Computer Security Symposium 2019
21 - 24 October 2019

秘密計算上の一括関数近似とそれを使った正確度の高いロジスティック回帰

濱田 浩気[1,a)]　五十嵐 大[1]　三品 気吹[1]　菊池 亮[1]

概要：本研究では，秘密計算で小さい誤差で関数の近似とロジスティック回帰の学習を行う手法を提案する．提案する秘密一括近似は秘密計算では計算が難しい指数関数などの計算を表参照により効率よく実現する方法であり，従来は計算コストが爆発的に増加してしまっていた許容される誤差が小さい場合にも効率よく計算できる．また，秘密一括近似を使ったロジスティック回帰の学習アルゴリズムは従来手法より高い収束速度の手法に基づいており，実際にいくつか入力例で高速かつ高精度にパラメータの学習ができていることが確認できた．

キーワード：秘密計算，ロジスティック回帰，近似

A secure batch function evaluation algorithm and its application to secure logistic regression algorithm with high accuracy

Koki Hamada[1,a)]　Dai Ikarashi[1]　Ibuki Mishina[1]　Ryo Kikuchi[1]

Abstract: In this paper, we propose two algortihms for secure multi-party computation. The first algorithm efficiently evaluate a single public function for a batch of data with high accuracy. We also propose a novel secure training algorithm for logistic regression. The high convergence speed of the proposed algorithm enable us to compute accurate trained parameters in a few minutes by the help of the first algorithm and other optimization techniques. We implemented our method and confirmed the efficiency and accuracy of our algorithms.

Keywords: Secure multi-party computation, logistic regression, approximation

1. はじめに

近年，個人に関する様々な情報を容易に取得できる環境が整い，データ分析技術の進歩と相まって，個人に関する情報の活用への期待が高まっている．その一方で，個人情報保護やプライバシの観点から個人に関する情報は極めて慎重な取扱いが必要とされ，データの保護と活用をどう両立させるかが問題となっている．

このようなデータの保護と活用の両立を目指して，プライバシ保護データ分析 (Privacy-Preserving Data Analysis: PPDA) 技術の研究が盛んになってきている．PPDA の有力なアプローチの一つに，秘密計算や秘匿関数計算，マルチパーティプロトコルと呼ばれる，暗号学に基づいた手法がある．秘密計算は Yao による基本的なアイディア [1] を端緒とする技術であり，暗号化などの方法でデータを秘匿化したまま一度も元のデータに戻すことなく任意の計算を行う．秘密計算を用いることで通常のデータ分析と同等の結果を高いプライバシ保護の下で得ることができるが，秘密計算では通常の計算機上の計算に比べて処理速度が低下してしまうことが実用上の課題となっている．その原因は大きく二つある．

一つは，基本演算のオーバーヘッドである．秘密計算ではデータの秘匿性を保つために，乗算のような通常の計算

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories
[a)] koki.hamada.rb@hco.ntt.co.jp

機では一命令で実行可能な基本演算にも複雑な処理を必要とする．その結果，処理時間も大きくなってしまう．これに対しては近年改良が進んでおり，Ben-David らによる FairplayMP [2] など，効率のよい基本演算を備えたフレームワークの提案，実装が行われている．

もう一つは，回路に基づいた構成による計算量の悪化である．秘密計算は Goldreich ら [3] や Ben-Or ら [4] により提案された回路に基づく構成方法を用いることで，任意の計算を実現することができる．しかしながら，この回路に基づいた一般的な構成方法を用いると，実用上重要な多くのアルゴリズムで通常の計算機上での計算に比べて計算量が大きくなってしまう．このため，特定の処理ごとに秘密計算上の効率的なアルゴリズムを設計する研究が進んでおり，Aggarwal らによる k 番目の要素の選択 [5] や Damgård らによるビット分解 [6] などが提案されている．

さらに最近では，秘密計算で実数を扱う手法が提案されてきている．固定小数点数は Catrina と Saxena [7] により実現され，さらに Aliasgari らにより浮動小数点数 [8] も提案されている．Aliasgari らはまた，指数関数や対数関数など科学技術計算で重要な関数も実現している．

近年では，データ分析でよく用いられるロジスティック回帰と呼ばれる分析も秘密計算で実現されるようになってきている．[9], [10], [11]

ロジスティック回帰は説明変数から 2 値の予測に用いられる分析手法である．利用方法は未知のデータに対する予測のみならず，各説明変数の重要性を把握するための統計量としてパラメータを計算する場合もあり，正確なデータを安全に計算できることが求められる．

Xie ら [9] は平文の場合と完全に一致する計算を実現している．ただし，学習された最終パラメータだけでなく，学習途中のパラメータもすべて参加者に明かされてしまう．そのため入力データの情報が漏れる危険が残る．Mohassel と Zhang [10] や三品ら [11] は反復回数と入出力サイズ以外の情報をすべて秘匿しながらパラメータの学習を実現している．ただし，評価は主に予測性能で行われており，学習されたパラメータ自体の誤差を数値的に評価はできていなかった．

1.1 本研究の貢献

本研究では，秘密計算で反復回数と入出力以外の情報を明かさずに，しかもパラメータの誤差を小さくロジスティック回帰の学習を行う手法を提案する．

提案手法は高精度なロジスティック回帰の実現のために，まず，秘密一括近似と呼ぶ新しい技法を提案する．秘密一括近似は秘密計算では計算が難しい指数関数などの計算を表参照により効率よく実現する方法である．これまでに一括写像 [12] と呼ばれる方法で関数の近似は行われてきた [11] が，誤差を小さく実現しようとすると参照表が爆発的に大きくなる問題があった．提案する秘密一括近似は従来より小さい参照表で高精度な関数の近似計算を実現する．

秘密一括近似を使って，高精度なロジスティック回帰を実現する．これまで秘密計算でのロジスティック回帰には比較的計算が容易だが収束速度が遅い手法が用いられてきた [9], [10], [11]．提案する秘密計算ロジスティック回帰アルゴリズムはいくつかの工夫により数値計算上の問題を回避し，秘密計算上の固定小数点で実現される．

