部分木を素性とする Decision Stumps と Boosting Algorithm の適用

工藤 拓　　松本 裕治

奈良先端科学技術大学院大学 情報科学研究科

〒 630-0192 奈良県生駒市高山町 8916-5

{taku-ku,matsu}@is.aist-nara.ac.jp

近年, テキスト分類タスクの多様化に伴い, 単語の集合 (Bag-of-Word) を素性とする古典的手法では, 十分な精度を得にくくなっている. 精度向上には, テキストの構造 (構文/レイアウト) を考慮する必要があるが, 恣意的に選択された部分構造のみを用いた手法が多い. 本稿では, 構造を考慮したテキスト分類 (半構造化テキスト分類) に向け, 部分木を素性とする Decision Stumps と, それを弱学習器とする Boosting アルゴリズムを提案する. また, Tree Kernel を用いた SVM との関連性, 及び本手法の利点について言及する. 実データを用いた実験により, 提案手法の有効性を検証する.

キーワード: Decision Stumps, Boosting, 半構造化テキスト分類, Tree Kernel

Boosting with Subtree-based Decision Stumps and Its application to Semi-strucutred Text Classification

Taku Kudo　　Yuji Matsumoto

Graduate School of Information Science, Nara Institute of Science and Technology

8916-5 Takayama, Ikoma Nara 630-0192 Japan

{taku-ku,matsu}@is.aist-nara.ac.jp

The research focus in text classification has expanded from a simple topic identification to a more challenging task, such as opinion/modality identification. For the latter, the traditional bag-of-word representations are not sufficient, and a richer, more structural representation will be required. Accordingly, learning algorithms must be able to handle such sub-structures observed in text. In this paper, we propose a Boosting algorithm that captures sub-structures embedded in text. The proposal consists of i) decision stumps that use subtree as features and ii) Boosting algorithm in which the subtree-based decision stumps are applied as weak learners. We also discuss a relation between our algorithm and SVM with Tree Kernel. Three experiments on the opinion/modality classification tasks confirm that subtree features are important. Our Boosting algorithm is computationally efficient for classification tasks involving discrete structural features.

Keywords: Decision Stumps, Boosting, Semi-structured Text Classification, Tree Kernel

1　はじめに

SVM といった機械学習手法が, テキスト分類に応用され, 多くの成功をおさめている. テキスト分類では, 一般に単語の集合 (Bag-of-Word 以下 BOW) を素性とし, 政治, 経済, スポーツといったカテゴリにテキストを分類する. 個々の単語が, これらのカテゴリを特徴付けるのに十分な情報を与えるため, BOW といった単純な素性でも十分な精度を達成できる.

一方, テキストマイニングの分野では, 「お客様の声」や「現場の声」といったテキストデータから, 早期に危機情報や要求を発見, 分析するための要素技術が求められている. 具体的には, コールセンターログからの知識発見, 自由記述アンケートや製品レビュー記事からの不満表現の発見といったタスクがそれにあたる. これらのタスクでは, 旧来の文書分類に典型的なカテゴリではなく, 主観的/客観的に述べているのか?, (ある製品を) 誉めてる/けなしているのか?, 背景/結果/結論を述べているのか? といった書き手の意図に関するカテゴリが使われる. さらに, 分類単位が, 文書といった大きな単位からパッセージ, 文のような小さな単位に変化している. これらのタスクは, テキスト分類には変わりないが, BOW といった素朴な素性では, 高い精度が得られにくい. 単純には, 品詞による単語のフィルタリング, 固定長の N-グラム, 係り受け関係 (係り元, 係り先のペア) といった, 有効そうな素性を発見的に追加することで, ある程度精度向上が期待できる. しかし, これらの手法は, あまりにも発見的であり, タスク依存性が強い.

本稿では, テキストが単語の集合であると仮定する従来手法に対し, テキストの「構造」を考慮する学習/分類アルゴリズムを提案する. ここでの構造とは, N-グラム, 係り受け構造, HTML/XML といった**ラベル付き順序木**を指す. この構造は, 自然言語処理に

おいて一般的なものである．さきほどの発見的な手法は, 構造情報を部分的に選択し, 利用しているモデルであると解釈できる．

本稿で提案する学習/分類アルゴリズムは, 以下の 3 点の特徴を持つ．

(1) テキストの構造 (構文構造やレイアウト構造) を考慮した, 学習/分類

(2) 学習の素性に, 構造の部分木を用いるが, そのサイズに制限を設けず, 全部分木を素性の候補とする．(係り受け木中の任意のサイズの部分木, 任意の長さの N-グラムが素性となる)

(3) 分析を容易にするために, 分類に必要な最小限の素性を学習アルゴリズムが自動的に選択する．

以下, 次章では提案手法の詳細, 3 章でその実装方法について説明する．4 章で Tree Kernel を用いた SVM と提案手法の関連性に言及する．5 章で, 評価実験とそれらの結果を報告し, 6 章でまとめと今後の課題について述べる．

2 提案手法

詳細に入る前に, 木に対するいくつかの定義, 表記について言及する．

2.1 準備

定義 1 ラベル付き順序木
ラベル付き順序木は, すべてのノードに一意のラベルが付与されており, 兄弟関係に順序が与えられる木である．

本稿では, ラベル付き順序木を単純に木と呼ぶ．

定義 2 部分木
t と u ラベル付き順序木とする．ある木 t が, u にマッチする, もしくは, t が u の部分木である *($t \subseteq u$)* とは, t と u ノード間に, *1* 対 *1* の写像関数 ψ が定義できることである．ただし, ψ は, 以下の *3* つの条件を満たす．*(1)* ψ は, 親子関係を保持する．*(2)* ψ は, 兄弟の順序関係を保持する．*(3)* ψ はラベルを保持する．

本稿では, 木 t 中のすべての部分木の集合を $\mathcal{S}(t)$ ($\mathcal{S}(t) = \{t' | t' \subseteq t\}$), さらに 木 t のノードの数を $|t|$ と表記する．

