Computer Security Symposium 2020
26 - 29 October 2020

秘密計算による効率的な決定木学習アルゴリズム

濱田 浩気[1,a)]

概要：プライバシーを保護しながら決定木を学習する研究は古く，2000 年には Lindell と Pinkas による手法が提案されているが，これまでの方式は学習した木の構造を隠せていなかった．本研究では，木の構造をも秘匿したまま決定木の学習を秘密計算で効率よく行う手法を提案する．

キーワード：秘密計算，決定木

An efficient decision tree training algorithm for secure multiparty computation

Koki Hamada[1,a)]

Abstract: There are some studies on learning decision trees while preserving privacy, but previous methods could not hide the structure of the learned trees. In this study, we propose a method for efficiently learning decision trees by for secure multi-party computation that can keep the tree structure secret.

Keywords: Secure multi-party computation, decision tree

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，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 [15] 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，桐渕らによる MEVAL？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 [4] や Ben-Or ら [2] 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．基本演算の効率化は定数倍の改善に過ぎず，多量のデータを扱うためには，計算量の悪化の解決が不可欠である．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

[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

んでおり，Aggarwal らによる k 番目の要素の選択 [1] や Damgård らによるビット分解 [3] Nishide と Ohta による比較 [12]，Ning と Xu による剰余計算 [11]，Goodrich によるソート [5] などが提案されている．

さらに，秘密計算上で機械学習も盛んに行われるようになってきており [10]，ニューラルネットワークによる予測のみならず，学習までもが秘密計算で実現されてきている．Mohassel と Zhang による SecureML [10] は semi-honest な攻撃者を仮定した 2 パーティ秘密計算でニューラルネットワークの学習を MNIST[8] と呼ばれるデータセットに対して実現した．Wagh らは SecureNN と呼ばれる 3 パーティ秘密計算システム [14]，Mohassel と Rindal は ABY3 と呼ばれる 3 パーティ秘密計算システム [9]，Rachuri と Suresh は TRIDENT と呼ばれる 4 パーティ秘密計算システム [13] を提案しており，MNIST の学習を行っている．SecureNN，ABY3，TRIDENT は semi-honest な攻撃者だけでなく，active な攻撃者を仮定した設定での実現方法も示されている．

1.1 貢献

本稿では，秘密計算で決定木の学習を行うアルゴリズムを提案する．決定木の学習は Lindell と Pinkas [7] により研究が行われるなどいくつかの手法が提案されている [6] が，これまでの方式では学習した木の構造を隠せていなかった．本研究では，木の構造をも秘匿したまま決定木の学習を秘密計算で効率よく行う手法を提案する．

本稿で提案する方式は，マルチパーティの秘密計算上で決定木の節点の情報のみならず，構造をも秘匿したまま学習を行う．前提として，決定木は 2 分木とし，木の高さの最大値 $h_{\max}$ は公開されているものとする．教師データの入力数が n，属性数が m であるとき，提案手法は $O(h_{\max}\ell mn\log n)$ の通信量で決定木の学習を実現する．通常の方法で同様の秘匿性を達成する場合は $\Omega(2^{h_{\max}}\ell mn^2)$ ビットの通信量が必要となるが，提案手法はこれに比べて $h_{\max}$ と n に関してそれぞれ指数的に効率よく実現している．

2. 準備

2.1 決定木

決定木は，データのある属性に対する知識を，木構造によるルールの組み合わせで表現したものである．目的変数と呼ぶ属性と，説明変数と呼ぶ属性からなる．決定木は説明変数の属性値を入力とし，目的変数の予測値を出力する．決定木は各節点に，例えば「年齢が 30 未満」などの説明変数に関する分割のルールを持つ木構造として表現される．また，決定木の葉 (終端の節点) には目的変数の属性値が割り当てられる．

決定木は説明変数を受け取ると，まず最初に根の節点のルールの判定を行う．判定結果に従い，子の節点のいずれかに進む．その後再帰的に各節点で判定を続け，最後に到達した葉に割り当てられた属性値が目的変数の予測値として出力される．

2.1.1 決定木の学習アルゴリズム

説明変数と目的変数からなるデータの集合から決定木を学習するアルゴリズムとして，CART，ID3，C4.5 などが提案されている．これらは細部で異なるが，いずれも以下の流れで学習を行う．

入力は，説明変数の行列と目的変数のベクトルからなる．出力は，根から葉へ向かう有向グラフである木である．木の各節点には条件が設定され，葉には目的変数の属性値が設定される．

目的変数の各属性はカテゴリ値または数値のいずれかをとる．説明変数はカテゴリ値とする．説明変数が数値の場合は回帰木と呼ばれるが，本稿では対象外とする．

2.1.1.1 制限

木は 3 以上への分岐を許す手法もある．本稿では 2 分木に限定するが，効率を犠牲にして拡張することが可能である．

2.1.1.2 アルゴリズム

決定木は，教師データを根の節点から葉の節点へとある目的関数を最大化するように貪欲的に再帰的に分割することで学習される．

入力:データ集合 $Q = (X, y)$

出力:木

(1) y がすべて同じであれば葉を作って出力して終了．葉の表現するクラスは y で最も多く現れる要素とする．

(2) 節点 v を作る．

(3) 適用可能なルール $r_1, r_2, \ldots$ を列挙する．

(4) 目的関数により各分割ルール r_i の評価値 s_i を計算する．

(5) $\{r_i\}$ から最大の評価値を出すルール r^* を求める

(6) r^* に基づいて分類した各データを $(X_1, y_1), (X_2, y_2), \ldots, (X_m, y_m)$ とする．

(7) 各 (X_j, y_j) について，再帰的に実行し，v からその根節点 v_j に枝を張る．

(8) v を出力して終了する．

2.1.1.3 分割ルール

分割のルールは説明変数を使ったあらゆる処理を使うことができるが，大小比較や集合に含まれるかどうかの判定など，単純なものを使うことが多い．本稿では，カテゴリ値に対しては $x \in X$ かどうかのテスト，数値に対しては $x \leq C$ かどうかのテストを扱うものとする．

