Towards a Homomorphic Machine Learning Big Data Pipeline for the Financial Services Sector

Oliver Masters*
oliver.masters@ibm.com

Hamish Hunt*
hamishun@uk.ibm.com

Enrico Steffinlongo*
enrico.steffinlongo@ibm.com

Jack Crawford*
jack.crawford@ibm.com

Flavio Bergamaschi*
flavio@uk.ibm.com

ABSTRACT

Machine Learning (ML) is today commonly employed in the Financial Services Sector (FSS) to create various models to predict a variety of conditions ranging from financial transactions fraud to outcomes of investments and also targeted upselling and cross-selling marketing campaigns. The common ML technique used for the modeling is supervised learning using regression algorithms and usually involves large amounts of data that needs to be shared and prepared before the actual learning phase. Compliance with recent privacy laws and confidentiality regulations requires that most, if not all, of the data and the computation must be kept in a secure environment, usually in-house, and not outsourced to cloud or multi-tenant shared environments. Our work focuses on how to apply advanced cryptographic schemes such as Homomorphic Encryption (HE) to protect the privacy and confidentiality of both the data during the training of ML models as well as the models themselves, and as a consequence, the prediction task can also be protected. We de-constructed a typical ML pipeline and applied HE to two of the important ML tasks, namely the variable selection phase of the supervised learning and the prediction task. Quality metrics and performance results demonstrate that HE technology has reached the inflection point to be useful in a financial business setting for a full ML pipeline.

KEYWORDS

homomorphic encryption; variable reduction; variable selection; feature selection; prediction

1 INTRODUCTION

Homomorphic encryption (HE) promises to generally transform and disrupt how business is currently done in many industries such as, but not limited to, healthcare, medical sciences, and finance. One particular area of interest and value to apply HE across numerous industries is in machine learning (ML). The ability to compute directly on the encrypted data allows that data to be shared in areas that were once considered impossible or highly undesirable due to data leaks through single point of failure (individuals or systems with the authority to see the data) which could be insecure.

Today, organizations make far more use of vast amounts of aggregated data to be able to perform data analytics and ML. Many organizations find themselves restricted from sharing data, internally and externally, due to legislation, regulation, and their other need-to-know policies coming into direct conflict with the need to collaborate by sharing the data (a.k.a. need-to-share). Approaches leveraging homomorphic encryption can overcome these restrictions by allowing homomorphic data aggregation intra- and/or inter-organization; meaning that a computation requiring the aggregated data can be performed without other parties having access to data shared in the aggregation.

HE as a technology has undergone accelerated progress since Gentry's influential work [13] showed how to construct a fully homomorphic encryption scheme based on lattices. Several schemes and algorithmic improvements have emerged since Gentry such as the BGV [4] and FV [11] schemes. The community is aware that the technology is becoming adequately performant to be useful and/or disrupt several areas [1]. In the last few years, the CKKS scheme [8] has emerged offering a more natural setting for performing operations on approximate numbers. CKKS is thus generally more suitable to analytics and ML problems.

The terminology *Machine Learning*, first introduced by Arthur Samuel in 1959 [24], today comprises several tasks with the fundamental goal of creating a model that can make predictions. Model generation by learning is the main focus of ML and the motivation in doing this homomorphically has been around for a few years. Many solutions have been shown to perform this task with varying times from minutes to hours using different HE schemes [3, 7, 9, 16, 19, 20]. However, practitioners are aware that in a typical ML pipeline this is but one necessary task.

Our work consists of exploring two tasks in the ML pipeline that HE can aid in the sharing of data. The first is running the prediction of a generated logistic regression model. This is the task that is the re-usable part of the typical ML endeavor. Businesses will want to ensure that only certain parts of the business will have access to the model and/or data. Although this tends to be inherently performed in the learning aspect it has had little attention to as a separate facet and metrics on it seem somewhat limited in the literature. Moreover, in previous works [5, 14] the speed of prediction was achieved through having the model itself unencrypted, thus only providing privacy of the input data. This work explores the concept of keeping the generated model private in addition to the data. The second task that we explore is performing variable reduction or more precisely variable selection (a.k.a. feature selection in the literature). With real data, this is a very common machine learning pipeline task and necessary to avoid overfitting of the data and/or only perform learning with variables of importance thus reducing resource required.

To achieve our goals, we applied state-of-the-art techniques in homomorphic encryption and ML. For our homomorphic encryption and computations, we used the homomorphic encryption library HElib [15], explicitly making use of its CKKS capabilities in

*IBM Research, Hursley, UK

the work presented. Firstly, we took an existing, encrypted logistic regression model that constitutes sensitive intellectual property and demonstrated the feasibility of running a large number of encrypted prediction operations on real, encrypted financial data while retaining acceptable performance with both 128 and 256 bits of security. Secondly, we built on work by Bergamaschi *et al.* [2] by exploring the feasibility of homomorphic variable selection.

2 BACKGROUND

In this section, we will introduce the key concepts which will be required throughout this work. All homomorphic computations were done using HElib's CKKS capabilities that were introduced to the library in 2018 [2, 15]. This allows us to code using approximates of real numbers. To solve both ML tasks of predicting and variable selection, this is required. The way we determine the importance of a variable for variable selection is to use the evaluation of a logistic regression model trained on that variable individually.

2.1 CKKS in HElib

The CKKS scheme [8] has provided a large change for certain problems of how we think about applying HE. In HElib's variant of the scheme the ciphertext mechanisms are mostly the same as they are for the BGV scheme [2]. The main difference lies in the CKKS plaintext space which we can therefore take advantage of.

CKKS has a decryption invariant form of $[\langle \mathsf{sk}, \mathsf{ct}\rangle]_q = \tilde{\mathsf{pt}}$, where sk and ct are the secret key and ciphertext vectors, respectively, $[\cdot]_q$ denotes reduction modulo q into the interval $(-q/2, q/2]$, and $\tilde{\mathsf{pt}}$ is an element that encodes the plaintext and includes also some noise. CKKS uses an element $\tilde{\mathsf{pt}}$ of low norm, $|\tilde{\mathsf{pt}}| \ll q$. Decoding to a plaintext, pt, is given by $\tilde{\mathsf{pt}} = \mathsf{e} + \Delta \cdot \mathsf{pt}$ where Δ is a scaling factor and, ideally, after performing our necessary computation we still have $|\mathsf{e}| < \Delta$.

Due to working with approximations of real numbers the scheme supports varying levels of precision determined by the accuracy parameter r. The noise, e, introduced during the encoding of the plaintext causes each operation performed in the CKKS scheme to be accurate up to an absolute bound on the magnitude of the additive noise, namely 2^{-r}.

The HElib implementation of the CKKS scheme maps to a plaintext space that is the integer polynomial ring $\mathbb{Z}[X]/\langle \Phi_m(X)\rangle$ where $\Phi_m(X)$ is the m^{th} cyclotomic polynomial with degree given by Euler's totient $\phi(m)$. The scheme provides encode and decode procedures to map the native plaintext elements to and from plaintext complex vectors $v = \mathbb{C}^l$ where $l = \phi(m)/2$ determines the number of complex numbers that can be packed into a single plaintext. For our purposes, we only make use of the real part of the numbers.

