# Privacy-Preserving Logistic Regression Training

Charlotte Bonte[1], Frederik Vercauteren[1]

imec-Cosic, Dept. Electrical Engineering, KU Leuven

**Abstract.** Logistic regression is a popular technique used in machine learning to construct classification models. Since the construction of such models is based on computing with large datasets, it is an appealing idea to outsource this computation to a cloud service. The privacy-sensitive nature of the input data requires appropriate privacy preserving measures before outsourcing it. Homomorphic encryption enables one to compute on encrypted data directly, without decryption, and can be used to mitigate the privacy concerns raised by using a cloud service. In this paper, we propose an algorithm (and its implementation) to train a logistic regression model on a homomorphically encrypted dataset. The core of our algorithm consists of a new iterative method that can be seen as a simplified form of the fixed Hessian method, but with a much lower multiplicative complexity. We test the new method on two interesting real life applications: the first application is in medicine and constructs a model to predict the probability for a patient to have cancer, given genomic data as input; the second application is in finance and the model predicts the probability of a credit card transaction to be fraudulent. The method produces accurate results for both applications, comparable to running standard algorithms on plaintext data. This article introduces a new simple iterative algorithm to train a logistic regression model that is tailored to be applied on a homomorphically encrypted dataset. This algorithm can be used as a privacy-preserving technique to build a binary classification model and can be applied in a wide range of problems that can be modelled with logistic regression. Our implementation results show that our method can handle the large datasets used in logistic regression training.

## Background

### Introduction

Logistic regression is a popular technique used in machine learning to solve binary classification problems. It starts with a training phase during which one computes a model for prediction based on previously gathered values for predictor variables (called covariates) and corresponding outcomes. The training phase is followed by a testing phase that assesses the accuracy of the model. To this end, the dataset is split into data for training and data for validation. This

This work was supported by the European Commission under the ICT programme with contract H2020-ICT-2014-1 644209 HEAT.

validation is done by evaluating the model in the given covariates and comparing the output with the known outcome. When the classification of the model equals the outcome for most of the test data, the model is considered to be valuable and it can be used to predict the probability of an outcome by simply evaluating the model for new measurements of the covariates.

Logistic regression is popular because it provides a simple and powerful method to solve a wide range of problems. In medicine, logistic regression is used to predict the risk of developing a certain disease based on observed characteristics of the patient. In politics, it is used to predict the voting behaviour of a person based on personal data such as age, income, sex, state of residence, previous votes. In finance, logistic regression is used to predict the likelihood of a homeowner defaulting on a mortgage or a credit card transaction being fraudulent.

As all machine learning tools, logistic regression needs sufficient training data to construct a useful model. As the above examples show, the values for the covariates and the corresponding outcomes are typically highly sensitive, which implies that the owners of this data (either people or companies) are reluctant to have their data included in the training set. In this paper, we solve this problem by describing a method for privacy preserving logistic regression *training* using somewhat homomorphic encryption. Homomorphic encryption enables computations on encrypted data without needing to decrypt the data first. As such, our method can be used to send encrypted data to a central server, which will then perform logistic regression training on this encrypted input data. This also allows to combine data from different data owners since the server will learn nothing about the underlying data.

### Related work

Private logistic regression with the aid of homomorphic encryption has already been considered in [NLV11,BLN14], but in a rather limited form: both papers assume that the logistic model has already been trained and is publicly available. This publicly known model is then evaluated on homomorphically encrypted data in order to perform classification of this data without compromising the privacy of the patients. Our work complements these works by executing the training phase for the logistic regression model in a privacy-preserving manner. This is a much more challenging problem than the classification of new data, since this requires the application of an iterative method and a solution for the nonlinearity in the minimization function.

Aono et al. [AHTPW16] also explored secure logistic regression via homomorphic encryption. However, they shift the computations that are challenging to perform homomorphically to trusted data sources and a trusted client. Consequently, their solution exists of letting the data sources compute some intermediate values, which they subsequently encrypt and send to the computation server. This allows them to only use an additively homomorphic encryption scheme to perform the second, easier, part of the training process. Finally, they require a trusted client to perform a decryption of the computed coefficients and use these coefficients to construct the cost function for which the trusted client needs to determine the minimum in plaintext space. Their technique is based on a polynomial approximation of the logarithmic function in the cost function and the trusted client applies the gradient descent algorithm as iterative method to perform the minimization of the cost function resulting from the homomorphic computations. Our method does not require the data owners to perform any computations (bar the encryption of their data) and determines the model parameters by executing the minimization directly on encrypted data. Again this is a much more challenging problem.

In [XWBB16] Xie et al. construct PrivLogit which performs logistic regression in a privacy-preserving but distributed manner. As before, they require the data owners to perform computations on their data before encryption to compute parts of a matrix used in the logistic regression. Our solution starts from the encrypted raw dataset, not from values that were precomputed by the centers that collect the data. In our solution all computations that are needed to create the model parameters, are performed homomorphically.

