Computer Security Symposium 2021
26 - 29 October 2021

秘密計算による決定木学習アルゴリズムの改良

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．近年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が提案されたが，平文の学習を理想的に秘密計算で実現できた場合と比較すると通信量に開きがあった．本稿では，より小さい通信量で決定木の学習を行うアルゴリズムを提案する．提案手法は平文でのアルゴリズムで単位演算に体の要素を 1 つ送るのに等しい通信量がかかる想定の理想的な通信量と漸近的に等しい通信量を達成した．

キーワード：秘密計算，決定木

An improved 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 an improved 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 [5] 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．

近年では，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 [4]，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．

濱田は木の情報を隠したまま決定木の学習・推定を行う，方法を提案した [6]．濱田の方法は説明変数が数値の場合の通信量が $\Omega(h\ell mn \log n)$ であった．ここで n は教師データのサンプル数，m は教師データの説明変数の属性数，h は学習する木の高さ，ℓ は秘密計算の体のビット長である．一方，平分の場合の決定木の効率的な学習の計算方法として，最初に一度だけ説明変数を属性ごとにソートし，その後は分割統治を行う学習アルゴリズムを考えると，$\Theta(mn(h+\log n))$ の計算量で学習ができる．秘密計算の乗算の通信量が $\Theta(\ell)$ であることを考えると，濱田の方法は平文に比べて $\Theta(h)$ 倍程度のオーバーヘッドがあると言える．

1.1 貢献

本稿では，説明変数が数値の場合の決定木や回帰木の学習の通信量を改善する．濱田の方法 [6] のボトルネックは，学習途中の階層ごとの整列，グループごとの最大値の計算，であった．そのため，これら両方の通信量を改善し，平文の場合と比べて漸近的に損失の少ない通信量の学習アルゴリズムを作る．本研究の貢献は以下の通りである．

- 任意の結合的な二項演算による秘密計算集約関数の一般的な構成方法．グループ内での二項演算による累積

[1] NTT 社会情報研究所
[a)] koki.hamada.rb@hco.ntt.co.jp

の計算を，全体での別の二項演算による累積の計算に変換する方法を示す．これと効率的な並列累積計算アルゴリズムを組み合わせることにより，例えば [3] と最大値の計算を組み合わせると五十嵐ら [10] の方法で $\Omega(n \log n)$ 回 $\Omega(\log n)$ 段必要だったグループごとの最大値の計算が $O(n)$ 回 $O(\log n)$ 段でできるようになる．

- 説明変数が数値の場合の秘密計算向けのより効率的な決定木や回帰木の学習アルゴリズム．先行研究は階層ごとにソートをしていたが，提案アルゴリズムはソートを最初に 1 回だけ行い，その後は階層ごとにビットによるソートを行う．さらに算集約関数の一般的な構成方法を使うことで，通信量が $O(mn(\ell \log n + h \log n + h\ell))$，ラウンド数が $O(h\ell \log n)$ の決定木の学習方法を実現する．
- 秘密計算決定木 (回帰木) の学習・予測方法の実装評価．サンプル数 10^5，説明変数の属性数 10，木の高さ 5 の場合に学習が 25.7 秒，予測が 1.94 秒であり，実用的な速度で動作することを確認できた．

$h < \ell$ かつ $h < \log n$ のときは提案手法の通信量は $O(\ell mn \log n)$，濱田の方法 [6] は通信量は $\Omega(h\ell mn \log n)$，前述の平文でのアルゴリズムの計算量は $\Theta(mn \log n)$ であり，提案手法は [6] に比べて通信量を $\Theta(h)$ 倍程度改善できており，さらに平文でのアルゴリズムで単位演算に $\Theta(\ell)$ の通信量がかかった場合の通信量と漸近的に等しい通信量を達成している．

2. 準備

2.1 記法

ベクトル $\vec{v}$ の i 番目の要素を $\vec{v}[i]$ で参照する．すなわち，$\vec{v}$ が長さ n のベクトルであるとき，$\vec{v} = (\vec{v}[1], \vec{v}[2], \ldots, \vec{v}[n])$ である．論理演算においては，0 を偽，1 を真とみなすものとする．

2.2 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) には予測値として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．

2.2.1 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または数値のいずれかである．説明変数が数値の場合は特に回帰木と呼ばれることもある．

2.2.1.1 本稿での制限

本稿では，目的変数が数値かつ説明変数がすべて数値の場合の決定木 (回帰木) を扱う．また，木は 3 つ以上の子への分岐を許す手法もあるが，本稿では 2 分木に限定する．さらに，各節点での分割のルールは特定の属性値があるしきい値以下かどうかを判定するルールに限定する．

2.2.1.2 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:教師データ集合 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．葉の表現する予測値は y の平均値とする．
(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 $(X_1, y_1), (X_2, y_2)$ とする．
(7) 各 (X_j, y_j) $(j \in \{1, 2\})$ について，学習を再帰的に実行し，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．

2.2.1.3 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，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．

2.2.1.4 不純度

分割の良し悪しを測るため，各データ集合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，目的変数がカテゴリの場合は gini 係数や，エントロピー，目的変数が数値の場合には平均二乗誤差，などがある．

教師データのサンプル数を n，説明変数が $(y_1, y_2, \ldots, y_n)$，現在の節点に到達しているサンプルの添字の集合を $Q \subseteq [1, n]$ とする．このとき，Q により表されるデータ集合の不純度を $H(Q)$ と表記する．

本稿では，純度の指標として，$\mu(Q) := \frac{1}{|Q|} \sum_{i \in Q} y_i$，$H(Q) := \frac{1}{|Q|} \sum_{i \in Q} (y_i - \mu(Q))^2$ のように定義される平均二乗誤差 (MSE) による不純度を用いる．

2.2.1.5 目的関数

分割ルールの決定の際には，ある関数値を最大化 (または最小化) するような分割ルール θ を選択する．この関数を**分割の目的関数**と呼ぶことにする．分割の目的関数は，不純度の関数をを内部で使う．

ある分割ルール θ が Q を Q_0 と Q_1 s.t. $Q = Q_0 \cup Q_1$ and $Q_0 \cap Q_1 = \emptyset$ に分割するとする．このとき，よく用いられる目的関数としては，$G(Q, \theta) := \frac{|Q_0|}{|Q|} H(Q_0) + \frac{|Q_1|}{|Q|} H(Q_1)$ や $\mathsf{Gain}(Q, \theta) := H(Q) - G(Q, \theta)$ などがある．通常，$G(\cdot)$ を使う場合は $G(\cdot)$ を最小化する分割ルールを，$\mathsf{Gain}(\cdot)$ を使う場合は $\mathsf{Gain}(\cdot)$ を最大化する分割ルールが選択される．

本稿では，Gain$(\cdot)$ を目的関数とし，これを最大化する分割ルールを選択する．

2.3 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．各値は特に記載のない限り ℓ ビットの素数 p に関する素体 $\mathbb{Z}_p$ 上の値とする．

2.3.1 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．

2.3.2 四則演算

加算，減算，乗算，除算の各演算は 2 つの暗号文 $[\![a]\!], [\![b]\!]$ を入力とし，それぞれ $a+b$，$a-b$，ab，$\lfloor a/b \rfloor$ の計算結果 c_1，c_2，c_3，c_4 の暗号文 $[\![c_1]\!]$，$[\![c_2]\!]$，$[\![c_3]\!]$，$[\![c_4]\!]$ を計算する．これらの演算の実行をそれぞれ，$[\![c_1]\!] \leftarrow \mathsf{Add}([\![a]\!], [\![b]\!])$，$[\![c_2]\!] \leftarrow \mathsf{Sub}([\![a]\!], [\![b]\!])$，$[\![c_3]\!] \leftarrow \mathsf{Mul}([\![a]\!], [\![b]\!])$，$[\![c_4]\!] \leftarrow \mathsf{Div}([\![a]\!], [\![b]\!])$ と記述する．誤解を招く恐れのない場合は，$\mathsf{Add}([\![a]\!], [\![b]\!])$，$\mathsf{Sub}([\![a]\!], [\![b]\!])$，$\mathsf{Mul}([\![a]\!], [\![b]\!])$，$\mathsf{Div}([\![a]\!], [\![b]\!])$ をそれぞれ $[\![a]\!] + [\![b]\!]$，$[\![a]\!] - [\![b]\!]$，$[\![a]\!] \times [\![b]\!]$，$[\![a]\!] / [\![b]\!]$ と略記する．

