最適な順序付き決定木の高速発見とその文書分類への応用

長部　和仁† 　宇野　毅明†† 　有村　博紀†

† 北海道大学情報科学研究科　〒 060–0814 札幌市北区北 14 条西 9 丁目
†† 国立情報学研究所　〒 101-8430 東京都千代田区一ツ橋 2-1-2
E-mail: †{kz-osabe,arim}@ist.hokudai.ac.jp, ††uno@nii.jp

あらまし　本研究では，分類ラベル付きの二値データベースから，制約をみたし，データ中での分類誤差を最小化するような決定木を厳密に求める問題，およびそれを用いた文書分類問題を考察する．変数順序なし決定木の族 $\mathcal{DT}$ に対して，Nijssen らは，頻出アイテム集合マイニングを応用した網羅的探索を用いて，訓練集合中で識別精度最大の決定木を求めるアルゴリズム DL8 を与えた．しかし，DL8 は大きな時間計算量と空間計算量を要するため，大規模なデータに適用することは困難である．そこで本研究では，決定木の部分族として，決定木中のすべてのパスの順序を一意に固定した順序付き決定木 (ordered decision tree) の族 $\mathcal{OT}$ を導入し，この族 $\mathcal{OT}$ に対して制約を満たす最適決定木を効率良く計算するアルゴリズム ODT を提案する．また，ODT アルゴリズムに対し，データベース縮約による処理の効率化と，データの不均衡に応じた重み付けを導入する．実験では，Bag of Words 形式の文書集合とアイテム集合形式のデータセットの関係に着目し，ODT アルゴリズムを用いたラベル付き文書集合の分類を行うことで，分類過程の解釈可能性，実行効率，および不均衡データへの対応における提案手法の有効性を検証する．

キーワード　決定木, パターン列挙, 頻出アイテム集合, データマイニング, テキストマイニング, 文書分類

1. はじめに

近年，実務的な機械学習の領域において，識別精度の向上だけではなく，なぜそのような識別結果が出たのか，その過程を明確にすることが求められている．この点において，深層学習などの精度に優れる機械学習の手法の一部は，直感的に理解しにくい識別過程を持つため，そうした実務上の解釈可能性の要請にそぐわない場合がある．

一方で，解釈可能性の高さで注目されているのが，**決定木**の手法である．決定木は，木構造で識別過程を可視化できるため，あるデータがなぜそのラベルに分類されたかを人間が理解することが容易である．こうした点に着目し，Eto, Fujimaki らは 2014 年に異種混合学習と呼ばれる学習モデルを提案した [2]．これは，決定木をベースに識別器を構成することで人間可読な識別過程を担保し，決定木の各葉に対して回帰モデルを割り振ることで，解釈可能性と識別精度の両立を図るものである．

このような背景から，より高性能な決定木構築アルゴリズムの開発が期待されているといえる．

1.1 関連研究

現在広く利用されている決定木アルゴリズムとして，C4.5 などのヒューリスティックに基づくものがある [7]．C4.5 は，決定木の各ノードを，ジニ係数やエントロピーなどの情報理論的スコア関数に基づいて選択する．

こうしたアルゴリズムはトップダウンの形で高速かつ省メモリに決定木を構築できる一方で，網羅的な探索を行っておらず，精度の最適性の保証を持たない．例えば，訓練集合に対し，C4.5 が算出した木よりも高い精度を実現する決定木は存在しうる．また，C4.5 が訓練集合からある制約条件のもとで木を一つも発見できなかったとしても，制約を満たす木が存在しないとは限らない．実際は算出可能な木に対し，単にヒューリスティックが到達しなかったというケースがありうる．

これに対し，2007 年に Nijssen らは DL8 という最適決定木の厳密発見アルゴリズムを提案した [5]．このアルゴリズムは，とりうるすべての木を網羅的に探索することで，訓練集合に対する識別精度最大の決定木を，与えられた制約のもとで厳密に発見することができる．

DL8 は枝刈り条件の導入などにより効率的に網羅的な探索を行うことができるが，速度やメモリの面で課題が残り，データセットによっては上手く動作しないことが著者らの論文において述べられている [6]．

1.2 研究目的

以上のように，従来の決定木構築アルゴリズムのうち，C4.5 などのヒューリスティックに基づくものは高速だが，探索の網羅性がなく算出する解の精度に問題がある．一方で，DL8 は厳密な最適解を発見できるが，実行上の効率の面で問題を抱えているということがいえる．

単一の決定木の精度を向上させる場合，訓練集合に対して精度最大な決定木を厳密発見できる DL8 は望ましいが，その一方で，決定木構築にかかる時間計算量，空間計算量があまりに大きい場合，実務領域での利用が困難になる．

こうした背景から，我々は，精度と計算効率の両面で，より適切な位置づけのアルゴリズムを開発する．C4.5 のようなヒューリスティックよりも高精度であり，DL8 のような厳密発見手法よりも省メモリであるアルゴリズム ODT を提案する．

また，提案アルゴリズムを用いて，文書分類の実験を行う．既存アルゴリズムと提案アルゴリズムの精度・計算効率の比較

を行い，実際に分類過程の可視化を試みることで，アルゴリズムの有効性を検証する．

2. 準　　備

2.1 データベース

$I = \{i_1, \ldots, i_n\}$ を n 個の**変数**からなる**変数空間**とする．n 個の変数（または**アイテム**）の組 $x = (i_1, \ldots, i_n)$ を，**タプル** (tuple) とよぶ．**タプル空間**を $\mathcal{X} \subseteqq I$ とおく．

タプルに対し，正例または負例を表すブール値 $y \in \mathcal{Y} = \{0, 1\}$ を，**分類ラベル**とよぶ．タプルと分類ラベルの組 $e = (x, y)$ を，**分類例**（または**データ**）とよぶ．**分類ラベル付きデータベース**（または**データベース**）は，m 個の例の組 $E = \{e_1, \ldots, e_m\} \subseteqq \mathcal{X} \times \mathcal{Y}$ を指す．

変数空間に含まれる変数の総数およびデータベース中の例数を，以後それぞれ $n = |I|$ と $m = |E|$ で表す．また，データベース中の個々の例 $e_i \in E$ の**添字** i を *TID* とよび，$Tid(E) := [1..m]$ で E の**添字集合**を表す．

2.2 パターンと出現リスト

I 上の論理式である**リテラル**と**パターン**を定義する．

各変数 $i \in I$ に対して，その**否定**を $\neg i$ と表記する．変数空間上の任意のタプル $x \in I$ について，x の i 番目の要素が 1 であるとき $i \in x$，0 であるとき $\neg i \in x$ である．