さらに，提案手法を実装して評価を行う．その結果，秘密一括近似は許容される誤差が小さい場合にも効率よく近似計算ができることがわかった．さらに，これを使ったロジスティック回帰では，相関係数が 0.99999 程度の正解に近い値が得られることが確認できた．

また，効率性をできるだけ損なわずに安全性をさらに高める方法の議論も行う．反復計算が多重に行われる場合に反復回数の秘匿を従来よりも小さいコストで実現される方法を提案する．

2. 準備

本稿で提案するアルゴリズムは，秘密計算上の演算の組み合わせにより構築される．本節では，本稿で用いる記法，定義と提案アルゴリズムがサブルーチンとして用いる既存の秘密計算上の各演算，元にする平文のアルゴリズムの説明を行う．

2.1 記法，定義

本稿で扱う値はすべて位数 q の有限体 K 上の値とする．$\ell := \lceil \log_2 q \rceil$ とする．ベクトルや行列の添字は 1 始まりとし，ベクトル $\vec{a}$ の第 i 要素を $\vec{a}[i]$ と書く．また，ベクトルは特に断りのない場合，列ベクトルとする．すなわち，ベクトル $\vec{a}$ の大きさが n のとき，$\vec{a} = (\vec{a}[1], \vec{a}[2], \ldots, \vec{a}[n])^{\mathrm{T}}$ である．ベクトル $\vec{a}$ とベクトル $\vec{b}$ を連結したベクトルを $\vec{a} \,||\, \vec{b}$ と書く．すなわち，$\vec{a} = (\vec{a}[1], \ldots, \vec{a}[m])^{\mathrm{T}}$，$\vec{b} = (\vec{b}[1], \ldots, \vec{b}[n])^{\mathrm{T}}$ のとき，$\vec{a} \,||\, \vec{b} = (\vec{a}[1], \ldots, \vec{a}[m], \vec{b}[1], \ldots, \vec{b}[n])^{\mathrm{T}}$ である．また，行列 M の第 i 行を $M[i]$ で，第 i 行第 j 列の要素を $M[i][j]$ で，それぞれ参照する．

2.2 秘密計算

本稿では有限体 K 上の秘密計算向けのアルゴリズムを提案する．提案するアルゴリズムは，本節で延べる各基本演算を提供する秘密計算であれば，手法に依存しない．

2.2.1 秘匿化，復元

$a \in K$ を暗号化や秘密分散などの手段で秘匿化した値を a の暗号文と呼び，$[\![a]\!]$ と表記する．また，a を $[\![a]\!]$ の平文と呼ぶ．ベクトル $\vec{v} = (\vec{v}[1], \ldots, \vec{v}[n])^{\mathrm{T}} \in K^n$ の各要素を秘匿化したベクトル $([\![\vec{v}[1]]\!], \ldots, [\![\vec{v}[n]]\!])^{\mathrm{T}}$ を $[\![\vec{v}]\!]$ と表記する．行列 M についても同様に，各要素を秘匿した行列を $[\![M]\!]$ と表記し，第 i 行第 j 列の要素の暗号文を $[\![M[i,j]]\!]$ で

参照する．a の暗号文 $[\![a]\!]$ を n 個並べた暗号文のベクトル $(\underbrace{[\![a]\!], \ldots, [\![a]\!]}_{n\text{個}})$ を $[\![a]\!]^n$ と略記する．

暗号文 $[\![a]\!]$ から，対応する平文である a を復元する演算の実行を

$$a \leftarrow \mathsf{Open}([\![a]\!])$$

と記述する．

2.2.2 算術演算

加算，減算，乗算の各演算は 2 つの値 $a, b \in K$ の暗号文 $[\![a]\!], [\![b]\!]$ を入力とし，それぞれ $a+b$，$a-b$，ab の計算結果 c_1，c_2，c_3 の暗号文 $[\![c_1]\!]$，$[\![c_2]\!]$，$[\![c_3]\!]$ を計算する．これらの演算の実行をそれぞれ，

$$[\![c_1]\!] \leftarrow \mathsf{Add}([\![a]\!], [\![b]\!]),$$
$$[\![c_2]\!] \leftarrow \mathsf{Sub}([\![a]\!], [\![b]\!]),$$
$$[\![c_3]\!] \leftarrow \mathsf{Mul}([\![a]\!], [\![b]\!])$$

と記述する．誤解を招く恐れのない場合は，$\mathsf{Add}([\![a]\!], [\![b]\!])$，$\mathsf{Sub}([\![a]\!], [\![b]\!])$，$\mathsf{Mul}([\![a]\!], [\![b]\!])$ をそれぞれ $[\![a]\!]+[\![b]\!]$，$[\![a]\!]-[\![b]\!]$，$[\![a]\!][\![b]\!]$ と略記する．

また，行列やベクトルの加算，減算，乗算の各演算は平文での計算と同様の順序で加算，減算，乗算を行うものとし，同様に $+, -$ を使って記述する．

2.2.3 比較

比較の演算は，$a, b \in K$ の暗号文 $[\![a]\!]$，$[\![b]\!]$ を入力とし，$a < b$ ならば $c = 1$，そうでないならば $c = 0$ である $c \in K$ の暗号文 $[\![c]\!]$ を出力する．この演算の実行を

$$[\![c]\!] \leftarrow ([\![a]\!] \overset{?}{<} [\![b]\!])$$

と記述する．

2.2.4 符号付き固定小数点数 [7]

本稿では，符号付きの固定小数点数を扱う．本来秘密計算では K の要素しか扱うことはできないが，符号付き固定小数点数を K の要素と対応付けることにより，符号付き固定小数点数を扱うことができるようになる．以降は [7] と同様に，K の要素と固定小数点数を対応付け，必要に応じて K の要素を固定小数点数と解釈することとする．