2.2 Homomorphic predictions

Given a trained ML model, its primary purpose is the generation of an output estimate of whether a given input has the condition or not. This is known as prediction. Many types of predictive models can be considered to be another form of data which can be encrypted homomorphically.

Depending on the scenario there are choices to be made as whether the data, the model or both are homomorphically encrypted. In all cases, the output will be encrypted as an operation between a ciphertext or a plaintext with a ciphertext always results in a ciphertext.

The first proposal of a privacy preserving Encrypted Prediction as a Service (EPaaS) solution was CryptoNets [14] in 2016. CryptoNets achieved 99% accuracy and a throughput of roughly 59000 predictions per hour.

When applying a prediction model in an HE context, careful consideration must be taken to find a balance between the accuracy and the computational complexity. This is due to the natural overhead that is introduced by encryption. Previous work has been carried out to reduce both the limitation on the depth and breadth of the circuit that can be computed as well as the latency of such applications.

One such notable work to produce a low latency, homomorphic neural network known as LoLa [5] presents an application that achieves considerable speedups without sacrificing on the level of security provided in previous attempts. This was achieved via the use of alternative data representations during the computation process. This application exhibited the feasibility of performing homomorphic predictions however left the exploration of homomorphically performing the training of ML models to further study.

It should be noted that these previous schemes do not perform the prediction using a homomorphically encrypted model thus making the prediction less computationally expensive.

2.3 Variable Selection via Logistic Regression

Variable selection is the process of deciding which of the variables (or features) of a given dataset are important to be kept when generating a predictive model. This also determines the variables which are not worth preserving as they have negligible or detrimental impact on the model's predictive quality [12, 18].

Homomorphic model generation by learning is a topic of increasing interest due to the ability to generate models with training data that is encrypted. This is important in scenarios where the data used for the training is private. In particular medical data is considered highly confidential and there is focus on applying HE to this sector. Another key industry in which data privacy and ML techniques are of particular interest is the financial sector.

Most notably, the work [3, 7, 9, 20] related to the 2017 iDASH competition [17] as well as [16, 19] explore the use of logistic regression on homomorphically encrypted data to generate models. These achieve varying computation times for data samples of differing sizes. Applications range from 6 minutes for over 1500 samples containing 18 features in [19] to generating a model from over 420000 samples containing over 200 features in approximately 17 hours [16]. These works demonstrate both the reality of generating a model homomorphically in a feasible amount of time as well as the scalability of such methods to handle large datasets.

When attempting to predict a binary condition or attribute (also known as *classification*) based on other attributes given (not necessarily binary themselves), logistic regression is a standard ML technique employed.

In this work, we are only dealing with the case where the condition that we want to predict is binary (i.e. with condition or without condition). The data, which one can consider to form a matrix, consists of n records or rows of the form $(y_i, \vec{x}_i)$ with $y_i \in \{0, 1\}$ and

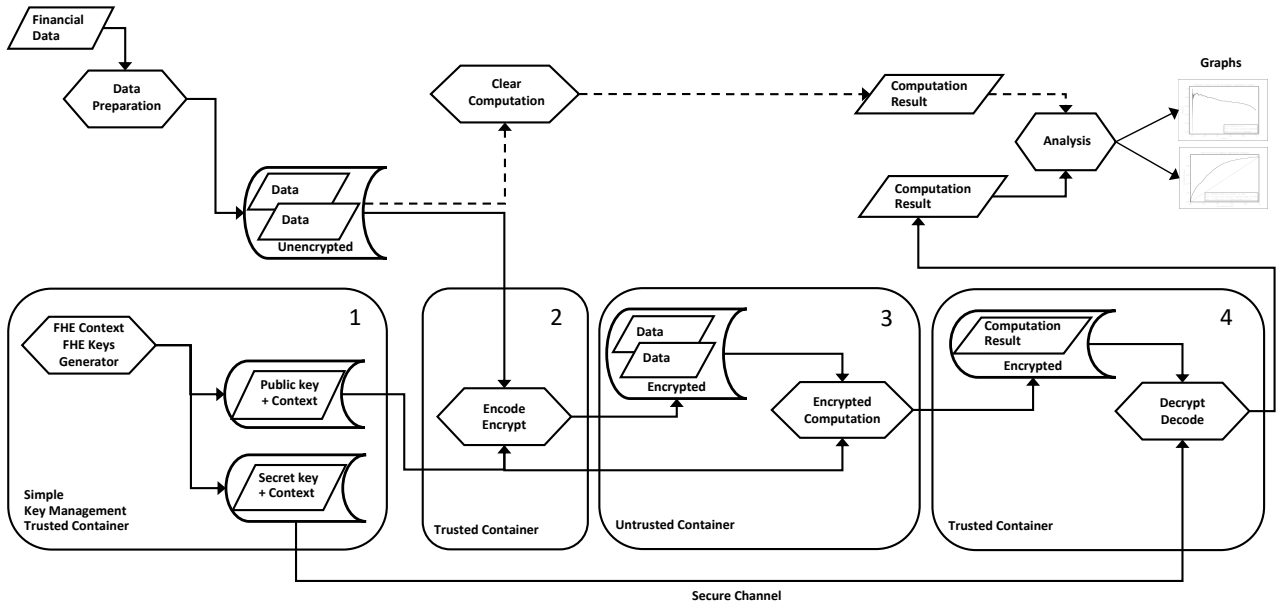

Figure 1: Homomorphic and plaintext pipelines.

$\vec{x}_i \in \mathbb{R}^d$. The aim is to predict the value of $y \in \{0,1\}$ given the attributes $\vec{x}$, and the logistic regression technique postulates that the distribution of y given $\vec{x}$ is given by

$$\Pr[y=1|\vec{x}] = \frac{1}{1+\exp\left(-w_0 - \sum_{i=1}^{d} x_i w_i\right)} = \frac{1}{1+\exp\left(-\vec{x'}^T \vec{w}\right)},$$

where $\vec{w} \in \mathbb{R}^{d+1}$ is a fixed vector of weights and $\vec{x'_i} = (1|\vec{x_i}) \in \mathbb{R}^{d+1}$ is a feature vector. Given the training data $\{(y_i, \vec{x_i})\}_{i=1}^n$, we can therefore make predictions if we can find the vector $\vec{w}$ that best matches this data, where the notion of *best match* is typically maximum likelihood. Such a weight vector, $\vec{w}^*$, can be expressed explicitly as

$$\vec{w}^* = \arg\max_{\vec{w}} \left\{ \prod_{y_i=1} \frac{1}{1+\exp\left(-\vec{x'_i}^T \vec{w}\right)} \cdot \prod_{y_i=0} \frac{1}{1+\exp\left(\vec{x'_i}^T \vec{w}\right)} \right\}$$

where we use the probability postulate given above in conjunction with the following identity

$$1 - \frac{1}{1+\exp(-z)} = \frac{1}{1+\exp(z)}.$$

The formula for $\vec{w}^*$ can be written more compactly by setting $y'_i = 2y_i - 1 \in \{\pm 1\}$ and $\vec{z}_i = y'_i \cdot \vec{x'}_i$, then our goal is to compute or approximate

$$\begin{aligned} \vec{w}^* &= \arg\max_{\vec{w}} \left\{ \prod_{i=1}^{n} \frac{1}{1+\exp\left(-\vec{z_i}^T \vec{w}\right)} \right\} \\ &= \arg\min_{\vec{w}} \left\{ \sum_{i=1}^{n} \log\left(1+\exp\left(-\vec{z_i}^T \vec{w}\right)\right) \right\}. \end{aligned}$$