変数とその否定を**リテラル**（literal）と呼ぶ．リテラルの全体集合を $\Sigma := I \cup \neg I$ とする．ただし，集合 $\neg I := \{\, \neg i \,|\, i \in I \,\}$ である．

Σ 上のサイズ d の**パターン**とは，リテラルの有限集合 $p = \{z_1, \ldots, z_d\} \subseteqq \Sigma$ であり，リテラルの論理積 $z_1 \wedge \cdots \wedge z_d$ を表す．ただし，任意のパターンにおいて，変数 i とその否定 $\neg i$ が同時に含まれることはない．したがって，任意のパターン p のサイズ $|p| \leqq n = |I|$ である．

［定義 1］（リテラルとパターンの評価値） 任意のタプル $x \in \mathcal{X}$ に対して，x に対するパターン p の**評価値** $\phi_p(x) \in \mathcal{Y} = \{0, 1\}$ を次のように定義する:

- 変数 $i \in I$ に対して，$\phi_i(x) := 1 \iff i \in x$.
- 変数の否定 $\neg i \in \neg I$ に対して，$\phi_{\neg i}(x) := \neg \phi_i(x)$.
- パターン $p = \{z_1, \ldots, z_d\} = z_1 \wedge \cdots \wedge z_d$ に対して，$\phi_p(t) := \phi_{z_1} \wedge \cdots \wedge \phi_{z_d}$.

データベースまたはその部分集合 E における**パターン** p **の出現リスト**とは，パターン p の評価値 $\phi_p(t_i)$ が真となる E の添字の集合であり，$Occ_E(p) := \{\, i \in [1..m] \,|\, \phi_p(t_i) = 1 \,\}$ と定義する．E 中のパターン p の**頻度** (frequency) を $freq_E(p) := |Occ_E(p)| \in [0..m]$ とする．

以後，文脈から明らかならば，E とその添字集合 $Tid(E)$，および添字集合 $Tid(E')$ と対応する $E' \subseteqq E$ 中の例集合を区別しない．出現リストは添字集合であるため，出現リスト E' 中でのパターン p の出現を $Occ_{E'}(p)$ などとも書く．

2.3 頻出アイテム集合マイニング問題

二値データベース E において，出現頻度が閾値 σ 以上であるようなパターンを σ-**頻出パターン**とよぶ．**頻出アイテム集合マイニング問題**とは，データベース E における σ-頻出パターンを全て出力する問題である．

［定義 2］（頻出アイテム集合マイニング問題） 入力として二値データベース E，制約パラメータとして最小頻度 $\sigma \in [0..m]$ が与えられたとき，$freq_E(p) \geqq \sigma$ となるようなすべてのパターン p を出力せよ．

頻出アイテム集合マイニング問題を解く代表的なアルゴリズムとして，Agrawal らによるアプリオリアルゴリズム [1]，Bayardo らによるバックトラックアルゴリズム [3] などがある．

2.4 決定木の族

本稿では，頂点のテストとしてアイテムをもち，内部頂点の分岐数が常に 2 である**二分決定木**のみを扱い，これを単に**決定木**とよぶ．はじめに，変数順序を持たない**順序なし決定木**のクラス $\mathcal{DT}$ を導入する．

［定義 3］（順序なし決定木） アイテム集合 I 上の**順序なし決定木**（decision tree）$\mathcal{DT}$ は，次のように定義される頂点ラベル付き 2 分木 $T = (V(T), E(T), root(T), label_T)$ である．

- $V(T)$ は頂点集合であり，$E(T)$ は 1-枝と 0-枝と呼ばれる有向辺の集合である．唯一の入次数 0 の頂点 $root(T) \in V(T)$ は**根**である．
- 頂点集合 $V(T)$ は**内部頂点**と**葉**からなる．各内部頂点 $v \in V(T)$ は，*1*-**枝**と *0*-**枝**と呼ばれる 2 つの有向辺をもち，それぞれ，*1*-**子**と *0*-**子**と呼ばれる子 $v.1$ と $v.0$ へと接続する．各葉 $l \in V(T)$ は枝および子をもたない．
- 関数 $label_T = test_T \cup class_T$ は，各頂点にラベルを対応づける**ラベル関数**である．各内部頂点 v に対して，$label_T(v) = test_T(v)$ は変数 $i \in I$ を返す．各葉 l に対して，$label_T(l) = class_T(l)$ は分類ラベル $y \in \mathcal{Y}$ を返す．

決定木 T 中の内部頂点の集合を $node(T)$，葉の集合を $leaf(T)$ と定義する．決定木 T 中の**パス**を，T 中の根から任意のノード v までの連続する有向辺の列と定義する．パスの**長さ** $|p|$ は，パスに含まれる辺の総数である．決定木 T 中の任意のノード v の**深さ**は，根から v へのパスの長さである．

決定木 T に対して，その**サイズ**を T の総頂点数 $k(T) = |V(T)| = |node(T)| + |leaf(T)|$ と定義し，その**深さ**を最長のパスの長さ $d(T) = \max_{p \in T} |p|$，**葉数**を T に含まれる葉の総数 $l(T) = |leaf(T)|$ と定める．T はすべての内部頂点が 2 つの子をもつ**全二分木**なので，常に $k(T) = 2l(T) - 1$ となる．

$T.1$ と $T.0$ で，それぞれ，根 $root(V)$ の 1 子と 0 子を根とする T の部分木を表し，T の *1* **木**と *0* **木**と呼ぶ．

決定木は，タプルに対して分類ラベルを返す**分類関数**と見ることができる．

［定義 4］（決定木が定義する分類関数） **決定木** T **が定める分類関数**は関数 $\psi_T : \mathcal{X} \to \mathcal{Y}$ であり，任意のタプル $x \in \mathcal{X}$ に対して $\psi_T(x) := \psi(root(T), x)$ と定義される．任意の頂点 $v \in V(T)$ に対して，$\psi_T(v, x)$ の値は次のように再帰的に定義される:

$$\psi(v, x) = \begin{cases} \ell, & \text{if } v \text{ は葉,} \\ \psi(v.1, x), & \text{if } v \text{ は内部頂点かつ } t \in x, \\ \psi(v.0, x), & \text{if } v \text{ は内部頂点かつ } t \notin x, \end{cases} \quad (1)$$

内部頂点 v に対して $t := test(v)$ は I 上の変数であり，葉 v に

表 1: サイズ n のデータベースにおいて決定木の予測関数 ψ_T が定める 2×2 分割表

	$\psi_T(x)=1$	$\psi_T(x)=0$	
$y=1$	m_1	n_1-m_1	n_1
$y=0$	m_0	m_0-m_0	n_0
	m	$n-m$	n