2.2.5 逆数

逆数の演算は，1 つの値 $a \neq 0$ の暗号文 $[\![a]\!]$ を入力とし，$1/a$ の計算結果 c の暗号文 $[\![c]\!]$ を計算する．この演算の実行を

$$[\![c]\!] \leftarrow \mathsf{Inv}([\![a]\!])$$

と記述する．

2.2.6 秘密一括写像

秘密一括写像 [12] の演算は，ソート済みの次数 m のベクトル $\vec{s}$，次数 m のベクトル $\vec{d}$，次数 n のベクトルの暗号文 $[\![\vec{x}]\!]$ を入力とし，各 $i \in [1, n]$ について，$\vec{y}[i] = \vec{d}[j]$ s.t. $\vec{s}[j] \leq \vec{x}[i] < \vec{s}[j+1]$ を満たすような次数 n のベクトル $\vec{y}$ の暗号文 $[\![\vec{y}]\!]$ を出力する．この演算の実行を

$$[\![\vec{y}]\!] \leftarrow \mathsf{BatchMap}([\![\vec{x}]\!], \vec{s}, \vec{d})$$

と記述する．

この処理はルックアップテーブルとして使うことができ，関数 f に対して，各区間 $[\vec{s}[j], \vec{s}[j+1])$ に対応するの関数値を $\vec{d}[j]$ としておくことで，$\vec{x}$ の各要素に対して f を適応した近似値を計算することができる．

Algorithm 1 共役勾配法

入力: d 次の正定値対称行列 A，d 次のベクトル $\vec{b}$
出力: d 次のベクトル $\vec{w}$ s.t. $A\vec{w} = \vec{b}$

$\vec{w}_0 \leftarrow 0^d$
$\vec{r}_0 \leftarrow \vec{b}$
for k = 0,1,... **do**
  $\rho_k \leftarrow \vec{r}_k^T \vec{r}_k$
  収束していたら $\vec{w}_k$ を出力して終了
  $\vec{p}_k \leftarrow \begin{cases} \vec{r}_0 & \text{if } k = 0, \\ \vec{r}_k + (\rho_k/\rho_{k-1})\vec{p}_{k-1} & \text{otherwise.} \end{cases}$
  $\vec{a}_k \leftarrow A\vec{p}_k$
  $\alpha_k \leftarrow \rho_k/(\vec{p}_k^T \vec{a}_k)$
  $\vec{w}_{k+1} \leftarrow \vec{w}_k + \alpha_k \vec{p}_k$
  $\vec{r}_{k+1} \leftarrow \vec{r}_k - \alpha_k \vec{a}_k$

2.3 共役勾配法

共役勾配法は，d 次の正定値対称行列 A と d 次のベクトル $\vec{b}$ を入力とし，$A\vec{w} = \vec{b}$ を満たす $\vec{w}$ を計算する反復手法である．アルゴリズムを Algorithm 2 に示す．数値誤差がない場合は高々 d 回の反復で解に収束することが知られている．

2.4 ロジスティック回帰

ロジスティック回帰は一般化線形回帰モデルの一種であり，d 個の説明変数と呼ばれる入力 $x_1, \ldots, x_d$ と $w_1, \ldots, w_d$ をパラメータとし，0,1 の 2 値の目的変数 y を

$$y = \mathsf{Sigmoid}(w_1 x_1 + w_2 x_2 + \cdots + w_d x_d) \qquad (1)$$

とするモデルである．ここで $\mathsf{Sigmoid}(x) = 1/(1+e^{-x})$ である．

2.4.1 予測

ロジスティック回帰で予測を行う場合は，あらかじめ多数の目的変数と説明変数の組から適切なパラメータ $w_1, \ldots, w_d$ を求めておき (学習と呼ぶ)，与えられた d 個の説明変数に対しては $w_1, \ldots, w_d$ を使って式 1 により目的変数の値を予測する．

2.4.2 学習

ロジスティック回帰の学習は，行列とベクトルで与えられた n 組の説明変数 X ($n \times d$ 行列) と目的変数 $\vec{t}$ (次数 d) を元に，定められた誤差関数を最小化するパラメータ $\vec{w}$ を

Algorithm 2 ニュートン法によるロジスティック回帰の学習

入力: 説明変数を表す $n \times d$ の行列 X，目的変数を表す n 次のベクトル $\vec{t}$
出力: 学習した重みを表す d 次のベクトル $\vec{w}$

$\vec{w} \leftarrow 0^d$
for k = 0,1,... **do**
  $\vec{y} \leftarrow \mathsf{Sigmoid}(X\vec{w})$
  $\vec{g} \leftarrow X^{\mathrm{T}}(\vec{y} - \vec{t})$
  $\vec{g}^{\mathrm{T}}\vec{g}$ が十分に小さければ $\vec{w}$ を出力して終了
  $H \leftarrow X^{\mathrm{T}}\,\mathsf{Diag}(\vec{y}\,(\cdot)\,(1^n - \vec{y}))X$
  $\vec{u} \leftarrow H^{-1}\vec{g}$
  $\vec{w} \leftarrow \vec{w} - \vec{u}$

求める．効率のよいアルゴリズムとして，誤差の最小化にニュートン法を用いたものが知られており，Algorithm 2 に示す．

3. 秘密一括近似アルゴリズム

本節では，複雑な関数の関数値を秘密計算上で効率よく求めることを目的とする，秘密一括近似アルゴリズムを提案する．秘密一括近似アルゴリズムは公開の関数 f と秘匿された n 個の入力 $[\![\vec{x}]\!] = ([\![x_1]\!], \ldots, [\![x_n]\!])$ が与えられたとき，秘匿された n 個の値 $[\![\vec{y}]\!] = ([\![y_1]\!], \ldots, [\![y_n]\!])$ $(y_i = f(x_i))$ の近似値を高精度にかつ効率良く求める．