また，行列やベクトルの加算，減算，乗算，除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，除算を要素ごとに行うものとし，同様に $+, -, \times, /$ を使って記述する．

2.3.3 論理和

論理和の演算は $a, b \in \{0, 1\}$ の暗号文 $[\![a]\!], [\![b]\!]$ を入力とし，$a \vee b$ の計算結果 c の暗号文 $[\![c]\!]$ を計算する．この演算の実行を $[\![c]\!] \leftarrow [\![a]\!]\ \mathsf{OR}\ [\![b]\!]$ と記述する．

2.3.4 比較

比較の各演算は，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_1 の暗号文 $[\![c_1]\!]$ を，$a < b$ ならば $c = 1$，そうでないならば $c = 0$ である c_2 の暗号文 $[\![c_2]\!]$ を出力する．これらの演算の実行を $[\![c_1]\!] \leftarrow ([\![a]\!] \stackrel{?}{=} [\![b]\!])$，$[\![c_2]\!] \leftarrow ([\![a]\!] \stackrel{?}{<} [\![b]\!])$ と記述する．Kikuchi らによって通信量 $O(\ell)$，ラウンド数 $O(\ell)$ の方法が提案されている [2].

2.3.5 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．

2.3.6 秘密一括写像

秘密一括写像 [8] の演算は，ソート済みの長さ m のベクトル $\vec{s}$ の暗号文 $[\![\vec{s}]\!]$，長さ m のベクトル $\vec{d}$ の暗号文 $[\![\vec{d}]\!]$，長さ 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. $\vec{s}[j] \le \vec{x}[i] < \vec{s}[j+1]$ を満たすような長さ 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．

2.3.7 最大値

最大値 [8] の演算は，比較対象の長さ n のベクトル $\vec{x}$ の暗号文 $[\![\vec{x}]\!]$，出力対象の長さ n のベクトル $\vec{y}$ の暗号文 $[\![\vec{y}]\!]$，を入力とし，$\vec{x}[i] = \max_{j \in [1,n]} \vec{x}[j]$ を満たす最小の i について，$z = \vec{y}[i]$ を満たす z の暗号文 $[\![z]\!]$ を出力する．この演算の実行を $[\![z]\!] \leftarrow \mathsf{VectMax}([\![\vec{x}]\!]; [\![\vec{y}]\!])$ と記述する．通信量 $O(nC_{\mathrm{cmp}})$，ラウンド数 $O(R_{\mathrm{cmp}} \log n)$ 段の方法が提案されている [7].

2.3.8 ソート

ソートの演算は，長さ n のベクトル $\vec{x}$ の暗号文 $[\![\vec{x}]\!]$ を入力とし，$\vec{x}$ の安定ソートを表す置換 π の暗号文 $[\![\pi]\!]$ を計算する．すなわち，π はどの $i, j \in [1, n]$ s.t. $i < j$ についても $\vec{x}[i] \le \vec{y}[j]$ ならば $\pi(i) < \pi(j)$ であるような $[1, n]$ から $[1, n]$ への全単射である．この演算の実行を $\langle\!\langle \pi \rangle\!\rangle \leftarrow \mathsf{Sort}([\![\vec{x}]\!])$ と記述する．通信量 $O(\ell n \log n)$，ラウンド数 $O(\ell)$ の手法が提案されている [1].

2.3.9 置換の適用

置換の適用の演算は，大きさ n の置換 π の暗号文 $\langle\!\langle \pi \rangle\!\rangle$ と長さ n のベクトル $\vec{x}$ の暗号文 $[\![\vec{x}]\!]$ を入力とし，各 $i \in [1, n]$ について $\vec{x}[i] = \vec{y}[\pi(i)]$ を満たす長さ n のベクトルの暗号文 $[\![\vec{y}]\!]$ を計算する．この演算の実行を $[\![\vec{y}]\!] \leftarrow \mathsf{Apply}(\langle\!\langle \pi \rangle\!\rangle, [\![\vec{x}]\!])$ と記述する．通信量 $O(n \log n + \ell n)$，ラウンド数 $O(1)$ の手法が提案されている [1].

2.3.10 置換の合成

置換の合成の演算は，大きさ n の置換 π の暗号文 $\langle\!\langle \pi \rangle\!\rangle$ と大きさ n の置換 σ の暗号文 $\langle\!\langle \sigma \rangle\!\rangle$ を入力とし，各 $i \in [1, n]$ について $\rho(i) = \pi(\sigma(i))$，すなわち $\rho = \pi \circ \sigma$ であるような置換 ρ の暗号文 $\langle\!\langle \rho \rangle\!\rangle$ を計算する．この演算の実行を $\langle\!\langle \rho \rangle\!\rangle \leftarrow \langle\!\langle \pi \rangle\!\rangle \circ \langle\!\langle \sigma \rangle\!\rangle$ と記述する．$\circ$ は右結合であるとする．すなわち，$a \circ b \circ c = a \circ (b \circ c)$ である．通信量 $O(n \log n)$，ラウンド数 $O(1)$ の手法が提案されている [1].

2.3.11 逆置換

逆置換の演算は，大きさ n の置換 π の暗号文 $\langle\!\langle \pi \rangle\!\rangle$ を入力とし，各 $i \in [1, n]$ について $\sigma(\pi(i)) = i$，すなわち $\sigma = \pi^{-1}$ であるような置換 σ の暗号文 $\langle\!\langle \sigma \rangle\!\rangle$ を計算する．この演算の実行を $\langle\!\langle \sigma \rangle\!\rangle \leftarrow \langle\!\langle \pi \rangle\!\rangle^{-1}$ と記述する．通信量 $O(n \log n)$，ラウンド数 $O(1)$ の手法が提案されている [1].

2.3.12 集約関数

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

グループフラグ $\vec{g}$ に含まれる 1 はグループの先頭を表す．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 $\vec{g} = (1, 0, 1, 0, 0, 1, 1)$ 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．

集約関数の総和はグループ内での $\vec{x}$ の総和を，累積和はグループ内での $\vec{x}$ の累積和を，最大値はグループ内で最大の $\vec{x}$ の要素 $\vec{x}[i]$ に対応した $\vec{z}$ の要素の値 $\vec{z}[i]$ 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．$[\![\vec{y}_3]\!] \leftarrow \mathsf{GroupBySum}([\![\vec{g}]\!], [\![\vec{x}]\!])$，$[\![\vec{y}_4]\!] \leftarrow \mathsf{GroupByPrefixSum}([\![\vec{g}]\!], [\![\vec{x}]\!])$，$[\![\vec{y}_6]\!] \leftarrow \mathsf{GroupByMax}([\![\vec{g}]\!], [\![\vec{x}]\!]; [\![\vec{z}]\!])$ 上述の $\vec{x}$, $\vec{g}$ および $\vec{z} = (6, 1, 2, 4, 2, 1, 3)$ に対する出力はそれぞれ $\vec{y}_3 = (8, 8, 12, 12, 12, 6, 2)$，$\vec{y}_4 = (1, 8, 4, 9, 12, 6, 2)$，$\vec{y}_6 = (1, 1, 4, 4, 4, 1, 3)$ のようになる．