(2)

対して $\ell := class(v) \in \mathcal{Y}$ はクラスラベルである．上記の評価において，ある頂点 v で式 $\psi(v,x)$ が評価された場合に，タプル x は葉 v へ**到達する**という．

データベース E における決定木 T の**頂点 v の頻度**を，v へ到達する E 中のタプルの総数 $\sigma_E(v) \in [1..m]$ と定義する．E における T の**葉の最小頻度**を，全ての葉に対する頻度の最小値 $\sigma_E(T) := \min_{v\in leaf(T)} \sigma_E(v) \in [0..m]$ と定義する．

以後，I 上の**決定木**（decision tree）の族を，$\mathcal{DT}^I$ で表す．ただし，文脈上変数空間が明らかなとき，I を省略して $\mathcal{DT}$ と書く．任意の部分族 $\mathcal{C}\subseteqq\mathcal{DT}$ に対して，$\mathcal{C}_{k,d,\sigma}\subseteqq\mathcal{C}$ で，サイズが k 以下で，深さ d 以下，葉の最小頻度が σ 以上となる $\mathcal{C}$ に属する決定木の族を表す．

2.5 分類スコア

本節では決定木 T の分類スコアを導入する [4]．

n 個の例を含むデータベース E を仮定する．本小節のみ，データベース中の例数を n で表記するので注意されたい．

各例 $e=(x,y)\in E$ に対して分類ラベル $y\in\mathcal{Y}$ と決定木 T による予測ラベル $\psi_T(x)\in\mathcal{Y}$ を考えて，次のように非負整数 $n_1,n_0,m_1,m_0\in\mathbb{N}$ を定める:

- 非負整数 n_1，n_0 は，それぞれ E 中の正例の数と負例の数である．
- 非負整数 m_1，m_0 は，それぞれ予測ラベルが正の例のうち，E 中のラベルが 1 である例の数，0 である例の数である．

表 1 に，関連する 2×2 **分割表**を示す．$n=n_1+n_0$ が成立すること，および決定木による予測値が 0 となる正例の数，決定木の予測値が 1 となる負例の数はそれぞれ n_1-m_1 と m_0 となることは，定義より明らかである．

分類関数 $\psi_T:\mathcal{X}\to\mathcal{Y}$ と例 $e=(x,y)\in\mathcal{X}\times\mathcal{Y}$ に対して，$y\neq\psi_T(x)$ のとき，e で**誤分類**（error）が起きたという．

［定義 5］（経験誤差と精度）サイズ $n\geqq 0$ の任意のデータベース $E\subseteqq\mathcal{X}\times\mathcal{Y}$ に対して，分類関数 ψ_T の E における**誤分類数**を，ψ_T が誤分類した例の個数 $\#Err(\psi_T,E)$ とおき，E における ψ_T の**経験誤差** (empirical error) を，E における ψ_T による誤分類数の比率

$$Err_n(\psi_T;E) := \frac{\#Err(\psi_T,E)}{n} \in [0,1] \qquad (3)$$

と定める．本節のように，$\mathcal{Y}=\{1,0\}$ の 2 クラス分類の場合，分割表より，$\#Err(\psi_T,E) := (n_1-m_1)+m_0 \in [0..n]$ である．さらに，E における ψ_T の**経験精度**（accuracy）を $Acc_n(\psi_T;E) := 1-Err_n(\psi_T;E)\in[0,1]$ とおく．

2.6 順序付き決定木の族

I 上の**変数順序** (variable ordering) $ord:[1..n]\to[1..n]$ は，変数の添字上の置換である．$<_I$ を，ord に対して定まる**変数間の全順序**とする．$ord(i)$ を変数 i の変数順序番号，$ord^{-1}(j)$ を ord における第 j 番目の変数とすると，$ord^{-1}(1)<_I\cdots<_I ord^{-1}(n)$ である．

変数順序 $<_I$ のもとで，順序付き決定木の族 $\mathcal{OT}$ を導入する．

［定義 6］ 決定木 T が**順序付き** (ordered) であるとは，I 上にある変数順序 ord が存在して，T 中の任意の根から葉までのパス上の変数が全順序 $<_I$ の昇順で並んでいることをいう．

以後，$\mathcal{OT}^{I,\mathrm{ord}}$ で，I 上の変数順序 ord にしたがう**順序付き決定木** (ordered decision tree) の族を表す．定義より，任意の ord に対して $\mathcal{OT}^{I,\mathrm{ord}}\subseteqq\mathcal{DT}^I$ である．文脈より明らかな時には添字 I と ord を略する．

3. 先行研究：DL8 アルゴリズム

DL8 アルゴリズムは，Njissen らによって提案された，最適決定木の厳密発見アルゴリズムである [5]．DL8 アルゴリズムは，頻出アイテム集合マイニングが列挙するアイテム集合と決定木のパスの関係性に着目し，訓練集合における精度最大の決定木を網羅的探索によって発見する．

3.1 アイテム集合とパスの関係

決定木 T 中のパスを考える．パス中の各有向辺は，隣接する内部頂点のうち，より深さが浅い点 (始点)v のラベル $label_T(v)=test_T(v)=i$，またはその否定 $\neg i$ がタプルに含まれるか否かの判定に対応する．したがって，決定木中の任意のパスはパターン $p=\{z_1,\ldots,z_d\}$ によって一意に表現することができる．

このとき，決定木 $T\in\mathcal{DT}$ によって，データベース E 中のすべての例を振り分けることを考えると，T 中の任意の葉 ℓ に到達する例 $e=(x,y)$ が満たすべき条件は，x が T 中の根から葉 ℓ までのパスに存在するすべての辺に対応するリテラルの集合 $p=\{z_1,\ldots,z_d\}$ を含むことと一致する．

［補題 1］ リテラルの全体集合 Σ におけるサイズ d のパターンの全体集合を $\mathcal{P}_\Sigma(d)$ としたとき，$\mathcal{P}_\Sigma(d)$ 上の長さ d の順列の全体集合 $\mathcal{P}^*_\Sigma(d)=perm(\mathcal{P}_\Sigma(d))$ は決定木の族 $\mathcal{DT}$ における長さ d のとりうるパスの全体集合 $\mathcal{P}_{\mathcal{DT}}(d)$ と等しい．