For a candidate weight vector $\vec{w}$, we denote the (normalized) *loss function* for the given training set by

$$J(\vec{w}) \stackrel{\text{def}}{=} \frac{1}{n} \cdot \sum_{i=1}^{n} \log\left(1+\exp\left(-\vec{z_i}^T \vec{w}\right)\right),$$

and our goal is to find $\vec{w}$ that minimizes that loss.

Nesterov's Accelerated Gradient Descent. We use Nesterov's accelerated gradient decent [22] which has been used successfully and applied previously in [2]. It is a variant of the iterative method used by Kim *et al.* in [19]. Let σ be the sigmoid function,

$$\sigma(x) \stackrel{\text{def}}{=} 1/(1+e^{-x}),$$

then the gradient of the loss function with respect to $\vec{w}$ can be expressed as

$$\begin{aligned} \nabla J(\vec{w}) &= -\frac{1}{n} \sum_{i=1}^{n} \frac{1}{1+\exp\left(\vec{z_i}^T \vec{w}\right)} \cdot \vec{z_i} \\ &= -\frac{1}{n} \sum_{i=1}^{n} \sigma\left(-\vec{z_i}^T \vec{w}\right) \cdot \vec{z_i}. \end{aligned}$$

Nesterov's method initializes two evolving vectors to the mean average of the input records. Then each iteration computes

$$\begin{aligned} \vec{w}^{(t+1)} &= \vec{v}^{(t)} - \alpha_t \cdot \nabla J\left(\vec{v}^{(t)}\right), \\ \vec{v}^{(t+1)} &= (1-\gamma_t) \cdot \vec{w}^{(t+1)} + \gamma_t \cdot \vec{w}^{(t)}, \end{aligned}$$

where α_t, γ_t are scalar parameters that change from one iteration to the next. The α parameter is known as the learning rate and γ is called the moving average smoothing parameter. For how they are set, see section 3.4.

3 IMPLEMENTATION

In this section, we will discuss our methodology for performing homomorphic predictions and homomorphic variable selection. In the case of the logistic predictions, we provide a description of the method used to efficiently pack data into CKKS ciphertexts as well as requisite function approximations employed. In the case of the variable selection, we provide greater detail of the technique adopted for obtaining relevant scores for each variable as well as parameters and configuration of the Nesterov gradient descent algorithm. We present the modular ML pipeline of our experimentation.

3.1 Pipeline Overview

Figure 1 illustrates the basic model of the computation and flow of data of the implemented system, used for both prediction and variable selection. In this model, we have several parties with the trusted parties operating in trusted containers (labeled 1, 2, and 4) and the untrusted party operating in the untrusted container 3. Typically, this trust model would correspond to a client-server relationship in which the server is considered to be acting under the honest-but-curious attacker model.

The trusted container 1, hosted in a hardware security module, is responsible for key management and generation of the public-private key pairs, and the key switching matrices required for the computation, as described in [15]. For the sake of conciseness, we will henceforth refer to the public key, the context, and the key switching matrices collectively as simply *the public key*.

Trusted container 2 is responsible for encrypting the plain data with the public key. The encrypted data is made available to container 3, the honest-but-curious untrusted environment where the homomorphic computation can be performed. Both containers require and have access to the public-key.

Trusted container 4 is responsible for decrypting the final results using the secret key which is accessed through a secure channel.

Considering the flow of data through the system, firstly raw financial data is sanitized and pre-processed by the *Data Preparation* module, which then flows into trusted container 2. The data is then encoded according to the relevant *Data packing* method as described later on, which differs depending on whether prediction or variable selection is being performed. The encoded data is then encrypted using the public key and then sent to the untrusted container.

The untrusted container 3 performs whichever homomorphic computation is required by using the public key and encrypted data. In the prediction case, this will be an operation between encrypted data and an existing encrypted model as described in 3.3. In the variable reduction case, this will be a large number of logistic regression model trainings followed by a log loss computation as described in 3.4. In both cases, the encrypted output is passed to the trusted container 4 for decryption. Trusted container 4 will decrypt the result with the secret key and then process it directly or pass it elsewhere for usage.

In addition to this workflow, figure 1 also contains more steps which would not be used in a typical system, but that we employed for evaluation purposes. These can be seen in the cells which are connected with dotted lines. The *Clear Computation* block performs analogous computations to the *Encrypted Computation* block, except they are performed with standard methods entirely on the plaintext data. The results of the *Clear Computation* block and the HE pipeline are then compared using standard statistical techniques. It is from this final analysis step that the figures seen in section 4.5 are derived.

3.2 Function Approximations

Our homomorphic computations necessitate the evaluation of several higher-order functions such as sigmoid and logarithm. Despite the fact that addition and multiplication are the only operations native to the CKKS scheme employed, we are able to use polynomial approximations of arbitrary continuous functions as guaranteed by the Weierstrass approximation theorem. It was important to strike a balance between degree of polynomial approximation with higher degrees increasing the depth of the calculation and accuracy of approximation which is harmed by lower-degree approximations. Due to the significant disadvantages inherent to high-degree polynomials, in terms of both computation time and noise growth, we use the lowest-degree approximations possible which still yield good results.

Sigmoid approximation. For sigmoid function approximation, we use the same low-degree polynomial function in a bounded symmetrical range around zero as in [2, 19], namely with degree-3 and degree-7 approximation polynomials in the interval $[-8, 8]$

$$SIG3(x) \stackrel{\text{def}}{=} 0.5 - 1.2\left(\frac{x}{8}\right) + 0.81562\left(\frac{x}{8}\right)^3 \text{ and} \quad (1)$$

$$SIG7(x) \stackrel{\text{def}}{=} 0.5 - 1.734\left(\frac{x}{8}\right) + 4.19407\left(\frac{x}{8}\right)^3 - 5.43402\left(\frac{x}{8}\right)^5 + 2.50739\left(\frac{x}{8}\right)^7 \quad (2)$$

Logarithm approximation. We apply the same technique to derive a quartic polynomial approximation function for the composition $\log \circ \sigma$ directly rather than composing approximations for both logarithm and sigmoid, since this allows us to perform the required computation with minimal computational depth. We again use an approximation minimizing mean squared difference in $[-8, 8]$:

$$LOGSIG4(x) \stackrel{\text{def}}{=} 0.000527x^4 - 0.0822x^2 + 0.5x - 0.78 \quad (3)$$

3.3 HE logistic regression predictions

In this section, we describe a general implementation to perform logistic regression predictions. This is achieved by encoding and encrypting both model and data. More precisely, taking data that was segregated for testing from a *real* financial dataset, the data was encoded and encrypted then passed to the predictor. The predictor loads the required model and performs the prediction algorithm. Essentially a inner product that is the input to a sigmoid function.

