are encrypted, so that the output $\vec{y}_j$'s are also encrypted.

As explained before, we copy each $\vec{x}_i$'s m times when encoding, where m is a power-of-2. Since each vector has the size s and the number of slots in a ciphertext is n, each ciphertext will contain $\ell = 2 \cdot \frac{n}{ms}$ number of different vectors (in both real and imaginary parts). Thus we need total $\frac{g}{\ell} = m \cdot \frac{sg}{2n}$ number of ciphertexts for encoding.

Precisely, we encode $\vec{x}_i$'s to each ciphertext by

$$\mathsf{ct}_i = \mathsf{Enc}\left(\|_{q=0}^{\frac{\ell}{2}-1}(\vec{x}_{i\cdot\ell+2q} + \mathsf{i} \cdot \vec{x}_{i\cdot\ell+2q+1})^m\right)$$

for $0 \le i \le \frac{g}{\ell} - 1$. Note that the massage vector in ct_i sequentially contains m copies of vector $\vec{x}_{i\cdot\ell+2q} + \mathsf{i} \cdot \vec{x}_{i\cdot\ell+2q+1}$ for each q. Also, we define $\tilde{w}_{ji}$ by

$$\tilde{w}_{ji} = \|_{q=0}^{\frac{\ell}{2}-1}\left(\|_{r=0}^{m-1}(\vec{w}_{jm+r}^{(i\cdot\ell+2q)} - \mathsf{i} \cdot \vec{w}_{jm+r}^{(i\cdot\ell+2q+1)})\right)$$

for $0 \le i \le \frac{g}{\ell} - 1$ and $0 \le j \le \lceil \frac{T}{m} \rceil - 1$. Here we define $\vec{w}_{jm+r} = \vec{0}$ if $jm + r \ge T$.

Next, we compute the multiplication step. For each j, we multiply ct_i and $\tilde{w}_{ji}$, add its conjugation and divide it by 2 to get $\mathsf{ct}'_{ji} = \left((\mathsf{ct}_i \cdot \tilde{w}_{ji}) + \overline{(\mathsf{ct}_i \cdot \tilde{w}_{ji})}\right)/2$. Then, by adding them to get $\mathsf{ct}_j^{\mathsf{sum}} = \sum_{i=0}^{\frac{g}{\ell}-1} \mathsf{ct}'_{ji}$, we get the ciphertext $\mathsf{ct}_j^{\mathsf{sum}}$ which is the encryption of the vector

$$\|_{q=0}^{\frac{\ell}{2}-1}\left(\|_{r=0}^{m-1}(\sum_{i=0}^{\frac{g}{\ell}-1}\sum_{k=0}^{1} \vec{x}_{i\cdot\ell+2q+k} \odot \vec{w}_{jm+r}^{(i\cdot\ell+2q+k)})\right).$$

for each j. Finally, since

$$\sum_{q=0}^{\frac{\ell}{2}-1}\sum_{i=0}^{\frac{q}{\ell}-1}\sum_{k=0}^{1}\vec{x}_{i\cdot\ell+2q+k}\odot\vec{w}_{jm+r}^{(i\cdot\ell+2q+k)}=\vec{y}_{jm+r},$$

we compute the desired sum using RotSum, so that the first ms slots contains the desired sum and it is copied to the remaining slots. Precisely, the jth group of m outputs $\vec{y}_{jm},\cdots,\vec{y}_{(j+1)m-1}$ are obtained as

$$\mathsf{RotSum}(\sum_{i=0}^{\frac{q}{\ell}-1}\mathsf{ct}'_{ji},ms,2,\log\frac{\ell}{2})=\mathsf{Enc}\left((\|_{r=0}^{m-1}\vec{y}_{jm+r})^{\frac{\ell}{2}}\right).$$

Note that the number of rotations used in the algorithm is $\log\frac{\ell}{2}=\log\frac{n}{ms}$ for each j, so total $\log\frac{n}{ms}\cdot\lceil\frac{T}{m}\rceil$.

In summary, if we duplicate $\vec{x}_j$'s m times for encryption, then the required number of rotations is reduced by m times, where the number of ciphertexts for encryption in increased by m times. Exactly, the number of rotations and multiplications are $\log\frac{n}{ms}\cdot\lceil\frac{T}{m}\rceil$ and $\frac{sg}{n}\cdot m\lceil\frac{T}{m}\rceil$, respectively, while the number of ciphertexts for encryption is increased by $m\cdot\frac{sg}{n}$ (note that the value $\frac{sg}{n}$ implies the minimum number of ciphertexts to encrypt X). Since the number of multiplication does not change asymptotically for m, our method provides a time cost trade-off between rotations and encryptions. In our implementation environment, the rotation costs the significant parts in total matrix-vector multiplication algorithm, so that we can reduce the total time cost by using more memory to encrypt data.

