ML with HE: Privacy Preserving Machine Learning Inferences for Genome Studies

ŞEYMA SELCAN MAĞARA, Sabanci University, Turkey
CEREN YILDIRIM, Sabanci University, Turkey
FERHAT YAMAN, North Carolina State University, USA
BERKE DILEKOĞLU, Sabanci University, Turkey
FURKAN REHA TUTAŞ, Sabanci University, Turkey
ERDINÇ ÖZTÜRK, Sabanci University, Turkey
KAMER KAYA, Sabanci University, Turkey
ÖZNUR TAŞTAN, Sabanci University, Turkey
ERKAY SAVAŞ, Sabanci University, Turkey

Preserving the privacy and security of big data in the context of cloud computing, while maintaining a certain level of efficiency of its processing remains to be a subject, open for improvement. One of the most popular applications epitomizing said concerns is found to be useful in genome analysis. This work proposes a secure multi-label tumor classification method using homomorphic encryption, whereby two different machine learning algorithms, SVM and XGBoost, are used to classify the encrypted genome data of different tumor types.

1 INTRODUCTION

Genome analysis is a popular and rapidly growing area; yet it is crucial to preserve the security and privacy of sensitive genome data while being able to work on it in an efficient manner. Our work is based on iDASH 2020 Competition\(^1\) Track 1, for which the goal was to classify genomes of different tumors with their respective types, such as skin, colon, kidney etc. Due to the high sensitivity of the genomic data, it is crucial to ensure confidentiality. For this purpose we implemented a homomorphic classification model and while doing that, we aimed to achieve a high rate of accuracy in a relatively short amount of time. Two different machine learning algorithms were selected to perform this classification task: SVM and XGBoost\(^2\). We initially worked with SVM over the encrypted space since it is useful for classifying models with high dimensional spaces. Furthermore, we implemented an XGBoost algorithm as well since tree based machine learning algorithms have not been widely used in homomorphic applications before. Finally, the accuracies of both algorithms were compared to each other.

2 BACKGROUND

2.1 Homomorphic Encryption

Homomorphic encryption (HE) is an encryption technique that allows computation on encrypted data without decrypting it first. This way the data remains confidential while processing it, hence private data can be outsourced to be processed in untrusted environments [Armknecht et al. 2015]. Fully homomorphic encryption (FHE) allows arbitrary computations on encrypted data without the decryption key. Given encryptions $E(m_1), \ldots, E(m_t)$ of messages $m_1, \ldots, m_t$, FHE scheme allows the user to apply the function $f$ on the encrypted messages and obtain a ciphertext which decrypts to $f(m_1, \ldots, m_t)$ [Gentry 2009]. There are also different FHE schemes such as BFV [Fan and Vercauteren 2012], CKKS [Cheon et al. 2017].

BFV scheme utilizes homomorphic operations on integer values while CKKS uses floating point numbers for approximate calculations.

2.2 SVM and XGBoost

2.2.1 SVM. Support Vector Machine (SVM) is a supervised machine learning algorithm that is used for classification and regression problems. SVM non-linearly maps the input vectors into a high-dimensional feature space and aims to construct a hyperplane that results in the largest margin $w$ between the classes. Given a set of labelled training patterns $(y_i, x_i), \ldots, (y_i, x_i), y_i \in \{-1, 1\}$, is said to

\(^1\)IDASH’20 competition details at http://www.humangenomeprivacy.org/2020/competition-tasks.html

\(^2\)Code is available at https://github.com/SU-CISEC/MLwithHE
be linearly separable if there exists a vector \( w \) and scalar \( b \) such that
\[
y_i (w \cdot x_i + b) \geq 1, \quad i = 1, ..., l
\]
The optimal hyperplane is \( w_0 \cdot x + b_0 = 0 \) the unique one which separates the training data with the maximal margin [Cortes and Vapnik 1995].

2.2.2 XGBoost. XGBoost is a supervised learning algorithm that uses gradient-boosted tree ensembles. Training data is used to construct the model that consists of a set of classification and regression trees (CART). Unlike regular decision trees, CARTs do not contain decision values on the leaves. Instead, each leaf holds a numerical score (i.e., similarity score). Inputs are evaluated on all the trees of the ensemble, which are then classified into one of the leaves. Finally, the numerical scores that we get from each tree are summed up to form a final prediction score. This tree ensemble model that is used to predict the outputs can be shown as
\[
\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in F
\]
where \( x_i \) is the input, \( K \) is the number of the trees in the ensemble, \( F \) is the space of regression trees and each \( f_k \) corresponds to an independent tree structure \( q \) and leaf weights \( w \) [Chen and Guestrin 2016].