Independently and in parallel with our research, Kim et al. [KSW$^+$18] investigated the same problem of performing the training phase of logistic regression in the encrypted domain. Their method uses a different approach than ours: firstly, they use a different minimization method (gradient descent) compared to ours (a simplification of the fixed Hessian method), a different approximation of the sigmoid function and a different homomorphic encryption scheme. Their solution is based on a small adaptation of the input values, which reduces the amount of homomorphic multiplications needed in the computation of the model. We assumed the dataset would be already encrypted and therefore adaptations to the input would be impossible. Furthermore, they tested their method on datasets that contain a smaller number of covariates than the datasets used in this article.

### Contributions

Our contributions in this paper are as follows: firstly, we develop a method for privacy preserving logistic training using homomorphic encryption that consists of a low depth version of the fixed Hessian method. We show that consecutive simplifications result in a practical algorithm, called the simplified fixed Hessian (SFH) method, that at the same time is still accurate enough to be useful. We implemented this algorithm and tested its performance and accuracy on two real life use cases: a medical application predicting the probability of having cancer given some genomic data and a financial application predicting the probability that a transaction is fraudulent. Our test results show that in both use cases the model computed is almost as accurate as the model computed by standard logistic regression tools such as the ones present in Matlab.

## Technical Background

### Logistic regression

Logistic regression can be used to predict the probability that a dependent variable belongs to a class, e.g. healthy or sick, given a set of covariates, e.g. some genomic data. In this article, we will consider binary logistic regression, where the dependent variable can belong to only two possible classes, which are labelled $\{\pm 1\}$. Binary logistic regression is often used for binary classification by setting a threshold for a given class up front, and comparing the output of the regression with this threshold. The logistic regression model is given by:

$$\Pr(y = \pm 1 | \mathbf{x}, \boldsymbol{\beta}) = \sigma(y \boldsymbol{\beta}^T \mathbf{x}) = \frac{1}{1 + e^{(-y \boldsymbol{\beta}^T \mathbf{x})}} , \tag{1}$$

where the vector $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_d)$ are the model parameters, $y$ the class label (in our case $\{\pm 1\}$) and the vector $\mathbf{x} = (1, x_1, \ldots, x_d) \in \mathbb{R}^{d+1}$ the covariates.

Because logistic regression predicts probabilities rather than classes, we can generate the model using the log likelihood function. The training of the model starts with a training dataset $(\mathbf{X}, \mathbf{y}) = [(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_N}, y_N)]$, consisting of $N$ training vectors $\mathbf{x_i} = (1, x_{i,1}, \ldots, x_{i,d}) \in \mathbb{R}^{d+1}$ and a corresponding observed class $y_i \in \{-1, 1\}$. The goal is to find the parameter vector $\boldsymbol{\beta}$ that maximizes the log likelihood function:

$$l(\boldsymbol{\beta}) = -\sum_{i=1}^{n} \log \left(1 + e^{(-y_i \boldsymbol{\beta}^T \mathbf{x_i})}\right) . \tag{2}$$

When the parameters $\boldsymbol{\beta}$ are determined, they can be used to classify new data vectors $\mathbf{x}^{\text{new}} = (1, x_1^{\text{new}}, \ldots, x_d^{\text{new}}) \in \mathbb{R}^{d+1}$ by setting

$$y^{\text{new}} = \begin{cases} 1 & \text{if } p(y = 1 | \mathbf{x}^{\text{new}}, \boldsymbol{\beta}) \geq \tau \\ -1 & \text{if } p(y = 1 | \mathbf{x}^{\text{new}}, \boldsymbol{\beta}) < \tau \end{cases}$$

in which $0 < \tau < 1$ is a variable threshold which typically equals $\frac{1}{2}$.

### Datasets

As mentioned before, we will test our method in the context of two real life use cases, one in genomics and the other in finance.

The genomic dataset was provided by the iDASH competition of 2017 and consists of 1581 records (each corresponding to a patient) consisting of 103 covariates and a class variable indicating whether or not the patient has cancer. The challenge was to devise a logistic regression model to predict the disease given a training data set of at least 200 records and 5 covariates. However, for scalability reasons the solution needed to be able to scale up to 1000 records with 100 covariates. This genomic dataset consists entirely of binary data.

The financial data was provided by an undisclosed bank that provided anonymized data with the goal of predicting fraudulent transactions. Relevant data fields that were selected are: type of transaction, effective amount of the transaction, currency, origin and destination, fees and interests, etc. This data has been subject to preprocessing to normalise the data. The resulting financial dataset consists of 20 000 records with 32 covariates, containing floating point values between 0 and 1.