2.1.1.4 純度の指標

分割の良し悪しを測るため，各データ集合 Q があいまいであるかどうかの純度の指標 $H(\cdot)$ を用いる．よく使われる指標には，gini 係数や，エントロピーなどがある．

データ集合 Q のうち，目的変数の値が k であるものの集合を Q_k とする．Q を入力とする節点におけるクラス k の割合を以下のように定義して，

$$p_k := \frac{|Q_k|}{|Q|}$$

本稿では純度の指標としてエントロピー

$$H(Q) = -\sum_k p_k \log_2 p_k$$

を用いる．

2.1.1.5 目的関数

各分割ルール θ の良し悪しは目的関数により評価される．よく利用される評価関数には，相互情報量，ゲイン率などがある．

データ集合 Q をあるルール θ で $Q(\theta,0), Q(\theta,1)$ の 2 つのデータ集合に分割するものとする．このとき，

$$p_i(Q,\theta) := \frac{|Q(\theta,i)|}{|Q|}$$

$$G(Q,\theta) := \sum_i p_i(Q,\theta) H(Q(\theta,i))$$

$$\mathsf{Gain}(Q,\theta) := H(Q) - G(Q,\theta)$$

$$\mathsf{SplitInfo}(Q,\theta) := -\sum_i p_i(Q,\theta) \log_2 p_i(Q,\theta)$$

$$\mathsf{GainRatio}(Q,\theta) := \frac{\mathsf{Gain}(Q,\theta)}{\mathsf{SplitInfo}(Q,\theta)}$$

により定められる GainRatio() はゲイン率と呼ばれる．本稿では，ゲイン率を目的関数とする．

2.2 秘密計算

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を説明する．

2.2.1 秘匿化，復元

a を暗号化や秘密分散などの手段で秘匿した値を 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 $\vec{v} = (\vec{v}[1], \ldots, \vec{v}[n])$ の各要素を秘匿したベクトル $([\![\vec{v}[1]]\!], \ldots, [\![\vec{v}[n]]\!])$ を $[\![\vec{v}]\!]$ と表記する．

2.2.2 加減乗算

加算，減算，乗算の各演算は 2 つの暗号文 $[\![a]\!], [\![b]\!]$ 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，2 つの暗号文 $[\![a]\!]$，$[\![b]\!]$ 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 の暗号文 $[\![c]\!]$ を出力する．この演算の実行を

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

と記述する．

2.2.4 選択

選択の演算は，$c \in \{0,1\}$ である暗号文 $[\![c]\!]$ と 2 つの暗号文 $[\![a]\!]$，$[\![b]\!]$ を入力とし，d が $c=1$ ならば $d=a$，$c=0$ ならば $d=b$ であるような 1 つの暗号文 $[\![d]\!]$ を出力する．この演算の実行を

$$[\![d]\!] \leftarrow \mathsf{IfElse}([\![c]\!], [\![a]\!], [\![b]\!])$$

と記述する．$\mathsf{IfElse}([\![c]\!], [\![a]\!], [\![b]\!]) = [\![b]\!] + [\![c]\!]([\![a]\!] - [\![b]\!])$ により実現される．

2.2.5 秘密一括写像

秘密一括写像? 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. $\vec{s}[j] \le \vec{x}[i] < \vec{s}[j+1]$ 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 を適用した近似値を計算することができる．

2.2.6 集約関数

集約関数は，グループフラグと呼ぶ $\{0,1\}^n$ の値 $\vec{e}$ と対応した長さ n のベクトル $\vec{x}$ に関し，$\vec{e}$ で表現される各グループ内で対応する $\vec{x}$ の統計量を計算する演算である．

グループフラグ $\vec{e}$ に含まれる 1 はグループの終端を表す．例えば，入力

$$\vec{x} = (1,7,4,5,3,6,2)$$

に対応したグループフラグ

$$\vec{e} = (0,1,0,0,1,1,1)$$

が与えられているとき，$\vec{x}$ はそれぞれ $(1,7), (4,5,3), (6), (2)$ のグループに分けられていることを表すものとする．

集約関数の度数はグループの度数を，番号はグループ内での番号を，総和はグループ内での総和を，累積和はグループ内での累積和を，逆順累積和グループ内での逆順の

累積和を，最大値はグループ内での最大値を，最大値フラグはグループ内で最大の要素を示すフラグを，それぞれ計算する演算で，各演算の実行はそれぞれ以下のように表記される．

$\vec{y}_1 \leftarrow \mathsf{GroupByCount}(\vec{e})$,

$\vec{y}_2 \leftarrow \mathsf{GroupByIndex}(\vec{e})$,

$\vec{y}_3 \leftarrow \mathsf{GroupBySum}(\vec{e}, \vec{x})$,

$\vec{y}_4 \leftarrow \mathsf{GroupByPrefixSum}(\vec{e}, \vec{x})$,

$\vec{y}_5 \leftarrow \mathsf{GroupByPrefixSumR}(\vec{e}, \vec{x})$,

$\vec{y}_6 \leftarrow \mathsf{GroupByMax}(\vec{e}, \vec{x})$,

$\vec{y}_7 \leftarrow \mathsf{GroupByMaxFlag}(\vec{e}, \vec{x})$.