2.2 ラベル付き順序木の分類問題

ラベル順序木の分類問題 (木の分類問題) とは, L 個の学習データ $T = \{\langle y_i, \mathbf{x}_i \rangle\}_{i=1}^{L}$ から, 分類関数 $f(\mathbf{x}) : \mathcal{X} \rightarrow \{\pm 1\}$ を導出することである．ただし, $\mathbf{x}_i \in \mathcal{X}$ はラベル付き順序木であり, $y_i \in \{\pm 1\}$ は, 木 $\mathbf{x}_i$ に付与されたラベルである (ここでは, 2 値分類問題を考える)．通常の学習, 分類問題との違いは, 事例 $\mathbf{x}_i$ が, 数値ベクトルで表現されるのではなく, ラベル付き順序木として表現される点にある．

2.3 Decision Stumps for Trees

Decision Stumps は, 1 つの素性の有無に基づき分類を行う, 単純な分類アルゴリズムである．Decision Stumps は, 深さ 1 の Decision Tree (決定木) とみなすことができる．ラベル付き順序木の分類問題に対し, 本稿では, 以下のような部分木の有無に基づく Decision Stumps を考える．

定義 3 *Decision Stumps for Trees*
t および $\mathbf{x}$ を, ラベル付き順序木, $y \in \{\pm 1\}$ を, クラスラベルとする．木を分類するための *Decision Stumps* を以下のように定義する．

$$h_{\langle t,y \rangle}(\mathbf{x}) \stackrel{\text{def}}{=} \begin{cases} y & t \subseteq \mathbf{x} \\ -y & otherwise. \end{cases} = y \cdot (2I(t \subseteq \mathbf{x}) - 1)$$

分類器のパラメータは, 木 t と ラベル y の組 $\langle t, y \rangle$, である．以後この組を, 分類器の **ルール** と呼ぶ．

Decision Stumps の学習は, 学習データ $T = \{\langle y_i, \mathbf{x}_i \rangle\}_{i=1}^{L}$ に対するエラー率を最小にするルール $\langle \hat{t}, \hat{y} \rangle$ を導出することで実現できる．

$$\langle \hat{t}, \hat{y} \rangle = \mathop{\mathrm{argmin}}_{t \in \mathcal{F}, y \in \{\pm 1\}} \frac{1}{2L} \sum_{i=1}^{L} (1 - y_i \cdot h_{\langle t,y \rangle}(\mathbf{x}_i)) \quad (1)$$

ただし, $\mathcal{F}$ は, 全部分木の集合 (**素性集合**) である (i.e., $\mathcal{F} = \bigcup_{i=1}^{L} \mathcal{S}(\mathbf{x}_i)$)．ここで, ルール $\langle t, y \rangle$ に対する $gain(\langle t, y \rangle)$ を以下のように定義する．

$$gain(\langle t, y \rangle) \stackrel{\text{def}}{=} \sum_{i=1}^{L} y_i \cdot h_{\langle t,y \rangle}(\mathbf{x}_i) \quad (2)$$

gain を用いると, (1) の最小化問題は, 以下の最大化問題と等価となる．

$$\langle \hat{t}, \hat{y} \rangle = \mathop{\mathrm{argmax}}_{t \in \mathcal{F}, y \in \{\pm 1\}} gain(\langle t, y \rangle)$$

本稿では, エラー率最小のかわりに, それと等価である gain の最大化を用いて, Decision Stumps の学習を定式化する．

2.4 Boosting の適用

部分木を素性とする Decision Stumps は, 1 つの部分木の有無に基づき分類を行うため, それ単独では精度が悪い．しかし, Boosting[6] を用いて, その精度を向上させることができる．Boosting は, 逐次 **弱学習器** (ここでは, 部分木を素性とする Decision Stumps) を構築し, それらの重み付き多数決によって, 最終的な分類器 f を構成する．k 番目の弱学習器は, それぞれ異なった事例分布 (重み) $\mathbf{d}^{(k)}$ を用いて学習される．$\mathbf{d}^{(k)} = (d_1^{(k)}, \ldots, d_L^{(k)})$, (ただし $\sum_{i=1}^{N} d_i = 1, d_i^k \geq 0$)．分布 $\mathbf{d}^{(k)}$ は, 分類が困難な事例に高い重みが与えられるように逐次更新される．弱

学習器が無作為分類よりも精度が良いことさえ満足すれば, 単一の弱学習器より, Boosting の汎化能力が高くなることが理論的に証明されている [6]. Decision Stumps を弱学習器とする Boosting を構築するには, 分布 $\mathbf{d}$ を考慮するよう 式 (2) を再定義すればよい.

$$gain(\langle t, y\rangle) \stackrel{\text{def}}{=} \sum_{i=1}^{L} d_i \cdot y_i \cdot h_{\langle t,y\rangle}(\mathbf{x}_i)$$

Boosting の中で, 最も有名なアルゴリズムは, AdaBoost[6] であろう. しかし, 本稿では, AdaBoost の代わりに, Breiman らによる Arc-GV [2] を用いる (Arc-GV を用いる理由は, 4 節で説明する). また, 以下断りが無い限り, Boosting は, Arc-GV のことを差すこととする.

3 実装

本章では, 最適なルール $\langle \hat{t}, \hat{y}\rangle$ を発見するための高速なアルゴリズムを提案する. 最適ルール発見問題は以下のように定式化される.