Data packing. To perform homomorphic logistic regression predictions, we require an encrypted model and encrypted data. The model consists of a vector of *weights* $\beta \in \mathbb{R}^{17}$, where the 0^{th} entry of β is the bias term. In order to fit best with our homomorphic implementation, we simply replicate each entry β_i, $0 \leq i \leq 16$, into its own ciphertext. That is to say, we let $\mathfrak{m}_i$ be an encryption of the vector $u_i \in \mathbb{C}^l$ where each entry of u_i is equal to β_i. For packing of the data, we describe first the case where we have l predictions to perform, i.e. a set D of data where $|D| = l$. We pack all l vectors $\{\vec{x}_i\}_{i=1}^{l} \in \mathbb{R}^{16}$ into 16 ciphertexts by mapping the first entry of

each $\vec{x}$ into one ciphertext, the second entry of every $\vec{x}$ into another ciphertext, and so on.

Prediction. If we denote the resulting ciphertexts $\{\mathfrak{c}_i\}_{i=1}^{16}$, we can perform l predictions by computing

$$\sigma\left(\mathfrak{m}_0 + \sum_{i=1}^{16} \mathfrak{c}_i \odot \mathfrak{m}_i\right)$$

where $\odot$ is the entrywise product and σ is the sigmoid function (also computed entrywise in this case). This amounts to an inner product operation on a vector of ciphertexts. See algorithm 1 for pseudocode for how this can be implemented in practice.

The resulting predictions will be one ciphertext which decrypts to a vector in $\mathbb{C}^l$ corresponding to l predictions. In order to perform $n > l$ predictions, we simply partition the n data vectors into $\lceil \frac{n}{l} \rceil$ *blocks*, perform the prediction on each block, then concatenate the $\lceil \frac{n}{l} \rceil$ vectors of size l at the end. This can be performed completely in parallel for a large n.

The inner product computation can be performed natively due to our ability to perform additions and multiplications. However, subsequent to the inner product computation a sigmoid approximation is applied to its result. As previously mentioned, the sigmoid function is approximated with a degree-3 polynomial and evaluated on the output of the previous step depending on the level of accuracy desired.

Input: A vector $\vec{D}$ of ciphertexts of size d.
A vector $\vec{M}$ of ciphertexts of size $d + 1$
Output: A single ciphertext
1 **for** $i \leftarrow 0$ **to** d **do**
2 | $P[i] := D[i] \times M[i + 1]$
3 **end**
4 $V := M[0]$ // Add the bias term
5 **for** $j \leftarrow 0$ **to** d **do**
6 | $V := V + P[j]$ // Aggregate the vector of products
7 **end**
8 output SIG(V)

Algorithm 1: Homomorphic prediction

Note that line 8 of algorithm 1 returns the sigmoid approximation of V of degree-3 as described previously in equation 1.

3.4 Homomorphic variable selection

Our variable selection method is to train a single-variable model for each of the variables in the dataset then evaluate the quality of each model via a statistical score returning the scores to a client. These scores are then used to sort the variables resulting in an ordering which should roughly correspond to *importance* or *predictive capability*.

To perform this variable selection method homomorphically, we generate logistic regression models and corresponding log loss scores for each of the variables in our dataset individually. In the language of our logistic regression discussion in section 2.3, for each j with $1 \le j \le d$ we generate a data set consisting solely of projections onto the j^{th} variable. That is to say, we map each datum $(y, \vec{x})$ to (y, x_j), then perform the logistic regression algorithm including log loss calculation on the resulting data set for each j.

Data packing. A naïve implementation of this might result in a large, albeit parallelizable, computational requirement. However, we are able to take advantage of the *slotwise* vector operations that the CKKS scheme gives us, packing each variable into an entry of a $\mathbb{C}^l$-vector, as discussed in section 2.1. More explicitly, we perform the following transformation. For a dataset of size n, $(y_i, \vec{x}_i)_{i=1}^n$, with each $\vec{x}_i \in \mathbb{R}^d$, $d \le l$, and $y_i \in \{0, 1\}$, we create $2n$ vectors in $\mathbb{C}^l$ in the following way:

For each datum $(y, \vec{x})$, compute $y' = 2y - 1$ as before, then create $a \in \mathbb{C}^l$ by setting $a_i = y' \mathbb{1}_{i \le d}$, which is a repetition of y' in the first d entries, padded with zeroes to the end of the ciphertext. Next, generate the vector $b \in \mathbb{C}^l$ by setting $b_i = y' x_i \mathbb{1}_{i \le d}$, which is a zero-padded version of x with y' multiplied in. Now, we can use $(a, b) \in \left(\mathbb{C}^l\right)^2$ in the same way as the $\vec{z}$ vectors are used in section 2.3, thinking of d as 1 and exploiting the independence of entries of a CKKS ciphertext. We henceforth refer to such vectors (a, b) as $\vec{z}$ with the understanding that all operations between a and b are performed entrywise.

Initializing the algorithm. Since we need to use a small number of iterations, the initial values of $\vec{v}, \vec{w}$ are important to the convergence of the weights. We set them as the average of the inputs,

$$\vec{v}^{(0)} = \vec{w}^{(0)} = \frac{1}{n} \sum_{i=1}^{n} \vec{z}_i$$

as this yields better results than choosing them at random [2].

The number of iterations. The number of iterations, τ, that can be performed is very limited as we are using a somewhat-homomorphic encryption scheme to implement the procedure on encrypted data. For our implementation and tests, we used $\tau = 5$ and $\tau = 6$ iterations.

The α and γ parameters. The learning-rate parameter α was set just as in [19], namely in iteration $t = 1, \ldots, \tau$ we used $\alpha_t = 10/(t + 1)$.

For setting the moving average smoothing parameter γ at each iteration, we used negative values for gamma as suggested in [6]. Setting $\lambda_0 = 0$, we can compute for $t = 1, \ldots, \tau$

$$\lambda_t = \frac{1 + \sqrt{1 + 4\lambda_{t-1}^2}}{2} \text{ and } \gamma_t = \frac{1 - \lambda_t}{\lambda_{t+1}}.$$

The values of γ for the first 6 iterations are $\gamma \approx (0, -0.28, -0.43, -0.53, -0.6, -0.65)$

Log loss. Logarithmic loss is a statistical measure commonly used in ML for evaluating the quality of a classification model which outputs probabilities. A log loss closer to zero implies a model with greater predictive quality. This technique takes into account the level of certainty of the prediction and compares it to the true value. For example, a probability prediction close to one will be rewarded heavily if correct, but heavily penalized if incorrect.