### The FV scheme

Our solution is based on the somewhat homomorphic encryption scheme of Fan and Vercauteren [FV12], which can be used to compute a limited number of additions and multiplications on encrypted data. The security of this encryption scheme is based on the hardness of the ring learning with error problem (RLWE) introduced by Lyubashevsky et al. in [LPR13]. The core objects in the FV scheme are elements of the polynomial ring $R = \mathbb{Z}[X]/(f(X))$, where typically one chooses $f(X) = X^D + 1$ for $D = 2^n$ (in our case $D = 4096$). For an integer modulus $M \in \mathbb{Z}$ we denote with $R_M$ the quotient ring $R/(MR)$.

The plaintext space of the FV scheme is the ring $R_t$ for $t > 1$ a small integer modulus, and the ciphertext space is $R_q \times R_q$ for an integer modulus $q \gg t$. The FV scheme uses two probability distributions on $R_q$: one is denoted by $\chi_{\text{key}}$ and is used to sample the secret key of the scheme, the other is denoted $\chi_{\text{err}}$ and will be used to sample error polynomials during encryption. The exact security level of the FV scheme is based on these probability distributions, the degree $D$ and the ciphertext modulus $q$ and can be determined using an online tool developed by Albrecht et al. [Alb04].

Given parameters $D$, $q$, $t$ and the distributions $\chi_{\text{key}}$ and $\chi_{\text{err}}$, the core operations are then as follows:

- `KeyGen`: the private key consists of an element $s \leftarrow \chi_{\text{key}}$ and the public key $\mathbf{pk} = (b, a)$ is computed as $a \leftarrow R_q$ uniformly at random and $b = [-(as+e)]_q$ with $e \leftarrow \chi_{\text{err}}$.
- `Encrypt`($\mathbf{pk}$, $m$): given $m \in R_t$, sample error polynomials $e_1, e_2 \in \chi_{err}$ and $u \in \chi_{key}$ and compute $c_0 = \Delta \cdot m + bu + e_1$ and $c_1 = au + e_2$ with $\Delta = \lfloor q/t \rfloor$. The ciphertext is then $\mathbf{c} = (c_0, c_1)$.
- `Decrypt`(sk, $\mathbf{c}$): compute $\tilde{m} = [c_0 + s \cdot c_1]_q$, divide the coefficients of $\tilde{m}$ by $\Delta$ and round, and reduce the result into $R_t$.

Computing the sum of two ciphertexts simply amounts to adding the corresponding polynomials in the ciphertexts. Multiplication however requires a bit more work and we refer to [FV12] for the precise details.

The relation between a ciphertext and the underlying plaintext can be described as $[c_0 + c_1 s]_q = \Delta \cdot m + e$, where $e$ is the noise component present in the ciphertext. This also shows that if the noise $e$ grows too large, decryption will no longer result in the original message, and the scheme will no longer be correct. Since the noise present in the resulting ciphertext will grow with each operation we perform homomorphically, it is important to choose parameters that guarantee correctness of the scheme. Knowing the computations that need to be performed up front enables us to estimate the size of the noise in the resulting ciphertext, which permits the selection of suitable parameters.

### w-NIBNAF

In order to use the FV scheme, we need to transform the input data into polynomials of the plaintext space $R_t$. To achieve this, our solution makes use of the $w$-NIBNAF encoding, because this encoding improves the performance of the homomorphic scheme. The $w$-NIBNAF encoding is introduced in [BBB$^+$17] and expands a given number $\theta$ with respect to a non-integral base $1 < b_w < 2$. By replacing the base $b_w$ by the variable $X$, the method encodes any real number $\theta$ as a Laurent polynomial:

$$\theta = a_r X^r + a_{r-1} X^{r-1} + \cdots + a_1 X + a_0 - a_{-1} X^{d-1} - a_{-2} X^{d-2} - \cdots - a_{-s} X^{d-s}.$$

A final step then maps this Laurent polynomial into the plaintext space $R_t$ and we refer the reader to [BBB$^+$17] for the precise details.

The $w$-NIBNAF encoding is constructed such that the encoding of a number will satisfy two conditions: the encoding has coefficients in the set $\{-1, 0, 1\}$ and each set of $w$ consecutive coefficients will have no more than one non-zero coefficient. Both conditions ensure that the encoded numbers are represented by very sparse polynomials with coefficients in the set $\{-1, 0, 1\}$, which can be used to bound the size of the coefficients of the result of computations on these representations. In particular, this encoding results in a smaller plaintext modulus $t$, which improves the performance of the homomorphic encryption scheme. Since larger values for $w$ increase the sparseness of the encodings and hence reduce the size of $t$ even more, one would like to select the value for $w$ to be as large as possible. However, similar to encryption one has to consider a correctness requirement for the encoding. More specifically, decoding of the final polynomial should result in the correct answer, hence the base $b_w$ and consequently also the value of $w$ should be chosen with care.