補題 1 より，深さ d の決定木の族 $\mathcal{DT}_{*,d,*}$ に存在しうるパスの全体集合は，$\mathcal{P}^*_\Sigma=\bigcup_{i=1}^d\mathcal{P}^*_\Sigma(i)$ である．したがって，$\mathcal{DT}_{*,d,*}$ における全ての木は $\mathcal{P}^*_\Sigma$ に含まれる要素の組み合わせで表現できる．また，データベース E に対する頻出アイテム集合マイニングによって列挙される σ-頻出パターンの全体集合を $\mathcal{P}^*_{\Sigma,\sigma}=\bigcup_{i=1}^d perm(p\in\{\mathcal{P}_\Sigma(i)|freq_E(p)\geqq\sigma\})$ とすると，$\mathcal{DT}_{*,d,\sigma}$ における全ての木は $P^*_{\Sigma,\sigma}$ に含まれる要素の組み合わせで表現できる．

3.2 DL8 アルゴリズムの構造

DL8 アルゴリズムは，前処理として頻出アイテム集合マイニングを実行し，その解集合をメモリ上に保持したうえで，制約を満たす決定木を全て試す再帰手続きを行う．

［補題 2］ 決定木の族 $\mathcal{DT}$ における任意の決定木 T について，T の根の左右の部分木 T_L と T_R が，それぞれ T のとりうる全ての候補部分木の中で経験精度最大であるとき，T は $\mathcal{DT}$ 中で経験精度最大である．

Algorithm 1: アルゴリズム ODT

Algorithm ODT$(I, ord, E, \sigma_{\min}, k_{\max}, d_{\max})$

Output: 最適木の根へのポインタ $root$、そのサイズ k、経験誤差 err からなる三つ組 $\tau_{opt} = (root, k, err)$

begin

1 $\Theta := (\sigma_{\min}, k_{\max}, d_{\max}, m := |E|, n := |I|, I, ord, E)$;
2 $p_0 := \emptyset$; $Tid(E) := [1..m]$; $tail_0 := n$, $d_0 := 0$;
3 $Opts :=$ RecODT$(p_0, Tid(E), tail_0, d_0, \Theta)$;
4 **return** $\tau_{opt} := \arg\min_{\tau \in Opts} \tau.err$;

補題 2 より，DL8 アルゴリズムは再帰的な分割統治法を用いて経験精度最大の決定木を厳密発見することができる．

4. 提案手法：ODT アルゴリズム

DL8 アルゴリズムは制約のもとで経験精度最大の決定木を厳密発見することができるが，順列の全列挙に相当する大きな時間計算量がかかり，頻出アイテム集合の総数に等しい数の候補パスをメモリ上に保持するために，入力に対して指数的な空間計算量がかかる [6]．

我々が提案する ODT アルゴリズムは，対象とする決定木の族を，ある**変数順序**のもとでの**順序付き決定木の族**に制約し，その中で経験精度最大の決定木を厳密発見することで，集合の全列挙に相当する時間，および入力に対する多項式領域へと計算量を改善する [10]．

本研究で考察する問題は次のとおりである．

[定義 7]（最適順序付き決定木発見問題） 順序付き決定木の族 $\mathcal{OT}$ に対して，入力として変数集合 $I = \{i_1, \ldots, i_n\}$ $(n \geqq 1)$ と，変数順序 $<_I$，I 上のデータベース $E = \{e_1, \ldots, e_m\} \subseteqq \mathcal{X} \times \mathcal{Y}$ $(m \geqq 1)$，制約パラメータとして最大サイズ $k_{\max} \geqq 0$ と，最大深さ $d_{\max} \geqq 0$，葉の最小頻度 $\sigma_{\min} \in [0..m]$ が与えられたとき，制約を満たす順序付き決定木 $T \in \mathcal{OT}_{k,d,\sigma}$ すべての中で，経験誤差 $Err_n(\psi_T; E)$ を最小化する順序付き決定木 $T_{\min}$ を出力せよ．

$\mathcal{OT} \subseteqq \mathcal{DT}$ であるために，最適木発見のために $\mathcal{OT}$ において試行すべき木の総数は $\mathcal{DT}$ におけるそれを下回り，問題の複雑性は減少する．一方で，$\mathcal{OT}$ における最大の経験精度は，$\mathcal{DT}$ の族における最大の経験精度よりも劣化しうる．

4.1 アルゴリズムの概要

図 1 に最適な順序決定木を計算する提案アルゴリズム ODT を示し，図 2 にその再帰手続き RecODT を示す．

アルゴリズム ODT は，頻出アイテム集合の列挙木をパスの探索空間として再帰的な深さ優先探索を行い，制約を満たす $\mathcal{OT}$ 上の最適決定木を発見する．

手続き RecODT は，探索木の根となる空パターン $\emptyset$ から，パスとなるパターンを構築しながら，再帰的にアイテム集合束上を探索し，分割統治法を用いてボトムアップに最適決定木を求めていく．手続き RecODT は，現在のパス p と，その出現リスト Occ，および p に追加する候補アイテムの最大の添字 $tail$ と再帰の深さ d を親から受け取り，Occ から二つの子のためのアイテムごとの出現リスト集合 $\mathcal{O}_1$ と $\mathcal{O}_0$ を計算した後に，自分自身を左右の子に対して再帰的に呼び出す．親へバックトラックする際には，各サイズ k ごとに，ボトムアップに求めた最適木のサイズ k，最適スコア err，最適木の根のポインタ $root$ からなる 3 つ組 $\tau = (root, k, err)$ を返す．

手続き RecODT における各繰り返しでは，左右の子の候補として，メモリ上に最大 $k_{\max}$ 個の決定木を構築し，それらを組み合わせた木のうち，各サイズ毎に経験誤差が最小の木を選択して，最適木プロファイルに登録する．手続き RecODT の