In our logistic regression case with weights vector $\vec{w}$ and input data vectors $\vec{z}_i$, $1 \le i \le n$, the log loss function l is given by

$$l(\vec{w}) = -\frac{1}{n} \sum_{i=1}^{n} \log\left(\sigma\left(\vec{z}_i^T \vec{w}\right)\right)$$

In this work, we make extensive use of the log loss function for two reasons: as a cost function of $\vec{w}$ to minimize during our logistic regression model fitting; and as a score by which to order variables. In order to compute this homomorphically, we use the *LOGSIG*4 approximation (equation 3) described in section 3.2 to give our (unscaled) log loss approximation function. We omit the $\frac{1}{n}$ term for ease of calculation since we are only concerned with the ordering resulting from these values.

$$LOGLOSS(\vec{w}) \stackrel{\text{def}}{=} -\sum_{i=1}^{n} LOGSIG4\left(\vec{z}_i^T \vec{w}\right) \tag{4}$$

Note that we also make use of the exact version of the log loss function in section 4 for assessing the quality of models.

Decorrelation. In order to improve the quality of models obtained homomorphically or otherwise, we apply decorrelation to the variables which is a standard technique in data analytics to improve model stability and mitigate overfitting [18]. However, rather than blindly applying a decorrelation policy during the data preparation phase (i.e. on the unordered set of variables), we delay the decorrelation until after the variable ordering has been obtained. This post-processing phase is advantageous to the resulting model as we are able to preferentially drop variables which are considered by the ordering to have lower predictive capability.

The precise method that we use for removing correlated variables is as follows: given an ordering of variables $(V_1, \ldots, V_d)$ where d is the number of variables, we consider the matrix M defined by $M_{ij} = \left|\rho(V_i, V_j)\right|$, where $\rho(X, Y)$ is the Pearson's correlation coefficient between two variables X and Y. We drop the variable V_j if and only if there exists an entry in the j^{th} column of the upper triangle of M with value greater than or equal to 0.75, i.e. if and only if there exists $i \in \mathbb{N}$, $1 \leq i < j$ such that $M_{ij} \geq 0.75$.

Upon first glance this might appear to go against the spirit of a homomorphic variable selection pipeline since the ρ values require the original data to be computed, however this is not the case. Notice that for any variable ordering, the utility matrix M is formed simply by rearranging the values of any other such matrix of correlation values. Thus, pre-computing the $\frac{d(d-1)}{2}$ real numbers in the upper triangle of M before performing the variable ordering gives us enough information to perform our decorrelation procedure without needing access to the data again.

Note that performing decorrelation before the variable selection phase would not result in any performance optimization, since the way in which we pack data into $\mathbb{C}^l$ vectors means that we can treat up to l variables without any slowdown. As can be seen in table 1, we always have $l \gg d$.

4 EXPERIMENTAL EVALUATION

The results from executing our pipeline are presented in this section. We primarily evaluate and compare quality of predictions as well as quality of the variable selection process.

Firstly, we will discuss the various metrics and methods we chose to evaluate the quality of predictions as well as those used to evaluate our method of performing variable selection. Next, we describe the configuration of our pipelines including hardware specifications and HE scheme parameters. Finally, we discuss and analyze the results of the implemented methodology with comparisons to plaintext equivalents.

4.1 Metrics

In section 3.4, we introduced and selected log loss as the metric for the ordering as it is a relatively simple-to-compute measure that can be calculated homomorphically. To demonstrate its benefits as a good common metric, we compared log loss to two other common metrics used to evaluate machine learning models.

Area under curve. The *receiver operating characteristic* (ROC) curve is a standard tool in evaluating the performance of predictive models. The ROC space is typically defined as $[0, 1]^2$ where a point $(a, b) \in [0, 1]^2$ has the false positive rate a and the true positive rate b of a given set of binary predictions. For a set of probability predictions, it is typical to trace out the curve in the ROC space parameterized by a threshold value. Attributes of the ROC curve, including the area under the curve (AUC) are considered to be superior measures of the quality of a set of predictions compared to a single accuracy value [23, 26].

Average precision. *Precision-recall* (PR) curves are another tool similar in use to ROC curves, but are more frequently used in information retrieval or situations in which the two classes are imbalanced in the dataset. With PR curves, a similar parameterization on threshold value is performed, but the points in $[0, 1]^2$ are (precision, recall) pairs instead of (false positive rate, true positive rate) pairs. The method of taking the area under this curve as a metric, known as the *average precision* (AP), is also a common practice. PR and ROC curves have been shown to have strong links to each other for a given predictor [10] as well as a direct relationship shown by Su *et al.* [25] between the AP and AUC scores.

In this work, we experimented with using AUC, AP, and log loss for selecting models and evaluating quality of derived models. However, we do not compute AUC or AP homomorphically for the purpose of variable ordering as this would require the application of a large number of threshold function approximations; likely requiring an extremely high-depth computation in comparison to our fourth-order log loss approximation in equation (4).

4.2 Testing Environment

Our approach has been tested on a hardware and software environment commonly available in the finance industry data centers and/or cloud settings, capable of high volume shared and multi-tenant workloads. The hardware used for our tests support 64 simultaneous threads over 64 cores, 1 TB RAM, and 1.2 TB HDD, running Linux Ubuntu 18.04 LTS.

4.3 CKKS Parameters

Parameters for the algebra used for CKKS were chosen to give at least 128 bit security while having enough *qbits* to support our required computational depth. Unlike the BGV scheme, parameters for the CKKS plaintext space in HElib are easier to find because there is no plaintext prime to consider. Moreover, m as a power of two works better for the deep circuit of the variable selection because the ciphertext sizes are a power of two, thus making the inherent FFTs that must be performed by HElib more efficient. Although not

recorded in this work, we found that non-power-of-two algebras slowed the computation down considerably.

The parameters selected for the experiments, in particular the variable selection, differ from those used by Bergamaschi *et al.* [2] because the security estimation in the newer version of HElib (commit 67abcebf1f8c1bae9d51c9352e6fef7d5b8d71a3) is more conservative. The initial parameter value to select is the m^{th} cyclotomic polynomial to use as this is the main factor on the security level λ and on the number of slots in each ciphertext. As mentioned previously, it is easier to select the value for this parameter in CKKS as the lack of a plaintext prime means that the number of slots will always be $l = \phi(m)/2$ as seen in table 1.

The value of the precision parameter r was set to 50 so as to ensure the highest level of precision with the aim to generate a model of a greater predictive quality. We conducted some preliminary investigations to determine a high value of r that we could use and not lead to decryption issues.

The next parameter to consider is *qbits*, the bitsize of the modulus of a freshly encrypted ciphertext. Since we are using a somewhat HE scheme this needs to be larger for evaluation of deeper circuits such as the variable selection, not so for homomorphic prediction. As seen in table 1, the number of bits used for prediction is 360 yet for the deeper circuit of variable selection we must select *qbits* to be over 2000. As operations are performed upon a ciphertext, the noise increases and this consumes the bits of modulus chain. It is important to ensure there are enough bits left of the modulus chain to allow for decryption of the result without any wraparound occurring.