上述の $\vec{x}$ および $\vec{e}$ に対する出力はそれぞれ

$\vec{y}_1 = (2, 2, 3, 3, 3, 1, 1)$,

$\vec{y}_2 = (1, 2, 1, 2, 3, 1, 1)$,

$\vec{y}_3 = (8, 8, 12, 12, 12, 6, 2)$,

$\vec{y}_4 = (1, 8, 4, 9, 12, 6, 2)$,

$\vec{y}_5 = (8, 7, 12, 8, 3, 6, 2)$,

$\vec{y}_6 = (7, 7, 5, 5, 5, 6, 2)$,

$\vec{y}_7 = (0, 1, 0, 1, 0, 1, 1)$

のようになる．

3. 提案手法: 決定木の学習アルゴリズム

本節では，決定木の学習を入出力を秘匿しながら行うアルゴリズムを提案する．提案アルゴリズムは，あらかじめ公開されている決定木の高さの最大値 $h_{\max}$ を利用し，2 分木の各節点の有無や分割ルールを秘匿したまま決定木の学習を行う．

3.1 データ構造

入力は n 行 m 列の説明変数と呼ばれる行列と長さ n の目的変数と呼ばれるベクトルである．説明変数の i 行目と目的変数の i 番目の要素は対応しており，学習では説明変数の i 行目に対して目的変数の i 番目の要素を予測できる決定木を得ることを目指す．

説明変数の各列は 0 または 1 の値をとる 2 値のカテゴリ値または数値のいずれかに属する．目的変数は C 値のカテゴリ値であり，属性値は $[1, C]$ のいずれかの値をとる．

説明変数で 3 値以上の値をとるカテゴリ値を扱いたい場合は，以下の方法で埋め込むことができる．まず説明変数にその属性を仮想的に作成し，その後その属性に対して行う可能性のある分割それぞれについて，各レコードが 0 に分類されるか 1 に分類されるかを計算した値を 2 値のカテゴリ値として持つ属性を追加する．そして，仮想的に定めた属性を削除する．実行時には，追加された 2 値属性による分割が選ばれることはこのカテゴリ属性を分割したことと等価であるので，これにより 3 値以上のカテゴリ値も扱うことができた．

3.2 アイディア

提案手法は，通常の方法では $\Omega(2^{h_{\max}} \ell m n^2)$ ビットの通信量が必要であった学習のアルゴリズムを $O(h_{\max} \ell m n \log n)$ へと，$h_{\max}$ と n に関してそれぞれ指数的に効率よく実現する．アイディアは以下の通りである．

- 通常の方法では各節点でその節点に分類されたレコード数の情報を隠すため，すべての節点で毎回入力全体を使用したように装う必要がある．そのため，節点数である $\Omega(2^{h_{\max}})$ 回入力全体を読まなくてはならない．提案手法は同じ高さの節点での分割方法を一括で見つけるため，全体を見る回数を $O(h)$ 回と指数的に小さく実現できる．一括で計算を行うために集約関数演算と呼ばれる処理を用いる．
- 数値属性のしきい値を決めて評価値を計算する際には，着目する属性と目的変数の属性値全体 ($\Theta(\ell n)$ ビット) の情報を必要とする．そのため，通常の方法では，各節点である属性の $\Theta(n)$ 通りの分割パターンそれぞれの評価値を計算する際に $\Omega(\ell n^2)$ ビットの通信量が必要となる．
  提案手法は $\Theta(n)$ 通りの分割パターンの評価値の計算の際に再利用可能な値があることに着目し，$\Theta(n)$ 通りの分割パターンの評価値の計算を $O(\ell n \log n)$ ビットの通信量で実現する．

3.3 ゲイン率の計算

各節点の分割ルールは，あらかじめ定められた目的関数をその節点で最大化するような分割ルールを選択することにより決められる．分割ルールの候補それぞれについて目的関数の値を計算する必要があるため，与えられた分割ルールに対して目的関数の値を効率よく計算できることは重要である．

本研究では，目的関数としてゲイン率を用いる．ゲイン率は 2.1.1 節で見たように，実際に分割を行った後の各目的変数の値の度数を求める入り組んだ計算をする必要がある．ここでは，秘密計算で複数の分割ルールに対するゲイン率の計算を一括で行えるようにゲイン率の計算方法を整理し単純化する．

ゲイン率の計算を単純化するために，ゲイン率では多くの割合が必要とされていることに注目する．割合は除算を必要とするためそのまま計算すると計算コストが大きくなるが，総数を掛けることで度数という計算しやすい統計量に変換することができる．この観察に基づき，以下のように $\mathsf{SplitInfo}, H, \mathsf{Gain}, G$ のそれぞれの関数の代わりに入力のデータ集合の大きさを乗じた $\mathsf{SplitInfo}^+, H^+, \mathsf{Gain}^+, G^+$ の