Approximation of Softmax

While softmax layer substantially enhances the performance of the classification, it cannot be directly computed by CKKS scheme since it comprises several non-polynomial operations. Recall that the softmax of a real vector $\vec{v}=(v_1,\cdots,v_t)$ is defined as followings.

$$\mathtt{softmax}(\vec{v})=\frac{1}{\sum_{i=1}^{t}\exp(v_i)}(\exp(v_1),\cdots,\exp(v_t))$$

Both exponential function and division are not polynomial, so we should replace them by their polynomial approximation. We introduce a proper polynomial approximation technique for each of exponential and division function. In general, in order to approximate a non-polynomial operation by a polynomial, minimax method that minimizes the maximum error value within an interval or Chebyshev approximation that express the function by the series of Chebyshev polynomials are mainly used. However, the minimax approximation loses the increasing property of exponential and the division function, because the sign of the error is not constant. Hence, this method is not suitable in terms of calculating microAUC score. Instead, we suggest an approximation method of the softmax layer that is more suitable for microAUC score. In particular, our approximation method consists of a less number of squaring operations, so it has an advantage to be evaluated over homomorphically encrypted data.

Approximation of Exponential Function

Our approximation method for exponential function comes from the elementary definition of $\exp(x)$. Precisely, for some r, we approximately compute $\exp(x)$ as

$$\exp(x)=\lim_{n\to\infty}(1+\frac{x}{n})^n\approx\left(1+\frac{x}{2^r}\right)^{2^r}.$$

Note that this formula can be computed by squaring n times, so it mitigates the computational overhead accompanied by the HE computation. Also, our approximation of $\exp(x)$ is monotone on $[-2^r,\infty)$, so it is more appropriate to approximate softmax functions in terms of getting a high microAUC score compared to other polynomial approximation techniques such as minimax and Chebyshev approximation.

Furthermore, to make the evaluation more stable, we rather consider the approximation of scaled exponential function, $\mathtt{AExp}_{r,L}(x)$ as followings:

$$\mathtt{AExp}_{r,L}(x):=\left(\frac{2^r+x}{L}\right)^{2^r}\approx\left(\frac{2^r}{L}\right)^{2^r}\exp(x)$$

If we carefully choose L large enough that satisfies $|\frac{2^r+x}{L}|<1$ for the all possible choices of x, then the values will not rapidly grow during the evaluation. This makes the computation more stable to the HE implementation which usually supports fixed precision bits.

Goldschdmidt's Algorithm

Goldschmidt's divison algorithm [37] is one of the most popular approximation algorithm to compute the inverse of a real number. For $x\in(0,2)$, Goldschmidt's algorithm uses the following property:

$$\begin{aligned}\frac{1}{x}&=\frac{1}{1-(1-x)}=\sum_{i=0}^{\infty}(1-x)^i\\&\approx\sum_{i=0}^{2^d-1}(1-x)^i=\prod_{i=0}^{d-1}\left(1+(1-x)^{2^i}\right).\end{aligned}$$

Note that the approximation converges rapidly as d grows, and it uses $2d-2$ multiplications to evaluate the approximate polynomial of degree 2^d-1. The

small number of multiplication gives a great advantage on implementation with HE, because HE multiplication between ciphertext is hugely time consuming. The detailed algorithm is described in algorithm 3. Here, note that the Algorithm 3 quickly converges to $1/x$ since the ratio between the error and true value is $\frac{a_d - 1/x}{1/x} = (1-x)^{2^{d+1}}$.

Algorithm 3: Goldschdmidt Method [37]

Input : $x \in (0,2)$, number of iteration $d \in \mathbb{N}$
Output: an approximate value of $1/x$

$a_0 \leftarrow 2 - x$
$b_0 \leftarrow 1 - x$
for $n \leftarrow 0$ *to* $d-1$ **do**
  $b_{n+1} \leftarrow b_n^2$
  $a_{n+1} \leftarrow a_n \cdot (1 + b_{n+1})$
end
return a_d

Moreover, we can easily exploit Goldschmidt's algorithm $\texttt{Gol}(\cdot)$ to approximately evaluate $1/x$ on $(0, 2M)$, instead of $(0,2)$, by using the following equation:

$$\frac{1}{x} = \frac{1}{M}\left(\frac{x}{M}\right)^{-1} \approx \frac{1}{M}\texttt{Gol}\left(\frac{x}{M}\right).$$

We note that the scaling factor M should be carefully chosen since the error accompanied by Goldschmidt's algorithm becomes non-negligiblly large as the input is near 0. Therefore, if we select an overly large M, the input values become smaller, resulting in an error that cannot be ignored.