In the model, each class holds the same number of trees that are used to predict the score of the class for the given input. Once the input is evaluated on the trees of a class, predicted scores from these trees are added up to form a final score for the respective class. The prediction result for the input is the class that holds the highest score value [Chen and Guestrin 2016].

2.3 Previous Works

Nowadays, machine learning models are used widely in real-life problems. Most of the data used for both training and inference are sensitive information. Thus, preserving privacy and the security of the data has become a priority issue that needs to be solved. Many research conducted on the inference phase such as [Teo et al. 2013] and [Omer et al. 2016]. Recently, [Park et al. 2020] proposed a secure training method to protect both model information and training data. In this work, it assumed that the model was stored in a secure machine. So we focus on the prediction phase.

Although XGBoost is relatively newer than SVM, it is a commonly used and effective algorithm. Yet, applying HE on XGBoost is a challenging task because the prediction phase of the XGBoost algorithm depends on comparison operation. [Meng and Feigenbaum 2020] proposed a privacy-preserving XGBoost inference method implemented on Amazon SageMaker. There are also secure and privacy-preserving XGBoost studies that don’t use HE. [Law et al. 2020] implemented a solution using hardware enclaves.

3 METHODOLOGY

To achieve the most accurate machine learning model for the given genome data, we implemented multiple algorithms with various parameters. Then we evaluated them based on accuracy, speed, and feasibility. Among these, the most proper two models, SVM and XGBoost, are selected to adapt to HE computations. First, we will explain the data set. Then, in the subsections, SVM and XGBoost adaptations are detailed.

The database of tumor classification task was provided by the IDASH’20 competition. There are total of 2713 unique patient id among 11 different cancer categories in the training data.

Two types of feature data set were given in the competition. The first one includes the variants of the somatic mutations for all the samples. The second one consists of the copy number states for each tumor type. Copy number states represent deletion and amplification on genes. They are given in 5 different levels where -2 and -1 represent a deletion while 1 and 2 represent an amplification. If none of these are present, it is indicated by 0. We encode our data by using -1 for deletion, 1 for amplification and 0 for neither of these to reduce the complexity.

3.1 SVM Model

We implemented one vs. all SVM classification. The algorithm finds optimal hyperplanes that separate unique class from all other classes in training phase. In the inference phase, confidence values \( y_i \) are calculated by dot product of feature vector \( X \) and support vector of hyperplane \( W_i \) with addition of bias value \( b_i \) for every class \( i \).
\[
y_i = (W_i \odot X + b_i), \quad i = 1, ..., s
\]
Feature values \( X \) are given encrypted and our model parameters \( W \) and \( B \) matrices used plaintext in computations. Converting those operations into homomorphic space efficiently requires adjustment of feature size and other homomorphic parameters. We decrease our feature size to minimize inference time while keeping accuracy high. We implemented our homomorphic computations using PALISADE[PAL 2021] HE library. Due to floating point numbers in \( W \) matrix, we adopted CKKS scheme. Encoding method in CKKS scheme allows to use single instruction multiple data (SIMD) parallelization. Implementation details of homomorphic SVM inference is given in Algorithm 1. Single ciphertext feature vector \( X \) and packed weight vector \( W \) are multiplied by ciphertext-plaintext homomorphic multiplication using library function. That produces elementwise multiplication results in one packed ciphertext. To calculate dot product, we utilized summation operation in single ciphertext of multiplication results. After that, bias values \( B \) added by ciphertext-plaintext homomorphic addition which returns confidence values \( Y \) of classification results in encrypted format.

3.2 XGBoost Model

The depth of the tree and the number of trees are two critical parameters that affect the performance. We prefer the depth of the tree to be small since it is directly proportional to the complexity of the model. Also, trees are grown one after another due to the boosting feature of the XGBoost. So, an increase up to a point in the number of trees will enhance accuracy. Considering these facts and evaluating models with various parameters, we selected the depth as two and the number of trees as 128. Hence, the model consists of 128*11 trees of depth 2. The model searches for optimal split-point for the feature values and leaf scores. For our model, the split points of the nodes are either -0.5 or +0.5. Test data is led into different leaves, and the score of the leaf was added to the corresponding class. A sample tree from the model is given in the 1. In this tree,
**Algorithm 1:** Homomorphic SVM Inference

\textbf{Input}: X: Encrypted feature values  
W: Weight Matrix  
B: Bias Vector  
s: Number of Classes  

\textbf{Output}: Y: Confidence values.