3.1 アイディア

提案アルゴリズムは，秘密一括写像を一般化し，秘密一括写像を高精度化しようとした場合の問題点を解決したものと見ることができる．

秘密一括写像は関数の定義域を細かい区間に区切り，区間ごとの関数値の近似値を参照表として用意しておき，与えられた入力それぞれに対して入力が含まれる区間に対応した値を参照表から読みだして近似値として出力していた．そのため，出力の誤差を小さくするためには区間をその分細かく区切る必要があった．例えば，$[0, 10^6]$ の定義域でシグモイド関数 $1/(1+e^{-x})$ を秘密一括写像で近似する場合，出力の誤差を 2^{-10} 以下にする場合は 339 個の区間を作ることで実現できるが，出力の誤差を 2^{-15}，2^{-20}，2^{-25} 以下と小さくしてくと区間数は 10814，346045，11073420 と爆発的に大きくなっていく．

秘密一括写像は内部で入力と参照表と入力を連結したベクトルのソートを使用するため，入力数を n，参照表の大きさ (区間数) を m とすると，$(m+n)\log(m+n)$ に比例した処理時間が必要となる．そのため，表の大きさ m が n に対して大きくなると効率が悪くなってしまう．

提案アルゴリズムは，各区間での近似値を参照表から読んだ値そのものとするのではなく，**読んだ値と入力から計算される値**とする．これにより，秘密一括写像で実現された場合に比べて同じ区間に対する出力の誤差を小さくする．言い換えると，秘密一括写像では各区間を定数関数で近似していたのに対し，提案アルゴリズムは一般の関数 (例えば多項式) で近似することで誤差を小さくする．これにより，誤差を一定の値以下に抑えるための区間の幅を大きくすることができ，従来よりも参照表の大きさを小さくすることができる．

3.2 提案アルゴリズム

提案アルゴリズムの入出力は一括写像と似ており，公開の関数 f に関する情報と秘匿された n 個の値 $[\![\vec{x}]\!] = ([\![x_1]\!], \ldots, [\![x_n]\!])$ を入力とし，秘匿された n 個の値 $[\![\vec{y}]\!] = ([\![y_1]\!], \ldots, [\![y_n]\!])$(ただし y_i は $f(x_i)$ の近似値) を出力する．また，許容する誤差の最大値を $\epsilon = 2^{-b}$ とする (b はパラメータ)．すなわち，すべての $i \in [1, n]$ に対して $|f(x_i) - y_i| \leq \epsilon$ となるような $[\![\vec{y}]\!]$ を計算する．

アルゴリズムの流れは以下の通りである．まず，入力を受け付ける前に，以下のようにあらかじめ f に関する事前計算を行う．

(1) 区間内での近似値の計算に使用する，ℓ 個のパラメータを持つ公開の関数 g を定める．例えば，g を $g(x) = a_3x^2 + a_2x + a_1$ のような 2 次多項式とすると，g は a_1, a_2, a_3 の 3 個のパラメータを含んでおり，$\ell = 3$ である．
(2) f の定義域を複数の区間に分割し，区間ごとに区間内での f と g の誤差が ϵ 以下となるように ℓ 個のパラメータを定める．i 番目の区間 D_i では，すべての $x \in D$ について $|f(x) - g(x, a_{i,1}, \ldots, a_{i,\ell})| \leq \epsilon$ となるように $a_{i,1}, \ldots, a_{i,\ell}$ を定める．
(3) 区間を表すベクトルを $\vec{s}$ とする．各 $i \in [1, \ell]$ について，$\vec{s}$ の各区間に対応した i 番目のパラメータの列 $\vec{a}_i$ を $\vec{a}_i = (a_{i,1}, \ldots, a_{i,m})$ とする．ここで m は区間数である．

続いて，入力 $[\![\vec{x}]\!] = ([\![x_1]\!], \ldots, [\![x_n]\!])$ に対して，以下のように $[\![\vec{y}]\!] = ([\![y_1]\!], \ldots, [\![y_n]\!])$ を計算する．

(1) 各 $i \in [1, \ell]$ について秘密一括写像を

$$[\![\vec{a}_i]\!] \leftarrow \mathsf{BatchMap}([\![\vec{x}]\!], \vec{s}, \vec{a}_i)$$

のように適用し，$\vec{x}$ に対応した ℓ 個のパラメータの組の暗号文 $[\![\vec{a}_i]\!]$ $(i \in [1, \ell])$ を求める．
(2) $[\![\vec{x}]\!]$ と $[\![\vec{a}_1]\!], \ldots, [\![\vec{a}_\ell]\!]$ を使って g に従って $[\![\vec{y}]\!]$ を計算する．例えば上述の例の場合は，$[\![\vec{y}]\!] \leftarrow [\![\vec{a}_3]\!][\![\vec{x}]\!][\![\vec{x}]\!] + [\![\vec{a}_2]\!][\![\vec{x}]\!] + [\![\vec{a}_1]\!]$ である．

秘密一括写像で各区間で一つの値を出力して近似値としていた代わりに，提案アルゴリズムでは ℓ 個の値を中間出力として得て，これらの値と入力を使った計算により出力を求めている．重要な点は，区間ごとに得られる中間出力は入力の各要素毎に異なるものの，(g で表される) その後の計算はすべての入力に対して一定であることである．こ

れにより，入力がどの区間に入っていたとしても動作が同じであるため，安全性が担保される．

計算効率は素直な実装では表サイズ m，入力数 n の一括写像 ℓ 回分であるが，区間を表すベクトル $\vec{s}$ と入力を結合してソートする部分を使いまわすことにより，ソートの回数は一括写像の場合を同じに抑えることができる．

3.3 参照表作成アルゴリズム

秘密一括近似の近似関数 g は任意に定めることができるが，g は秘密計算で計算されるため，f よりも計算コストが大きい関数では無意味である．ここでは，秘密計算でも計算しやすい g の一例と，g に対する具体的な区間とパラメータの計算方法を提案する．簡単のため，f の定義域は単一の閉区間であることとする．