To put approximate exponential function and Goldschmidt's algorithm together, we now can approximately compute the softmax layer by using HE operations. Denoting $L_r = \left(\frac{L}{2^r}\right)^{2^r}$, we utilize the following equation:

$$\texttt{softmax}(\vec{v}) = \left(\sum_{i=1}^{t} \frac{\exp(v_i)}{L_r}\right)^{-1} \left(\frac{\exp(v_1)}{L_r}, \cdots, \frac{\exp(v_t)}{L_r}\right)$$
$$\approx \texttt{Gol}\left(\sum_{i=1}^{t} \texttt{AExp}_{r,L}(v_i)\right)\left(\texttt{AExp}_{r,L}(v_1), \cdots, \texttt{AExp}_{r,L}(v_t)\right).$$

The detailed algorithm is described in Algorithm 4.

Algorithm 4: Approximate Softmax Algorithm

Input : iteration number $d \in \mathbb{N}$, the pre-determined bounds $r \in \mathbb{N}$ and $M \in \mathbb{R}$, a vector $\vec{v} = (v_1, \cdots, v_t) \in (-2^r, 2^r)^t$
Output: an approximate output of $\texttt{softmax}(\vec{v})$

for $i \leftarrow 1$ *to* t **do**
  $w_i \leftarrow (v_i + 2^r)/2^{r+1}$
  for $j \leftarrow 1$ *to* r **do**
    $w_i \leftarrow w_i^2$
  end
end
$G \leftarrow 1/M \cdot \texttt{Goldschmidt}\left(\sum_{i=1}^{t} w_i/M, d\right)$
for $i \leftarrow 1$ *to* t **do**
  $w_i \leftarrow G \cdot w_i$
end
return $(w_1, \cdots, w_t)$

Results

Parameter Selection for HE

We used CKKS parameters $N = 2^{17}$, $q = 2^{2670}$ with (uniformly random) signed binary secret, which satisfy more than 128-bit security according to Albrecht's LWE estimator [38]. For the scaling factor of CKKS, we choose $\Delta = 2^{60}$ for ciphertexts and $\Delta_{\mathsf{const}} = 2^{40}$ to encode plain vectors for constant multiplication or maksing vectors. All experiments were performed on Intel Xeon CPU E5-2620v4 at 2.10GHz processor and used 8 threads.

Experimental Results

Plain Model Construction

In our experiment, we use the TCGA data that is imported in the same way as used in the 2020 iDASH competition with the same format. In the analysis, we use *Challange* dataset for training and *Evaluation* dataset for inference step, respectively. In order to use each filtering effectively, we experiment for various combinations in plain model. To be suitable for implementation with HE, we fix the gene bound by 2^B for $B = 9, 10$, or 11 first, then change parameters d_{cn} and k_{var} until the filtered number of genes is close to the bound. Here, in order to compare the importance of the CN data and the Variants data, experiments are carried out for the case of fixing one of the parameters d_{cn} or k_{var} to 0 and the case of controlling both to obtain the chosen gene bound. Then, using each chosen pair of parameters, we build plain model by train dataset and then check microAUC value with test dataset. Our experimental results are summarized in Table 2.

According to Table 2, we obtain a much better microAUC by mixing the two filtering than when using only one of the CN data filtering and Variants data filtering. And when more genes are extracted from Variants data filtering than CN data filtering, they tend to have better accuracy. Since we are using a 1-layer neural network, increasing the genes does not necessarily increase the accuracy.