## Privacy preserving training of the model

### Newton-Raphson method

To estimate the parameters of our logistic regression model, we need to compute the parameter vector $\boldsymbol{\beta}$ that maximizes Equation (2). Typically, one would differentiate the log likelihood equation with respect to the parameters, set the derivatives equal to zero and solve these equations to find the maximum. The gradient of the log likelihood function $l(\boldsymbol{\beta})$, i.e. the vector of its partial derivatives $[\partial l/\partial \beta_0,\ \partial l/\partial \beta_1,\ \ldots,\ \partial l/\partial \beta_d]$ is given by:

$$\nabla_{\boldsymbol{\beta}} l(\beta) = \sum_i (1 - \sigma(y_i \boldsymbol{\beta}^T \mathbf{x}_i)) y_i \mathbf{x}_i \,.$$

In order to estimate the parameters $\boldsymbol{\beta}$, this equation will be solved numerically by applying the Newton-Raphson method, which is a method to numerically determine the zeros of a function. The iterative formula of the Newton-Raphson method to calculate the root of a univariate function $f(x)$ is given by:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}\,, \tag{3}$$

with $f'(x)$ the derivative of $f(x)$. Since we now compute with a multivariate objective function $l(\boldsymbol{\beta})$, the $(k+1)^{\text{th}}$ iteration for the parameter vector $\boldsymbol{\beta}$ is given by:

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - H^{-1}(\boldsymbol{\beta}_k)\nabla_{\boldsymbol{\beta}} l(\boldsymbol{\beta}_k)\,, \tag{4}$$

with $\nabla_{\boldsymbol{\beta}} l(\boldsymbol{\beta})$ as defined above and $H(\boldsymbol{\beta})$ the Hessian of $l(\boldsymbol{\beta})$, being the matrix of its second partial derivatives $H_{i,j} = \partial^2 l/\partial\beta_i\partial\beta_j$, given by:

$$\nabla^2_{\boldsymbol{\beta}} l(\beta) = \sum_i (1 - \sigma(y_i \boldsymbol{\beta}^T \mathbf{x}_i))\sigma(y_i \boldsymbol{\beta}^T \mathbf{x}_i)(y_i \mathbf{x}_i)^2\,.$$

### Homomorphic logistic regression

The downside of Newton's method is that exact evaluation of the Hessian and its inverse are quite expensive in computational terms. In addition, the goal is to estimate the parameters of the logistic regression model in a privacy-preserving manner using homomorphic encryption, which will further increase the computational challenges. Therefore, we will adapt the method in order to make it possible to compute it efficiently in the encrypted domain.

The first step in the simplification process is to approximate the Hessian matrix with a fixed matrix instead of updating it every iteration. This technique is called the fixed Hessian Newton method. In [BL88], Böhning and Lindsay investigate the convergence of the Newton-Raphson method and show it converges if the Hessian $H(\boldsymbol{\beta})$ is replaced by a fixed symmetric negative definite matrix $B$ (independent of $\boldsymbol{\beta}$) such that $\nabla^2 l(\boldsymbol{\beta}) \geq B$ for all feasible parameter values $\boldsymbol{\beta}$, where " $\geq$ " denotes the Loewner ordering. The Loewner ordering is defined for symmetric matrices $A,\ B$ and denoted as $A \geq B$ iff their difference $A - B$ is non-negative definite. Given such $B$, the Newton-Raphson iteration simplifies to

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - B^{-1}\nabla_{\boldsymbol{\beta}} l(\boldsymbol{\beta}_k)\,.$$

Furthermore, they suggest a lower bound specifically for the Hessian of the logistic regression problem, which is defined as $\bar{H} = -\frac{1}{4}X^T X$ and demonstrate that this is a good bound. This approximation does not depend on $\boldsymbol{\beta}$, consequently it is fixed throughout all iterations and it only needs to be computed once as desired. Since the Hessian is fixed, so is its inverse, which means it only needs to be computed once.

In the second step, we will need to simplify this approximation even more, since inverting a square matrix whose dimensions equal the number of covariates (and thus can be quite large), is nearly impossible in the encrypted domain. To this end, we replace the matrix $\bar{H}$ by a diagonal matrix for which the method still converges. The entries of the diagonal matrix are simply the sums of the rows of the matrix $\bar{H}$, so our new approximation $\tilde{H}$ of the Hessian becomes:

$$\tilde{H} = \begin{bmatrix} \sum_{i=0}^{d} \bar{h}_{0,i} & 0 & \dots & 0 \\ 0 & \sum_{i=0}^{d} \bar{h}_{1,i} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sum_{i=0}^{d} \bar{h}_{d,i} \end{bmatrix} .$$

To be able to use this approximation as lower bound for the above fixed Hessian method we need to assure ourselves it satisfies the condition $\nabla^2 l(\boldsymbol{\beta}) \geq \tilde{H}$. As mentioned before we already know from [BL88] that $\nabla^2 l(\boldsymbol{\beta}) \geq \frac{-1}{4} X^T X$, so it is sufficient to show that $\frac{-1}{4} X^T X \geq \tilde{H}$, which we know prove more generally.