問題 1 最適ルール発見問題
学習データ $T = \{\langle \mathbf{x}_1, y_1, d_1\rangle \ldots, \langle \mathbf{x}_L, y_L, d_L\rangle\}$ *(ただし*, $\mathbf{x}_i$ は順序木, $y_i \in \{\pm 1\}$ は木に対応するクラスラベル, d_i $(\sum_{i=1}^{L} d_i = 1,\ d_i \geq 0)$ は, 木の重み*)* が与えらえた時, *gain* を最大にするルール $\langle \hat{t}, \hat{y}\rangle$ を発見せよ. *(ただし*, $\langle \hat{t}, \hat{y}\rangle = \mathrm{argmax}_{t\in\mathcal{F}, y\in\{\pm 1\}} d_i y_i h_{\langle t,y\rangle}(\mathbf{x}_i)$, $\mathcal{F} = \bigcup_{i=1}^{L} \mathcal{S}(\mathbf{x}_i)$*).*

単純な方法は, 全部分木 $\mathcal{F}$ を陽に列挙し, そこから最大の gain を与える部分木を選択することであろう. しかし, 部分木の個数は, 木のサイズに対し指数的に増えていくために, こういったしらみつぶしの方法は実際の問題には適用困難である. 実際に, 木の集合から全部分木を完全に列挙する問題は, NP 困難であることが分かっている.

本稿では, 最適ルール発見問題のための効率良いアルゴリズムを提案する. 提案手法は以下の 3 つの戦略から構成される.

(1) 木の集合から, 全部分木を完全に重複なく列挙するための正準的な探索空間を定義する
(2) 1 で定義した探索空間を深さ優先探索し, 最大の gain を与える部分木を発見する.
(3) 1, 2. だけでは, 全部分木をしらみつぶしに列挙してることと変わりない. そこで, gain の上限値を用い, 探索空間を枝刈りする.

この 3 つの戦略について, 順に説明する.

3.1 部分木の列挙方法

Abe と Zaki は, 木から, その部分木を, 完全に重複なく列挙するアルゴリズム**最右拡張**を, それぞれ独立に提案している [1, 17]. 最右拡張は, 部分集合を列挙するアルゴリズム Set Enumuration Tree [8] の部分木への自然な拡張となっている. まずはじめに, サイズ 1 の木が列挙される. そして, サイズ $(k-1)$ の木に, 1 つのノードを追加することでサイズ k の木を構築する. この手続きを再帰的に適用することで全部分木を列挙する. しかし, 任意の位置にノードを追加すると重複する木を生成してしまうため, ノードの追加に一種の制限を設ける. この制限が最右拡張の鍵となるアイデアである. 以下に最右拡張の定義を与える.

定義 4 最右拡張 *[1]*
3 つの木 $t(\neq \phi)$, $t'(\neq \phi)$ $u(\neq \phi)$, が与えらた時, t' が, t の u に基づく最右拡張であるとは, 以下の *3* つの条件が満たされる時であり, かつその時に限る. *(1)* t と t' は u の部分木である *(i.e.,* $t \subseteq u$ かつ $t' \subseteq u$*). (2)* t' は, t に *1* つの ノードを追加することで構築される *(i.e.,* $t \subset t'$ かつ $|t| + 1 = |t'|$*). (3)* ノードは, t の最右パス上のノード *(*ルートから常に右端のノードを葉まで選ぶ*)* に追加される. *(4)* 追加されるノードは, 最右の兄弟である.

便宜上, ここでは次のような表記を用いる: $\{\langle t'_1, i'_1\rangle, \ldots, \langle t'_n, i'_n\rangle\} = \mathrm{RME}(t, i, u)$. ただし $t, t'_j\,(j = 1, \ldots, n)$ 及び u は, 定義 4 で導入した木である. つまり, $\{t'_j | j = 1, \ldots, n\}$ は t の u に基づく最右拡張となる木の集合である. また, i (および i'_j) は, t (および t'_j) の u に基づく最右葉の**位置** である (u 中のノードは, 前順序 (pre-order) で順序が付与されているものとする). 図 1 に 最右拡張の具体例を示す.

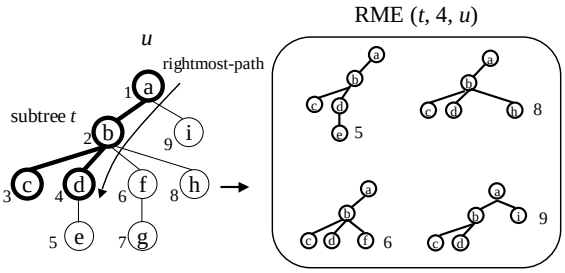


図 1: 最右拡張

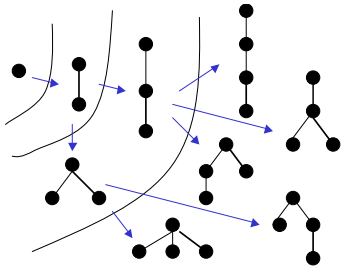


図 2: 最右拡張の再帰 (探索木)

最右拡張を, 再帰的に適用することで, 一種の「探索木」を構築するとができる. 図 2 に, その探索木の例を示す (図 2 中では, 便宜的にラベルの種類を 1 種類のみとしている).

3.2 探索空間の枝刈り

本章では, 前章で定義した探索空間を, 効率良く枝刈りする手法について議論する. 定理 1 (Morishita[10] の拡張) は, t の上位木 t' に対する $gain(\langle t', y\rangle)$ の上限値を与える.