Encrypted Inference Results

Using filtered gene in preprocessing step and the model built in training step, we compute inference step over encrypted data as stated in previous sections. To

compute approximate softmax, we choose parameters $(r, L) = (4, 64)$ for $B = 9, 10$ and $(4, 80)$ for $B = 11$. Then, the parameters d, M for Goldschmidt algorithm are chosen by 30 and 64, respectively. As a result, we estimate the time cost for each round-trip step including encryption, constant multiplication, rotate and summation, Approximate softmax evaluation, and decryption step. The results are stated in Table 3. In the table, the column *Encoding Dup.* means the parameter m that we used as the number of duplication in encryption. As m increases, the number of required ciphertexts increases linearly so the time cost for encryption also similarly increases and the number of rotations decreases as $O(\frac{1}{m} \log \frac{1}{m})$ scale. With such trade-offs, denoting the bound of the number of genes by 2^B, we notice that the time cost is minimized when $m = 4$ for $B = 9$, and $m = 2$ for $B = 10, 11$, respectively. We illustrate the scalability of time cost from m in Figure 2 for $B = 10$. Moreover, the microAUC-enc, real, and linear columns in Table 3 calculate the microAUC values with an approximate softmax algorithm (in encrypted state), an exact softmax algorithm (in plain), and without a softmax (in plain), respectively, with the same model for each B built in the previous step. Since our model was trained with the softmax activation function, if we omit the calculation of the softmax function in the inference step, the microAUC value is greatly reduced. However, using our softmax approximation method, the microAUC value hardly decreases compared to the correct softmax.

Discussion

The main purpose of our data preprocessing is to remove irrelevant genes in the data. The genes removed have a similar effect to the remaining genes, or are have a weak effect on the cancer type classification. CN data filtering extracts the genes with the hamming distance of the samples' copy number, based on the fact that the adjacent genes have copy number similarity. This filtering technique can be applied to other types of the genetic dataset with an appropriate threshold d_{cn} for each dataset. Variants data filtering can be also applied to other types of datasets that have a lot of empty information. Since various genetic datasets are 'empty-sparse', not sparse with lots of 0s but no information, our filtering technique can be used to efficiently extract the relevant features.

Our softmax approximation algorithm works well for any type of data. However, the size of the input data must be adjusted through normalization, and to compute the approximation of exponential function and the Goldschmidt algorithm, the process of predicting the range of the maximum and minimum values of the matrix product XW as a result of the train data should be required. In addition, since our algorithm consumes a lot of depth of HE in approximating softmax, it may not be suitable to compute a deep neural network model having multiple softmax activation functions for a limited time.

Conclusions

In this paper, we adapted the previous data preprocessing methods to reduce the size of the dataset from $25,128$ to 2^B where $B = 9, 10$, or 11. Also, our approximation method for the softmax function enables us to apply the softmax activation function for learning shallow neural network model. As a result, we obtain inference results with microAUC values of about 0.985 to classify multi-label tumor data in 5 minutes. If the time given for the inference step is not limited, our method would be sufficiently scalable to a general deep neural network with softmax activations.

Abbreviations

HE: Homomorphic encryption; DNN: Deep neural network; CNN: Convolutional neural network; SIMD: Single instrument multiple data; CN: Copy number;

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Availability of data and materials

The datasets generated and analysed during the current study are available in [39].

Competing interests

The authors declare that they have no competing interests.

Funding

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2020-0-00840, Development and Library Implementation of Fully Homomorphic Machine Learning Algorithms supporting Neural Network Learning over Encrypted Data).

Author's contributions

SH designed the overall algorithms for this study. JHP contributed to build the structure of model and approximate softmax algorithm. WC contributed to train model and find the best parameters. HC contributed to preprocess data and construct filtered dataset. JHC contributed in writing the manuscript. All authors have read and approved the manuscript. SH takes final responsibility for the manuscript.

Acknowledgements

We would like to acknowledge Dr. Duhyeong Kim for helping to optimize the HE algorithm and having a useful discussion. Furthermore, we would like to acknowledge Prof. Arif Harmanci, who helped us reprocess and publish the dataset in the same way it was used in the iDASH competition.

References

1. Sung, H., Ferlay, J., Rebecca, L.S., Laversanne, M., Soerjomataram, I., Jemal, A., Bray, F.: Global cancer statistics 2020: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries. CA: a cancer journal for clinicians, 209–249 (2021)
2. Yu, J., Ongarello, S., Fiedler, R., Chen, X., Toffolo, G., Cobelli, C., Trajanoski, Z.: Ovarian cancer identification based on dimensionality reduction for high-throughput mass spectrometry data. Bioinformatics **21**(10), 2200–2209 (2005)