各関数を考える．

$$f(x) := x \log_2 x$$

を使うと，それぞれ以下のように整理できる．

$$\begin{aligned}
\mathsf{SplitInfo}^+(Q,\theta) &:= |Q|\,\mathsf{SplitInfo}(Q,\theta) \\
&= -\sum_i |Q(\theta,i)| \log_2(|Q(\theta,i)|/|Q|) \\
&= -\sum_i |Q(\theta,i)|(\log_2|Q(\theta,i)| - \log_2|Q|) \\
&= |Q|\log_2|Q| - \sum_i |Q(\theta,i)|\log_2|Q(\theta,i)| \\
&= f(|Q|) - \sum_i f(|Q(\theta,i)|)
\end{aligned}$$

$$\begin{aligned}
H^+(Q) &:= |Q|H(Q) \\
&= -|Q|\sum_k p_k \log_2 p_k \\
&= -\sum_k |Q_k|(\log_2|Q_k| - \log_2|Q|) \\
&= |Q|\log_2|Q| - \sum_k |Q_k|\log_2|Q_k| \\
&= f(|Q|) - \sum_k f(|Q_k|)
\end{aligned}$$

$$\begin{aligned}
G^+(Q,\theta) &:= |Q|\sum_i p_i(Q,\theta)H(Q(\theta,i)) \\
&= \sum_i |Q(\theta,i)|H(Q(\theta,i)) \\
&= \sum_i H^+(Q(\theta,i))
\end{aligned}$$

$$\begin{aligned}
\mathsf{Gain}^+(Q,\theta) &:= |Q|\,\mathsf{Gain}(Q,\theta) \\
&= |Q|H(Q) - |Q|G(Q,\theta) \\
&= H^+(Q) - G^+(Q,\theta)
\end{aligned}$$

これらはいずれも Q や Q のうち条件を満たすレコードの度数と $f(\cdot)$，加減算のみで構成される．GainRatio は

$$\mathsf{GainRatio}(Q,\theta) = \frac{|Q|\,\mathsf{Gain}(Q,\theta)}{|Q|\,\mathsf{SplitInfo}(Q,\theta)} = \frac{\mathsf{Gain}^+(Q,\theta)}{\mathsf{SplitInfo}^+(Q,\theta)}$$

であるから，データ集合 Q に対する分割ルール θ の GainRatio の分子，分母は結局

- Q のレコード数
- Q のうち目的変数の各クラスのレコード数
- Q を θ で分割した各データ集合のレコード数
- Q を θ で分割した各データ集合のうち目的変数の各クラスのレコード数

と，$f(\cdot)$，加減算により計算できることがわかる．

$f(\cdot)$ の入力は上述の 4 種類の度数のいずれかであるので必ず n 以下の整数である．よって，$f(\cdot)$ は大きさ $\Theta(n)$ の $[0,n] \ni x \mapsto x\log x$ を表す対応表を使った秘密一括写像を用いることで $\Theta(n)$ 回の $f(\cdot)$ の計算を $O(\ell n \log n)$ ビットの通信量で実現できる．

非負の分子と分母の対として与えられる 2 つの値 (a,b) と (c,d) の比較結果は ad と bc の比較結果と等しくなる．GainRatio の分子分母はいずれも非負であるので，GainRatio の比較を行う際には上述の方法で代用することで除算を回避する．

Algorithm 1 グループ内ソート

コスト: $O(\ell n \log n)$ com
Notation: $\sigma \leftarrow \mathsf{SortG}(\vec{e},\vec{a})$
Input: 最後フラグ $\vec{e}$，属性値 $\vec{a}$
Output: σ ($\sigma(\vec{a})$ が $\vec{e}$ のグループ内で昇順になった $\vec{a}$ と一致)
1: $\vec{d} \leftarrow \mathsf{Sum}(\vec{e}) - \mathsf{PrefixSumR}(\vec{e})$. // グループ番号に変換する．例えば，$(0,0,0,1,1,2,3,3,3,3) \leftarrow (0,0,1,0,1,1,0,0,0,1)$
2: $\sigma \leftarrow \mathsf{SortPerm}(\vec{d},\vec{a})$

3.4 分割テスト

Algorithm 2 分割テスト (ソート済み数値属性)