提案するアルゴリズムは貪欲法の一種である．まず最初に g を定め，定義域の左端から順に，誤差が ϵ を超えないようなパラメータ $a_1 \ldots, a_\ell$ が存在するような可能な限り大きい区間を求めていく．

初期化$k \geq 1$ とし，g を $g(x) = \sum_{i=0}^{k} a_i (x - a_*)^i$ (a_* を中心とした k 次のテイラー展開) とする．r_0 を f の定義域の左端，r_1 を右端とする．

(1) g を f の r_* を中心とした k 次のテイラー展開とするとき，g の区間 $[r_0, r_*]$ での誤差が ϵ を超えないような最大の $r_* \in [r_0, r_1]$ を，r_* についての二分探索で求める．すなわち，初期の探索区間を $[r_0, r_1]$ とし，区間が十分に狭くなるまで以下を繰り返す．
  (a) r_* を区間の中心とする
  (b) r_* を中心とした k 次のテイラー展開によりパラメータを仮に設定し，テイラーの定理により区間 $[r_0, r_*]$ の誤差の最大値 e を見積もる．
  (c) $e > \epsilon$ ならば区間の右端を r_* に，そうでなければ区間の左端を r_* に更新する．

(2) 区間の左端をテイラー展開の中心 a_* とする．

(3) 同様の二分探索により，区間 $[r_0, r_*]$ で今設定したパラメータの g の誤差が ϵ 以下となるような最大の r_* を求める

(4) 区間 $[r_0, r_*]$ に現在のパラメータを対応させる．

(5) 定義域の左端を r_* に更新し，上述の手続きをを区間 $[r_*, r_1]$ がなくなるまで繰り返し行う．

ここではテイラー展開の誤差評価が容易である性質を利用した．また，$k = 0$ の場合はこれまでの一括写像による実現と一致する．

3.3.1 例

いくつかの関数に対して，上述の手法を適用して実際にできた区間数 m を表 1 に示す．ここで，e^{-x} と $\mathsf{Sigmoid}(x)$ の定義域は $[0, 10^6]$，$1/x$ と $1/\sqrt{x}$ の定義域は $[1, 10^6]$ とした．表からわかるように，区間数 m はいずれの場合も指数的に増加しているが，k を大きくすることによって区間数を小さくすることができている．これにより，一括写像 ($k = 0$) では区間数 m が大きすぎて計算が困難だったような高精度に関数値の計算を行いたい場合 (b が大きい場合) にも提案手法の適用により現実的になる可能性がある．

3.3.2 区間の判定に必要なビット長

提案手法は，b が比較的小さい場合でも，従来手法よりも高速になる場合がある．提案手法の適用による区間数の減少に伴い，一括写像で入力と $\vec{s}$ のソートの際に必要なビット数 κ を小さくできる場合がある．上述の例では表 1 のように κ は計算され，例えば $b = 10$ の場合でも $k = 2$ のときは $k = 0$ の場合に比べてソートに必要なビット数が半分以下となり，ソートの部分の時間は半減させることができる．

4. 高精度な秘密計算ロジスティック回帰

次に，秘密一括近似を利用し，秘密計算で高精度なロジスティック回帰を行う手法を提案する．

4.1 アイディア

提案手法はニュートン法によるロジスティック回帰を秘密計算で実現する．ニュートン法は収束の速い最適化手法として知られており，R などの統計ソフトウェアでも使われている．秘密計算の文脈ではこれまでニュートン法よりも計算は容易であるものの収束が遅い再急降下法や Nesterov の加速法 [11] や準ニュートン法 [9] が使われていた．

ニュートン法の計算で問題となるのは，ヘッセ行列の逆行列とベクトルの積の計算であるが，これに対しては以下のように対処する．逆行列の直接的な計算は，その計算コストの大きさと計算の不安定さから，通常の数値計算でも可能な限り回避される．提案手法はニュートン法のヘッセ行列が正定値であることを利用し，共役勾配法により直接逆行列を計算せずにヘッセ行列の逆行列とベクトルの積の計算を行う．

また，結果の高精度化のためにはすべての計算がより正確に行われる必要がある．ロジスティック回帰の学習のうち秘密計算で正確な値の計算が難しいのはシグモイド関数 $\mathsf{Sigmoid}(x)$ の計算であった．これに対しては，前節で提案した方法により高精度で計算する．

4.2 提案アルゴリズム

提案アルゴリズムを Algorithm 4，Algorithm 3 に示す．平文での方法 (Algorithm 2，Algorithm 1) と比較すると，主な違いは

- 各値が暗号文に各値が暗号文に置き換わっている
- 除算の代わりに $\mathsf{Inv}(\cdot)$ との積が使われている
- ヘッセ行列の逆行列を直接求める代わりに共役勾配法で $[\![\vec{u}]\!]$ を求めている

などであり，平文の場合と数値誤差を除いてほとんどの部分の挙動は同じである．以下では違いについて言及する．

表 1 許容する誤差 $\epsilon = 2^{-b}$ とテイラー展開の次数 k ごとの区間数 m と一括写像のソートに必要なビット数 κ．ここで $k = 0$ は従来手法，$k = 1, 2$ は提案手法である．

		$b = 10$		$b = 15$		$b = 20$		$b = 25$	
f	k	m	κ	m	κ	m	κ	m	κ
e^{-x}	0	611	13	19,541	19	625,317	24	20,010,133	45
	1	32	8	155	11	959	29	4,838	17
	2	13	6	35	8	112	25	351	12
$1/x$	0	584	30	18,647	35	596,680	40	19,093,755	45
	1	41	16	237	23	1,282	30	7,651	32
	2	24	15	89	25	376	26	1,658	28
$\mathsf{Sigmoid}(x)$	0	339	28	10,814	17	346,045	38	11,073,420	28
	1	24	23	130	26	677	12	4,063	31
	2	10	21	29	7	87	9	266	26
$1/\sqrt{x}$	0	569	29	18,189	34	582,043	39	18,625,350	44
	1	67	24	506	26	3,838	29	21,868	31
	2	51	21	369	24	2,892	26	15,647	27