GroupBySum，GroupByPrefixSum は [10] に [9] の手法を適用することで，$O(n \log n + \ell n)$ 通信量，$O(1)$ ラウンドで実現できる．GroupByMax は [10] の手法により通信量 $O(C_{\mathrm{cmp}} n \log n)$，ラウンド数 $O(R_{\mathrm{cmp}} \log n)$ で実現できる．

2.3.13 最初の要素の検出

最初の要素の検出の演算は，大きさ n のグループフラグ $\vec{g}$ の暗号文 $[\![\vec{g}]\!]$ と大きさ n のビットのベクトル $\vec{b} \in \{0,1\}^n$ の暗号文 $[\![\vec{b}]\!]$ を入力として，$\vec{b}[i]$ がグループ内で最初の 0 または 1 である場合に $\vec{z}[i] = 1$，そうでない場合に $\vec{z}[i] = 0$ であるようなベクトル $\vec{z}$ の暗号文 $[\![\vec{z}]\!]$ を計算する．この演算の実行を $[\![\vec{z}]\!] \leftarrow \mathsf{DetectFirst}([\![\vec{g}]\!], [\![\vec{b}]\!])$ と記述する．通信量 $O(n \log n + nC_{\mathrm{cmp}}(\log n))$，ラウンド数 $O(R_{\mathrm{cmp}}(\log n))$ の方法が提案されている [6]．

3. 秘密計算集約関数計算アルゴリズム

本節では，任意の結合的な二項演算 $\oplus$ に対し，$\oplus'$ に関する累積が $\oplus$ に関するグループごとの累積となるような別の結合的な二項演算 $\oplus'$ を作る方法を提案する．この方法を用いて作った二項演算 $\oplus'$ に関する累積を，二項演算の累積の計算を行う効率的なアルゴリズム A で計算することにより，A の二項演算の実行回数・段数と同じ回数・段数の $\oplus$ の計算で $\oplus$ に関するグループごとの累積が計算できるようになる．

例えば，Ladner と Fischer による二項演算の実行回数が $O(n)$，実行段数が $O(\log n)$ の累積計算のアルゴリズム [3] を用いる場合，$\oplus$ に関するグループごとの累積を $O(n)$ 回，$O(\log n)$ 段で計算することができる．これは五十嵐らの方法 [10] の $\Omega(n \log n)$ 回，$\Omega(\log n)$ 段と比べると，漸近的に同等の段数でありながら実行回数が漸近的に少ない．

3.1 準備

ある二項演算 $\oplus$ が任意の a, b, c に対して $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ を満たすとき，$\oplus$ は**結合的**であるという．例えば，実数の加算や乗算，2 つの数の最大値の計算は結合的であるが，減算や除算は結合的ではない．

長さ n のベクトル $\vec{x}$ の二項演算 $\oplus$ による**累積**とは，長さ n のベクトル $\vec{s}$ で，各 $i \in [1,n]$ が $\vec{s}[i] = \vec{x}[1] \oplus \vec{x}[2] \oplus \cdots \oplus \vec{x}[i]$ を満たすものである．

3.2 グループごとの累積を計算する二項演算

本節では，定義 1 で定義される変換により，任意の結合的な二項演算からグループごとの累積を計算する二項演算への変換できることを示す．

定義 1. $\oplus$ をある集合 S 上の二項演算とする．

$$(x, f_x) \oplus' (y, f_y) = (y \text{ if } f_y; x \oplus y \text{ otherwise}, f_x \vee f_y)$$

により定義される $S \times \{0,1\}$ 上の二項演算 $\oplus'$ を $\oplus$ の拡張二項演算と呼ぶ．

定義 1 から，直ちに以下が成り立つことがわかる．

命題 1. $\oplus'$ が $\oplus$ の拡張二項演算のとき，$(x, f_x) \oplus' (y, 1) = (y, 1)$.

命題 2. $\oplus'$ が $\oplus$ の拡張二項演算のとき，$(x, f_x) \oplus' (y, 0) = (x \oplus y, f_x)$.

定理 1. 二項演算 $\oplus$ が結合的であるならば，$\oplus$ の拡張二項演算 $\oplus'$ も結合的である．

Proof. $S' := S \times \{0,1\}$，$(a, f_a), (b, f_b), (c, f_c) \in S'$，$(d, f_d) := ((a, f_a) \oplus' (b, f_b)) \oplus' (c, f_c)$，$(e, f_e) := (a, f_a) \oplus' ((b, f_b) \oplus' (c, f_c))$ とする．常に $(d, f_d) = (e, f_e)$ が成り立つこと示す．

(1) $f_c = 1$ のとき．命題 1 より $(d, f_d) = ((a, f_a) \oplus' (b, f_b)) \oplus' (c, 1) = (c, 1)$，$(e, f_e) = (a, f_a) \oplus' ((b, f_b) \oplus' (c, 1)) = (a, f_a) \oplus' (c, 1) = (c, 1)$ であるので，$(d, f_d) = (e, f_e)$ である．

(2) $f_b = 1$ かつ $f_c = 0$ のとき．命題 1 と 2 より $(d, f_d) = ((a, f_a) \oplus' (b, 1)) \oplus' (c, 0) = (b, 1) \oplus' (c, 0) = (b \oplus c, 1)$，$(e, f_e) = (a, f_a) \oplus' ((b, 1) \oplus' (c, 0)) = (a, f_a) \oplus' (b \oplus c, 1) = (b \oplus c, 1)$ であるので，$(d, f_d) = (e, f_e)$ である．

(3) $f_b = 0$ かつ $f_c = 0$ のとき．命題 2 より $(d, f_d) = ((a, f_a) \oplus' (b, 0)) \oplus' (c, 0) = (a \oplus b, f_a) \oplus' (c, 0) = (a \oplus b \oplus c, f_a)$，$(e, f_e) = (a, f_a) \oplus' ((b, 0) \oplus' (c, 0)) = (a, f_a) \oplus' (b \oplus c, 0) = (a \oplus b \oplus c, f_a)$ であるので，$(d, f_d) = (e, f_e)$ である．

いずれの場合も $(d, f_d) = (e, f_e)$ であるので，$\oplus'$ は結合的である． □

定理 2. $\oplus'$ を結合的な二項演算 $\oplus$ の拡張二項演算，i を 1 以上の整数，$P_i = (X_i, G_i) = (x_1, g_1) \oplus' (x_2, g_2) \oplus' \cdots \oplus' (x_i, g_i)$，$g_1 = 1$ とする．$j \in [1, i]$ を $g_j = 1$ かつすべての $k \in [j+1, i]$ について $g_k = 0$ を満たす整数とする．このとき，$X_i = x_j \oplus x_{j+1} \oplus \cdots \oplus x_i$ が成り立つ．

Proof. 定理 1 より $\oplus'$ は結合的だから，命題 1 より $(x_1, g_1) \oplus' (x_2, g_2) \oplus' \cdots \oplus' (x_j, g_j) = (x_1, g_1) \oplus' (x_2, g_2) \oplus' \cdots \oplus' (x_j, 1) = ((x_1, g_1) \oplus' (x_2, g_2) \oplus' \cdots \oplus' (x_{j-1}, 1)) \oplus' (x_j, 1) = (x_j, 1)$. よって，命題 1 と 2 より $P_i = ((x_1, g_1) \oplus' (x_2, g_2) \oplus' \cdots \oplus' (x_j, g_j)) \oplus' (x_{j+1}, g_{j+1}) \oplus' \cdots \oplus' (x_i, g_i) = (x_j, 1) \oplus' (x_{j+1}, 0) \oplus' \cdots \oplus' (x_i, 0) = (x_j \oplus x_{j+1}, 1) \oplus' (x_{j+2}, 0) \oplus' \cdots \oplus' (x_i, 0) = \cdots = (x_j \oplus x_{j+1} \oplus \cdots \oplus x_i, 1)$. □