3. Nguyen, C., Wang, Y., Nguyen, H.N.: Random forest classifier combined with feature selection for breast cancer diagnosis and prognostic (2013)
4. Yuan, Y., Shi, Y., Li, C., Kim, J., Cai, W., Han, Z., Feng, D.D.: Deepgene: an advanced cancer type classifier based on deep learning and somatic point mutations. BMC bioinformatics **17**(17), 243–256 (2016)
5. He, Z., Zhang, J., Yuan, X., Zhang, Y.: Integrating somatic mutations for breast cancer survival prediction using machine learning methods. Frontiers in genetics **11**, 1853 (2021)
6. Chaudhuri, K., Monteleoni, C., Sarwate, A.D.: Differentially private empirical risk minimization. Journal of Machine Learning Research **12**(3) (2011)
7. Jagannathan, G., Pillaipakkamnatt, K., Wright, R.N.: A practical differentially private random decision tree classifier. In: 2009 IEEE International Conference on Data Mining Workshops, pp. 114–121 (2009). IEEE
8. Ball, M., Carmer, B., Malkin, T., Rosulek, M., Schimanski, N.: Garbled neural networks are practical. IACR Cryptol. ePrint Arch. **2019**, 338 (2019)
9. Chen, H., Gilad-Bachrach, R., Han, K., Huang, Z., Jalali, A., Laine, K., Lauter, K.: Logistic regression over encrypted data from fully homomorphic encryption. BMC medical genomics **11**(4), 3–12 (2018)
10. Kim, D., Son, Y., Kim, D., Kim, A., Hong, S., Cheon, J.H.: Privacy-preserving approximate gwas computation based on homomorphic encryption. BMC Medical Genomics **13**(7), 1–12 (2020)
11. Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. (2020)
12. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. **2012**, 144 (2012)
13. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Annual Cryptology Conference, pp. 75–92 (2013). Springer
14. Wu, X.-Z., Zhou, Z.-H.: A unified view of multi-label performance measures. In: International Conference on Machine Learning, pp. 3780–3788 (2017). PMLR
15. Halevi, S., Shoup, V.: Algorithms in helib. In: Annual Cryptology Conference, pp. 554–571 (2014). Springer
16. 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
17. Chen, Y., Sun, J., Huang, L.-C., Xu, H., Zhao, Z.: Classification of cancer primary sites using machine learning and somatic mutations. BioMed research international **2015** (2015)
18. Yuan, Y., Shi, Y., Su, X., Zou, X., Luo, Q., Feng, D.D., Cai, W., Han, Z.-G.: Cancer type prediction based on copy number aberration and chromatin 3d structure with convolutional neural networks. BMC genomics **19**(6), 1–8 (2018)
19. Sun, Y., Zhu, S., Ma, K., Liu, W., Yue, Y., Hu, G., Lu, H., Chen, W.: Identification of 12 cancer types through genome deep learning. Scientific reports **9**(1), 1–9 (2019)
20. Lee, K., Jeong, H.-o., Lee, S., Jeong, W.-K.: Cpem: Accurate cancer type classification based on somatic alterations using an ensemble of a random forest and a deep neural network. Scientific reports **9**(1), 1–9 (2019)
21. Kim, A., Song, Y., Kim, M., Lee, K., Cheon, J.H.: Logistic regression model training based on the approximate homomorphic encryption. BMC medical genomics **11**(4), 23–31 (2018)
22. Bonte, C., Vercauteren, F.: Privacy-preserving logistic regression training. BMC medical genomics **11**(4), 13–21 (2018)
23. Crawford, J.L.H., Gentry, C., Halevi, S., Platt, D., Shoup, V.: Doing real work with fhe: The case of logistic regression. In: Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. WAHC '18, pp. 1–12. Association for Computing Machinery, New York, NY, USA (2018). doi:10.1145/3267973.3267974. https://doi.org/10.1145/3267973.3267974
24. Blatt, M., Gusev, A., Polyakov, Y., Rohloff, K., Vaikuntanathan, V.: Optimized homomorphic encryption solution for secure genome-wide association studies. BMC Medical Genomics **13**(7), 1–13 (2020)
25. Kim, M., Song, Y., Li, B., Micciancio, D.: Semi-parallel logistic regression for gwas on encrypted data. BMC Medical Genomics **13**(7), 1–13 (2020)
26. Sim, J.J., Chan, F.M., Chen, S., Tan, B.H.M., Aung, K.M.M.: Achieving gwas with homomorphic encryption. BMC Medical Genomics **13**(7), 1–12 (2020)
27. Carpov, S., Gama, N., Georgieva, M., Troncoso-Pastoriza, J.R.: Privacy-preserving semi-parallel logistic regression training with fully homomorphic encryption. BMC Medical Genomics **13**(7), 1–10 (2020)
28. Kim, M., Harmanci, A., Bossuat, J.-P., Carpov, S., Cheon, J.H., Chillotti, I., Cho, W., Froelicher, D., Gama, N., Georgieva, M., et al.: Ultra-fast homomorphic encryption models enable secure outsourcing of genotype imputation. bioRxiv (2020)
29. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
30. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast Fully Homomorphic Encryption Library. https://tfhe.github.io/tfhe/ (August 2016)
31. Kim, A., Song, Y., Kim, M., Lee, K., Cheon, J.H.: Logistic regression model training based on the approximate homomorphic encryption. BMC Medical Genomics **11**(4), 83 (2018)
32. 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)
33. Cheon, J.H., Kim, D., Kim, Y., Song, Y.: Ensemble method for privacy-preserving logistic regression based on homomorphic encryption. IEEE Access **6**, 46938–46948 (2018)
34. Han, K., Hong, S., Cheon, J.H., Park, D.: Logistic regression on homomorphic encrypted data at scale. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 9466–9471 (2019)
35. Cheon, J.H., Han, K., Hong, S.M., Kim, H.J., Kim, J., Kim, S., Seo, H., Shim, H., Song, Y.: Toward a secure drone system: Flying with real-time homomorphic authenticated encryption. IEEE Access **6**, 24325–24339 (2018). doi:10.1109/ACCESS.2018.2819189
36. Chollet, F., et al.: Keras. https://keras.io (2015)
37. Goldschmidt, R.E.: Applications of division by convergence. PhD thesis, Massachusetts Institute of Technology (1964)
38. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of Learning with Errors. Cryptology ePrint Archive, Report 2015/046. https://eprint.iacr.org/2015/046 (2015)
39. Harmanci, A.O.: TCGA Dataset Processed in the Same Way as Idash 2020. https://drive.google.com/drive/folders/1r9VV5D6S0aOo9aW1wVE2vC_lrS9KTkfj?usp=sharing