Algorithm 3 秘密計算共役勾配法

記法: $[\![\vec{w}]\!] \leftarrow \mathsf{CGD}([\![X]\!], [\![\vec{y}]\!], [\![\vec{b}]\!])$
入力: $[\![X]\!]$, n 次のベクトルの暗号文 $[\![\vec{y}]\!]$，d 次のベクトルの暗号文 $[\![\vec{b}]\!]$
出力: d 次のベクトルの暗号文

1: $[\![\vec{w}_0]\!] \leftarrow 0$
2: $[\![\vec{r}_0]\!] \leftarrow [\![\vec{b}]\!]$
3: **for** k = 0,1,... **do**
4: $\quad [\![\vec{r}'_k]\!] \leftarrow [\![\vec{r}_k]\!]\,\mathsf{Inv}(\mathsf{L1}([\![\vec{r}_k]\!]))$
5: $\quad [\![\rho'_k]\!] \leftarrow [\![\vec{r}_k]\!]^T [\![\vec{r}'_k]\!]$
6: $\quad [\![\rho'_k]\!]$ が十分に小さければ $[\![\vec{w}_k]\!]$ を出力して終了
7: $\quad [\![\vec{p}'_k]\!] \leftarrow \begin{cases} [\![\vec{r}'_0]\!] & \text{if } k = 0, \\ [\![\vec{r}'_k]\!] + ([\![\rho'_k]\!]\,\mathsf{Inv}([\![\rho'_{k-1}]\!]))[\![\vec{p}'_{k-1}]\!] & \text{otherwise.} \end{cases}$
8: $\quad [\![\vec{a}'_k]\!] \leftarrow A[\![\vec{p}'_k]\!]$
9: $\quad [\![\vec{a}'_k]\!] \leftarrow [\![X]\!]^{\mathrm{T}}(([\![\vec{y}]\!] \cdot (1^d - [\![\vec{y}]\!])) \cdot ([\![X]\!][\![\vec{p}]\!]))$
10: $\quad [\![\alpha^+_k]\!] \leftarrow [\![\rho'_k]\!]\,\mathsf{Inv}([\![\vec{p}'_k]\!]^T [\![\vec{a}'_k]\!])$
11: $\quad [\![\vec{w}_{k+1}]\!] \leftarrow [\![\vec{w}_k]\!] + [\![\alpha^+_k]\!][\![\vec{p}'_k]\!]$
12: $\quad [\![\vec{r}_{k+1}]\!] \leftarrow [\![\vec{r}_k]\!] - [\![\alpha^+_k]\!][\![\vec{a}'_k]\!]$

Algorithm 4 秘密計算ニュートン法

入力: 説明変数を表す $n \times d$ の行列の暗号文 $[\![X]\!]$，目的変数を表す n 次のベクトルの暗号文 $[\![\vec{t}]\!]$
出力: 学習した重みを表す d 次のベクトルの暗号文 $[\![\vec{w}]\!]$

1: $[\![\vec{w}]\!] \leftarrow 0^d$
2: **for** k = 0,1,... **do**
3: $\quad [\![\vec{y}]\!] \leftarrow \mathsf{Sigmoid}([\![X]\!][\![\vec{w}]\!])$
4: $\quad [\![\vec{g}]\!] \leftarrow [\![X]\!]^{\mathrm{T}}([\![\vec{y}]\!] - [\![\vec{t}]\!])$
5: $\quad [\![\vec{g}]\!]^{\mathrm{T}} [\![\vec{g}]\!]$ が十分に小さければ $[\![\vec{w}]\!]$ を出力して終了
6: $\quad [\![\vec{u}]\!] \leftarrow \mathsf{CGD}([\![\vec{w}]\!], [\![\vec{y}]\!], [\![\vec{g}]\!])$
7: $\quad [\![\vec{w}]\!] \leftarrow [\![\vec{w}]\!] - [\![\vec{u}]\!]$

なお $\vec{x}\ (\cdot)\ \vec{y}$ は 2 つのベクトル $\vec{x}$ と $\vec{y}$ の要素ごとの積の計算を表す．

4.2.1 共役勾配法の変形

Algorithm 3 のステップ 4 と 10 では，$\vec{r}$ を $\mathsf{L1}(\vec{r})$ で割っている．ここで $\mathsf{L1}(\cdot)$ は入力のベクトルの各要素の絶対値の総和を計算する処理である．これは値の正規化によるオーバーフロー対策である．共役勾配法では，各変数の値は非常に大きい値から小さい値までの広い範囲の値を取ることが多い．しかしながら今回は固定小数点で実現するため，大きい値が出現するとオーバーフローの危険が生じる．そのため，各値を 0 に近づけるためにこのような正規化を追加で行っている．これにより，x' のような「′」のついた値は元々の $1/\mathsf{L1}(\vec{r}_k)$ 倍に，α^+ のような「+」のついた値は元々の $\mathsf{L1}(\vec{r}_k)$ 倍になる．その他の変数の値は，この変形を行うことにより変わらない．よって，この変形は出力の影響を与えない．

4.2.2 ヘッセ行列の計算回避

Algorithm 3 のステップ 8 は，平文の処理と大きく異なっている．ここでは，ヘッセ行列とベクトル $\vec{b}$ の積の計算の必要があり，ヘッセ行列の計算に用いられる X や $\vec{y}$ を $\vec{b}$ に順に掛けていくことにより，計算コストの大きいヘッセ行列の直接の計算を回避している．これは数値計算でよく用いられる手法である．

4.3 ループ数の秘匿

提案アルゴリズムは反復計算であり，反復の終了判定を明かして動作する．これはすべての情報を隠す場合には隠す必要がある情報である．反復の回数を隠すためには，十分に大きい回数の反復計算を収束後も続けたふりをする手法が元いられる．提案アルゴリズムもこの方法により，安全性を高めることができる．