Algorithm 2: 再帰手続き RecODT

Procedure RecODT$(p, Occ, tail, d, \Theta)$

Output: サイズごとの最適決定木 τ_k からなる決定木プロファイル $Opts = \{\tau_1, \ldots, \tau_{k_{\max}}\}$

begin

1 $(\sigma_{\min}, k_{\max}, d_{\max}, m, n, I, \mathrm{ord}, E) := \Theta$;
// Step1: サイズ $k = 1$ の最適木を見つける
2 $y_1 := \arg\max_{y \in \mathcal{Y}} |Occ_y|$; $err_1 := |Occ| - |Occ_{y_1}|$;
3 $Opts := \emptyset$; $Opts[1] :=$ a new leaf l with $l.label := y, l.err := err_1$;
4 **if** $d + 1 > d_{\max}$ **then return** $Opts$;
// Step2: サイズ $k > 1$ の最適木を見つける
5 $(\mathcal{O}_0, \mathcal{O}_1) :=$ OccDeliver$(p.last, Occ, I)$;
6 **for** $i := 1, \ldots, tail$ **do**
7 $item := ord^{-1}(i)$;
8 **if** $(|\mathcal{O}_1[item]| \geqq \sigma_{\min})$ *and* $(|\mathcal{O}_0[item]| \geqq \sigma_{\min})$ **then**
9 $Opts_1 :=$ RecODT$(p \cup \{item\}, \mathcal{O}_1[item], i - 1, d + 1, \Theta)$;
10 $Opts_0 :=$ RecODT$(p \cup \{\neg item\}, \mathcal{O}_0[item], i - 1, d + 1, \Theta)$;
11 **foreach** $\tau_1 \in Opts_1$ *and* $\tau_0 \in Opts_0$ *with* $\tau_1.k + \tau_0.k < k_{\max}$ **do**
12 $\tau :=$ a new optimal tree triple;
13 $\tau.k := 1 + \tau_1.k + \tau_0.k$;
14 $\tau.err := \tau_1.err + \tau_0.err$;
15 **if** $(Opts[k] = null)$ *or* $(\tau.err < Opts[k].err)$ **then**
16 **if** $(Opts[k] \neq null)$ **then**
17 DecNodeRefCount$(Opts[k].root)$;
18 IncNodeRefCount(τ_1);
19 IncNodeRefCount(τ_0);
20 $\tau.root :=$ a new node v with $v.test := item$, $v.1 := \tau_1$, $v.0 := \tau_0$;
21 $Opts[k] := \tau$;
22 **foreach** $\tau \in (Opts_1 \cup Opts_0)$ **do**
23 DecNodeRefCount$(\tau.root)$;
24 **return** $Opts$;

Algorithm 3: 出現リストの振り分けを行う副手続き OccDeliver および、不要なメモリの回収を行う副手続き IncNodeRefCount と DecNodeRefCount

```
Procedure OccDeliver(item, Occ, I)
Output: (O_1, O_0) = (⋃_{t∈Occ, i∈I, i<_I item} {i | i ∈ t.x},
                      ⋃_{t∈Occ, i∈I, i<_I item} {i | i ∉ t.x})
begin
1   (O_1, O_0) := (∅, ∅);
2   for each t ∈ Occ do
3       for each i ∈ I such that i <_I item do
4           if i ∈ t.x then Add {i} to O_1[t] ;
5           else Add {i} to O_0[t] ;
6   return (O_1, O_0);
Procedure IncNodeRefCount(v)
begin
7   v.refc := v.refc + 1;
Procedure DecNodeRefCount(v)
begin
8   v.refc := v.refc − 1;
9   if v.refc = 0 then
10      DecNodeRefCount(v.0);
11      DecNodeRefCount(v.1);
12      delete v;
```

各繰り返しで親にバックトラックする前に，$Opts_1 \cup Opts_0$ に含まれる部分木のうち，選択されなかったものすべてについて，参照カウンタを用いてそれらの頂点を削除することでメモリを回収する．

ODT の基本的なアイデアは次の通りである．

まず，変数順序 ord を仮定することで，各パスについて，アイテム集合との一意な正規形を仮定できる．そのため，頻出アイテム集合の列挙木と ODT の列挙木が一致し，パターンを列挙しながら同時に決定木の構築を行うことができる．これにより，DL8 が要していた，頻出アイテム集合マイニングの解集合を保持するための入力サイズに対して指数的な領域が不要になる．また，変数順序の導入により，頻出アイテム集合の列挙木と ODT の列挙木を一致させることができる．こうすることで，探索木のノードの個数が順列の総数から集合の総数へと減少し，総実行時間の減少が見込まれるほか，頻出アイテム集合マイニングで開発された様々な技術の流用や，双方のアルゴリズムで必要とされる情報の共有などが可能となる．

4.2 アルゴリズムの計算量

本小節では，アルゴリズム ODT の計算量を解析する．

[定理 1]（ODT の計算量） 順序付き決定木の族 $\mathcal{OT}$ に対して，図 1 のアルゴリズム ODT は，変数集合 I と，変数順序 ord，入力データベース E，制約パラメータとして最大サイズ $k \geqq 0$，最大の深さ $d \geqq 0$，葉の最小頻度 $\sigma \in [0..m]$ を受け取り，与えられた制約を満たす全ての可能な順序付き決定木の中で，経験誤差を最小化する順序付き決定木 $T_{\mathrm{opt}} \in \mathcal{ODT}^I_{k,d,\sigma}$ を，$O(N + k^2) = poly(k, n, m)$ 作業領域と $O(M(N + nk^3)) = O(M \cdot poly(k, m, n))$ 計算時間で出力する．ここで，$n = |I|$ は変数の個数であり，$m = |E|$ はデータベースの例数，$N = O(mn)$ は E の総アイテム数であり，$M = O(2^n)$ は E における頻出パターンの総数である．

証明: まず，領域計算量について検討する．アルゴリズム RecODT の入力となる構造体 Θ の領域計算量は，データベース E の総要素数 $O(mn) = N$ によって N に抑えられる．パス p の長さは高々 $d < n$，出現リスト Occ の長さは高々 m，出現リスト集合 $\mathcal{O}_1, \mathcal{O}_0$ の総要素数は合わせて mn である．候補木を格納する配列 $Opts$ は，探索木のルートと，再帰のアクティブな深さに一つずつ存在し，それぞれ候補木へのポインタを k 個もつため，候補木配列に必要な領域は $O(k)$ となる．また，個々の木のノード数は定義より $O(k)$ である．前述の参照カウンタにより，探索木のルートと，再帰のアクティブな深さの候補木集合 $Opts$ に含まれる以外の木は全てメモリから回収されるため，ある時点においてメモリ上に存在する実ノード数は $O(k^2)$ 個である．したがって総領域計算量は $O(N + k^2)$ 領域となる．

続いて，時間計算量について検討する．RecODT の探索木の総ノード数は頻出アイテム集合マイニングの解総数 $M = O(2^n)$ である．この M 個のノードそれぞれに，手続き RecODT の計算時間がかかる．RecODT の各行のうち，5 行目の OccDeliver については，$|Occ||I| = O(mn) = N$ 時間がかかる．6 行目のループは $O(n)$ 回，11 行目のループは $O(k^2)$ 回実行され，17 行目，22 行目の処理はいずれも $O(k)$ 時間で実行できる．他の行の処理は全て定数時間である．よって，手続き RecODT にかかる計算時間は $O(mn + n \cdot k^3)$ 時間であり，アルゴリズム全体の総計算時間は $O(M(N + nk^3))$ となる．以上で示された． □