**Lemma 1** *Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix with all entries non-positive, and let $B$ be the diagonal matrix with diagonal entries $B_{k,k} = \sum_{i=1}^{n} A_{k,i}$ for $k = 1, \dots, n$, then $A \geq B$.*

*Proof.* By definition of the matrix $B$, we have that $C = A - B$ has the following entries: for $i \neq j$ we have $C_{i,j} = A_{i,j}$ and $C_{i,i} = -\sum_{k=1, k \neq i}^{n} A_{i,k}$. In particular, the diagonal elements of $C$ are minus the sum of the off-diagonal elements on the $i$-th row. We can bound the eigenvalues $\lambda_i$ of $C$ by Gerschgorin's circle theorem [Ger31], which states that for every eigenvalue $\lambda$ of $C$, there exists an index $i$ such that

$$|\lambda - C_{i,i}| \leq \sum_{j \neq i} |C_{ij}| \qquad i \in \{1, 2, \dots, n\} .$$

Note that by construction of $C$ we have that $C_{i,i} = \sum_{j \neq i} |C_{ij}|$, and so every eigenvalue $\lambda$ satisfies $|\lambda - C_{i,i}| < C_{i,i}$ for some $i$. In particular, since $C_{i,i} \geq 0$, we conclude that $\lambda \geq 0$ for all eigenvalues $\lambda$ and thus that $A \geq B$.

Our approximation $\tilde{H}$ for the Hessian also simplifies the computation of the inverse of the matrix, since we simply need to invert each diagonal element separately. The inverse will be again computed using the Newton-Raphson method: assume we want to invert the number $a$, then the function $f(x)$ will be equal to $\frac{1}{x} - a$ and the iteration is given by $x_{k+1} = x_k(2 - a x_k)$. For the Newton-Raphson method to converge, it is important to determine a good start value. This can be done by estimating the order of magnitude of the solution that is expected to be found by this algorithm. By doing this, we can already have an acceptable approximation of the inverse by performing only one iteration of the method. Based on the dimensions of the input data we estimate the size of the number we want to invert. We use this knowledge to set the initial value of our Newton-Raphson iteration.

In the third and final step, we simplify the non-linearity coming from the sigmoid function. Here we simply use the Taylor series: extensive experiments with plaintext data showed that approximating $\sigma(y_i\boldsymbol{\beta}^T\mathbf{x}_i)$ by $\frac{1}{2}+\frac{y_i\boldsymbol{\beta}^T\mathbf{x}_i}{4}$ is enough to obtain good results.

The combination of the above techniques finally results in our simplified fixed Hessian (SFH) method given in Algorithm 1.

**Algorithm 1** $\boldsymbol{\beta} \leftarrow$ simplified fixed Hessian$(X, Y, u_0, \kappa)$

```
**Input:** $X(N, d+1)$: training data with in each row the values for the covariates for one record and starting with a column of ones to account for the constant coefficient
$Y(N,1)$: labels of the training data
$u_0$: start value for the Newton-Raphson iteration that computes the inverse
$\kappa$: the required number of iterations
**Output:** $\boldsymbol{\beta}$: the parameters of the logistic regression model

$\boldsymbol{\beta} = 0.001 * \text{ones}(d+1, 1)$
$\text{sum} = \text{zeros}(N, 1);$
**for** $i = 1 : N$ **do**
    **for** $j = 1 : d+1$ **do**
        $\text{sum}(i) += X(i)(j)$
    **end for**
**end for**
**for** $j = 1 : d+1$ **do**
    $\text{temp}=0;$
    **for** $i = 1 : N$ **do**
        $\text{temp} += X(i)(j) \cdot \text{sum}(i);$
    **end for**
    $\tilde{H}(j) = -\frac{1}{4} \cdot \text{temp};$
    $\tilde{H}^{-1}(j) = 2 \cdot u_0 - \tilde{H}(j) \cdot u_0^2;$
**end for**
**for** $k = 1 : \kappa$ **do**
    **for** $i = 1 : N$ **do**
        $g = (\frac{1}{2} + \frac{1}{4} \cdot Y(i) \cdot X(i,:) \cdot \boldsymbol{\beta}) \cdot Y(i) \cdot X(i,:);$
    **end for**
    $\boldsymbol{\beta} = \boldsymbol{\beta} - \tilde{H}^{-1} \cdot g;$
**end for**
```

We implemented the SFH algorithm in Matlab and verified the accuracy for a growing number of iterations. One can see from Algorithm 1 that each iteration requires 5 homomorphic multiplications, so performing one iteration is quite expensive. In addition, Table 1 indicates that improving the accuracy significantly requires multiple iterations. We will therefore restrict our experiments to one single iteration.