ただし，提案手法は二重の反復計算となっており，それぞれに大きい反復回数を設定すると全体での処理時間が非常に大きくなってしまう．これに対しては，以下に提案する，多重で反復計算を行う場合の反復回数削減方法を用いることで緩和が可能である．

まず，内側の反復計算の最大反復数を小さめに設定する．こうすると内側の反復計算の収束が保証されなくなり，全体の計算が正しく行われない可能性がある．これを回避するため，内側の反復計算が終了しなかった場合にはそのと

きの状態を保存して，その後の処理を進める「ふり」をする．この間はすべての値は更新されない．そして，再度内側の反復計算に入ったところで保存しておいた状態を使って再度計算を行う．内側の反復数が最大に達したときに内側の反復計算が終了していた場合は，それを使って通常どおりの計算を進める．これにより，内側が早めに終わってしまったときの無駄な計算のを軽減することができる．

提案手法では，ある入力に対して，内側の反復計算が順に $27, 28, 30, 29, 28, 29, 23, 15, 1$ 回で終了した場合があった．このときは，従来の方法では，たまたま例えば内側の最大反復数を 30，外側を 9 回に設定できれば内側の見かけも含めた反復計算の総実行回数を 270 回に最小化できていたことになる．ここで提案するループ数秘匿方法を使うと，例えば内側を 15 回，外側を 16 回に設定することで，さらに 240 回に軽減できる．

5. 評価

提案手法の実用性を確認するため，実装して評価を行った．実装は MEVAL と呼ばれる 3 パーティ秘密計算システム?上で行った．測定は以下に示すマシン 3 台を用いて行った．

- OS: CentOS Linux release 7.3.1611
- CPU: Intel Xeon Gold 6144k(3.50GHz 8 コア/16 スレッド) × 2
- メモリ: 768GB
- ネットワーク: Intel Ethernet Controller X710/X557-AT 10G リング構成

5.1 秘密一括近似の処理時間

秘密一括近似の有効性を見るため，3.3.1 節で用いた 4 種類の関数について，3.3.1 節と同じ条件で入力数 n を変えながら処理時間の測定を行った．結果を 1 に示す．横軸が許容誤差 $\epsilon = 2^{-b}$ のパラメータ b (b が大きいほど高精度)，縦軸が処理時間である．図が示すように，$\leq 10^6$ 程度の範囲では，b が大きくなると一括写像 $k = 0$ は $b = 20$ あたりから爆発的に処理時間が大きくなっていることがわかる．このように，b が大きい場合には秘密一括近似 ($k > 0$) は一括写像 $k = 0$ よりも有効である．

5.2 ニュートン法以外の方法との比較

ニュートン法以外のロジスティック回帰の学習方法と提案手法の比較を行った．データセットには Wine Quality Data Set を用いた．[*1] 説明変数の次元数は $d = 12$，サンプル数は $n = 6497$ である．説明変数の値は予め属性ごとに $[-1, 1]$ に正規化を行い，小数点以下 20 ビットとした．

比較対象は，最急降下法と Nesterov の加速法とした．それぞれ，学習率は 0.1 と 0.01 の 2 パターンを試した．Nesterov の加速法のモーメンタムの減衰率は 0.9 とした．これらを stochastic gradient descent でバッチ数 128 で実行した．各種法は収束判定が成功したら終了とし，最大でも反復回数は 500 回とした．結果を表 2 に示す．重み i は学習されたパラメータのうちの最初の 3 つのパラメータである．正解の重みは R を用いて計算した結果とした．いずれも同程度の実行時間であったが，提案手法は他の手法に比べて大きく正解から離れていた．打ち切り時も誤差の減少は続いていたため，これは収束速度の違いによると考えられる．

[*1] https://archive.ics.uci.edu/ml/datasets/wine+quality

5.3 人工データとの比較

スケーラビリティの確認のため，人工データを作成して入力の大きさを変えてロジスティック回帰の学習を行った．人工データは，あらかじめ各説明変数ごとに $[-1, 1]$ のランダムな重みを設定し，$[0, 1)$ でランダムに設定された各説明変数に上述の重みを使って計算された結果が正の場合は目的変数を 1 に，そうでない場合は 0 に設定した．ただし，確率 0.9 で目的変数の値を反転させた．入力は小数点以下 16 ビットとした．

説明変数の数を d，入力の数を n とするとき，$(n, d) \in \{(10, 10^3), (10, 10^4), (10, 10^5), (10, 10^6), (100, 10^3), (100, 10^4), (100, 10^5)\}$ のそれぞれに対して人工データを作成し，提案手法を適用した．実行時間と R の計算結果との比較を表 3 に示す．実行時間は入力により差があるもののいずれも 1 分以内であり，入力の表の要素数が 10^7 程度であれば十分に実用的であることが確認できた．また，誤差も最大で 0.0002 程度と小さかった．

6. おわりに

本研究では，秘密計算で小さい誤差で関数の近似とロジスティック回帰の学習を行う手法を提案した．提案した秘密一括近似は秘密計算では計算が難しい指数関数などの計算を表参照により効率よく実現する方法であり，従来は計算コストが爆発的に増加してしまっていたような許容される誤差が小さい場合にも効率よく計算できる．また，秘密一括近似を使ったロジスティック回帰の学習アルゴリズムは実際にいくつか入力例で高速かつ高精度にパラメータの学習ができていることが確認できた．

謝辞 本論文の実験にあたっては，竹之内大地様の多大なご支援を賜りました．ここに感謝します．

参考文献

[1] Yao, A. C.: How to Generate and Exchange Secrets (Extended Abstract), *FOCS*, pp. 162–167 (1986).