The final parameter shown in table 1 is c. This parameter determines the number of columns of our key switching matrices. The key switching is used to relinearize the ciphertext after each multiplication operation. This was selected to be 2 so as to minimize the size of the key switching matrices which reduces the size of the files being sent across the pipeline as well as reducing the computation time of the relinearization process.

4.4 Dataset preparation

Table 1 specifies the parameters selected for the homomorphic prediction and homomorphic variable selection experiments. The raw datasets used in the experiments represent *real* financial transactions over a 24-month period comprising a table of 360000 entries with 564 features. Although a large data set, the data is very sparse and the condition to be modeled is a rare event in the dataset (only $\sim$ 1%) where it would lead to a biased model that would underestimate the condition and overestimate the non-condition [21]. During data preparation the input data was diligently sanitized for missing values, categorical variable processing performed, and the data balanced; resulting in a balanced set with approximately 7500 entries with 546 explanatory features. The plaintext reference model for the prediction experiment was generated using the Python scikit-learn library.

4.5 Results and discussion

We now present the results of the pipeline described in section 3 applied to both homomorphic prediction and homomorphic variable reduction. These experiments were performed using the parameters given in table 1.

Table 1: CKKS parameters used for homomorphic prediction and homomorphic variable selection.

	Prediction		Variable Selection	
	128 bit security	256 bit security	sig3 5 steps	sig3 6 steps
m	21491	33689	262144(=2^{18})	262144(=2^{18})
r	50	50	50	50
$qbits$	360	360	2000	2400
c	2	2	2	2
$\phi(m)$	21490	33060	131072(=2^{17})	131072(=2^{17})
l	10745	16530	65536(=2^{16})	65536(=2^{16})
λ	128	256	193	140

Table 2: CPU time and RAM usage of prediction.

	Prediction Security	
	128 bit	256 bit
# Predictions per thread	10745	16530
# Threads	1	1
Encrypted Model size	40 MB	61 MB
Model input time	1 sec	1.3 sec
Encrypted Data size	37 MB	57 MB
Data input time	0.8 sec	1.2 sec
Prediction time	5.4 sec	9.4 sec

Homomorphic predictions. We evaluated the pipeline for homomorphic predictions with several configurations. In terms of CKKS parameters, we performed predictions with parameters which result in 128 and 256 bits of security. The prediction computation consisted of an inner product followed by application of an approximated sigmoid function. In order to approximate the sigmoid function while still minimizing the computation depth of performing predictions. Then we experimented with our degree-3 sigmoid approximation, *SIG*3. The results of these prediction operations were then analyzed by means of comparison with predictions run entirely in plaintext against the same model.

Figure 2 depicts the comparison between the predictions performed in plaintext and homomorphically. This is done by means of a ROC curve using a size 2271 sample with known condition to test against. Both ROC curves are practically indistinguishable, demonstrating that any inaccuracies resulting from performing the predictions homomorphically do not significantly impact the quality of the predictions. Table 2 shows performance information including memory usage for the aforementioned prediction pipeline. Due to the low depth of computation required for performing a logistic regression prediction operation, our solution achieves acceptable performance even in the case of 256 bit security. Based on these results, selecting *SIG*3 provides the solution with the adequate balance of accuracy and performance.

Homomorphic variable selection. We performed extensive experimentation in order to determine the quality of the homomorphic log loss calculations (section 3.4) compared to a fully-plaintext

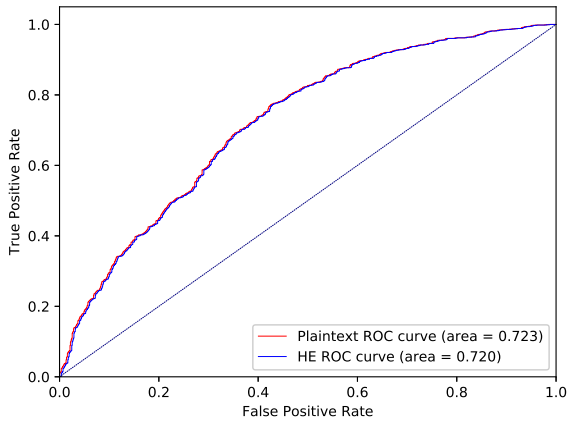

Figure 2: ROC curve for plaintext and homomorphic prediction.

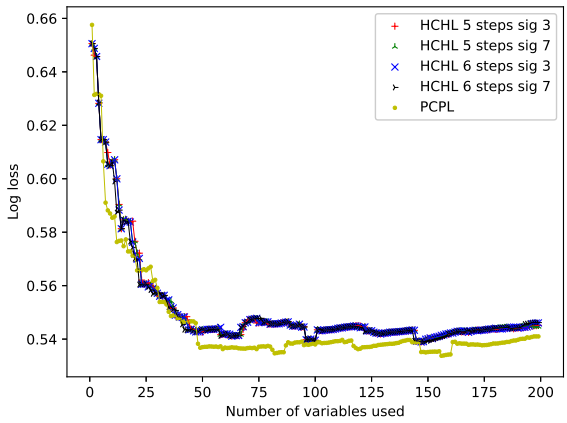

Figure 3: Log loss for several homomorphic parameters.

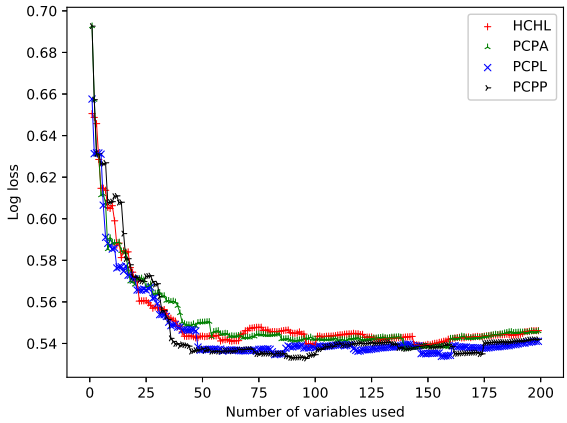

Figure 4: Log loss for HE and plain.

pipeline which performs similar calculations. In this experimentation, we consider not the log loss values themselves, but the quality of relative ordering which results from sorting based on these values. Once an adequate set of parameters were derived for homomorphic log loss calculations, we compared the results with various different plaintext-based orderings, namely ordering by AUC and by AP.

Our method for evaluating the quality of the selected ordering was as follows. We took the first k variables ordered by score, then used only these k variables to create a penalized logistic regression model. We then evaluated the quality of the resultant model using 10-fold cross-validation to derive a value for the typical scores: AUC, AP, and log loss. This procedure was carried out for each k between 1 and 200. The results were then plotted on a scatter to evaluate any trends of differing performance.

The convention used for the curves with four-letter labels (e.g. HCHL) in the graphs below is the following. The first two letters indicate how the variable selection was computed; either HC or PC for homomorphically computed or plaintext computed, respectively. The third letter H or P indicate how ordering score was calculated, namely, homomorphically or in the plain. The fourth and last letter indicates which metric was used for ordering (computing a score). The last letter can take L, A or P for ordering by log loss, AUC, or AP, respectively. Thus in combination, the last two letters should be read as how the variable ordering was performed, e.g. HL for homomorphic log loss or PL for plaintext log loss.