3.3 グループごとの累積の計算

アルゴリズム 1 結合的な演算によるグループごとの累積

Notation: $[\![\vec{p}]\!] \leftarrow \mathsf{GroupByPrefixX}(\oplus, [\![\vec{g}]\!], [\![\vec{x}]\!])$

Input: 結合的な二項演算 $\oplus$，$[\![\vec{g}]\!]$，$[\![\vec{x}]\!]$．ただし，$\vec{g} \in \{0,1\}^n$ は $\vec{g}[1] = 1$ を満たすグループを表すベクトル，$\vec{x}$ は累積の計算対象の長さ n のベクトル．

Output: $[\![\vec{p}]\!]$．ただし，$\vec{p}$ はすべての $i \in [1,n]$ について，$\vec{p}[i] = \vec{x}[j_i] \oplus \vec{x}[j_i + 1] \oplus \cdots \oplus \vec{x}[i]$ であるような長さ n のベクトル．ここで，j_i は i 以下で $\vec{g}[j_i] = 1$ かつすべての $k \in [j+1, i]$ について $\vec{g}[k] = 0$ である整数．

1: 二項演算 $\oplus'$ を以下のように定める．$([\![x]\!], [\![g_x]\!]) \oplus' ([\![y]\!], [\![g_y]\!]) = (\mathsf{IfElse}([\![g_y]\!], [\![y]\!], [\![x]\!] \oplus [\![y]\!]), [\![g_x]\!] \mathsf{\ OR\ } [\![g_y]\!])$

2: $[\![\overrightarrow{x_+}]\!][i] := ([\![\vec{x}]\!][i], [\![\vec{g}]\!][i])$ for $i \in [1,n]$

3: Ladner と Fischer のアルゴリズム [3] により，$[\![\overrightarrow{x_+}]\!]$ の $\oplus'$ による累積を並列に計算し，出力を $[\![\overrightarrow{p_+}]\!]$ とする．

4: 各 $i \in [1,n]$ について，$[\![\vec{p}]\!][i]$ を $[\![\overrightarrow{p_+}]\!][i]$ の第一要素の値とする．

前節で示した性質を用いて，グループを表すフラグ $\vec{g}$ と値の列 $\vec{x}$ に対し，$\vec{x}$ の結合的な二項演算 $\oplus$ に関する $\vec{g}$ で表されるグループ内の累積の計算を $\oplus$ の拡張二項演算 $\oplus'$ による列全体での累積の計算により実現する方法を具体的

に示す．

グループごとの累積を計算するアルゴリズムをアルゴリズム 1 に示す．このアルゴリズムでは，まず，$\oplus$ の拡張二項演算を作る (ステップ 1)．定理 1 より，$\oplus'$ もまた結合的である．よって，結合的な二項演算に関する累積を計算する Ladner と Fischer のアルゴリズム [3] を用いて計算される $\overrightarrow{x_+}$ は $\oplus'$ に関する累積である (ステップ 3)．定理 2 より，$\vec{p}[i]$ は $\vec{x}$ の $\vec{g}$ により表されるグループでの $\oplus$ に関する累積となっている (ステップ 4)．

結合的な二項演算 $\oplus$ の 1 回の実行の通信量とラウンド数を C_{op} と R_{op} とし，$\vec{x}$ の各要素のビット長を ℓ_x とすると，アルゴリズム 1 の通信量は $O(n(\ell_x + C_{\mathrm{op}}))$，ラウンド数は $O(R_{\mathrm{op}} \log n)$ である．

3.4 グループごとの総和の計算

アルゴリズム 2 結合的な演算によるグループごとの総和

Notation: $[\![\vec{s}]\!] \leftarrow \mathsf{GroupByX}(\oplus, [\![\vec{g}]\!], [\![\vec{x}]\!])$

Input: 結合的な二項演算 $\oplus$，$[\![\vec{g}]\!]$，$[\![\vec{x}]\!]$．ただし，$\vec{g} \in \{0,1\}^n$ は $\vec{g}[1] = 1$ を満たすグループを表すベクトル，$\vec{x}$ はグループごとの総和の計算対象の長さ n のベクトル．

Output: $[\![\vec{s}]\!]$．ただし，$\vec{p}$ は各 $\vec{p}[i]$ が $\vec{g}$ で表されるグループ内での $\vec{x}$ の $\oplus$ についての総和となっている長さ n のベクトル．

1: $[\![\vec{p}]\!] \leftarrow \mathsf{GroupByPrefixX}(\oplus, [\![\vec{g}]\!], [\![\vec{x}]\!])$
2: // グループ内の最後の要素以外を 0 にする
3: $[\![\vec{l}]\!][i] := \begin{cases} [\![\vec{p}]\!][i] \times [\![\vec{g}]\!][i+1] & \text{if } i < n, \\ [\![\vec{p}]\!][i] & \text{otherwise} \end{cases}$ for $i \in [1, n]$
4: // グループ内の最後の要素の値をグループ内にコピー
5: $[\![\vec{s}]\!] \leftarrow \mathsf{GroupBySum}([\![\vec{g}]\!], [\![\vec{l}]\!])$

効率的なグループごとの累積の計算を用いると，グループごとの総和の計算も効率的に実現できる．グループごとの累積では，各グループの最後の要素の値が各グループの総和となっていることを利用する．具体的なアルゴリズムをアルゴリズム 2 に示す．

アルゴリズム 2 では，まずアルゴリズム 1 を使ってグループごとの累積を計算する (ステップ 1)．次に，各グループの最後の要素以外を 0 で上書きする (ステップ 3)．これはグループを表すフラグ $[\![\vec{g}]\!]$ を用いて容易に計算できる．最後に，各グループでは最後の要素以外は 0 となっていることから，グループ内の通常の加算に関する総和を計算することにより，グループ内のすべての要素の値を最後の要素の値，すなわちグループ内での $\oplus$ に関する総和で上書きする (ステップ 5)．

結合的な二項演算 $\oplus$ の 1 回の実行の通信量とラウンド数を C_{op} と R_{op} とし，$\vec{x}$ の各要素のビット長を ℓ_x とすると，アルゴリズム 2 の通信量は $O(n(\ell_x + C_{\mathrm{op}} + \log n))$，ラウンド数は $O(R_{\mathrm{op}} \log n)$ である．

4. 属性ごとの分割テストと予測値の計算

本節では，回帰木の学習のうち目的関数と不純度に依存した処理である，(1) 属性ごとの分割テストと (2) 予測値の計算，の秘密計算向けのアルゴリズムを提案する．本稿では回帰木の学習の際に不純度として MSE を，目的関数として Gain をそれぞれ用いて Gain の最大化を行うことを仮定している．そのため，提案するアルゴリズムはこの仮定を置く場合に特化したものである．

4.1 属性ごとの分割テスト

アルゴリズム 3 目的関数が Gain，不純度が MSE のときの属性ごとの分割テスト

Notation: $[\![\vec{b}]\!], [\![\overrightarrow{t_g}]\!], [\![\overrightarrow{e_g}]\!] \leftarrow \mathsf{AttributewiseSplit}([\![\vec{g}]\!], [\![\vec{x}]\!], [\![\vec{y}]\!])$

Input: $[\![\vec{g}]\!]$, $[\![\vec{x}]\!]$: groupwise sorted, $[\![\vec{y}]\!]$: sorted by x,

Output: $[\![\vec{b}]\!] \in \{0,1\}^n$: 各レコードの分割結果, $[\![\overrightarrow{t_g}]\!]$: グループで最良の分割のしきい値の 2 倍の値, $[\![\overrightarrow{e_g}]\!]$: グループで最良の分割の評価値,