[2] Ben-David, A., Nisan, N. and Pinkas, B.: FairplayMP: a system for secure multi-party computation, *ACM Conference on Computer and Communications Security*

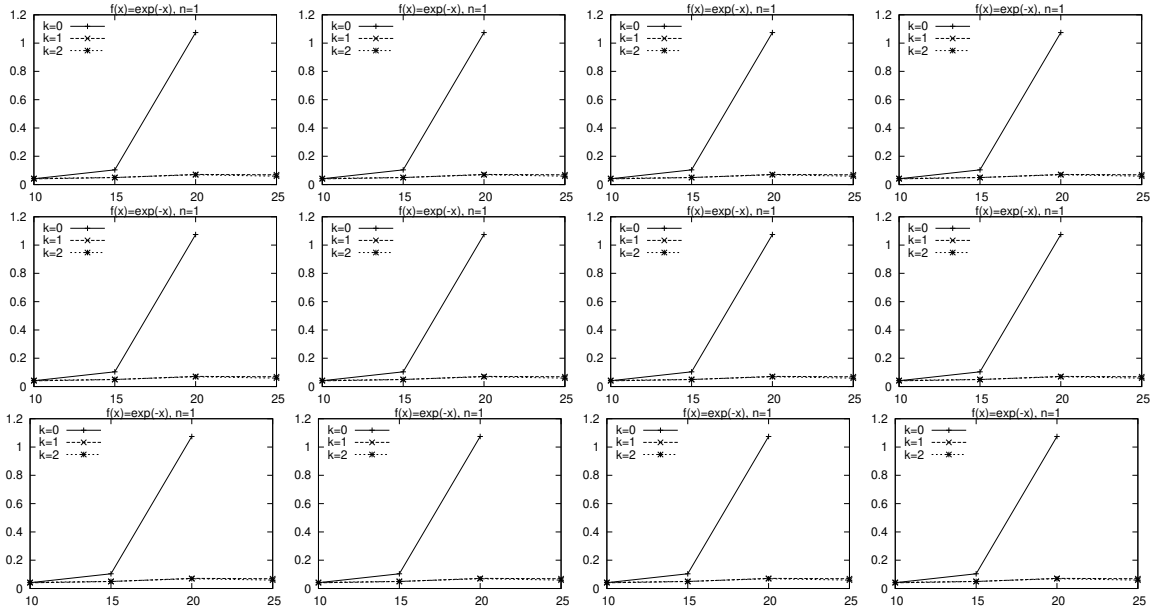

図 1 k 次の多項式による一括近似の処理時間の違い ($k = 0$ は従来手法，$k > 0$ は提案手法)．各関数，入力の数 n に対する k を変えた時の処理時間の違い．横軸は許容誤差 $\epsilon = 2^{-b}$ の b，縦軸は処理時間 (秒)．

表 2 Wine Quality Data Set に対する学習結果の比較．手法のカッコ内は学習率を表す．

手法	反復回数	時間 (s)	重みの最大誤差	重みの相関係数	重み 1	重み 2	重み 3
提案手法	9	129.849	0.08064	0.99999	-23.5650	-23.3508	2.1911
最急降下法 (0.1)	500(打ち切り)	136.407	89.06975	0.47481	-0.3198	-0.8865	0.4230
Nesterov の加速法 (0.1)	500(打ち切り)	141.400	88.27612	0.45452	-3.4434	-5.7254	1.7416
最急降下法 (0.01)	500(打ち切り)	138.249	89.27411	0.43055	0.2022	-0.0056	0.1291
Nesterov の加速法 (0.01)	500(打ち切り)	143.719	89.07218	0.47531	-0.3077	-0.8682	0.4180
R による計算結果					-23.5797	-23.3558	2.1894

表 3 人工データに対する実行結果．実行時間の"-" は未測定を表す．

d	n	重みの最大誤差	重みの相関係数	実行時間 [s]
10	1000	3.492^{-5}	0.9999999993865	11.114
10	10000	5.382^{-6}	0.9999999999939	-
10	100000	5.561^{-6}	0.9999999999809	-
10	1000000	9.372^{-7}	0.9999999999992	38.691
100	1000	0.0001958	0.9999999966676	11.140
100	10000	9.282^{-6}	0.9999999999718	-
100	100000	1.659^{-5}	0.9999999996981	18.390

(Ning, P., Syverson, P. F. and Jha, S., eds.), ACM, pp. 257–266 (2008).

[3] Goldreich, O., Micali, S. and Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority, *STOC*, ACM, pp. 218–229 (1987).

[4] Ben-Or, M., Goldwasser, S. and Wigderson, A.: Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract), *STOC* (Simon, J., ed.), ACM, pp. 1–10 (1988).

[5] Aggarwal, G., Mishra, N. and Pinkas, B.: Secure Computation of the k th-Ranked Element, *EUROCRYPT*, pp. 40–55 (2004).

[6] Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J. B. and Toft, T.: Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation, *TCC*, pp. 285–304 (2006).

[7] Catrina, O. and Saxena, A.: Secure Computation with Fixed-Point Numbers, *Financial Cryptography* (Sion, R., ed.), LNCS, Vol. 6052, Springer, pp. 35–50 (2010).

[8] Aliasgari, M., Blanton, M., Zhang, Y. and Steele, A.: Secure Computation on Floating Point Numbers, *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*, The Internet Society (2013).

[9] Xie, W., Wang, Y., Boker, S. M. and Brown, D. E.: Privlogit: Efficient privacy-preserving logistic regression by tailoring numerical optimizers, *arXiv preprint arXiv:1611.01170* (2016).

[10] Mohassel, P. and Zhang, Y.: SecureML: A System for Scalable Privacy-Preserving Machine Learning, *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, IEEE Computer Society, pp. 19–38 (online), DOI: 10.1109/SP.2017.12 (2017).

[11] 三品気吹，濱田浩気，五十嵐大，　菊池亮：秘密計算によるロジスティック回帰は本当に使えるか？，*SCIS* (2019).

[12] 濱田浩気，五十嵐大，千田浩司：秘匿計算上の一括写像アルゴリズム，電子情報通信学会論文誌. A, 基礎・境界, Vol. 96, No. 4, pp. 157–165 (2013).