Figures
Tables

Figure 1 Workflow of the data preprocessing. (a) CN data filtering. CN data matrix C consists of $s = 2{,}713$ samples and $g = 25{,}128$ genes, where each entry represents the copy number of the corresponding sample and gene in 5 levels: 0, ± 1, ± 2. The CN data filtering is based on the copy number similarity of adjacent genes. (b) Variants data filtering. Variants data matrix V_t for each tumor type $t = 1, 2, \cdots, 11$ consists of mutations' effect data. The data is encoded to real values 0, 0.2, 0.5, 0.9, and 1.0 where 0 is for the entry with no information. The Variants data filtering removes irrelevant genes with ineffective mutations. (c) Input data to Shallow Neural Network. The filtered CN and Variants data is concatenated and input to our Neural Network.

Figure 2 Time Graph for $B = 10$. As encoding duplication number increaes, we can check that the time cost for Encryption increases linearly, the cost for RotSum decrease approximately log scale, and the other parts are stationary. As a result, the total time cost is minimal when duplication number is 2.

Table 1 Sample and mutation statistics of the dataset on 11 cancer types. Note that the total number of genes are 25,128 and the number in Mutation's Effect column means that the number of non-zero values in Variants dataset (which is less than # of samples $\times$ # of genes)

Cancer Site	Samples	Mutation's Effect			
		LOW	MODERATE	MODIFIER	HIGH
Bladder	258	25,641	61,862	11,303	9,616
Breast	201	11,915	30,966	8,335	7,828
Bronchus / Lung	638	61,277	166,945	28,039	25,819
Cervix uteri	149	12,084	28,515	14,715	4,278
Colon	256	42,501	105,179	29,525	24,320
Corpus uteri	219	139,405	364,241	167,526	60,846
Kidney	149	5,425	13,772	3,684	3,155
Liver / Intrahepatic bile ducts	189	7,198	19,697	6,186	3,149
Ovary	151	6,477	17,218	3,569	3,663
Skin	254	107,923	197,015	34,699	22,051
Stomach	249	32,972	78,593	14,630	20,571
Total	2,713	452,818	1,084,003	322,211	185,296

Table 2 Table of experiment results for plain model construction. In this experiment, we use 70% of training dataset to find best parameters in training phase. We compute microAUC using exact softmax function in inference step. the column *filtered number of genes* means the number of genes which are extracted by CN data filtering or Variants data filtering and the column *Parameter* $d_{\text{cn}}, k_{\text{var}}$ means the threshold values which divide the group in each filtering. Each column with asterisk(*) in microAUC shows the best result for each bound.