コスト: $O(\ell n \log n)$
Notation: $\vec{s},\vec{g},\vec{q} \leftarrow \mathsf{SplitNumericSorted}(\vec{x},\vec{y},\vec{e})$
Input: グループ内で整列済みの説明変数 $\vec{x}$，目的変数 $\vec{y}$，グループフラグ $\vec{e}$.
Output: 評価値 $\vec{s}$，分割フラグ $\vec{g}$，分割パラメータ $\vec{q}$
1: $\vec{y}_1,\ldots,\vec{y}_C \leftarrow \mathsf{ToHotVector}(\vec{y})$
2: **for** $i \in [1,C]$ **do**
3: $\quad \vec{c}_{i,*} \leftarrow \mathsf{GroupBySum}(\vec{e},\vec{y}_c)$ //各クラスの総数を計算
4: $\quad \vec{c}_{i,1} \leftarrow \mathsf{GroupByPrefixSum}(\vec{e},\vec{y}_c)$ // 1 の各クラスの数を計算
5: $\quad \vec{c}_{i,0} \leftarrow \vec{c}_{c,*} - \vec{c}_{c,1}$ // 0 の各クラスの数を計算
6: $\vec{c}_{*,*} \leftarrow \mathsf{GroupByCount}(\vec{e})$ // 総数
7: $\vec{c}_{*,1} \leftarrow \mathsf{GroupByIndex}(\vec{e})$ // 各分割の 1 数を計算
8: $\vec{c}_{*,0} \leftarrow \vec{c}_{*,*} - \vec{c}_{*,1}$ // 各分割の 0 数を計算
9: 3.3 節の方法で $\vec{c}_{\cdot,\cdot}$ からゲイン率 $(\vec{a},\vec{b})$ を計算
10: $\vec{z} \leftarrow \mathsf{OR}(\mathsf{EQ}(\vec{x},\mathsf{Shift}(\vec{x})),\vec{e})$ // 1 つ後と同じか末尾なら対象外にするための除外フラグ
11: $(\vec{a},\vec{b}) \leftarrow \mathsf{IfElse}(\vec{z},(0^n,1^n),(\vec{a},\vec{b}))$
12: $\vec{f} \leftarrow \mathsf{GroupByMaxFlag}(\vec{e},(\vec{a},\vec{b}))$ でゲイン率の最大値のフラグ
13: $\vec{f} \leftarrow \mathsf{IfElse}(\vec{z},0,\vec{f})$
14: $\vec{t} \leftarrow \vec{x} + \mathsf{Shift}(\vec{x})$ // しきい値の 2 倍の値
15: $(\vec{a},\vec{b},\vec{t}) \leftarrow \mathsf{IfElse}(\vec{f},(\vec{a},\vec{b},\vec{t}),(0^n,0^n,0^n))$
16: $(\vec{a},\vec{b},\vec{t}) \leftarrow \mathsf{GroupBySum}(\vec{e},(\vec{a},\vec{b},\vec{t}))$
17: $\vec{f} \leftarrow \mathsf{GroupByPrefixSumR}(\vec{e},\vec{f})$
18: **return** $(\vec{a},\vec{b}),\vec{f},\vec{t}$

次に，分割テストについて説明する．提案手法では，グループフラグにより示された各グループごとに複数の分割ルールを試し，その中で最もゲイン率が高い分割ルールをそのグループの分割ルールとして採用する．分割テストは，指定された分割ルールに対応したゲイン率をグループごとに計算する処理である．アルゴリズムを Algorithm 6 に示す．

Algorithm 3 分割テスト (数値属性)

Notation: $\vec{s}, \vec{g}, \vec{q} \leftarrow \mathsf{SplitNumeric}(\vec{x}, \vec{y}, \vec{e})$
Input: 説明変数 $\vec{x}$, 目的変数 $\vec{y}$, グループフラグ $\vec{e}$
Output: 分割フラグ $\vec{g}$, 分割パラメータ $\vec{q}$

$\sigma \leftarrow \mathsf{SortG}(\vec{e}, \vec{x})$ // 段数多い
$\vec{y}', \vec{x}' \leftarrow \mathsf{Apply}(\sigma; \vec{x}, \vec{y})$
$\vec{s}', \vec{g}', \vec{q}' \leftarrow \mathsf{SplitNumericSorted}(\vec{x}', \vec{y}', \vec{e})$
$\vec{s}, \vec{g}, \vec{q} \leftarrow \mathsf{ApplyInv}(\sigma; \vec{s}', \vec{g}', \vec{q}')$ // 入力の並びに戻す
return $\vec{s}, \vec{g}, \vec{q}$

Algorithm 4 分割テスト (分割なし)

Notation: $\vec{s} \leftarrow \mathsf{SplitNOP}(\vec{x}, \vec{y}, \vec{e})$
Input: 説明変数 $\vec{x}$, 目的変数 $\vec{y}$, 最後フラグ $\vec{e}$
Output: スコア $\vec{s}$

$\vec{y}_1, \ldots, \vec{y}_C \leftarrow \mathsf{ToHotVector}(\vec{y})$
$\vec{a} \leftarrow \mathsf{GroupByCount}(\vec{e})$
for $c \in [1, C]$ **do**
  $\vec{b}_c \leftarrow \mathsf{GroupBySum}(\vec{e}, \vec{y}_i)$
  $\vec{c}_c \leftarrow \mathsf{EQ}(\vec{b}_c, \vec{a})$
$\vec{d} \leftarrow \vec{c}_1 + \cdots + \vec{c}_C$
return $(\vec{d}, 1^n - \vec{d})$ // グループごとに一定

Algorithm 5 分割テスト (2 値カテゴリ属性)

Notation: $\vec{s}, \vec{g}, \leftarrow \mathsf{SplitBinary}(\vec{x}, \vec{y}, \vec{e})$
Input: 説明変数 $\vec{x}$, 目的変数 $\vec{y}$, 最後フラグ $\vec{e}$
Output: 分割フラグ候補 $\vec{g}$