1: // レコードごとに直後で分割したときの $F(\cdot)$ の評価値を計算
2: $[\![s_1]\!] \leftarrow \mathsf{GroupByPrefixSum}([\![\vec{g}]\!], [\![\vec{y}]\!])$
3: $[\![s_0]\!] \leftarrow \mathsf{GroupBySum}([\![\vec{g}]\!], [\![\vec{y}]\!]) - [\![s_1]\!]$
4: $[\![n_1]\!] \leftarrow \mathsf{GroupByPrefixSum}([\![\vec{g}]\!], 1^n)$
5: $[\![n_0]\!] \leftarrow \mathsf{GroupBySum}([\![\vec{g}]\!], 1^n) - [\![n_1]\!]$
6: $[\![\vec{e}]\!] \leftarrow [\![s_0]\!] \times [\![s_0]\!] / [\![n_0]\!] + [\![s_1]\!] \times [\![s_1]\!] / [\![n_1]\!]$
7: // 直後で分割できない (グループの最後または次が同じ値である) 場合は評価値を小さい値 (-1) に設定
8: $[\![\vec{d}]\!][i] \leftarrow \begin{cases} [\![\vec{g}]\!][i+1] \ \mathsf{OR}\ ([\![\vec{x}]\!][i] \stackrel{?}{=} [\![\vec{x}]\!][i+1]) & \text{if } i < n, \\ 1 & \text{otherwise} \end{cases}$ for $i \in [1, n]$
9: $[\![\vec{e}]\!] \leftarrow \mathsf{IfElse}([\![\vec{d}]\!], (-1)^n, [\![\vec{e}]\!])$
10: // 各レコードごとに直後分割したときのしきい値の 2 倍を計算
11: $[\![\vec{t}]\!][i] \leftarrow \begin{cases} [\![\vec{x}]\!][i] + [\![\vec{x}]\!][i+1] & \text{if } i < n, \\ 0 & \text{otherwise} \end{cases}$ for $i \in [1, n]$
12: // グループごとに最大の評価値と対応するしきい値を計算
13: $[\![\overrightarrow{e_g}]\!] \leftarrow \mathsf{GroupByMax}([\![\vec{g}]\!], [\![\vec{e}]\!]; [\![\vec{e}]\!])$
14: $[\![\overrightarrow{t_g}]\!] \leftarrow \mathsf{GroupByMax}([\![\vec{g}]\!], [\![\vec{e}]\!]; [\![\vec{t}]\!])$
15: // レコードごとに分岐の結果を計算
16: $[\![\vec{b}]\!] \leftarrow (2 \times [\![\vec{x}]\!] \stackrel{?}{<} [\![\overrightarrow{t_g}]\!])$

属性ごとの分割テストは，特定の属性 j に関して，グループを表すフラグ $\vec{g} \in \{0,1\}^n$，グループごとに昇順で整列された属性 j の説明変数 $\vec{x}$，$\vec{x}$ と同じ順序で整列された目的変数 $\vec{y}$ を入力とし，グループごとの最良の分割のしきい値 $\overrightarrow{t_g}$ と評価値 $\overrightarrow{e_g}$，レコードごとの分岐 $\vec{b}$ を計算する処理である．ここで，分割ルールは [6] と同様に，説明変数の値 x としきい値 t に対して，$2x < t$ とする．

提案アルゴリズムは，本来の目的関数である $\mathsf{Gain}(Q, \theta)$ を最大化する θ を直接計算する代わりに，Gain よりも計算が容易で Gain と同じ最適解を持つ関数 $F(Q, \theta)$ に対し，$F(Q, \theta)$ を最大化する分割ルール θ^* に対応した評価値 $F(Q, \theta^*)$ と分割ルール θ^* のしきい値を計算する．

関数 $F(\cdot)$ を $F(Q, \theta) := (S(Q_0))^2/|Q_0| + (S(Q_1))^2/|Q_1|$ とする．不純度が MSE のとき，$S(Q) := \sum_{i \in Q} \vec{y}[i]$ とすると，$|Q|H(Q) = \sum_{i \in Q}(y_i^2 - 2\mu(Q)y_i + (\mu(Q))^2) = \sum_{i \in Q} y_i^2 - (S(Q))^2/|Q|$，$|Q|G(Q, \theta) = |Q_0|H(Q_0) + |Q_1|H(Q_1) = \sum_{i \in Q} y_i^2 - (S(Q_0))^2/|Q_0| - (S(Q_1))^2/|Q_1|$ が成り立つので，Gain は $\mathsf{Gain}(Q, \theta) = \frac{1}{|Q|}F(Q, \theta) - (\mu(Q))^2$ と表される．$|Q|$ と $\mu(Q)$ は θ に依らず一定であり，$|Q| > 0$ であるから，ある θ^* が $F(Q, \theta)$ を最大化するとき，θ^* は同時に $\mathsf{Gain}(Q, \theta)$ も最大化する．

属性ごとの分割テストを行うアルゴリズムをアルゴリズム 3 に示す．アルゴリズムはまずレコードごとに，その直後で分割をした場合の分割より前のグループ内の $\vec{y}$ の要素の総和を s_1，分割より後のグループ内の $\vec{y}$ の要素の総和を s_0，分割より前のグループ内の $\vec{y}$ の要素数を n_1，分割より後のグループ内の $\vec{y}$ の要素数を n_0，としてそれぞれ計算する (ステップ 2 から 5)．続いて，レコードごとに

その直後で分割した場合の $F(\cdot)$ の値を $\vec{t}$ として計算する (ステップ 6)．i 番目のレコードがグループの末尾の場合や $\vec{x}[i] = \vec{x}[i+1]$ の場合には直後で分割できないため，どの評価値よりも小さい値 -1 で $\vec{t}[i]$ を上書きする (ステップ 8 と 9)．さらに，i 番目のレコードの直後で分割した場合の分割ルールのしきい値を $\vec{x}[i] + \vec{x}[i+1]$ により計算する (ステップ 11)．最後に，グループごとに最大の評価値 $\vec{e}[i]$ を与える i に対応した評価値 $\vec{e}[i]$ としきい値 $\vec{t}[i]$ を $\vec{e_g}$ と $\vec{t_g}$ として計算し (ステップ 13 と 14)，レコードごとにしきい値 $\vec{t_g}$ による分岐の結果を $\vec{b}$ として出力する (ステップ 16)．

除算の通信量とラウンド数を C_{div} と R_{cmp}，比較の通信量とラウンド数を C_{cmp} と R_{div} とするとき，アルゴリズム 3 の通信量は $O(n(\log n + \ell + C_{\mathrm{div}} + C_{\mathrm{cmp}}))$，ラウンド数は $O(R_{\mathrm{cmp}} \log n + R_{\mathrm{div}})$ である．

4.2 予測値の計算

アルゴリズム 4 不純度が MSE の場合の予測値の計算

Notation: $[\![\mathrm{Label}^{(h)}]\!] \leftarrow \mathsf{ComputeLabel}([\![\vec{g}^{(h)}]\!], [\![\vec{y}^{(h)}]\!])$
Input: $[\![\vec{g}^{(h)}]\!]$, $[\![\vec{y}^{(h)}]\!]$
Output: $[\![\mathrm{Label}^{(h)}]\!]$
1: $[\![s_0]\!] \leftarrow \mathsf{GroupBySum}([\![\vec{g}^{(h)}]\!], [\![\vec{y}^{(h)}]\!])$
2: $[\![n_0]\!] \leftarrow \mathsf{GroupBySum}([\![\vec{g}^{(h)}]\!], 1^n)$
3: $[\![\mathrm{Label}^{(h)}]\!] \leftarrow [\![s_0]\!] / [\![n_0]\!]$

予測値の計算は，グループごとに最良の予測値を計算する処理である．不純度として用いる MSE は $\vec{y}$ のグループ内での平均値を予測値とする場合に最小となるため，ここでは予測値として $\vec{y}$ のグループ内での平均値を計算するアルゴリズムを作成する．$\vec{y}$ のグループ内での平均値は，グループ内での総和と要素数を計算し，総和を要素数で割ることで得られる．これは，GroupBySum と要素ごとの除算により，アルゴリズム 4 のように計算できる．