定理 1 より，ODT の時間計算量のオーダーは DL8 と同様に入力サイズに対する指数時間である一方で，領域計算量は入力サイズの多項式メモリを達成し，DL8 の指数メモリに対して大幅な改善を行なっている．

5. データベース縮約

頻出アイテム集合マイニングにおいて，入力パラメータや制約条件を利用して事前にデータベースを圧縮する手法を，**データベース縮約**とよぶ．

ODT アルゴリズムの実装では，Uno らの 2004 年の論文に示されるデータベース縮約の手法 [9] を取り入れている．

まず，ODT の入力パラメータのうち，葉の最小頻度 $\sigma_{\min}$ が定まると，頻度が $\sigma_{\min}$ 未満または $1 - \sigma_{\min}$ より大きい全てのアイテム，およびそれを含むアイテム集合が候補パスとならないことが直ちにいえる．したがって，アルゴリズムを実行する前に，前処理として頻度制約を満たさないアイテムをデータベースから削除しても，得られる解は同一となる．

また，このようにしてアイテムを削除したデータベース中に，タプルとラベルの構成が全く同じである例が生じることがある．こうした同一の例を一つにまとめ，まとめた個数分の重みで扱うことで，アルゴリズム中で走査するデータ数を減らしつつ，

Algorithm 4: 重み付きエラー導入時の RecODT の 2 行目

// Step1: サイズ $k = 1$ の最適木を見つける

$y_1 := \arg\max_{y \in \mathcal{Y}} (\sum_{e_y \in Occ_y} w[e_y.label])$;

$err_1 := (\sum_{e \in Occ} w[e.label]) - (\sum_{e_{y_1} \in Occ_{y_1}} w[e_{y_1}.label])$;

同一の結果を得ることができる．

ODT では，データベースの読み込み時にそれぞれのアイテムの頻度を数え上げ，頻出でないアイテムはメモリ上に格納せず除外して処理する．さらに，非頻出なアイテムを除いたデータベースのうち，タプルとラベルが同一のデータは一つに縮約し，計算時にそのデータ一つの重みを縮約した個数倍して扱う．

データベース縮約によってアルゴリズムの最悪時計算量は変化しないが，特に $\sigma_{\min}$ が大きい場合に，候補アイテムの総数とデータ件数を削減し，時間計算量とメモリ使用量を減少させることができる．

6. 不均衡データに対する重み付け

訓練集合中の分類ラベルのごとのデータ数に偏りのあるデータを，**不均衡データ**とよぶ．不均衡データ上で分類誤差を基準にモデル選択を行うと，誤差は小さいものの，識別モデルとしての意味が薄い学習結果が生じる場合がある．たとえば，正例と負例の割合が 1:9 であるようなデータベースに対して決定木学習を行うと，全ての要素を負とするサイズ 1 の木が，精度 90 %を達成するモデルとして選択されてしまうことが考えられる．このモデルは，正例と負例の比率が既知であれば自明であり，また正例の予測には全く役立たない．負例を誤って正例としてしまう**偽陽性** (false positive) の削減よりも，存在する正例を見逃してしまう**偽陰性** (false negative) を減らすことに重きが置かれる応用では，こうした学習結果は望ましくない場合がある．

このような不均衡データに関する問題に対して，いくつかのアプローチが提案されている [8]．データレベルのアプローチには，正例を水増しする over sampling，負例を間引く under sampling などがあり，アルゴリズムレベルのアプローチには，データベース中のラベル分布の偏りに応じて各訓練データの重みを補正する**重み付け** (example weighting) などがある．

ODT アルゴリズムはデータベース縮約の同一データ縮約により既にそれぞれのデータに対する重み情報を持っており，自然な拡張で重み付けの補正を導入できる．ODT アルゴリズムに対して，重み付けの補正を導入した際に，アルゴリズム 2 の 2 行目を書き換えたものをアルゴリズム 4 に示す．ここで，w は分類ラベルごとに設定する重みの配列である．

7. ODT アルゴリズムによる文書分類

BoW (Bag of Words) は，文章にある単語が含まれていれば 1，そうでなければ 0 を格納したバイナリ列によって文章を表現する文書モデルである．BoW の構築にあたっては，単語集合である**ボキャブラリ**を用意し，ボキャブラリ中のそれぞれの語が含まれているか否かを 1 または 0 で表現することで，ボキャブラリ長ぶんの二値ベクトルを生成する．

このとき，BoW 構築に用いる単語のボキャブラリをアイテム空間 $\mathcal{I}$ に，BoW の各単語をアイテムに対応させると，任意の BoW はタプルとみなすことができる．

それぞれの BoW に，文書が所属するカテゴリを示すラベルを紐付けた**識別ラベル付き文書集合**を考えると，これは ODT が入力として受け取るデータベースと同一の形式とみなすことができ，単語の有無の情報を用いて文書分類を行う決定木を，ODT アルゴリズムによって学習することができる．

表 2: データセットの属性

name	$\|E\|$	$\|I\|$	正例	負例
MBSM	374	2611	29	345
CORP	374	2611	45	329

8. 実　　験

提案手法 ODT を用いた文書分類の実験を行う．

8.1 デ ー タ

実験対象の文書集合として，DEIM2016(注1) の最終論文集を用いた．論文集のうち，pdf が存在する日本語の論文のみを選び，タイトル・アブストラクト・キーワードを抽出したものを文書とする．これを形態素解析ソフト MeCab(注2) によって分かち書きし，BoW 表現に変換したものをタプルとして用いた．タプルに対し，以下の分類ラベル列を用意した．

- マイクロブログ·ソーシャルメディア (MBSM): DEIM2016 のセッション表で,「マイクロブログ分析」1〜3 か,「ソーシャルメディア」1〜3 に登録されていれば 1，さもなくば 0
- 企業 (CORP): 論文のいずれかの著者の所属が企業またはその研究所であれば 1，さもなくば 0

各タプルに上記の分類ラベルをそれぞれ割りつけたものを同名の分類ラベル付きデータベースとし，決定木学習を行った．

8.2 プログラム

使用したプログラムは次のとおりである．

- ODT: 4. 節のアルゴリズム ODT を C++で実装したもの．最適化手法として，4. 節に述べた Occurrence Deliver とデータベース縮約を組み込んだ．候補アイテムの順序としては，データベース中のアイテムの頻度が大きいものからの降順を採用している．
- ODT-W: ODT に対し，6. 節で述べた重み付け補正を加えたもの．補正として，訓練データの正例の重み合計と負例の重み合計が等しくなるような w を設定している．
- DL8: 文献 [5, 6] の著者らによるオリジナルの C++実装を入手して用いた．
- C4.5: データマイニングツール Weka(注3) に搭載されている C4.5 の Java 版実装である J48 を用いた．