Gene Bound	Parameters		Filtered number of genes			microAUC
	d_{cn}	k_{var}	CN	Variants	Total	
512	0.103	·	510	·	510	0.9459
	·	102	·	510	510	0.9216
	0.17	50	241	252	493	0.9795*
1024	0.064	·	1023	·	1023	0.9486
	·	205	·	1007	1007	0.9294
	0.103	102	510	510	1020	0.9834
	0.12	120	596	420	1016	0.9845*
2048	0.0372	·	2031	·	2031	0.9485
	·	430	·	1975	1975	0.9313
	0.064	205	1023	1007	2030	0.9831
	0.1	300	1447	538	1985	0.9868*

Table 3 Table of experiment results for inference step over encrypted state. For each bound of genes, the number of chosen genes for CN ard variants data are as in Table 2. Encoding Dup. means the number of duplications for X in encoding step, referes the parameter m. For microAUC, the column *enc* means the microAUC value with results computed with encrypted algorithm, while the values in column *real* are computed with exact formula and the values in column *linear* are computed without softmax function.

Gene Bound	Encoding Dup.	Enc Computation (sec)						microAUC		
		Encryption	Const. Mult	RotSum	A. Softmax	Decryption	Total	enc	real	linear
512	1	7.005	16.235	62.967	134.041	0.1	220.348	0.9853	0.9854	0.9547
	2	13.555	17.661	29.202	137.136	0.1	197.654			
	4	26.453	17.827	11.685	138.154	0.1	194.219			
	8	52.501	23.489	6.415	135.881	0.1	218.386			
1024	1	13.395	32.761	62.530	135.503	0.1	244.289	0.9873	0.9867	0.9660
	2	26.861	35.550	29.552	135.727	0.1	227.790			
	4	53.586	35.751	12.277	134.690	0.1	236.404			
	8	105.695	48.212	7.097	135.603	0.1	296.707			
2048	1	26.984	64.518	65.919	135.716	0.1	293.237	0.9884	0.9880	0.9617
	2	52.952	71.689	30.074	136.038	0.1	290.853			
	4	107.086	72.634	13.032	139.519	0.1	332.371			
	8	211.831	100.192	9.119	148.258	0.1	469.500			