Table 1: Performance for the financial dataset with 31 covariates and 700 records

| # iterations | AUC |
|---:|---|
| 1 | 0.9418 |
| 5 | 0.9436 |
| 10 | 0.9448 |
| 20 | 0.9466 |
| 50 | 0.9517 |
| 100 | 0.9599 |

## Results

### Accuracy of the SFH method

Table 2 shows the confusion matrix of a general binary classifier. From the confusion matrix we can compute the true positive rate (TPR) and the false positive rate (FPR) which are given by

$$\mathrm{TPR} = \frac{\#\mathrm{TP}}{\#\mathrm{TP} + \#\mathrm{FN}} \qquad \text{and} \qquad \mathrm{FPR} = \frac{\#\mathrm{FP}}{\#\mathrm{FP} + \#\mathrm{TN}} \,. \tag{5}$$

Table 2: Comparing actual and predicted classes

| | | **actual class** | |
|---|---|---|---|
| | | -1 | 1 |
| **predicted** | -1 | true negative (TN) | false negative (FN) |
| **class** | 1 | false positive (FP) | true positive (TP) |

By computing the TPR and FPR for varying thresholds $0 \leq \tau \leq 1$, we can construct the receiver operating characteristic curve or ROC-curve. The ROC-curve is constructed by plotting the $(\mathrm{FPR}, \mathrm{TPR})$ pairs for each possible value of the threshold $\tau$. The ideal situation would be there exists a point with $(\mathrm{FPR}, \mathrm{TPR}) = (0, 1)$, which would imply that there exists a threshold for which the model classifies all test data correctly.

The area under the ROC-curve or AUC-value will be used as the main indicator of how well the classifier works. Since our SFH method combines several approximations, we need to verify the accuracy of our model firstly on unencrypted data and finally on encrypted data. For well chosen system parameters there will be no difference between accuracy for unencrypted vs. encrypted data since all computations on encrypted data are exact.

The first step is performed by comparing our SFH method with the standard logistic regression functionality of Matlab. This is done by applying our method with all its approximations to the plaintext data and comparing the result to the result of the the “glmfit” function in Matlab. The function $b = \text{glmfit}(X, y, \text{distr})$ returns a vector $b$ of coefficient estimates for a generalized linear model of the responses $y$ on the predictors in $X$, using distribution distr. Generalized linear models unify various statistical models, such as for example linear regression, logistic regression and Poisson regression, by allowing the linear model to be related to the response variable via a link function. We use the “binomial” distribution, which corresponds to the “logit” link function and $y$ a binary vector indicating success or failure to compute the parameters of the logistic regression model with “glmfit”.

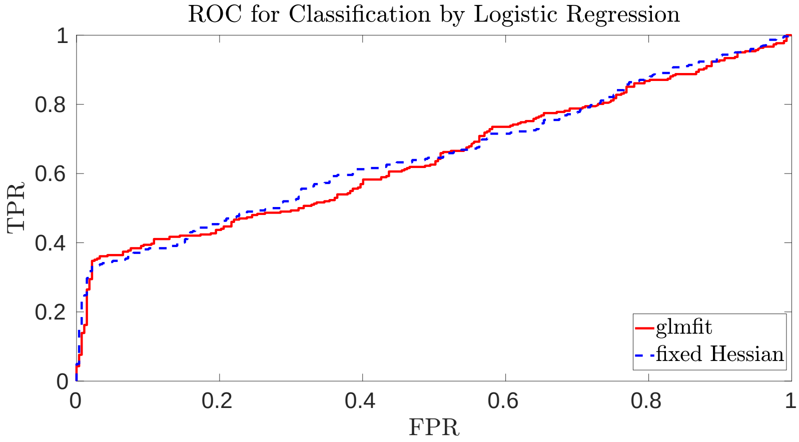

Fig. 1: ROC curve for the cancer detection scenario of iDASH with 1000 training vectors with 20 covariates.

From Figures 1 and 2 one can see that the SFH method classifies the data approximately as well as “glmfit” in Matlab, in the sense that one can always select a threshold that gives approximately the same true positive rate and false positive rate. One can thus conclude that the SFH method, with all its approximations, performs well compared to the standard Matlab method, which uses much more precise computations. By computing the TPR and FPR for several thresholds, we found that the approximations of our SFH method shifts the model a bit such that we need a slightly larger threshold to get approximately the same TPR and FPR as for the Matlab model. Since the ideal situation would be to end up with a true positive rate of 1 and false positive rate of 0, we see from Figure 1 that for the genomics dataset both models are performing rather poorly. The financial fraud use case is however much more amenable to binary classification as shown in Figure 2. The main conclusion is that our SFH method performs almost as well as standard methods such as those provided by Matlab.