定理 1 *gain* の上限値*:* $\mu(t)$
t の全上位木 $t' \supseteq t$, $y \in \{\pm 1\}$ について, ルール $\langle t', y\rangle$ の *gain* は $\mu(t)$ を上限値とする *(i.e.,* $gain(\langle t'y\rangle) \leq \mu(t)$*).* ただし, $\mu(t)$ は以下で与えられる.

$$\mu(t) \stackrel{\text{def}}{=} \max\Big(2\sum_{\{i|y_i=+1, t\subseteq \mathbf{x}_i\}} d_i - \sum_{i=1}^{L} y_i\cdot d_i,\ 2\sum_{\{i|y_i=-1, t\subseteq \mathbf{x}_i\}} d_i + \sum_{i=1}^{L} y_i\cdot d_i\Big).$$

証明 1

$$\begin{aligned} gain(\langle t', y\rangle) &= \sum_{i=1}^{L} d_i y_i h_{\langle t', y\rangle}(\mathbf{x}_i) \\ &= \sum_{i=1}^{L} d_i y_i \cdot y \cdot (2I(t' \subseteq \mathbf{x}_i) - 1) \end{aligned}$$

ここで, $y = +1$ の場合に着目すると

$$\begin{aligned} gain(\langle t', +1\rangle) &= 2\Big(\sum_{\{i|y_i=+1, t'\subseteq \mathbf{x}_i\}} d_i - \sum_{\{i|y_i=-1, t'\subseteq \mathbf{x}_i\}} d_i\Big) - \sum_{i=1}^{L} y_i\cdot d_i \\ &\leq 2\sum_{\{i|y_i=+1, t'\subseteq \mathbf{x}_i\}} d_i - \sum_{i=1}^{L} y_i\cdot d_i \\ &\leq 2\sum_{\{i|y_i=+1, t\subseteq \mathbf{x}_i\}} d_i - \sum_{i=1}^{L} y_i\cdot d_i \end{aligned}$$

(∵ t の全上位木 $t' \supseteq t$ について $|\{i|y_i = +1, t' \subseteq \mathbf{x}_i\}| \leq |\{i|y_i = +1, t \subseteq \mathbf{x}_i\}|$ が成立するため*.)* 同様に,

$$gain(\langle t', -1\rangle) \leq 2\sum_{\{i|y_i=-1, t\subseteq \mathbf{x}_i\}} d_i + \sum_{i=1}^{L} y_i\cdot d_i$$

以上より, t の全上位木 $t' \supseteq t$, $y \in \{\pm 1\}$ について

$$\begin{aligned} gain(\langle t', y\rangle) &\leq \max\Big(2\sum_{\{i|y_i=+1, t\subseteq \mathbf{x}_i\}} d_i - \sum_{i=1}^{L} y_i\cdot d_i,\ 2\sum_{\{i|y_i=-1, t\subseteq \mathbf{x}_i\}} d_i + \sum_{i=1}^{L} y_i\cdot d_i\Big) \\ &= \mu(t) \quad \blacksquare \end{aligned}$$

最右拡張が定義する探索空間を, 深さ優先探索しながら, 準最適 gain τ と準最適ルールを保持する. ここで, 任意の木 t について, もし $\mu(t) \leq \tau$ ならば, t の全上位木 t' の gain は, τ 以下であるため, t が張る部分空間は安全に枝刈りできる. 逆に $\mu(t) > \tau$ ならば, $gain(t', y) > \tau$ となる上位木 t' が存在する可能性があるため, 枝刈りできない.

また, $\mu(t) > \tau$ となり, ノード s が, 木 t に最右拡張により追加される場合でも, $\mu(s) < \tau$ ならば, 拡張された木 t' が張る部分空間を枝刈りできる.

図 3 にアルゴリズム **Find Optimal Rule** の擬似コードを示す. 上記の 2 つの枝刈りの条件を それぞれ (1) と (2) に記す.

Algorithm: Find Optimal Rule
argument: 学習データ: $T = \{\langle \mathbf{x}_1, y_1, d_1\rangle \ldots, \langle \mathbf{x}_L, y_L, d_L\rangle\}$
(ただし, $\mathbf{x}_i$ は順序木, $y_i \in \{\pm 1\}$ はクラス,
d_i $(\sum_{i=1}^{L} d_i = 1,\ d_i \geq 0)$ は重み)
returns: 最適ルール $\langle \hat{t}, \hat{y}\rangle$
begin
  $\tau = 0$ // 準最適 gain
  function project $(t, loc = \{\langle i_1, j_1\rangle, \ldots\})$
    if $\mu(t) \leq \tau$ **then return (1)**
    $y' = argmax_{y\in\{\pm 1\}}\, gain(\langle t, y\rangle)$
    if $gain(\langle t, y'\rangle) > \tau$ **then**
      $\langle \hat{t}, \hat{y}\rangle = \langle t, y'\rangle$
      $\tau = gain(\langle \hat{t}, \hat{y}\rangle)$ // 準最適
    end

    $idx = \phi$ // hash
    foreach $\langle i, j\rangle \in loc$
      foreach $\langle t', j'\rangle \in \mathbf{RME}(t, j, \mathbf{x}_i)$
        if $\mu(t' - t) \leq \tau$ **then continue (2)**
        $idx[t'] = idx[t'] \cup \{\langle i, j'\rangle\}$
      end
    end
    foreach keys $t' \in idx$
      project$(t', idx[t'])$
    end
  end

  foreach $t' \in \{t | t \in \cup_{i=1}^{L} \mathcal{S}(\mathbf{x}_i),\ |t| = 1\}$
    $loc_{t'} \leftarrow t'$ を含む 事例 i と位置のペアの集合
    project $(t', loc_{t'})$
  end
  return $\langle \hat{t}, \hat{y}\rangle$
end

図 3: アルゴリズム: Find Optimal Rule

3.3 Boosting の高速化

Boosting の逐次計算の度に, 準最適解 τ は 0 にリセットされる. しかし, もしタイトな上限値をあらかじめ見積もることができれば, 探索空間を, より効率良く枝刈りできる. この目的のために, 逐次計算中に発見されたルールをキャッシュに保持していく. 準最適解 τ は, キャッシュ中のルール集合から最大の gain を与えるルールを選択し, 計算する. このアイデアは, Boosting の逐次計算が進むにつれ, 同じルールが繰り返し使われるという我々の観察に基づいている.