$\vec{y}_1, \ldots, \vec{y}_C \leftarrow \mathsf{ToHotVector}(\vec{y})$
for $i \in [1, C]$ **do**
  $\vec{c}_{i,*} \leftarrow \mathsf{GroupBySum}(\vec{e}, \vec{y}_c)$ // 各クラスの総数を計算
  $\vec{c}_{i,1} \leftarrow \mathsf{GroupBySum}(\vec{e}, \vec{y}_c \times \vec{x})$ // 1 の各クラスの数を計算
  $\vec{c}_{i,0} \leftarrow \vec{c}_{i,*} - \vec{c}_{i,1}$ // 0 の各クラスの数を計算
$\vec{c}_{*,*} \leftarrow \mathsf{GroupByCount}(\vec{e})$ // 総数
$\vec{c}_{*,1} \leftarrow \mathsf{GroupBySum}(\vec{e}, \vec{x})$ // 分割の 1 数を計算
$\vec{c}_{*,0} \leftarrow \vec{c}_{*,*} - \vec{c}_{*,1}$ //各分割の 0 数を計算
3.3 節の方法で $\vec{c}_{\cdot,\cdot}$ からゲイン率 $(\vec{a}, \vec{b})$ を計算
return $(\vec{a}, \vec{b}), \vec{x}$

Algorithm 6 分割テスト

Notation: $\vec{s}, \vec{g}, \vec{q} \leftarrow \mathsf{Split}(X, \vec{y}, \vec{e}, j)$
Input: 説明変数 X, 目的変数 $\vec{y}$, グループフラグ $\vec{e}$, ルール番号 j
Output: 評価値 $\vec{s}$, 分割フラグ $\vec{g}$, 分割パラメータ $\vec{q}$

if $j = 0$ **then**
  $\vec{s} \leftarrow \mathsf{SplitNOP}(X, \vec{y}, \vec{e})$
  $\vec{g} \leftarrow 0^n$
  $\vec{q}[i] = (0, j, 0)$ により $\vec{q}$ を作る // 種別 0: 分割なし
else if 属性 j が数値 **then**
  $\vec{s}, \vec{g}, \vec{q}' \leftarrow \mathsf{SplitNumeric}(X, \vec{y}, \vec{e})$
  $\vec{q}[i] = (1, j, \vec{q}'[i])$ により $\vec{q}$ を作る // 種別 1: 数値
else if 属性 j がバイナリ **then**
  $\vec{s}, \vec{g} \leftarrow \mathsf{SplitBinary}(X, \vec{y}, \vec{e})$
  $\vec{q}[i] = (2, j, 0)$ により $\vec{q}$ を作る // 種別 2: バイナリ
return $(\vec{s}, \vec{g}, \vec{q})$

分割テストはグループフラグと説明変数, 目的変数, ルール番号を入力として受取り，ルール番号により指定された分割ルールによる評価値，分割フラグ，分割パラメータを計算して出力する．評価値はゲイン率の値の分子と分母を表すベクトルの対である．分割フラグはその分割ルールを使用した場合に 0 か 1 かのどちらに分類されるかを示す．

Algorithm 6 は指定されたルール番号に従い，ルール番号が 0 の場合は分割を行わない可能性を試み，1 以上の場合は対応する属性による分割を試みる．Algorithm 5 は 2 値カテゴリ属性による分割による評価値の計算を，Algorithm 3 は数値属性による分割による評価値の計算を，Algorithm 4 は分割を行う必要の有無の判定を，それぞれ行う．

Algorithm 3 で行う数値属性の分割を行う際には，しきい値を決めて分割を行うため，説明変数を昇順に並べて隣接する要素間にしきい値を入れることにするとグループの大きさ w に対して最大 $w-1$ 通りの分割候補がある．これらの候補を一度にまとめて計算し，最大値を求めるため，Algorithm 3 ではまず Algorithm 1 により各グループ内で説明変数を昇順に並び替え，その後各グループ内ですべてのしきい値の候補を試して最大の値をとった分割を出力する．

説明変数を整列後の処理は Algorithm 2 で行う．ここでは 3.3 節の方法でまとめてゲイン率を計算するために集約関数を使ってゲイン率の計算に必要なレコード数，目的変数の各クラスのレコード数，分割した各データ集合のレコード数，分割した各データ集合のうち目的変数の各クラスのレコード数をすべてレコード内で計算している．

3.5 葉の計算

Algorithm 7 葉の計算

Notation: $\vec{l} \leftarrow \mathsf{Leaf}(\vec{y}, \vec{e})$
Input: 目的変数 $\vec{y}$, グループフラグ $\vec{e}$
Output: 予測結果 $\vec{l}$
コスト: $O(\ell n)$ com, $O(1)$ rnd (C は定数)

for $c \in [1, C]$ **do**
  $\vec{t} = \mathsf{EQ}(\vec{y}, c)$
  $\vec{u}_c = \mathsf{GroupBySum}(\vec{e}, \vec{t})$
$\vec{l}' = \mathsf{ArgMax}(\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_C)$
$\vec{l}[i] = (3, 0, \vec{l}'[i])$ により $\vec{l}$ を作る // 種別 3: 葉

決定木の終端の節点である葉では，分岐の代わりにその節点にたどり着いた入力に対して与える目的変数の予測値を保持する．予測値は学習の際に葉にたどり着いたレコードの目的変数の中で出現回数が最多の属性値が選ばれる．アルゴリズムを Algorithm 7 に示す．

3.6 学習アルゴリズム

Algorithm 8 は，$\vec{e}$ で指定される各グループ内で，$\vec{f}$ の各異なる値 (0 または 1) ごとに最後に出現する位置にだけ 1 を，その他に 0 を入れたベクトル $\vec{g}$ の暗号文を計算する．$\vec{g}$ はこの後 $\vec{f}$ により安定ソートした場合に作られる最大 2 個のグループの最後の要素の位置を指し示しており，$\vec{f}$ により安定ソートした後の層でのグループフラグとして使われる．