The first step in our assessment was to compare how homomorphic variable selection by logistic regression ordered by homomorphic log loss (HCHL) really compares with the plaintext version of variable selection by logistic regression ordered by plaintext-computed log loss (PCPL). This comparison is illustrated in figure 3. Furthermore, we compared variations of the different HCHL configurations, as described in section 4.3. The figure shows the comparison between different numbers of Nesterov steps and different degrees of sigmoid approximations, alongside PCPL as a baseline for comparison. It is clearly shown that ordering by log loss homomorphically is comparable to computing it in the plain. We can also see that the HCHL configurations have negligible difference. This is significant because of the consequences of requiring a higher depth of computation; namely its considerable effect on computation time and the adverse impact that the requisite increase in *qbits* has on security. Nonetheless, for all remaining evaluations, we used the degree-7 sigmoid and 6 Nesterov steps.

Contemporary metrics commonly used for evaluation are AUC and AP. As discussed in section 4.1, these are considered computationally heavy to implement homomorphically. However, we compare the performance of ordering with these metrics in plaintext only. This comparison is performed by measuring against all three of the aforementioned evaluation scoring methods. Evaluation by log loss can be seen in figure 4, AUC in figure 5, and AP in figure 6. All three of these figures support the same conclusion: there is not significant difference between the different methods of ordering,

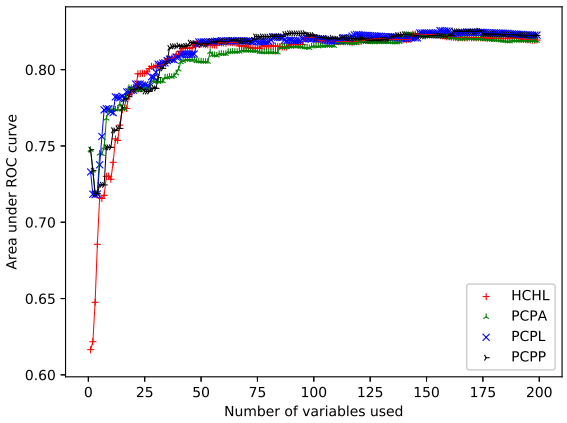

Figure 5: Evaluation by AUC.

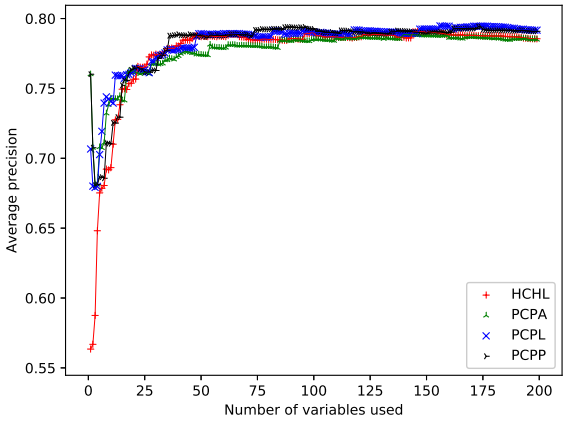

Figure 6: Evaluation by average precision.

including the homomorphic methodology. One can read from any of the figures that by around the time the 50 best-scoring variables have been included the model quality stabilizes at around the same level.

Table 3 depicts the performance of the homomorphic variable selection with log loss ordering comparing 5 and 6 Nesterov steps with degree-3 sigmoid approximation. These were run with the algebras given in table 1. The 6 step version requires deeper computation, thus requiring an algebra with a larger value for *qbits*. Consequently the ciphertexts are larger resulting in higher memory usage than the 5 steps version. In both cases, increasing the number of threads decreases the running time. However, in the shared and multi-tenant environment we observed that using more than 48 threads for computation did not further decrease the running time of the training phase, which is a deep computation. This behavior is likely to be caused by memory locality issues resulting from the large ciphertexts required. Since there is negligible difference in the

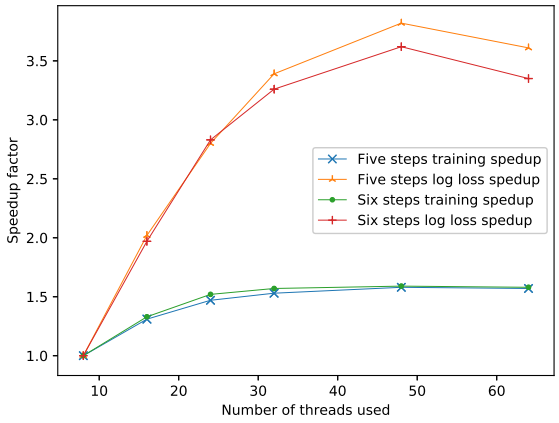

Figure 7: Computation speed-up of training and log loss versus the number of threads.

quality of the results for 5 and 6 Nesterov steps as seen in figure 3, we choose 5 steps as a good compromise between memory usage, performance, and quality of the results.

5 CONCLUSION

To progress towards a *real-world* ML pipeline, we investigated two common pipeline tasks. These tasks need to be further considered when assessing if HE can be utilized to address whether data can be aggregated. We have demonstrated that predictions can be performed in a typical business setting with a powerful architecture in a reasonable amount of time for realistic workloads using *real* financial data.

Prediction on the encrypted reference model took less than 10 seconds with a security level of 256 bits. It was shown that over 16500 predictions can be performed in this time. Variable selection, while preserving the privacy and confidentiality of the input data, took 1 hour and 43 minutes to perform for a security level above 128 bits, which is adequate considering that most training tasks run as batch processes. To achieve these levels of security, we used algebras not previously used in related work [2] with $m = 2^{18}$ allowing for variable selection to be performed for the depth required. The CKKS scheme has demonstrated to be invaluable to achieving good accuracy despite its approximate nature, and with HElib it is now possible to have high accuracy by having the r parameter set as high as 50.

Moreover, we have shown through comparison that log loss is an adequate metric for ordering during the homomorphic variable selection. The experimentation demonstrated comparable results to ordering by common ML metrics such as AUC or AP. This is a good result as log loss is considered to be of low depth computationally as opposed to homomorphically calculating the other metrics.

6 FURTHER WORK

Due to time constraints, we were not able to explore performing the decorrelation homomorphically. This would be of interest and the

Table 3: CPU time and RAM usage of the degree-3 sigmoid with 5-6 Nesterov iterations vs. number of threads.