3.4 分類の高速化

最終的な分類関数 $f(\mathbf{x})$ は, $2|\mathcal{F}|$ $(= |\mathcal{F}| \times |\{\pm 1\}|)$ 個の仮説の線形結合として表現できる.

$$f(\mathbf{x}) = sgn\Big(\sum_{t\in\mathcal{F}, y\in\{\pm 1\}} w_{\langle t,y\rangle} \cdot h_{\langle t,y\rangle}(\mathbf{x})\Big) \quad (3)$$

各仮説 $h_{\langle t,y\rangle}$ に対する重み $w_{\langle t,y\rangle}(\geq 0)$ が, Boosting の逐次計算で導出される. さらに, 式 (3) は, 以下のような線形分類器に変形できる.

$$\begin{aligned} f(\mathbf{x}) &= sgn\Big(\sum_{t\in\mathcal{F}, y\in\{\pm 1\}} w_{\langle t,y\rangle} \cdot y \cdot (2I(t \subseteq \mathbf{x}) - 1)\Big) \\ &= sgn\Big(\sum_{t\in\mathcal{G}} \lambda_t \cdot I(t \subseteq \mathbf{x}) - b\Big) \quad (4) \end{aligned}$$

ただし,

$$\begin{aligned} \mathcal{G} &= \{t | t \in \mathcal{F}, (w_{\langle t,+1\rangle} - w_{\langle t,-1\rangle}) \neq 0\}, \\ \lambda_t &= 2 \cdot (w_{\langle t,+1\rangle} - w_{\langle t,-1\rangle}), \\ b &= \sum_{t\in\mathcal{G}} (w_{\langle t,+1\rangle} - w_{\langle t,-1\rangle}) \end{aligned}$$

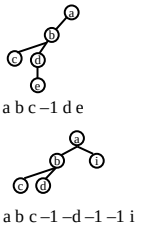
$\mathcal{G}$ は, **分類に必要な素性集合** (以後, **サポート 素性** と呼ぶ) である. 式 (4) の計算量は, 以下の TreeMatcher 問題のそれと等価であるため, ここではこの問題に着目する.

問題 2 *TreeMatcher*
木の集合 $\mathcal{G}$ と, 木 $\mathbf{x}$ が与えられた時, $\mathbf{x}$ 中のすべての部分木のうち, $\mathcal{G}$ に含まれるものを列挙せよ. *(i.e.,* 集合 $\{t | t \subseteq \mathbf{x}\} \cap \mathcal{G}$ を構築せよ*)*

実際に, この問題も, 最右拡張を用いて高速に実現できる. 詳細の前に, 木の文字列表現について言及する.

定義 5 木の文字列表現 *[17]*
木 t を, 以下の方法で一意の文字列表現 $str(t)$ に変換する. *(1)* 初期化 $str(t) = \phi$ *(2)* 前順序探索 *(pre-order)* で, 木のノードを列挙し, 各ノードのラベルを $str(t)$ の末尾に追加する. *(3)* 探索中に, 子から親に後戻りする場合, ラベル集合に含まれない特殊なラベル *-1* を $str(t)$ の末尾に追加する.

このような文字列表現の例を図 4 に示す. さらに, $\mathcal{G}$ 中のすべての木を, 文字列表現に変換し, それらすべてを単一の TRIE で表現する (図 5). 文字列表現中のラベルの順番と, 最右拡張が広げていくラベルの順番は同一なので, 構築された TRIE は, 最右拡張によって定義される「探索木」と同一になる. つまり, 木 $\mathbf{x}$ が与えられると, $\mathbf{x}$ の各ノードに対し, 最右拡張を行いながら, TRIE を探索していくことで, TreeMatcher が実現できる. TreeMatcher の計算量は, TRIE の深さに依存するが, 深さは定数で抑えられるため, $O(|\mathbf{x}|)$ となる. これは Tree Kernel に基づく SVM の計算量に比べ小さい (詳細は次章で説明する).



subtrees	λ
a b c	0.3
a b d	-0.2
a b –1 c	0.5
a d	0.1
b c	0.2
b d –1 e	-0.3

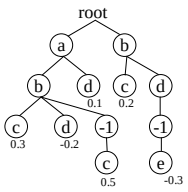


図 4: 文字列表現　図 5: 素性集合の TRIE

4 SVM との関連性

本節では, Tree Kernel[5, 9] を用いた Support Vector Machine (SVM) と提案手法の関連性を 文献 [11] の分析に基づき論じる. 近年の研究 [2, 12, 11] により, Boosting と SVM は本質的に同一であり, 共に **最小マージンの最大化**に基づく学習アルゴリズムとして単一化できることが明らかになった. SVM と Boosting の唯一違いは, マージンのノルムである.

AdaBoost 及び, Arc-GV は, 漸近的に以下の線形計画問題を解いていることが知られている [2, 12, 11].

$$\max_{\mathbf{w}\in\Re^J, \rho\in\Re^+} \quad \rho \quad (5)$$

$$s.t. \quad y_i \sum_{j=1}^{J} w_j h_j(\mathbf{x}_i) \geq \rho \quad (6)$$

$$||\mathbf{w}||_1 = 1, \quad (7)$$

ただし, J は, 仮説の個数である. (Decision Stumps を用いた場合は, $J = |\{\pm 1\} \times \mathcal{F}| = 2|\mathcal{F}|$ となる.) Breiman らは, Arc-GV が上記の問題を漸近的に解いていることを理論的に示している [2]. この理論的漸近性から, 本稿では AdaBoost を用いず Arc-GV を用いることとした.