(注1)：http://db-event.jpn.org/deim2016/

(注2)：http://taku910.github.io/mecab/

(注3)：http://www.cs.waikato.ac.nz/ml/weka/

表 3: 決定木学習に要した実行時間 (sec)

	MBSM			CORP		
$\sigma_{\min}$	ODT	DL8	C4.5	ODT	DL8	C4.5
50	0.16	0.65	2.59	0.15	15.69	2.43
40	0.93	–(b)	2.61	0.92	–(b)	2.64
30	4.71	–(b)	2.76	4.61	–(b)	2.78
20	62.33	–(b)	2.77	52.69	–(b)	2.84
10	577.96	–(b)	2.85	562.21	–(b)	3.10
5	–(a)	–(b)	3.46	–(a)	–(b)	3.99
2	–(a)	–(b)	4.03	–(a)	–(b)	4.90

表 4: 最大経験精度とその際の木サイズ

	MBSM				CORP			
$\sigma_{\min}$	ODT		C4.5		ODT		C4.5	
	Acc_n	k	Acc_n	k	Acc_n	k	Acc_n	k
50	0.8797	1	0.8797	1	0.9225	1	0.9225	1
40	0.8797	1	0.8797	1	0.9225	1	0.9225	1
30	0.8797	1	0.8797	1	0.9251	7	0.9225	1
20	0.8797	1	0.8797	1	0.9305	7	0.9225	1
10	0.8957	7	0.8797	1	0.9332	7	0.9225	1
5	–(a)	–(a)	0.8957	7	–(a)	–(a)	0.9305	3
2	–(a)	–(a)	0.9519	41	–(a)	–(a)	0.9599	19

a ：実行時間 600 秒を超えたため未計測
b ：使用メモリ 64GB を超えたため実行不可

可視化時の解釈可能性と実行速度を考慮し，ODT および DL8 には，木のサイズ制約 $k_{\max}$ として 7 を指定した．C4.5 には木サイズ制約のパラメータが存在しないため，サイズ制限は与えず，ヒューリスティックによる探索が終了するサイズまで実行している．

実験環境は，PC（CPU Intel Core i7 3.3GHz, Memory 64GB, OS Ubuntu 14.04），コンパイラ g++ ver4.8.4 を用いた．

8.3 実験 1：実行効率

MBSM および CORP のデータセットについて，それぞれ ODT，DL8，C4.5 の 3 つのアルゴリズムで決定木の学習を行った．

速度に関する実験結果を表 3 に示す．まず，DL8 アルゴリズムは，ほとんどの実験設定において実験環境上限の 64GB 以上のメモリを消費し，動作しなかった．これは，本稿の実験で用いたデータセットのアイテム種別数が約 2600 と通常の頻出マイニング用データセットに比して多く，前処理のアイテム集合列挙によって生じる解の個数（アイテム種別数 n に対して $O(2^n)$）が膨大になっているためだと考えられる．一方で，ODT は前処理の結果をメモリに保持せず実行しており，データベース縮約で制約を満たさないアイテムを除去しているため，また，C4.5 はヒューリスティックにより全解探索をしていないためこうした問題を回避できている．

それぞれのアルゴリズムの実行時間については，ODT はパラメータの変化に対して指数的に実行時間が増加し，$\sigma_{\min} = 5, 2$ の設定では 600 秒以内に実行が完了しなかった．C4.5 は常にパラメータに線形程度のごく短時間で実行が完了している．

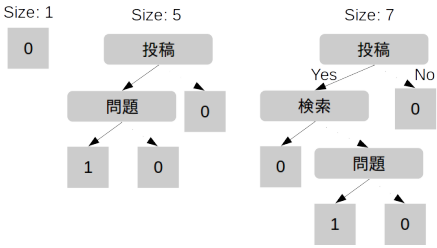

図 1: MBSM データセットに $\sigma_{\min} = 10$ で ODT を適用した結果得た木の一覧

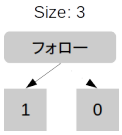

図 2: MBSM データセットに $\sigma_{\min} = 5$ で C4.5 を適用した結果得た木

以上の結果より，DL8 アルゴリズムはメモリ使用量の面で本研究の実験に適さないこと，および実行時間の面では ODT に比して C4.5 が優れていることがわかる．

8.4 実験 2：経験精度と木サイズ

メモリ使用量の面で実験に適さないとわかった DL8 アルゴリズムを除き，ODT アルゴリズムと C4.5 アルゴリズムについて，経験精度と出力された木を比較する．表 4 に，学習の結果得られた木の経験精度とサイズを示す．

6. 節の議論より，サイズ 1 の木は自明な木であり，学習結果として意味が薄い．また，$\sigma_{\min}$ は一つの葉に分類される最小データ数を制約するパラメータであり，同時に決定木の一つの分割を決定するために用いられる最小データ数を制約している．分割を決定する例数が少ないことは，過学習のリスクを高めるため，より大きな $\sigma_{\min}$ パラメータで，サイズ 1 よりも大きい木が得られることが分析上望ましい．

この観点では，ODT は網羅的探索により C4.5 よりも大きな $\sigma_{\min}$ で非自明な木を得られており，アルゴリズムの有効性が確認できる．一方で，C4.5 は ODT が時間的に実行困難なパラメータでも動作し，サイズは大きいものの，経験精度の高い木を得ることができている点で優れているといえる．

図に，MBSM データセットに $\sigma_{\min} = 10$ で ODT が算出した木，および $\sigma_{\min} = 5$ で C4.5 が算出した木を示す．ODT は，一度の実行で経験精度最大の木をすべてのとりうる深さについて出力することができるため，3 種類の木を出力している．サイズ 7 の木を解釈すると，「投稿」を含み，「検索」を含まず，「問題」を含むデータはマイクロブログ・ソーシャルメディアカテゴリであるという決定木になっている．マイクロブログの根幹をなす「投稿」という単語を根に持つのは対象カテゴリ上自然であり，「検索」を含まないのはこの語を含む論文は情報検索のセッションに割り振られているためであると推測できる．「問題」に関しては普遍的な語であるため，過学習の可能性も考えられる．C4.5 が算出した木は，「フォロー」という，マイクロブログサービスである Twitter(注4) の機能名を含むものをクラス 1 に割り振っており，直感的な結果である．