Algorithm 8 グループ内の各分類値の最後の出現フラグ

Notation: $\vec{g} \leftarrow \mathsf{IsLastElement}(\vec{e}, \vec{f})$
Input: グループフラグ $\vec{e}$，分類フラグ $\vec{f}$
Output: グループ内の各分類値の最後の出現を表すフラグ $\vec{g}$
コスト: $O(\ell n)$ com, $O(1)$ rnd
1: $\vec{b}_0 \leftarrow 1^n - \vec{f}$ // $\vec{f}[i] = 0$ ならば $\vec{b}_0[i] = 1$
2: $\vec{b}_1 \leftarrow \vec{f}$ // $\vec{f}[i] = 1$ ならば $\vec{b}_1[i] = 1$
3: **for** $i \in \{0, 1\}$ **do**
4: $\quad \vec{v} \leftarrow \mathsf{GroupByPrefixSumR}(\vec{e}, \vec{b}_i)$
5: $\quad \vec{g}_i \leftarrow \mathsf{EQ}(\vec{v}, 1)$
6: $\vec{g} \leftarrow \vec{b}_0 + \vec{b}_1$
7: **return** $\vec{g}$

提案する決定木学習アルゴリズムを Algorithm 9 に示す．Algorithm 9 は二分木の決定木を根から葉へと順に，同一の高さの節点を並列に構築していく．構築の過程で説明変数や目的変数は，これまでの分岐に従って，同じ節点に属するレコードが互いに隣接するように順序を入れ替えられる．そして，各高さでは現在の並びに従って各グループで選択した分割ルールに従った分類結果を計算し，0 または 1 の分類結果に関する安定ソートによりレコードの順序を入れ替える．

この入れ替え後の順序は，通常の決定木を図示したときの並びとは異なる．より最近の分岐の結果の値 (0 または 1) が同一であったレコードがより近くなるように並ぶ．

$$|1, 2, 3, 4, 5, 6, 7|$$

グループフラグが

$$|0, 0, 0, 0, 0, 0, 1|$$

であるような入力に対する根での分類結果が

$$|0, 1, 0, 0, 1, 1, 0|$$

であった場合には，次の高さでは分類結果による安定ソートが行われ，説明変数が

$$|1, 3, 4, 7|, 2, 5, 6|$$

に，グループフラグが

$$|0, 0, 0, 1|, 0, 0, 1|$$

になる．ここで次の分類結果が

$$|1, 1, 0, 1|, 1, 0, 1|$$

であった場合にはさらに次の高さではこの分類結果による安定ソートが行われ，説明変数が

$$|4|, 5|, 1, 3, 7|, 2, 6|$$

に，グループフラグが

$$|1|, 1|, 0, 0, 1|, 0, 1|$$

になる．ここで，縦棒はグループの境界を示している．1 回目と 2 回目の分類結果が $(0, 0)$ であった説明変数は 1,3,7 であり，これらは同じグループに入っている．また，1 回目と 2 回目の分類結果が $(0, 1)$ であった説明変数は 4 だけであり，これは単一要素のグループに入っている．1,3,7 と 4 は最初の分類結果は同じ 0 であったが，直近の 2 回目の分類結果が異なっていたため遠く配置されている．

Algorithm 9 は，このようにレコードの順序を変えながら，各高さでグループごとに目的関数であるゲイン率を最大化するような分割ルールを選択し，その選択結果にあたる分類結果に従ってレコードの並び替えを行うことを繰り返す．アルゴリズムは高さが $h_{\max}$ に達したところで分割ルールの選択を終え，葉での目的変数の予測値を計算する．最後に，これまでの各高さでの分類結果と葉で設定した予測値に従って木を構築して出力する．