一方, SVM は, 以下の 2 次計画問題の最適化として定式化される [14] [1].

$$\max_{\mathbf{w}\in\Re^J, \rho\in\Re^+} \quad \rho \quad (8)$$

$$s.t. \quad y_i \cdot (\mathbf{w} \cdot \Phi(\mathbf{x}_i)) \geq \rho \quad (9)$$

$$||\mathbf{w}||_2 = 1. \quad (10)$$

関数 $\Phi(\mathbf{x})$ は, 事例 $\mathbf{x}$ を J 次元空間に写像する関数である (i.e., $\Phi(\mathbf{x}) \in \Re^J$). l_2-norm のマージンの場合, 分離平面の構築には, 素性を陽に展開する必要はなく, 事例間の内積 (Kernel) $K(\mathbf{x}_1, \mathbf{x}_2) = \Phi(\mathbf{x}_1) \cdot \Phi(\mathbf{x}_2)$ さえ与えられればよい. このような考えに基づき, ラベル付き順序木の内積を定義したものが Tree Kernel[5, 9] である. Tree Kernel は, 個々の部分木を個別の素性とする. つまり, 写像関数 $\Phi(\mathbf{x})$ は, $\Phi(\mathbf{x}) = (I(t_1 \subseteq \mathbf{x}), \ldots, I(t_{|\mathcal{F}|} \subseteq \mathbf{x}))$ となる. ここで, 全ルール $\langle t, y\rangle \in \mathcal{F} \times \{+1, -1\}$ それぞれの分類結果を, 個々の次元に対応させるよう, 写像関数 $\Phi(\mathbf{x})$ を再設計してみる $(\Phi(\mathbf{x}) = (2I(t_1 \subseteq \mathbf{x}) - 1, \ldots, 2I(t_{|\mathcal{F}|} \subseteq \mathbf{x}) - 1, -2I(t_1 \subseteq \mathbf{x}) + 1, \ldots, -2I(t_{|\mathcal{F}|} \subseteq \mathbf{x}) + 1))$.

[1] 議論を単純にするために, バイアス項 (b), ソフトマージンは考えない

全弱学習器を個々の次元とする素性性空間 (仮説素性空間) は, 全部分木を用いる Tree Kernel の基礎となる概念を変えない. つまり, 部分木を素性とする Decision Stumps と Tree Kenrnel に基づく SVM は, 素性空間という観点において, 本質的に同一であるといえる.

2 つのアルゴリズムの唯一の違いは, $||\mathbf{w}||$ のノルムである (Boosting は l_1, 式 (7), SVMs は l_2, 式 (10)). 文献 [11] では, このノルムの違いを, モデルのスパース性という観点で議論している. 詳細は [11] に譲るが, SVM は, **事例スパース**の解を導出することが知られている. つまり, できるだけ少数の事例で $\mathbf{w}$ を表現しようとする. SVM の分離平面 ($\mathbf{w}$) は, 学習事例の線形結合で与えられる ($\mathbf{w} = \sum_{i=1}^{L} \lambda_i \Phi(\mathbf{x}_i)$). 非 0 の係数が付いた事例が, **サポートベクター** (分離平面を支える事例) と呼ばれるのは上記の理由からである. 一方, Boosting は, **素性スパース**の解を導出する. つまり, できるだけ少数の素性で $\mathbf{w}$ を表現しようとする.

SVM と Boosting を精度という観点で比較することは, タスク依存性が強いため, ここでは行わない. しかし, Boosting を用いる提案手法は以下のような「現実的な」利点があると考える.

- 解釈のしやすさ
  できるだけ少数の素性で分類し, その分類自身が陽に実行されるため, どのような素性が分類に使用されているか, 分類に寄与する素性は何か, といった分析が行いやすい. SVM では, 素性空間が陰に表現されるため, そのような分析を与えにくい.
- 高速な分類
  スパースな素性空間は, 高速な分類を可能にする. Tree Kernel の計算量は, $O(|N_1||N_2|)$ である (ただし N_1 と N_2 は内積をとる 2 つの木である). さらに Kernel 法に基づく分類アルゴリズムはサポートベクターの数 L' に依存する. つまり, SVM の分類コストは, $O(L'|N_1||N_2|)$ となり非常に大きい. この問題は, 実際の応用において障壁になっている.

5 実験と考察

5.1 実験環境, 設定

実験は, 以下の 3 種類のタスクで行った.

- MEDLINE 文の役割分類 (**MEDLINE**)
  MEDLINE には, 構造化アブストラクトと呼ばれる, 各段落の先頭にその段落の役割 (背景, 目的など) が明記されたものが含まれる. これらのデータを用い, 与えられた文がどの役割の段落に含まれているかを同定するタスクを考える (詳細は, 文献 [15] を参照). 分類単位は「文」, 文数は 2 万, カテゴリは「背景」「目的」「手法」「結果」「結論」の 5 つである.
- PHS レビュー分類 (**PHS**)
  PHS ユーザに, 良い点/悪い点を区別してレビューを投稿するよう指示した掲示板のデータである. 分類単位は「文」, 文数は 5,741, カテゴリは「良い点」「悪い点」の 2 つである.
- 文のモダリティ分類 (**MOD**)
  被験者 1 名が, 田村ら [18] の大分類に従い, 99 年毎日新聞の社説からランダムに選んだ 60 記事に, 文単位のモダリティ付与したデータである. 分類単位は「文」, 文数は 1,710, カテゴリは, 「意見」「断定」「叙述」の 3 つである.

3 つのタスクのデータの一例を図 6 に示す.

文の表現方法として, 以下の 3 つを用いた

- Bag-of-Word (**bow**), ベースライン
  構造は考慮せず, 単語の有無を素性とする. 英語は Charniak Parser[3] を, 日本語は ChaSen[2]を用いて単語を切り出す. 日本語のみ, 単語の表層ではなく, 原形を用いた. Boosting を用いた提案手法では, サイズ 1 の構造のみが素性集合となる.
- 係り受け (**dep**)
  単語単位の係り受け構造として文を表現する. 英語は, Charniak Parser と文献 [4] の主辞規則を用い単語単位の係り受けに変換する. 日本語は, CaboCha[3]を使い, 文節単位の係り受け構造を導出した後, 単語 (原形) 単位の係り受け構造に変換する. 基本的に, 文節中の単語は, 直後の単語に係け, 文節中の最後の単語は, 係り先文節の主辞となる単語に係ける. ただし, 文頭, 文末にはダミーのノードを置く.
- N-グラム (**ngram**)
  各単語 (原形) の係り先が, 直後の単語であるとみなした係り受け木である. 任意の部分木は, 単語 N-グラムになる.