除算の通信量とラウンド数を C_{div}，R_{div} とするとき，アルゴリズム 4 の通信量は $O(n \log n + \ell n + n C_{\mathrm{div}})$，ラウンド数は $O(R_{\mathrm{div}})$ である．

5. 秘密計算決定木学習アルゴリズム

本節では，秘密計算で説明変数が数値の場合に木の構造を秘匿したまま決定木や回帰木の学習を行うアルゴリズムを提案する．

5.1 学習アルゴリズムの概要

5.1.1 濱田の方法 [6] と共通の基本構造

提案する学習アルゴリズムは濱田の方法 [6] の改良であり，m 属性 n サンプルの教師データを m 個の長さ n の説明変数のベクトル $\vec{x_j}$ ($j \in [1, m]$) の暗号文と長さ n の目的変数のベクトル $\vec{y}$ の暗号文として受け取り，高さ h (h は公開値) で各節点の分割ルールが「属性 j の値が (しきい値)/2 未満」である 2 分木を学習する．

各節点の情報を隠すため，[6] と同様に，明示的に教師データを分割しないようにしながら学習を行う (もし分割すると，各節点に分類されたサンプルの数が漏洩してしまう)．具体的には，教師データを分割する代わりに同じ節点 (グループと呼ぶ) に分類されたサンプルが互いに隣接するようにサンプル同士の並べ替えを行っていく．このとき，グループ間の境界を管理するために，グループフラグと呼ぶ長さ n のベクトル $\vec{g}$ を用いる．グループフラグ $\vec{g}$ の各要素は 0 または 1 であり，$\vec{g}[i] = 0$ のときは現在 i 番目のサンプルが $i-1$ 番目のサンプルと同じグループに分類されていることを，$\vec{g}[i] = 0$ のときは現在 i 番目のサンプルがグループ内で先頭であることを，それぞれ表す．

学習の流れも [6] と同様であり，グループフラグを利用して，以下を木の高さの順に繰り返す．

(1) 説明変数の各 j 番目の属性ごとに，各グループ内で $\vec{x_j}$ に関してソート済みの $\vec{x_j}$ と $\vec{y}$ を作る
(2) 説明変数の各 j 番目の属性ごとに，ソート済みの $\vec{x_j}$ と $\vec{y}$ から各グループ内で最良の分割ルールを選択する
(3) 説明変数の属性間で最良の分割ルールを選択する
(4) 前ステップで選択した分割ルールにより各グループを分割する
(5) 計算された分割ルールを現在の高さの節点の情報として出力する

5.1.2 改良のアイディア

提案する学習アルゴリズムは，濱田の方法 [6] のボトルネックとなっていた 2 箇所の通信量を改善することにより，全体の通信量を漸近的に小さくする．

1 点目は，前述のステップ (1) である．濱田の方法 [6] では，前述のステップ (1) で毎回明示的に $\vec{x_j}$ と $\vec{y}$ のソートを行っていた．本稿で提案するアルゴリズムは，ステップ (1) で $\vec{x_j}$ と $\vec{y}$ を明示的にソートする代わりに，前の高さでのステップ (4) の時点では分割前のグループで $\vec{x_j}$ と $\vec{y}$ がソート済みであったことを利用し，分割を安定に行うことによって明示的なソートを回避する．

2 点目は，ステップ (2) の最良の分割ルールの選択である．濱田の方法 [6] では，ステップ (2) では GroupByMax を用いて，グループ内で最良の分割を与えるしきい値を選択していた．本稿で提案するアルゴリズムも同様に GroupByMax を用いるが，3 節で示したように，GroupByMax 自体の通信量を改善した．

5.2 データ構造

学習アルゴリズムの入力は，説明変数と呼ばれる m 個の長さ n のベクトル $\vec{x_j}$ の暗号文 $[\![\vec{x_j}]\!]$ ($j \in [1, m]$) と，目的変数と呼ばれる長さ n のベクトル $\vec{y}$ の暗号文 $[\![\vec{y}]\!]$ である．各ベクトルの i 番目 ($i \in [1, n]$) の要素は対応しており，学習では説明変数の各 i 番目の要素の値 $\vec{x_1}[i], \vec{x_2}[i], \ldots, \vec{x_m}[i]$ から目的変数の i 番目の要素の値 $\vec{y}[i]$ を予測できるモデルの構築を目指す．

学習を行う木は高さ h の二分木で，h は公開値である．各節点は，根からの距離 (枝数) に応じて，距離が k の節点は k 段目の節点と呼ぶ．根は 0 段目であり，最大で h 段目までである．

k 段目 ($k \in [0, h)$) の各節点には，節点番号 j，属性番号 a，しきい値 t が割り当てられており，その節点における分割ルールが $\vec{x_a}$ の値が $t/2$ 未満か否かの判定であることを表す．節点番号は $[1, 2^k]$ の整数値であり，同じ段数の節点同士で互いに異なる．根の節点番号は 1 である．k 段目 ($k \in [0, h)$) の各節点には，高々 2 個の子節点があり，節点番号が j の節点の分割ルールが偽であった場合の行き先の子節点は $k+1$ 段目の節点番号が j の節点，真であった場合の行き先は $k+1$ 段目の節点番号が $j + 2^k$ の節点である．

表 1 回帰木の学習，予測の実行時間．n はサンプル数，m は説明変数の属性数，h は木の高さ

n	m	h	学習 [s]	予測 [s]
10	10	5	1.9712	0.3190
10^3	10	5	2.6892	0.3286
10^5	10	1	6.9917	0.5364
10^5	10	2	11.0750	0.8580
10^5	1	5	6.7317	1.3810
10^5	2	5	9.4616	1.5756
10^5	5	5	14.5912	1.6548
10^5	10	5	25.6586	1.9368

h 段目の節点は葉である．各葉には節点番号 j，予測値 p が割り当てられており，その節点に到達した場合の予測値が p であることを表す．

学習アルゴリズムの出力は，学習した木の情報を表す $h+1$ 組のベクトルの暗号文 $[\![L_0]\!], [\![L_1]\!], \ldots, [\![L_h]\!]$ で，各 $[\![L_k]\!] = ([\![\vec{n}_k]\!], [\![\vec{a}_k]\!], [\![\vec{t}_k]\!])$ $(k \in [0,h))$ は長さ $\min(n, 2^k)$ の 3 つのベクトル $\vec{n}_k, \vec{a}_k, \vec{t}_k$ の暗号文からなる 3 つ組，$[\![L_h]\!] = ([\![\vec{n}_h]\!], [\![\vec{p}_h]\!])$ は長さ $\min(n, 2^h)$ の 2 つのベクトル $\vec{n}_h, \vec{p}_h$ の暗号文からなる対である．各 $(\vec{n}_k[i], \vec{a}_k[i], \vec{t}_k[i])$ は k 段目の節点番号 $\vec{n}_k[i]$ の節点の属性番号が $\vec{a}_k[i]$，しきい値が $\vec{t}_k[i]$ であることを表す．各 $(\vec{n}_h[i], \vec{p}_h[i])$ は h 段目の節点番号 $\vec{n}_h[i]$ の節点の予測値が $\vec{a}_h[i]$ であることを表す．各 $\vec{n}_k$ $(k \in [0,h])$ は昇順で，$\vec{n}_k$ の各要素は $[1, 2^k+1]$ の値である．2^k+1 はその節点が存在しないことを示すダミーであり，$\vec{n}_k$ の末尾に並ぶ．$[1, 2^k]$ の各値は $\vec{n}_k$ に高々 1 つしか含まれない．