1. \textbf{for} i = 0, 1, \ldots, s - 1 \textbf{do}
2. \hspace{1em} W[i] \leftarrow \text{CKKS.PackedPlaintext}(W[i]);
3. \hspace{1em} t \leftarrow X \cdot W[i];
4. \hspace{1em} t \leftarrow \text{CKKS.Sum}(t);
5. \hspace{1em} Y[i] \leftarrow t + B[i];
6. \textbf{end}
7. \textbf{return} Y

---

**Table 1:** encoding table

| test data value | x2 | x1 | x0 |
|-----------------|----|----|----|
| -1              | 0  | 0  | 1  |
| 0               | 1  | 0  | 0  |
| +1              | 1  | 0  | 0  |

---

the root node compares the value of the 25486th feature of the test data to -0.5. If it is smaller it goes to the left node, otherwise to the right node. The challenging part is comparing the node values with the test data because the data should be encrypted for the sake of privacy. Hence, we decided to convert the comparison operations to logical gate operations and implemented an efficient encoding method similar to one-hot encoding.

Since the split points of the nodes can take only two values, 1 bit is sufficient to encode the node values. Therefore, split values equal to 0.5 are replaced with 0 and -0.5 is replaced with 1. We show the encoded split-points with letter y in this paper. The test data is represented by only 3 possible values: -1, 0, 1. Therefore, 3 bits are needed to represent test data with one-hot encoding, namely x0, x1 and x2. If, value of a feature is -1 then x2, x1 and x0 values are mapped to 0, 0, 1. Full encoding is given in Table 1.

We used Karnaugh maps (kmap) in order to achieve an optimized boolean expression that has the same behaviour as the comparison operation. As a result, we expressed a single comparison as

\[ \neg x_2 \land (\neg y \lor x_0) \]

Notice that we were able to formulate it without the x1 term. This significantly reduces the complexity. Since we are working with BFV and CKKS HE schemes, we are limited to arithmetical operations. However all the terms are in the binary field. Thus we can convert this boolean expression into an arithmetical expression effortlessly:

\[ z = (1 - x_0) \cdot (x_1 + (y - 1) - y) + 1 \]

Using this formula we can compare the value of a feature with the value of a node. If it is smaller result will be 1, otherwise 0. Remember that, prediction data is encrypted. Thus, the result of the comparison is also encrypted. We need to reach the correct leaf node without decrypting it. To solve this issue, we took advantage of node values being boolean.

In complete binary trees of depth 2, there are four possible paths. As shown in the Fig. 2, truth value of a node i is shown with z_i and the score of the leaves with c_i. Then, the values of paths can be written as:

\[
\begin{align*}
    p_1 &= z_1 \cdot z_2 \cdot c_1 \cdot p_2 = z_1 \cdot (1 - z_2) \cdot c_2 \\
    p_3 &= (1 - z_1) \cdot z_3 \cdot c_3 \cdot p_4 = (1 - z_1) \cdot (1 - z_3) \cdot c_4
\end{align*}
\]

Note that, only one of these, the correct path, can be nonzero at a time. Others will be zero. So, summation of the all p_i’s gives the score of a single tree. Simplified score expression will be:

\[
\text{score} = (z_1 - 1) \cdot l_3 + z_3 + (z_2 - l_1 + l_2) \cdot z_1 + l_4
\]

4 IMPLEMENTATION

The basic outline of how this detection system works is given as below:

1. The model is not encrypted. On the other hand, the genomic input data is given in encrypted form. Hence, the client sends their data to the server and computation is done on the cloud side.

2. Once the input data is received, the cloud server computes the decision values for each node of every tree in the model. This is done by comparing the numeric values given in the inner
### Algorithm 2: Comparing Feature Values With Leaf Values

**Input:** \( X_i = \{x_{1i}, x_{2i}, x_{3i}\} \): Encoded feature values of all trees appended in order.
\( y_i \): Encoded node values of all trees appended in order.

**Output:** \( z \): Comparison result of a single node value with the feature value it holds.

1. For \( i = 0, 1, \ldots, m - 1 \)
2. \( z[i] \leftarrow (1 - x_{1i}) \times (x_{2i} \times (y_i - 1) - y_i) + 1 \)
3. Return \( z \)

### Algorithm 3: Calculating Tree Score

**Input:** \( (z_1, z_2, z_3) \): Comparison results for each inner node.
\( L = \{l_1, l_2, l_3, l_4\} \): List of leaf values.
\( m \): Number of genetic information inputs.

**Output:** \( R \): List of final scores of each genetic information input.

1. \( R \leftarrow (0)^{1 \times m} \)
2. For \( i = 0, 1, \ldots, m - 1 \)
3. \( R[i] \leftarrow (z_1 - 1) \times l_3 + z_3 + (z_2 \times l_2 + l_4) + z_1 + l_4 \)
4. Return \( R \)

There are three inner nodes in each tree. Hence, `compareNode` function is called three times for each inner node and the comparison values of the root node, left child and right child are stored in \( z_1, z_2 \) and \( z_3 \) respectively. Once we have the comparison results for every node, `calculateTree` function is called. This function returns an array that holds the leaf score of the predicted output for each tree.