8.5 実験 3：重み付けによる出力木の変化

実験で用いている NBSM および CORP のデータは，いずれ

(注4)：http://twitter.com/

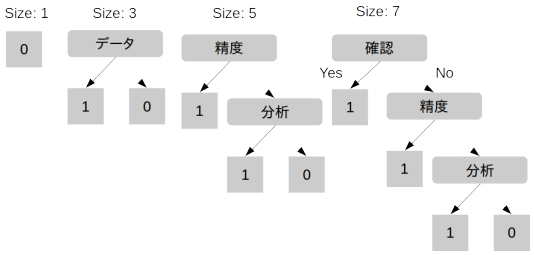

図 3: CORP データセットに $\sigma_{\min} = 50$ で ODT-W を適用した結果得た木の一覧

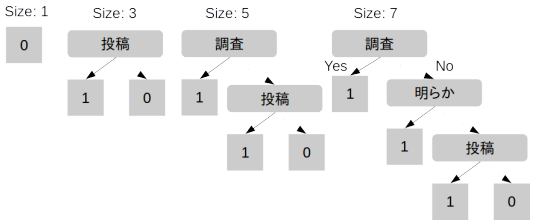

図 4: MBSM データセットに $\sigma_{\min} = 10$ で ODT-W を適用した結果得た木の一覧

も 6. 節で述べた不均衡データであり，例の割合が負例に偏っている．前小節の実験において，高い $\sigma_{\min}$ の設定でサイズ 1 の自明な木が出力されてしまうのは，こうしたデータの偏りによる．

ここでは，ODT に対して，訓練データの正例の重み合計と負例の重み合計を等しくするような重みベクトル $w = (1, \frac{y_0}{y_1})$ を設定した ODT-W を用いて分析を行い，不均衡データに対する対策を試みた．実験の設定上，ODT が算出する木はエラー数最小ではなく，エラーにかかる重みの総和最小となるため，最大の精度の木といった捉え方ができなくなり，精度の定量的評価が難しくなる．したがって，本稿では，いくつかの出力解の例を示すに留める．

図 3 は，CORP データセットに $\sigma_{\min} = 50$ で ODT-W を適用して得た木の一覧である．同一のパラメータでは，ODT,DL8,C4.5 のいずれもサイズ 1 の木しか算出することができなかったが，ODT-W では重み付けにより，有意味な木を算出できている．各ノードには,「精度」「分析」「確認」「データ」など，データ分析に関する用語が並び，それらのクラスが 1 に割り振られている．ここから，企業に所属する著者を含む研究グループの関心がデータ分析に向いていることが推察される．

図 4 は，MBSM データセットに $\sigma_{\min} = 10$ で ODT-W を適用して得た木の一覧である．前節に記載した重みなしの分析結果と比べ，語の構成がやや変化している.「投稿」を含むものを 1 に割り振るのは同様だが,「調査」「明らか」という語もクラス 1 への割り振り条件になっている．マイクロブログやソーシャルメディアを扱う論文が，何かを「調査」し,「明らか」にする傾向を持っていると考えれば，筋の通る結果であるといえる．

9. おわりに

本稿では，決定木の部分族である順序付き決定木の族 $\mathcal{OT}$ に対して，入力の多項式領域のメモリしか用いずに，先行研究 DL8 と同一のオーダーの時間計算量で，制約を満たす最適決定木を厳密に発見する提案アルゴリズム ODT を与えた．また，提案アルゴリズム ODT を用いて，文書分類問題を効率的に解く手法を考察した．

実験の結果，Bag of Words 形式の文書のように候補アイテム数が多くスパースなデータにおいて，ODT アルゴリズムは最小例数制約の変化に対して指数的な時間計算量がかかるが，最小例数制約の大きい設定でも有意味な木を発見できること，一方で既存研究である C4.5 アルゴリズムは高速であるが，最小例数制約を小さくしなければ有意味な木を発見できないことを確認した．また，DL8 アルゴリズムは候補アイテム数が多い設定ではメモリ消費が大きく動作困難であることがわかった．

本稿の実験では，一種類の実データのみを用いたため，実験の網羅性に欠けているほか，データ件数の少なさと不均衡性により，テスト精度の計測ができていない．したがって，より適切なデータを用いた実験が求められる．

また，タプルに連続値を含むデータベースに対応したアルゴリズムの改良や，ODT アルゴリズムによって発見した決定木を複数組み合わせたモデルによる精度向上，より効果的な辞書順の発見などは，今後の課題である．

謝　辞

第一著者の長部および第三著者の有村は，産業技術総合研究所の瀬々 潤氏，東京大学大学院の津田 宏治氏，寺田 愛花氏，美添 一樹氏に は，データベースからの決定木構築および，木探索を用いたパターン発見，大規模化について，また東京大学大学院の小宮山 純平氏および湊基盤 S プロジェクトの石畠 正和氏には，頻出アイテム集合発見の信頼度解析について貴重なご議論とご教示をいただきました．ここに謝意を表します．

文　献

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. the 20th International Conference on Very Large Data Bases (VLDB'94)*, pages 487–499. Morgan Kaufmann, 1994.

[2] R. Eto, R. Fujimaki, S. Morinaga, and H. Tamano. Fully-Automatic Bayesian Piecewise Sparse Linear Models. *Proc. AISTATS*, 33:238–246, 2014.

[3] R. J. B. Jr. Efficiently mining long patterns from databases. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*, pages 85–93. ACM Press, 1998.

[4] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[5] S. Nijssen and E. Fromont. Mining optimal decision trees from itemset lattices. In *Proc. the 13th ACM SIGKDD int'l. conf. on Knowledge Discovery and Data Mining*, pages 530–539. ACM, 2007.

[6] S. Nijssen and E. Fromont. Optimal constraint-based decision tree induction from itemset lattices. *Data Mining and Knowledge Discovery*, 21(1):9–51, 2010.

[7] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann,, 1993.

[8] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, 40(12):3358–3378, 2007.

[9] T. Uno, M. Kiyomi, and H. Arimura. LCM ver. 2: Efficient Mining Algorithms for Frequent / Closed / Maximal Itemsets. *Workshop on Frequent Itemset Mining*, 2004.

[10] 長部 和仁, 宇野 毅明, and 有村博紀. アイテム集合列挙に基づく最適な順序付き決定木の高速発見. Technical report, 人工知能基本問題研究会資料 102, 人工知能学会 SIG-FPAI, 2016 年 12 月.