これらのデータ, 文の表現方法を用い, 提案手法 (Boosting) と, Tree Kernel による SVM の比較実験を行った. Tree Kernel には, 文献 [9] で提案されている部分木のサイズに対する事前分布や, 親子間の伸縮等は考慮せず, 全部分木に展開するオリジナルの Tree Kernel を用いた. これは, 学習に使われる素性を同一にし, 比較を公平にするためである. SVM のソフトマージンパラメータ, 及び Boosting の繰り返し回数は, それぞれ適当に変化させながら, 最良の結果を採用した. 複数クラスの分類には, 2 値分類器を多値分類問題へ拡張する手法の 1 つである one-vs-rest を用いた. 評価は, 5-fold 交差検定により得られた F 値の平均にて行う.

5.2 実験結果

表 1,2,3 に, MEDLINE,PHS,MOD の実験結果を示す. B は, 提案手法 (Boosting) は, S は, SVM を用いた結果である.

[2] http://chasen.org/
[3] http://cl.aist-nara.ac.jp/~ taku-ku/software/cabocha/

MEDLINE
背景: It has been suggested that bipyridines or their derivatives may have a selective pulmonary ...
目的: To investigate the perceptions of patients about delivery of their babies.
手法: Meta-analysis was used to analyze data obtained from a search ...
結果: Review 486 courses of ECT yielded a total of 3936 treatments.
結論: The data suggest that postmenopausal estrogen treatment reduces the risk for angiographically....

PHS
良い点: メールを送受信した日付、時間が表示されるのも結構ありがたいです。
悪い点: なんとなく、レスポンスが悪いように思います。

mod
断定: 「ポケモン」の米国での成功を単純に喜んでいてはいけない。
意見: その論議を詰め、国民に青写真を示す時期ではないのか。
叙述: バブル崩壊で会社神話が崩れ、教育を取り巻く環境も変わった。

図 6: データの例

表 1: 結果 (MEDLINE)

	背景	目的	方法	結果	結論
B/bow	53.7	58.2	80.9	78.0	58.6
B/dep	55.1	60.2	82.2	80.4	60.5
B/ngram	56.1	66.1	82.5	79.7	61.4
S/bow	53.3	59.4	81.8	79.1	57.8
S/dep	26.7	8.6	71.0	70.0	22.3
S/ngram	41.5	65.7	79.8	76.4	43.8

表 2: 結果 (PHS)

	F 値
B/bow	76.6
B/dep	79.0
B/ngram	79.3
S/bow	77.2
S/dep	77.2
S/ngram	79.4

表 3: 結果 (MOD)

	断定	意見	叙述
B/bow	71.2	62.1	83.0
B/dep	87.5	80.5	91.9
B/ngram	87.6	78.4	91.9
S/bow	72.1	59.2	82.5
S/dep	81.7	26.1	88.1
S/ngram	81.7	26.1	88.1

5.3 考察

5.3.1 構造を考慮する有効性

全タスクにおいて, 構造を用いないベースライン (bow) に比べ, 構造を考慮する提案手法が顕著に精度が高いことが確認できる. 係り受け (dep) と N-グラム (ngram) を比較すると, タスク依存性が強く, 顕著な差は確認できない.

5.3.2 提案手法 vs SVM + Tree Kernel

bow を素性とした場合, Boosting と SVM には顕著な差は確認されない. しかし, SVM を用いた場合, タスクによっては著しく精度が悪い. これらの「顕著な」精度低下の要因として, Tree Kernel をはじめとする Convolution Kernel 全体が持つ欠点が挙げられる. 言語データといった疎データに Convolution Kernel を用いると, 素性数が指数的に増えるため, 自分以外との内積と比較して, 自分自身との内積が極めて大きくなる傾向にある. これにより, 学習事例に酷似した事例のみを分類し, 残りはデフォルトクラスに分類するような, 丸暗記学習が行われやすくなる. これを回避する方法として, 1) 大きな部分構造の重みを減衰させる [9], 2) 正規分布の分散と同形の平滑化を与える [7], 3) カイ 2 乗値により素性選択する [13], といった手法が提案されている. これらの手法を用いることで, 精度という意味では, 提案手法を上まわる可能性はある. しかし, 使用される素性そのものが大きく変わり, 減衰率といったハイパーパラメータの設定方法についての議論の余値が残るため, 公平な比較ができない.

5.3.3 提案手法の利点

4 章にて, 提案手法の「現実的な」利点について述べた. ここでは, 実験結果 (PHS) から, それらの利点を検証する.

提案手法は, 必要最小限の素性を自動的に選択する能力を備える. PHS タスクにおいて, 実際に使われた素性 (サポート素性) の数は, 1,793 であり, 人手による分析に耐えうるサイズであった. 学習データから抽出した全 1-gram, 2-gram, 3-gram の異なり数がそれぞれ 4,211, 24,206, 43,658 であることから, いかに少数の素性で分類を行ってるかが分かる. もし, SVM のサポートベクターの集合から, サポート素性を列挙すると, 数十万から数百万の素性数になると予想される.

サポート素性の分析の例として, 図 7 に, 個々のサポート素性 $t \in \mathcal{G}$ と, Boosting が算出した素性重み λ_t の一部を示す.

(A)「(〜し) にくい」を含む素性
「(〜し) にくい」は, 一般に否定的な意味の形容詞であり, 大くの素性は負 (悪い点) の重みが与えらえている. しかし,「切れにくい」のみ正の重みとなり, ドメイン知識 (PHS) をよく反映している.

(B)「使う」を含む素性
「使う」は, 評判には中立な意味の単語だが, 周辺のコンテキストで重みが変化している (「使いたい」→正,「使いやすい」→正). さらに, 過去形 (「使いやすかった」) になったり, 他の要素との比較 (「(〜の/〜する) 方が使いやすい」) になると, 負の重みが与えられており, 興味深い.