Algorithm 9 決定木の学習

Input: 説明変数 X，目的変数 $\vec{y}$，木の高さの最大値 $h_{\max}$
Output: 決定木 T
1: $X_1 := X$, $\vec{y}_1 := \vec{y}$, $\vec{e}_1 := (0, 0, \ldots, 0, 1), \vec{d}_1 = 1^n$
2: **for** $i = 1$ **to** $h_{\max} - 1$ **do**
3: $\quad$ // X_i と $\vec{y}_i$ は $(\vec{f}_i, \vec{f}_{i-1}, \ldots, \vec{f}_1)$ でソート済み，$\vec{e}_i$ は X_i の各グループの末尾だけ 1
4: $\quad \vec{s}_{i,j}, \vec{g}_{i,j}, \vec{q}_{i,j} \leftarrow \mathsf{Split}(X_i, \vec{y}_i, \vec{e}_i)$ for $j \in [0, m]$
5: $\quad$ 各 $k \in [1, n]$ について，$\vec{s}_k := (\vec{s}_{i,1}[k], \ldots, \vec{s}_{i,m}[k])$，$\vec{g}_k := (\vec{g}_{i,1}[k], \ldots, \vec{g}_{i,m}[k])$，$\vec{q}_k := (\vec{q}_{i,n}[k], \ldots, \vec{q}_{i,m}[k])$ として，$\vec{f}_i[k], \vec{p}_i[k] \leftarrow \mathsf{MaxElement}(\vec{s}_k; \vec{g}_k, \vec{q}_k)$ により $\vec{f}_i$ と $\vec{p}_i$ を計算
6: $\quad \vec{u}_i \leftarrow 2\vec{d}_i + \vec{f}_i$
7: $\quad \vec{t}_i \leftarrow \mathsf{IsLastElement}(\vec{e}_i, \vec{f}_i)$
8: $\quad X_{i+1}, \vec{y}_{i+1}, \vec{e}_{i+1}, \vec{d}_{i+1} \leftarrow \mathsf{Sort}(\vec{f}_i; X_i, \vec{y}_i, \vec{t}_i, \vec{u}_i)$ // $\vec{f}_i$ に従って分割
9: $\vec{l} \leftarrow \mathsf{Leaf}(\vec{y}_{h_{\max}}, \vec{e}_{h_{\max}})$ // 葉
10: **for** $i = 1$ **to** $h_{\max} - 1$ **do**
11: $\quad (\vec{d}_i, \vec{p}_i) \leftarrow \mathsf{IfElse}(\vec{e}_i, (\vec{d}_i, \vec{p}_i), (0^n, 0^n))$
12: $\quad (\vec{d}_i, \vec{p}_i) \leftarrow \mathsf{Sort}(\vec{e}_i; (\vec{d}_i, \vec{p}_i))$
13: $\quad (\vec{d}_i, \vec{p}_i)$ のうち末尾 2^{i-1} 個を木 T の i 層目に設定
14: $(\vec{d}_{h_{\max}}, \vec{l}) \leftarrow \mathsf{IfElse}(\vec{e}_{h_{\max}}, (\vec{d}_{h_{\max}}, \vec{l}), (0^n, 0^n))$
15: $(\vec{d}_{h_{\max}}, \vec{l}) \leftarrow \mathsf{Sort}(\vec{e}_{h_{\max}}; (\vec{d}_{h_{\max}}, \vec{l}))$
16: $(\vec{d}_{h_{\max}}, \vec{l})$ のうち末尾 $2^{h_{\max}-1}$ 個を木 T の $h_{\max}$ 層目に設定
17: **return** T

3.7 木を用いた予測

提案アルゴリズム Algorithm 9 の出力 T は $h_{\max}$ 層からなる 2 分木を表す暗号文であり，第 i 層に対してはそれぞれ長さ 2^{i-1} のベクトル $\vec{d}_i$ と $\vec{p}_i$ が格納されている．それぞれの j 番目の要素は節点の番号 d と (その節点におけるルールの種別，属性番号，パラメータ) もしくは 0 と (0,0) のダミーとなっている．各節点での入力の評価はこれらの値を使って $O(\ell m)$ の通信量ででき，また分割後の子節点の番号は $2d, 2d + 1$ で計算可能であるので，例えば市川らの手法を使うことで秘密計算上で効率よく予測ができる．

3.8 効率

体の大きさを $\ell \geq \log n, \log m$ とする n は学習データ数，m は属性数とするとき，Algorithm 9 の通信量は全体で $O(h\ell mn \log n)$ である．目的変数のクラス数 C は定数とみなした．

4. おわりに

本稿では，秘密計算で決定木の学習を行うアルゴリズムを提案した．本稿で提案した方式は，マルチパーティの秘密計算上で決定木の節点の情報のみならず，構造をも秘匿したまま学習を行う．木の高さの最大値 $h_{\max}$ は公開されているものとし，教師データの入力数が n，属性数が m であるとき，体の大きさを ℓ として，提案手法は $O(h_{\max}\ell mn \log n)$ ビットの通信量で決定木の学習を実現する．通常の方法で同様の秘匿性を達成する場合は $\Omega(2^{h_{\max}}\ell mn^2)$ ビットの通信量が必要となるが，提案手法はこれに比べて $h_{\max}$ と n に関してそれぞれ指数的に効率よく実現した．

参考文献

[1] Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the k th-ranked element. In *EUROCRYPT*, pp. 40–55, 2004.

[2] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *STOC*, pp. 1–10. ACM, 1988.

[3] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC*, pp. 285–304, 2006.

[4] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pp. 218–229. ACM, 1987.

[5] Michael T. Goodrich. Randomized shellsort: A simple oblivious sorting algorithm. In *SODA*, pp. 1262–1277, 2010.

[6] Ye Li, Zoe L Jiang, Lin Yao, Xuan Wang, Siu-Ming Yiu, and Zhengan Huang. Outsourced privacy-preserving c4. 5 decision tree algorithm over horizontally and vertically partitioned dataset among multiple parties. *Cluster Computing*, Vol. 22, No. 1, pp. 1581–1593, 2019.

[7] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In Mihir Bellare, editor, *CRYPTO*, Vol. 1880 of *LNCS*, pp. 36–54. Springer, 2000.

[8] MNIST database. `http://yann.lecun.com/exdb/mnist/`.

[9] Payman Mohassel and Peter Rindal. ABY3: a mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 35–52. ACM, 2018.

[10] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pp. 19–38. IEEE Computer Society, 2017.

[11] Chao Ning and Qiuliang Xu. Multiparty computation for modulo reduction without bit-decomposition and a generalization to bit-decomposition. In *ASIACRYPT*, pp. 483–500, 2010.

[12] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *PKC*, pp. 343–360, 2007.

[13] Rahul Rachuri and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. *arXiv preprint arXiv:1912.02631*, 2019.

[14] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, Vol. 1, p. 24, 2019.

[15] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pp. 162–167, 1986.