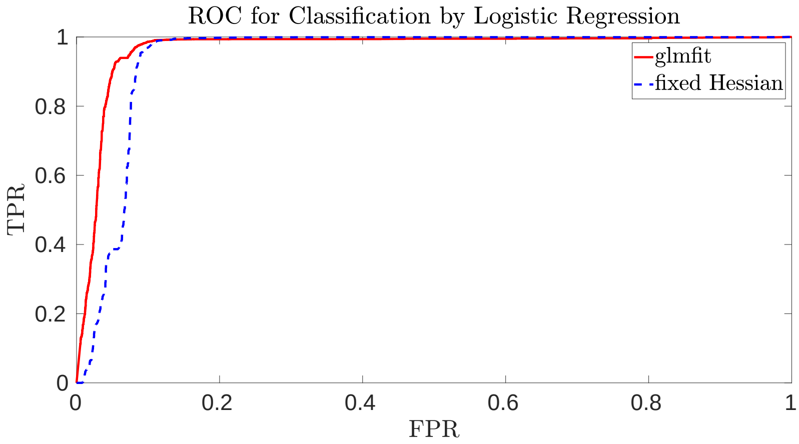

Fig. 2: ROC curve for the financial fraud detection with 1000 training vectors with 31 covariates.

### Implementation details and performance

Our implementation uses the FV-NFLlib software library [Cry16] which implements the FV homomorphic encryption scheme. The system parameters need to be selected taking into account the following three constraints:

1. the security of the somewhat homomorphic FV scheme,
2. the correctness of the somewhat homomorphic FV scheme,
3. the correctness of the $w$-NIBNAF encoding.

The security of a given set of system parameters can be estimated using the work of Albrecht, Player and Scott [APS15] and the open source learning with error (LWE) hardness estimator implemented by Albrecht [Alb04]. This program estimates the security of the LWE problem based on the following three parameters: the degree $D$ of the polynomial ring, the ciphertext modulus $q$ and $\alpha = \frac{\sqrt{2\pi}\sigma}{q}$ where $\sigma$ is the standard deviation of the error distribution $\chi_{\mathrm{err}}$. The security estimation is based on the best known attacks for the learning with error problem. Our system parameters are chosen to be $q = 2^{186}$, $D = 4096$ and $\sigma = 20$ (and thus $\alpha = \frac{\sqrt{2\pi}\sigma}{q}$) which results in a security of 78 bits.

As explained in the section on the FV scheme, the error in the ciphertext encrypting the result, should be small enough to enable correct decryption. By estimating the infinity norm of the noise we can select parameters that keep this noise under the correctness bound and in particular, we obtain an upper bound $t_{\mathrm{max}}$ of the plaintext modulus. Similarly, to ensure correct decoding the coefficients of the polynomial encoding the result, must remain smaller than the size of the plaintext modulus $t$. This condition results in a lower bound on the plaintext modulus $t_{\mathrm{min}}$.

It turns out that these bounds are incompatible for the chosen parameters, so we have to rely on the Chinese Remainder Theorem to decompose the plaintext space into smaller parts that can be handled correctly. The plaintext modulus $t$ is chosen as a product of small prime numbers $t_1,\ t_2,\ \ldots,\ t_n$ with $\forall i \in \{1,\ \ldots,\ n\} : t_i \leq t_{\max}$ and $t = \prod_{i=1}^{n}\ t_i \geq t_{\min}$, where $t_{\max}$ is determined by the correctness of the FV scheme and $t_{\min}$ by the correctness of the $w$-NIBNAF decoding. The CRT then gives the following ring isomorphism:

$$R_t \to R_{t_1} \times \ldots \times R_{t_n} : g(X) \mapsto (g(X) \bmod t_1,\ \ldots,\ g(X) \bmod t_n)\,.$$

and instead of performing the training algorithm directly over $R_t$, we compute with each of the $R_{t_i}$'s by reducing the $w$-NIBNAF encodings modulo $t_i$. The resulting choices for the plaintext spaces are given in Table 3.

Table 3: The parameters defining plaintext encoding

| | w | t |
|---|---|---|
| **genomic data** (1) | 71 | $5179 \cdot 5189 \cdot 5197$ |
| **financial data** (2) | 150 | $2237 \cdot 2239$ |

Since we are using the Chinese Remainder Theorem, each record will be encrypted using two (for the financial fraud case) or three (for the genomics case) ciphertexts. As such, a time-memory trade off is possible depending on the requirements of the application. One can choose to save computing time by executing the algorithm for the different ciphertexts in parallel; or one can choose to save memory by computing the result for each plaintext space $R_{t_i}$ consecutively and overwriting the intermediate values of the computations in the process.

The memory required for each ciphertext is easy to estimate: a ciphertext consists of 2 polynomials of $R_q = \mathbb{Z}_q[X]/(X^D+1)$, so its size is given by $2D \log_2 q$ which is $\approx$ 186kB for the chosen parameter set. Due to the use of the CRT, we require $T$ (with $T = 2$ or $T = 3$) ciphertexts encrypt each record, so the general formula for the encrypted dataset size is given by:

$$T \cdot (d+1) \cdot N \cdot 2D \log_2 q \text{ bits}\,,$$

with $T$ the number of prime factors used to split the plaintext modulus $t$ and $d+1$ (resp. $N$) the number of covariates (resp. records) used in the training set.