(C)「充電」を含む素性
ドメイン知識を反映した素性 (「充電時間が短い」→正,「充電時間が長い」→負) が抽出されている. また, これらは, 係り受け構造を考慮した素性である[4].

さらに, 図 8 に, 分類の実行例を示す. 入力文「液晶が大きくて, 綺麗, 見やすい」に対し, どういう素性が適用されたか陽に出力し, 分析を容易にしている. このような分析は, Tree Kernel では困難である.

最後に, 分類速度であるが, 提案手法の分類速度が 0.135 秒/1,149 事例に対し, SVM は, 57.91 秒/1,149 事例 であった. 提案手法が, およそ 400 倍高速であることが確認された.

[4]「充電時間がとても短い」は, 係り受けパターンならマッチできるが, ngram だとできない.

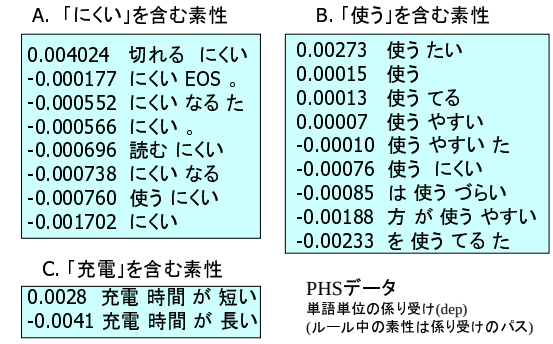


図 7: サポート素性の一部

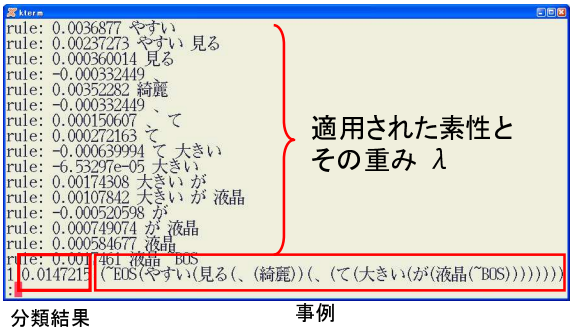


図 8: 分類の実行例
(注: 前順序の S 式で木を表現しているため, 語順が逆になる)

6 おわりに

本稿では, テキストの構造 (構文構造やレイアウト構造) を考慮したテキスト分類 (半構造化テキスト分類) に向け, 部分木を素性とする Decision Stumps と, それらを弱学習器として用いる Boosting アルゴリズムを提案した. さらに, 部分木列挙アルゴリズムを拡張し, 弱学習器の効率良い学習/分類手法を提案した. 実データを用いた実験にて, 文の構造を考慮する本手法の有効性を示した. さらに, 提案手法の 2 つの利点 (解釈のしやすさ, 高速分類) を実タスクにおいて確認した.

今後の計画として, 以下が挙げられる.

- 他のタスク
  本手法は, 言語処理全般に適用できる汎用的な手法である. 今後はテキスト分類だけではなく, 係り受け解析, 構文解析といった言語解析タスクでの有効性を検証したいと考える.
- グラフ構造
  これまでは, 木構造のみに着目していたが, より複雑な情報を表現するには, グラフを用いるほうがよい. 効率良いグラフ列挙アルゴリズムが提案されているので [16], 今後は, それらを適用したいと考える.

謝辞

MEDLINE のタスク設定について, 奈良先端大学の山崎貴宏氏に助言をいただきました. また, 文のモダリティ判別データを NTT コミュニケーション科学基礎研究所の平尾努氏に提供していただきました. ここに感謝いたします.

参考文献

[1] Kenji Abe, Shinji Kawasoe, Tatsuya Asai, Hiroki Arimura, and Setsuo Arikawa. Optimized substructure discovery for semi-structured data. In *Proc. 6th European Conference on PKDD*, 2002.

[2] Leo Breiman. Prediction games and arching algoritms. *Neural Computation*, 11(7):1493 – 1518, 1999.

[3] Eugene Charniak. A maximum-entropy-inspired parser. In *In Proc. of NAACL*, pages 132–139, 2000.

[4] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[5] Michael Collins and Nigel Duffy. Convolution kernels for natural language. In *Proc. of Neural Information Processing Systems (NIPS)*, 2001.

[6] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sicences*, 55(1):119–139, 1996.

[7] David Haussler. Convolution kernels on discrete structures. Technical report, UC Santa Cruz (UCS-CRL-99-10), 1999.

[8] Roberto J. Bayardo Jr. Efficiently mining long patterns from databases. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*. ACM Press, 1998.

[9] Hisashi Kashima and Teruo Koyanagi. Svm kernels for semi-structured data. In *Proceedings of the ICML-2002*, pages 291–298, 2002.

[10] Shinichi Morhishita. Computing optimal hypotheses efficiently for boosting. In *Progress in Discovery Science*, pages 471–481. Springer, 2002.

[11] Gunnar. Rätsch. *Robust Boosting via Convex Optimization*. PhD thesis, Department of Computer Science, University of Potsdam, 2001.

[12] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proc. 14th International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann, 1997.

[13] Jun Suzuki, Tsutomu Hirao, Hideki Isozaki, and Eisaku Maeda. String kernel with feature selection function (in Japanese). In *IPSJ SIG Technical Reports 2003-FI-72/2003-NL-157*, 2003.

[14] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[15] Takahiro Yamasaki, Masashi Shimbo, and Yuji Matsumoto. Automatic classification of sentences using sequential patterns (in Japanese). In *Technical Report of IECE AI2002-83*, 2003.

[16] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proc. of ICDM*, pages 721–724, 2002.

[17] Mohammed Zaki. Efficiently mining frequent trees in a forest. In *Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining KDD*, pages 71–80, 2002.

[18] 和田啓二 田村直良. セグメントの分割と統合による文章の構造解析. **自然言語処理**, 5(1), 1996.