Figures

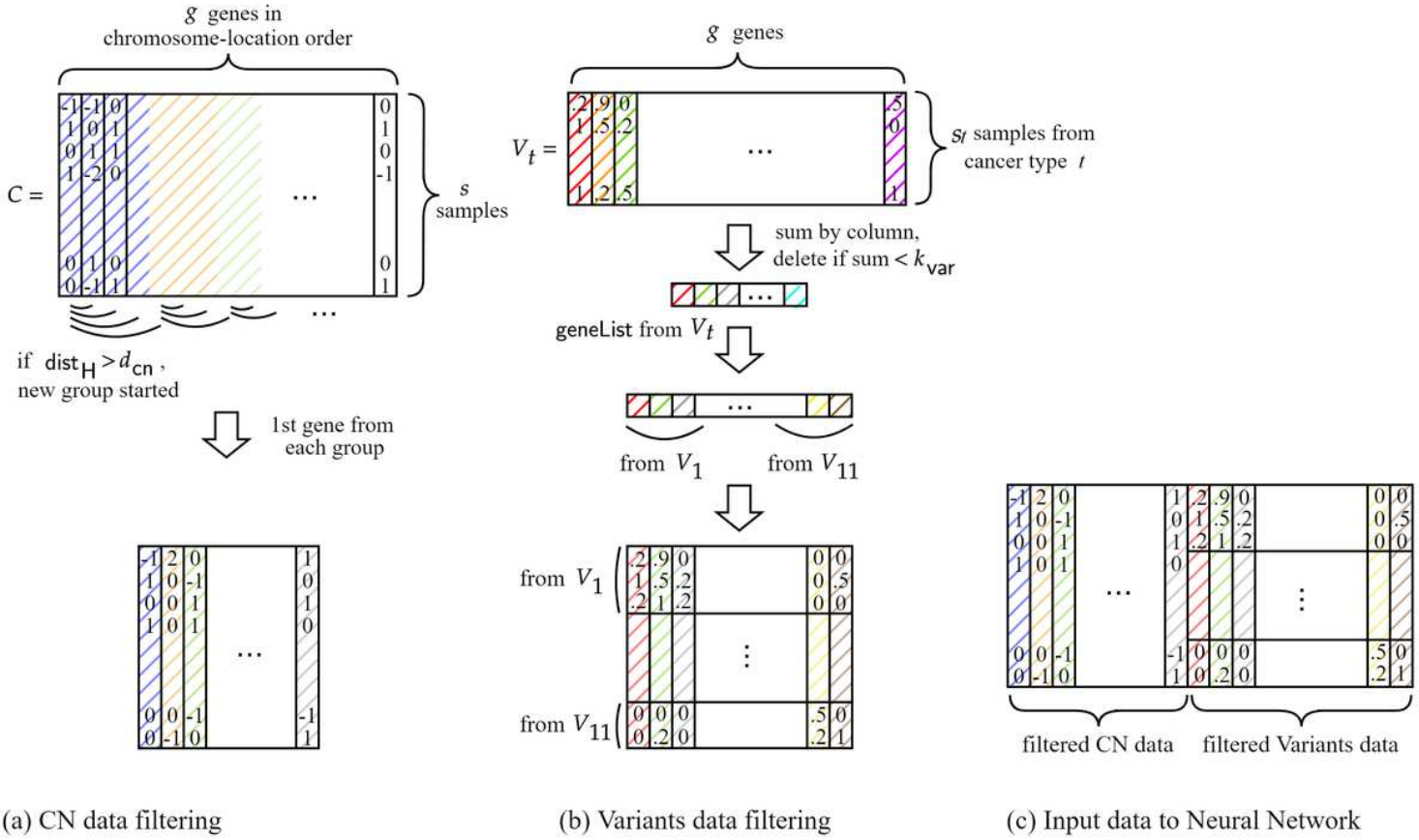

Figure 1

Work flow of the data preprocessing. (a) CN data filtering. CN data matrix C consists of s = 2,713 samples and g = 25,128 genes, where each entry represents the copy number of the corresponding sample and gene in 5 levels: 0, ± 1, ± 2. The CN data filtering is based on the copy number similarity of adjacent genes. (b) Variants data filtering. Variants data matrix Vt for each tumor type t = 1; 2; • • • ; 11 consists of mutations' effect data. The data is encoded to real values 0, 0.2, 0.5, 0.9, and 1.0 where 0 is for the entry with no information. The Variants data filtering removes irrelevant genes with ineffective mutations. (c) Input data to Shallow Neural Network. The filtered CN and Variants data is concatenated and input to our Neural Network.

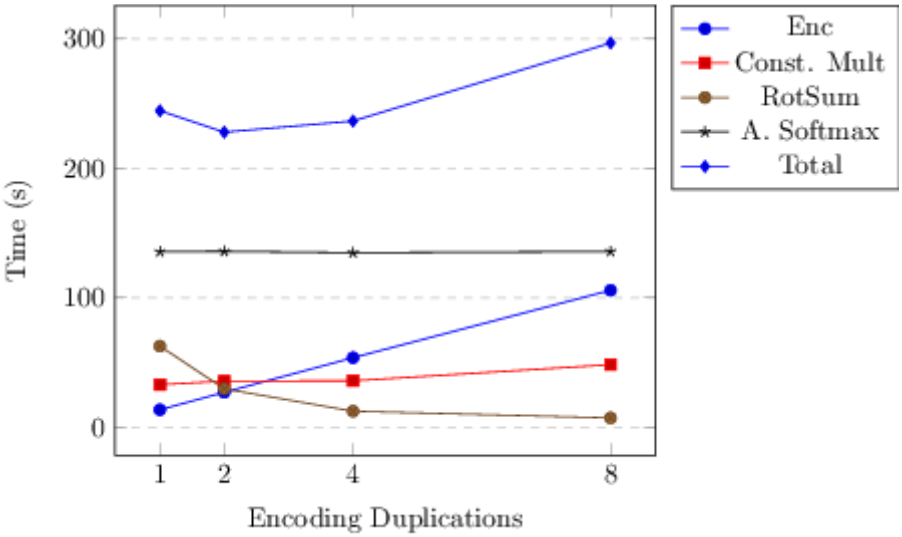

Figure 2

Time Graph for B = 10. As encoding duplication number increaes, we can check that the time cost for Encryption increases linearly, the cost for RotSum decrease approximately log scale, and the other parts are stationary. As a result, the total time cost is minimal when duplication number is 2.