The time complexity of our SFH method is also easy to estimate, but one has to be careful to perform the operations in a specific order. If one would naively compute the matrix $\tilde{H}$ by first computing $\bar{H}$ and subsequently summing each row, the complexity would be $O(N \cdot d^2)$. However, the formula of the $k$-th diagonal element of $\tilde{H}$ is given by $\frac{-1}{4} \sum_{j=1}^{d+1} \big( \sum_{i=1}^{N} x_{k,i} x_{j,i} \big)$, which can be rewritten as $\frac{-1}{4} \sum_{i=1}^{N} x_{k,i} \big( \sum_{j=1}^{d+1} x_{j,i} \big)$. This formula shows that it is more efficient to first sum all the rows of $X$ and then perform a matrix vector multiplication with complexity $O(N \cdot d)$.

Table 4: Performance for the genomic dataset with a fixed number of covariates equal to 20

| # records | computation time | AUC | AUC glmfit |
|---|---|---|---|
| 500 | 22 min | 0.6348 | 0.6287 |
| 600 | 26 min | 0.6298 | 0.6362 |
| 800 | 35 min | 0.6452 | 0.6360 |
| 1000 | 44 min | 0.6561 | 0.6446 |

Table 5: Performance for the genomic dataset with a fixed number of records equal to 500

| # covariates | computation time | AUC | AUC glmfit |
|---|---|---|---|
| 5 | 7 min | 0.65 | 0.6324 |
| 10 | 12 min | 0.6545 | 0.6131 |
| 15 | 17 min | 0.6446 | 0.6241 |
| 20 | 22 min | 0.6348 | 0.6272 |

Table 6: Performance for the financial dataset with a fixed number of covariates equal to 31

| # records | computation time | AUC | AUC glmfit |
|---|---|---|---|
| 700 | 30 min | 0.9416 | 0.9619 |
| 800 | 36 min | 0.9411 | 0.9616 |
| 900 | 40 min | 0.9409 | 0.9619 |
| 1000 | 45 min | 0.9402 | 0.9668 |

Table 7: Performance for the financial dataset with a fixed number of records equal to 500

| # covariates | computation time | AUC | AUC glmfit |
|---|---|---|---|
| 5 | 5 min | 0.8131 | 0.8447 |
| 10 | 8 min | 0.9403 | 0.9409 |
| 15 | 11 min | 0.9327 | 0.9492 |
| 20 | 15 min | 0.9401 | 0.9629 |

This complexity is clearly visible in the tables, more specifically in Table 4 and Table 5 for the genomic use case, and Table 6 and Table 7 for the financial use case. These tables show a linear growth of the computation time for a growing number of records or covariates.

## Conclusions

The simple, but effective, iterative method presented in this paper allows one to train a logistic regression model on homomorphically encrypted input data. Our method can be used to outsource the training phase of logistic regression to a cloud service in a privacy preserving manner. We demonstrated the performance of our logistic training algorithm on two real life applications using different numeric data types. In both cases, the accuracy of our method is only slightly worse than standard algorithms to train logistic regression models. Finally, the time complexity of our method grows linearly in the number of covariates and the number of training input data points.

## References

[AHTPW16] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. Scalable and secure logistic regression via homomorphic encryption. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pages 142–144. ACM, 2016.

[Alb04] Martin Albrecht. Complexity estimates for solving LWE. `https://bitbucket.org/malb/lwe-estimator/raw/HEAD/estimator.py`, 2000–2004.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015.

[BBB+17] Charlotte Bonte, Carl Bootland, Joppe W. Bos, Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Faster homomorphic function evaluation using non-integral base encoding. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 579–600, Cham, 2017. Springer International Publishing.

[BL88] Dankmar Böhning and Bruce G Lindsay. Monotonicity of quadratic-approximation algorithms. *Annals of the Institute of Statistical Mathematics*, 40(4):641–663, 1988.

[BLN14] Joppe W Bos, Kristin Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data. *Journal of biomedical informatics*, 50:234–243, 2014.

[Cry16] CryptoExperts. FV-NFLlib. `https://github.com/CryptoExperts/FV-NFLlib`, 2016.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[Ger31] Semyon Aranovich Gershgorin. Uber die abgrenzung der eigenwerte einer matrix. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na*, (6):749–754, 1931.

[KSW+18] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang. Secure logistic regression based on homomorphic encryption. Cryptology ePrint Archive, Report 2018/074, 2018. `https://eprint.iacr.org/2018/074`.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. *J. ACM*, 60(6):Art. 43, 35, 2013.

[NLV11] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124. ACM, 2011.

[XWBB16] Wei Xie, Yang Wang, Steven M Boker, and Donald E Brown. Privlogit: Efficient privacy-preserving logistic regression by tailoring numerical optimizers. *arXiv preprint arXiv:1611.01170*, 2016.