# 5 RESULTS

We participated in the IDASH’20 competition with the SVM model, and achieved remarkable results as seen in the Figure 4. In addition, an XGBoost model is developed using different HE libraries(SEAL, SEAL 2020 and Palisade) and schemes. The comparison between XGBoost and SVM models is shown in Table 2. Based on the measurements, the XGBoost model is superior to SVM in terms of accuracy while being slower. This is expected because XGBoost has a higher multiplication depth, and it uses larger data for the prediction phase. Since the competition ranked the participants using a separate data set, a direct comparison is not possible. However, it is possible to make an inference about the performance of the XGBoost model by comparing it to other participants’ results. We have achieved a significant increase in the accuracy with the XGBoost model. While the speed of the XGBoost model is lower than that of the SVM model’s, it still compares well to other participants’ timing results. We believe this improvement over the accuracy makes up for the increase in end-to-end timing.

### Table 2. Performance Table: Execution times (Key generation, encryption, computation, decryption and total timing) and MicroAUCs

| Team Name            | Institution          | Country       | End-End Time (Sec.) | AUC   |
|----------------------|----------------------|---------------|---------------------|-------|
| SVM                  | Palisade             | 0.06s         | 0.078s              | 2.748s| 1.029s | 3.915s | 0.96  |
| XGBoost              | SEAL                 | 0.31s         | 1.31s               | 5.106s| 1.671s | 8.397s | 0.981 |
| XGBoost              | Palisade             | 0.257s        | 4.984s              | 10.501s| 10.218s| 25.96s | 0.981 |
| XGBoost(Encrypted Model) | Palisade             | 0.07s         | 5.799s              | 16.664s| 1.399s | 23.932s| 0.981 |

Fig. 4. IDASH’20 End-End time sorted rankings Accessed 2 August 2021, <http://www.humangenomeprivacy.org/2020/agenda.html>
7 ACKNOWLEDGEMENT
C. Yıldırım, Dr. Öztürk and Dr. Savaş are supported by TUBITAK under grant number 118E725.

REFERENCES
2021. PALISADE Lattice Cryptography Library (release 1.11.2). https://palisade-crypto.org/
Frederik Armknecht, C. Boyd, Christopher Carr, K. Gjøsteen, Angela Jäschke, Christian A. Reuter, and Martin Strand. 2015. A Guide to Fully Homomorphic Encryption. IACR Cryptol. ePrint Arch. 2015 (2015), 1192.
Tianqi Chen and Carlos Guestrin. 2016. XGBoost. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (Aug 2016). https://doi.org/10.1145/2939672.2939785
Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In Advances in Cryptology – ASIACRYPT 2017, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer International Publishing, Cham, 469–487.
Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. Machine Learning 20, 3 (1995), 273–297. https://doi.org/10.1007/bf00994018
Junfeng Fan and F. Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. IACR Cryptol. ePrint Arch. 2012 (2012), 144.
Craig Gentry. 2009. A fully homomorphic encryption scheme. Ph.D. Dissertation.
Andrew Law, Chester Leung, Rishabh Poddar, Rahul Ada Popa, Chenyu Shi, Octavian Sima, Chaofan Yu, Xingmeng Zhang, and Wenting Zheng. 2020. Secure Collaborative Training and Inference for XGBoost. arXiv:2010.02524 [cs.CR]
Xianru Meng and Joan Feigenbaum. 2020. Privacy-Preserving XGBoost Inference. arXiv:2011.04789 [cs.CR]
Mohammed Z. Omer, Hui Gao, and Faisal Sayed. 2016. Privacy Preserving in Distributed SVM Data Mining on Vertical Partitioned Data. In 2016 3rd International Conference on Soft Computing Machine Intelligence (ISCMI). 84–89. https://doi.org/10.1109/ISCMI.2016.40
Saerom Park, Junyoung Byun, Joohee Lee, Jung Hee Cheon, and Jaewook Lee. 2020. HE-Friendly Algorithm for Privacy-Preserving SVM Training. IEEE Access 8 (2020), 57414–57425. https://doi.org/10.1109/ACCESS.2020.2981818
SEAL 2020. Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL
Microsoft Research, Redmond, WA.
Sin G. Teo, Shuanguo Han, and Vincent C.S. Lee. 2013. Privacy Preserving Support Vector Machine Using Non-linear Kernels on Hadoop Mahout. In 2013 IEEE 16th International Conference on Computational Science and Engineering. 941–948. https://doi.org/10.1109/CSE.2013.200