# Nesterov iterations	Data input ciphertext	# Threads	Data input time	Data input speedup	Training time	Training speedup	LogLoss time	LogLoss speedup	RAM usage
5	64 GB	64	30 sec	6.00 ×	6062 sec	1.57 ×	217 sec	3.61 ×	228 GB
		48	37 sec	4.86 ×	6000 sec	1.58 ×	205 sec	3.82 ×	220 GB
		32	47 sec	3.83 ×	6186 sec	1.53 ×	231 sec	3.39 ×	217 GB
		24	62 sec	2.90 ×	6467 sec	1.47 ×	280 sec	2.80 ×	210 GB
		16	92 sec	1.96 ×	7255 sec	1.31 ×	388 sec	2.02 ×	206 GB
		8	180 sec	1.00 ×	9491 sec	1.00 ×	784 sec	1.00 ×	200 GB
6	80 GB	64	32 sec	6.81 ×	9584 sec	1.58 ×	295 sec	3.35 ×	284 GB
		48	58 sec	3.76 ×	9481 sec	1.59 ×	273 sec	3.62 ×	271 GB
		32	58 sec	3.76 ×	9658 sec	1.57 ×	303 sec	3.26 ×	260 GB
		24	75 sec	2.91 ×	9920 sec	1.52 ×	349 sec	2.83 ×	257 GB
		16	113 sec	1.93 ×	11349 sec	1.33 ×	502 sec	1.97 ×	252 GB
		8	218 sec	1.00 ×	15119 sec	1.00 ×	987 sec	1.00 ×	243 GB

Note: The timings in this table are for reference only as the HE code implementation was focused on achieving numerical fidelity and adequate security.

next logical step to attempt to tie together a more complete machine learning pipeline. This might involve homomorphic calculations of correlation coefficients such as the Pearson correlation coefficient used in this work, then elimination of variables with a sufficiently high correlation. At the time of writing, the authors are unaware of any works which attempt to achieve this and any such scheme would certainly push the depth of computation beyond what this work performed.

Other future works may include attempting to calculate other model scores such as AUC or AP in a novel homomorphic way, the latter of which might be of particular interest for heavily imbalanced datasets. However, the homomorphic application of various threshold values may prove problematic and high-depth in the absence of any innovative scheme for efficiently doing so.

It is reasonable to expect that more complete ML pipelines would require higher depth of computation thus necessitating the requirement for bootstrapping. This would need to be taken into consideration in implementation.

REFERENCES

[1] David Archer, Lily Chen, Jung Hee Cheon, Ran Gilad-Bachrach, Roger A. Hallman, Zhicong Huang, Xiaoqian Jiang, Ranjit Kumaresan, Bradley A. Malin, Heidi Sofia, Yongsoo Song, and Shuang Wang. 2017. *Applications of Homomorphic Encryption.* Technical Report. HomomorphicEncryption.org, Redmond WA, USA.

[2] Flavio Bergamaschi, Shai Halevi, Tzipora T. Halevi, and Hamish Hunt. 2019. Homomorphic Training of 30,000 Logistic Regression Models. In *Applied Cryptography and Network Security*, Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung (Eds.). Springer International Publishing, Cham, 592–611.

[3] Charlotte Bonte and Frederik Vercauteren. 2018. Privacy-preserving logistic regression training. *BMC Medical Genomics* 11, (Suppl 4) (2018). https://doi.org/10.1186/s12920-018-0398-y.

[4] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Transactions on Computation Theory* 6, 3 (2014), 13. https://doi.org/10.1145/2633600

[5] Alon Brutzkus, Oren Elisha, and Ran Gilad-Bachrach. 2019. Low Latency Privacy Preserving Inference. In *International Conference on Machine Learning*, Proceedings of Machine Learning Research (Ed.). Long Beach, CA, USA.

[6] Sebastien Bubeck. 2013. ORF523: Nesterov's Accelerated Gradient Descent. https://blogs.princeton.edu/imabandit/2013/04/01/acceleratedgradientdescent, accessed January 2019.

[7] Hao Chen, Ran Gilad-Bachrach, Kyoohyung Han, Zhicong Huang, Amir Jalali, Kim Laine, and Kristin Lauter. 2018. Logistic regression over encrypted data from fully homomorphic encryption. *BMC Medical Genomics* 11, (Suppl 4) (2018). https://doi.org/10.1186/s12920-018-0397-z.

[8] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *ASIACRYPT (1) (Lecture Notes in Computer Science)*, Vol. 10624. Springer, 409–437.

[9] Jack L. H. Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup. 2018. Doing Real Work with FHE: The Case of Logistic Regression. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2018*, Michael Brenner and Kurt Rohloff (Eds.). ACM, 1–12. https://doi.org/10.1145/3267973.3267974 https://eprint.iacr.org/2018/202.

[10] Jesse Davis and Mark Goadrich. 2006. The Relationship Between Precision-Recall and ROC Curves. In *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*. ACM, New York, NY, USA, 233–240. https://doi.org/10.1145/1143844.1143874

[11] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive* 2012 (2012), 144.

[12] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning*. Vol. 1. Springer series in statistics New York.

[13] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st ACM Symposium on Theory of Computing – STOC 2009*. ACM, 169–178.

[14] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *ICML (JMLR Workshop and Conference Proceedings)*, Vol. 48. JMLR.org, 201–210.

[15] Shai Halevi and Victor Shoup. Accessed August 2019. HElib - An Implementation of homomorphic encryption. https://github.com/homenc/HElib.

[16] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. 2018. Efficient Logistic Regression on Large Encrypted Data. Cryptology ePrint Archive, Report 2018/662. https://eprint.iacr.org/2018/662.

[17] iDASH. [n.d.]. Integrating Data for Analysis, Anonymization and SHaring (iDASH). http://www.humangenomeprivacy.org.

[18] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An introduction to statistical learning*. Vol. 112. Springer.

[19] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. 2018. Logistic regression model training based on the approximate homomorphic encryption. *BMC Medical Genomics* 11, 4 (11 Oct 2018), 83. https://doi.org/10.1186/s12920-018-0401-7

[20] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang. 2018. Secure Logistic Regression based on Homomorphic Encryption: Design and Evaluation. *JMIR Med. Inform.* 6(2) (2018), e19. DOI: 10.2196/medinform.8805. Also available from https://eprint.iacr.org/2018/074.

[21] Gary King and Langche Zeng. 2001. Logistic Regression in Rare Events Data. *Political Analysis* 9 (April 2001), 137–163.

[22] Yurii Nesterov. 2004. *Introductory Lectures on Convex Optimization: A Basic Course*. Applied Optimization, Vol. 87. Springer US. https://doi.org/10.1007/978-1-4419-8853-9.

[23] Foster J. Provost, Tom Fawcett, and Ron Kohavi. 1998. The Case Against Accuracy Estimation for Comparing Induction Algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 445–453. http://dl.acm.org/citation.cfm?id=645527.657469

[24] A L Samuel. 1959. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development* 3, 3 (July 1959), 210–229.

[25] Wanhua Su, Yan Yuan, and Mu Zhu. 2015. A Relationship Between the Average Precision and the Area Under the ROC Curve. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval (ICTIR '15)*. ACM, New York, NY, USA, 349–352. https://doi.org/10.1145/2808194.2809481

[26] M H Zweig and G Campbell. 1993. Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine. *Clinical Chemistry* 39, 4 (1993), 561–577. arXiv:http://clinchem.aaccjnls.org/content/39/4/561.full.pdf http://clinchem.aaccjnls.org/content/39/4/561