5.3 学習アルゴリズム

アルゴリズム 5 最良の分割ルールの選択

Notation: $[\![\vec{b}]\!], [\![\vec{a}]\!], [\![\vec{t}]\!] \leftarrow \mathsf{Split}((\langle\!\langle \sigma_j \rangle\!\rangle)_j, ([\![\vec{x_j}]\!])_j, [\![\vec{y}]\!], [\![\vec{g}]\!])$
Input: $(\langle\!\langle \sigma_j \rangle\!\rangle)_j$, $([\![\vec{x_j}]\!])_j$, $[\![\vec{y}]\!]$, $[\![\vec{g}]\!]$,
Output: スコアが最大だった属性の分岐 $[\![\vec{b}]\!]$，スコアが最大だった属性の番号 $[\![\vec{a}]\!]$，スコアが最大だった属性の分割ルールのしきい値 $[\![\vec{t}]\!]$

1: // $\vec{y}$ を $\vec{x_j}$ と同じ順序に並べ替えた $\vec{y_j}$ を計算
2: $[\![\vec{y_j}]\!] := \mathsf{Apply}(\langle\!\langle \sigma_j \rangle\!\rangle, [\![\vec{y}]\!])$ for $j \in [1,m]$
3: // グループ内で j 番目の属性に関してソート済みの $\vec{x_j}$ と $\vec{y_j}$ を使い j 番目の属性で最良の分割の分岐，しきい値，評価値を計算
4: $[\![\vec{c_j}]\!], [\![\vec{t_j}]\!], [\![\vec{s_j}]\!] \leftarrow \mathsf{AttributewiseSplit}([\![\vec{g}]\!], [\![\vec{x_j}]\!], [\![\vec{y_j}]\!])$ for $j \in [1,m]$
5: // 分岐を $\vec{y}$ の順序に並び替える．しきい値 $\vec{t_j}$，スコア $\vec{s_j}$ はグループ内で同一なので並べ替え不要
6: $[\![\vec{d_j}]\!] \leftarrow \mathsf{ApplyInv}(\langle\!\langle \sigma_j \rangle\!\rangle, [\![\vec{c_j}]\!])$ for $j \in [1,m]$
7: // 各行で評価値が最大の属性の分岐，属性番号，しきい値を選択
8: **for each** $i \in [1,m]$ **do**
9: // スコアが最大だった属性の分岐
10: $[\![\vec{b}]\!][i] \leftarrow \mathsf{VectMax}(([\![\vec{s_1}]\!][i], \ldots, [\![\vec{s_m}]\!][i]); ([\![\vec{d_1}]\!][i], \ldots, [\![\vec{d_m}]\!][i]))$
11: // スコアが最大だった属性の番号
12: $[\![\vec{a}]\!][i] \leftarrow \mathsf{VectMax}(([\![\vec{s_1}]\!][i], \ldots, [\![\vec{s_m}]\!][i]); (1, \ldots, m))$
13: // スコアが最大だった属性の分割ルールのしきい値
14: $[\![\vec{t}]\!][i] \leftarrow \mathsf{VectMax}(([\![\vec{s_1}]\!][i], \ldots, [\![\vec{s_m}]\!][i]); ([\![\vec{t_1}]\!][i], \ldots, [\![\vec{t_m}]\!][i]))$

提案する学習アルゴリズムをアルゴリズム 6 と部分アルゴリズムアルゴリズム 5 に示す．提案する学習アルゴリズムは属性ごとの最良の分割ルールの計算 (分割テスト) と予測値の計算とは独立に作られており，これらはそれぞれ AttributewiseSplit，ComputeLabel として呼び出す．目的関数が Gain かつ不純度が MSE の場合は 4 節で定義された AttributewiseSplit と ComputeLabel を使えばよいし，別の目的関数や不純度を使う場合には別途定義したものを呼び出せばよい．

アルゴリズム 6 説明変数が数値の決定木・回帰木の学習

Input: m 属性 n サンプルの説明変数の暗号文 $([\![\vec{x_j}]\!])_{j\in[1,m]}$，目的変数の暗号文 $[\![\vec{y}]\!]$
Output: $([\![L_k]\!])_{k\in[0,h]}$

1: // 初期化 (0 段目の計算)
2: $\langle\!\langle \sigma_j^{(0)} \rangle\!\rangle := \mathsf{Sort}([\![\vec{x_j}]\!])$ for $j \in [1,m]$
3: $[\![\vec{x}_j^{(0)}]\!] := \mathsf{Apply}(\langle\!\langle \sigma_j^{(0)} \rangle\!\rangle, [\![\vec{x_j}]\!])$
4: $[\![\vec{y}^{(0)}]\!] := [\![\vec{y}]\!]$
5: $[\![\vec{g}^{(0)}]\!] := (1, 0, 0, \ldots, 0)$ // すべてのサンプルは根に所属
6: $[\![\mathrm{NodeID}^{(0)}]\!] := 1^n$ // 根の節点番号は 1
7: **for** $k = 1$ **to** h **do**
8: // 最良の分割ルールの計算
9: $[\![\vec{b}^{(k-1)}]\!], [\![\mathrm{AttrID}^{(k-1)}]\!], [\![\mathrm{Threshold2}^{(k-1)}]\!] \leftarrow \mathsf{Split}(\langle\!\langle \sigma_j^{(k-1)} \rangle\!\rangle, [\![\vec{x}_j^{(k-1)}]\!], [\![\vec{y}^{(k-1)}]\!], [\![\vec{g}^{(k-1)}]\!])$
10: // $k-1$ 段目の出力
11: $[\![L_{k-1}]\!] \leftarrow ([\![\mathrm{NodeID}^{(k-1)}]\!], [\![\mathrm{AttrID}^{(k-1)}]\!], [\![\mathrm{Threshold2}^{(k-1)}]\!])$
12: // k 段目の計算のための置換の計算
13: $\langle\!\langle \pi^{(k)} \rangle\!\rangle := \mathsf{BinarySort}([\![\vec{b}^{(k-1)}]\!])$
14: $\langle\!\langle \rho_j^{(k)} \rangle\!\rangle := \mathsf{BinarySort}(\mathsf{Apply}(\langle\!\langle \sigma_j^{(k-1)} \rangle\!\rangle, [\![\vec{b}^{(k-1)}]\!]))$ for $j \in [1,m]$
15: // 更新 (k 段目の計算)
16: $\langle\!\langle \sigma_j^{(k)} \rangle\!\rangle := \langle\!\langle \rho_j^{(k)} \rangle\!\rangle \circ \langle\!\langle \sigma_j^{(k-1)} \rangle\!\rangle \circ \langle\!\langle \pi^{(k)} \rangle\!\rangle^{-1}$ for $j \in [1,m]$
17: $[\![\vec{x}_j^{(k)}]\!] := \mathsf{Apply}(\langle\!\langle \rho_j^{(k)} \rangle\!\rangle, [\![\vec{x}_j^{(k-1)}]\!])$ for $j \in [1,m]$
18: $[\![\vec{y}^{(k)}]\!] := \mathsf{Apply}(\langle\!\langle \pi^{(k)} \rangle\!\rangle, [\![\vec{y}^{(k-1)}]\!])$
19: $[\![\vec{g}^{(k)}]\!] := \mathsf{Apply}(\langle\!\langle \pi^{(k)} \rangle\!\rangle, \mathsf{DetectFirst}([\![\vec{g}^{(k-1)}]\!], [\![\vec{b}^{(k)}]\!]))$
20: $[\![\mathrm{NodeID}^{(k)}]\!] := \mathsf{Apply}(\langle\!\langle \pi^{(k)} \rangle\!\rangle, ([\![\vec{b}^{(k-1)}]\!] \times 2^{k-1} + [\![\mathrm{NodeID}^{(k-1)}]\!]))$
21: // h 段目の出力
22: $[\![\mathrm{Label}^{(h)}]\!] \leftarrow \mathsf{ComputeLabel}([\![\vec{g}^{(h)}]\!], [\![\vec{y}^{(h)}]\!])$
23: $[\![L_h]\!] \leftarrow ([\![\mathrm{NodeID}^{(h)}]\!], [\![\mathrm{Label}^{(h)}]\!])$

アルゴリズム 6 は 0 段目から順に，各 k 段目で以下の性質が保たれるように計算を行う．

- $\vec{g}^{(k)}$ が k 段目の分類でのグループフラグ
- $\mathrm{NodeID}^{(k)}$ が k 段目の分類での各サンプルの節点番号
- $\vec{y}^{(k)}$ が k 段目の分類に従って並べ替えられた $\vec{y}$
- $\vec{x}_j^{(k)}$ が k 段目の分類で各グループで $\vec{x_j}$ を昇順に並べ替えたベクトル
- $\sigma_j^{(k)}$ が $\vec{y}^{(k)}$ を $\vec{x}_j^{(k)}$ と同じ順序に並べ替える置換

これらはステップ 2 から 6 で，すべてのサンプルが単一の節点である根に分類された状態で初期化される．k 段目の計算はステップ 9 で得られた最良の分割ルールによる分類結果 $\vec{b}^{(k-1)}$ による分割を表す置換 $\pi^{(k)}$ を使ってステップ 16 から 20 で実行される．

$\vec{x}_j^{(k)}$ の計算の際には，$\vec{x}_j^{(k-1)}$ が $k-1$ 段目でソート済みであることを利用して，分類結果 $\vec{b}^{(k-1)}$ を $\vec{x}_j^{(k-1)}$ の順序に並び替え (ステップ 14)，並び替えた分類結果に従って $\vec{x}_j^{(k-1)}$ を安定に分類する (ステップ 17) ことで明示的なソートなしにグループ内でソートされた $\vec{x}_j^{(k)}$ を得ている．

最良の分割ルールの計算はステップ 9 から呼び出されたアルゴリズム 5 で行われる．アルゴリズム 5 では，まず $\vec{y}$

をソート済みの各 $\vec{x_j}$ と同じ順序に並び替え (ステップ 2)，AttributewiseSplit により j 番目の属性で最良の分割ルールを計算する (ステップ 4)．続いて，各属性ごとに得た分割ルールの情報を $\vec{y}$ と同じ順序に並び替え (ステップ 6)，サンプルごとに最良のルールを選択する (ステップ 10 から 14)．

5.4 コスト

比較の通信量を C_{cmp}，ラウンド数を R_{cmp}，除算の通信量を C_{div}，ラウンド数を R_{div} とすると，アルゴリズム 6 のコストは，通信量が $O(mn(\ell \log n + h \log n + h\ell + hC_{\mathrm{cmp}} + hC_{\mathrm{div}}))$，ラウンド数が $O(\ell + hR_{\mathrm{cmp}} \log n + hR_{\mathrm{div}})$ となる．

除算を??と同様に有理数として扱うこととし，比較は?の方法を使うこととして通信量 $O(\ell)$，ラウンド数 $O(\ell)$ とすると，アルゴリズム 6 のコストは通信量が $O(mn(\ell \log n + h \log n + h\ell))$，ラウンド数が $O(h\ell \log n)$ となる．

5.5 予測アルゴリズム

予測アルゴリズムとして，以下のアルゴリズムを考える．
入力:サンプル数 n，属性数 m の説明変数
出力:各サンプルに対する予測値
(1) 各サンプルの節点番号を 1 (根の節点番号) で初期化
(2) for $i = 0$ to $h - 1$
  (a) 各サンプルの節点番号に $[\![L_i]\!]$ による一括写像を適用し，サンプルごとにしきい値と属性番号を得る
  (b) 線形スキャンにより属性番号に対応する属性値を得る
  (c) 属性値としきい値の比較により分類結果を計算
  (d) 分類結果を使って次段の節点番号を計算
(3) 各サンプルの節点番号に $[\![L_h]\!]$ による一括写像を適用し，各サンプルの予測値を獲得

6. 実装評価

提案手法の実用性の確認のため，5 節で提案した学習アルゴリズム (アルゴリズム 6) と 5.5 項の予測アルゴリズムを実装して，実行時間を測定した．ただし，実用上効率的と考えられる実装として，除算は一括写像を用いた逆数計算による実装，集約関数最大値は [10] の実装を用いた．実装は MEVAL と呼ばれる 3 パーティ秘密計算システム?上で semi-honest な攻撃者を想定する設定で行った．

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

回帰木の目的関数は Gain，不純度は MSE とし，分割テストと予測値の計算は 4 節で提案したアルゴリズム 3 とアルゴリズム 4 をそれぞれ用いた．

教師データのサンプル数を n，説明変数の属性数を m，木の高さを h として，m，n，h を変えながらランダムな入力に対して学習，予測を行った．実行時間を各 3 回測定し，処理時間の平均値を求めた．測定結果を table 1 に示す．

また，提案手法を実装し，処理性能を測定した．サンプル数 10^5，説明変数の属性数 10，木の高さ 5 の場合に学習が 25.7 秒，予測が 1.94 秒であり，実用的な速度で動作することを確認できた．

7. おわりに

本稿では，秘密計算で決定木の学習を効率よく行うアルゴリズムを提案した．提案アルゴリズムは目的変数，説明変数がすべて数値，不純度が MSE，分割ルールの目的関数が Gain の場合に教師データのサンプル数を n，説明変数の属性数を m，木の高さを h として，通信量 $O(mn(\ell \log n + h \log n + h\ell))$，ラウンド数 $O(h\ell \log n)$ で決定木の学習を実現する．

$h < \ell$ かつ $h < \log n$ のときは提案手法の通信量は $O(\ell mn \log n)$，濱田の方法 [6] は通信量は $\Omega(h\ell mn \log n)$，前述の平文でのアルゴリズムの計算量は $\Theta(mn \log n)$ であり，提案手法は [6] に比べて通信量を $\Theta(h)$ 倍程度改善できており，さらに平文でのアルゴリズムで単位演算に $\Theta(\ell)$ の通信量がかかった場合の通信量と漸近的に等しい通信量を達成している．

参考文献

[1] Chida, K., Hamada, K., Ikarashi, D., Kikuchi, R., Kiribuchi, N. and Pinkas, B.: An Efficient Secure Three-Party Sorting Protocol with an Honest Majority, *IACR Cryptol. ePrint Arch.*, Vol. 2019, p. 695 (online), available from ⟨https://eprint.iacr.org/2019/695⟩ (2019).

[2] Kikuchi, R., Ikarashi, D., Matsuda, T., Hamada, K. and Chida, K.: Efficient Bit-Decomposition and Modulus-Conversion Protocols with an Honest Majority, *Information Security and Privacy - 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings* (Susilo, W. and Yang, G., eds.), Lecture Notes in Computer Science, Vol. 10946, Springer, pp. 64–82 (online), DOI: 10.1007/978-3-319-93638-3_5 (2018).

[3] Ladner, R. E. and Fischer, M. J.: Parallel Prefix Computation, *J. ACM*, Vol. 27, No. 4, pp. 831–838 (1980).

[4] 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).

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

[6] 濱田浩気：秘密計算による効率的な決定木学習アルゴリズム，*CSS* (2020).

[7] 濱田浩気，三品気吹，五十嵐大，菊池　亮：秘密計算による畳み込みニューラルネットワークの学習，*SCIS* (2020).

[8] 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（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009596932/⟩ (2013).

[9] 濱田浩気，五十嵐大，千田浩司：秘密計算一括写像計算の効率化，*SCIS* (2014).

[10] 五十嵐大，千田浩司，濱田浩気，高橋克巳：軽量検証可能 3 パーティ秘匿関数計算の効率化及びこれを用いたセキュアなデータベース処理，*SCIS*, pp. 1–8 (2011).