半構造化テキストの分類のためのブースティングアルゴリズム

工藤 拓[†,☆] 松本 裕治[†]

近年，テキスト分類は，単純なトピック分類から，文のモダリティ，意見性，主観性といった書き手の意図に基づく分類へと，そのタスクの多様化が進んでいる．それにともない，単語の集合（bag-of-words）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，およ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．

A Boosting Algorithm for Semi-structured Text Classification

Taku Kudo[†,☆] and Yuji Matsumoto[†]

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, 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 subtrees 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. Two 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.

1. はじめに

SVM や Boosting 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[1),2)]．テキスト分類では，一般に単語の集合（bag-of-words 以下 BOW）を素性として用い，政治，経済，スポーツといった粒度の粗い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，早期に危機の芽や要求を発見，分析するための要素技術が求められている．たとえば，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要素の発見といったタスクがそれらにあたる．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は用いられず，主観的/客観的に述べているのか，（ある製品を）誉めている/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．精度向上には，単語の並び（可変長の N-グラム），係り受け関係，特徴的言語パターンといったテキストの部分構造を考慮する必要がある．

分類において構造が重要となるタスクは自然言語処理分野だけにとどまらない．その背景には，テキストが任意の構造を内包する「構造化テキスト」，特に自明なスキーマがなく要素の追加や削除が比較的自由に行

† 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology
☆ 現在，日本電信電話株式会社 NTT コミュニケーション科学基礎研究所
Presently with NTT Communication Science Laboratories, NTT Corporation

える「半構造化テキスト」の普及がある．XML 文書は半構造化テキストの代表例であり，任意の構造が汎用的な手続きによって表現される．XML のサブセットである HTML は，Web 文書のレイアウトやスタイルといった構造を規定するための言語であり，HTML 文書はそれらの構造を内包する．半構造化テキストの普及により，それらに対する検索，分類，情報抽出といった機械処理の需要が高まりつつある．形態素，構文解析済の自然言語テキストも一種の半構造化テキストであり，統語的，意味的な構造がテキスト中に陽に表現される．このような一般的な視点に立つと，自然言語テキストの機械処理は，半構造化テキストの機械処理の一応用ととらえることができる．

半構造化テキストの機械処理，特に学習，分類において BOW（一般には任意のアイテムの集合）といった素朴な素性を用いることは，テキストが内包する構造を無視していることにほかならない．特に，構文情報やレイアウトといった構造がテキストの分類に寄与する場合，BOW では十分な精度が達成できないであろう．単純にはどのような部分構造が分類に寄与するか人手により調査し，それらを素性として用いることである程度の精度向上が期待できる．しかし，分類に有効な部分構造が事前に分からないことが多く，人手だけでは有効な素性を取りこぼす可能性がある．理想的な手法とは，半構造化テキストを直接処理できる手法，つまり半構造化テキストから有効な部分構造を自動的に抽出し，学習，分類を行う手法であろう．本稿では，このような考えに基づき，半構造化テキストのための学習，分類アルゴリズムを提案する．具体的な構造として，本稿ではラベル付き順序木を考える．ラベル付き順序木は XML，HTML，単語の並び，構文木などの多くのデータをモデル化できる一般的なデータ構造である．本稿で提案する学習，分類アルゴリズムは，以下の特徴を持つ．

(1) テキストの構造を考慮した学習，分類が可能．
(2) 学習の素性に，構造の部分構造（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．
(3) 分析を容易にするために，分類に必要な最小限の部分構造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． □

XML/HTML，単語の並び，係り受け構造は，ラベル付き順序木で定式化できる．XML/HTML では，ラベルは XML/HTML のタグや属性に対応する．言語データでは，ラベルは単語や文節となる．兄弟関係の順序が与えられるため，言語データにおける語順の違いは区別される．たとえば「太郎が本を読む」と「本を太郎が読む」はそれぞれ別々の木となる．

単語の並びは，ラベル付き順序木の特殊形であるラベル付き系列でも表現可能である．しかし，単語そのものが品詞や意味属性といった付加情報を持つ場合，系列では不十分である．たとえば，(名詞 (魚)(助詞 (が)(動詞 (好き)))) は，単語の並びには変わりないが，単語は品詞と語彙から構成されているために系列では表現できない．言語処理においてこのような付加情報を用いる応用は少なくないために，ラベル付き系列としての定式化は行わず，より一般的なラベル付き順序木を考える．

定義 2 部分木
t と u を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 のノード間に，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，親子関係を保存する．(2) ψ は，兄弟の順序関係を保存する．(3) ψ はラベルを保存する． □

本稿では，ラベル付き順序木を単純に順序木，もしくは木と呼ぶ．さらに，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．**図 1** にラベル付き順序木と部

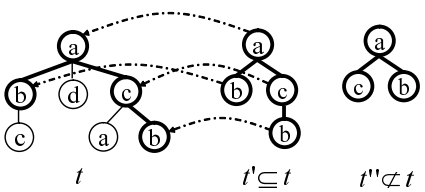

図 1 ラベル付き順序木，部分木
Fig. 1 Labeled ordered tree and subtree relation.

分木の例を示す．

2.2 ラベル付き順序木の分類問題

ラベル付き順序木の分類問題（木の分類問題）とは，L 個の学習データ $T = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{L}$ から，分類関数 $f(\mathbf{x}) : \mathcal{X} \to \{\pm 1\}$ 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，単純なアルゴリズムである．Decision Stumps は，深さ 1 の Decision Tree（決定木）と同一である．Schapire らは，個々の単語を 1 つの素性とする Decision Stumps を用い，テキスト分類を行っている[2]，本稿では，ラベル付き順序木の分類問題に対し，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．

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

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**ルール**と呼ぶ．

Decision Stumps の学習は，学習データ $T = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{L}$ 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．

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

ただし，$\mathcal{F}$ は，全部分木の集合（**素性集合**）である（$\mathcal{F} = \bigcup_{i=1}^{L} \mathcal{S}(\mathbf{x}_i)$）．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．

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

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

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

Algorithm: Arc-GV
argument: 学習事例
$T = \{\langle \mathbf{x}_1, y_1 \rangle, \ldots, \langle \mathbf{x}_L, y_L \rangle\} \subset \mathcal{X} \times \{\pm 1\}$
繰返し回数 K
returns: 分類器 $f(\mathbf{x})$
begin
$d_i^{(1)} = 1/L, \quad \rho_i^{(0)} = 0 \ \ \forall i = 1, \ldots, L$
foreach $k = 1, \ldots, K$
$\langle t_k, y_k \rangle = \mathrm{argmax}_{t \in \mathcal{F}, y \in \{\pm 1\}} gain(\langle t, y \rangle)$
$\rho^{(k-1)} = \min_{i=1,\ldots,L} \rho_i^{(k-1)}$
$\alpha_k = \frac{1}{2} \log(\frac{1+gain(\langle t_k, y_k \rangle)}{1-gain(\langle t_k, y_k \rangle)}) + \frac{1}{2} \log(\frac{1-\rho^{(k-1)}}{1+\rho^{(k-1)}})$
$\beta_k = \sum_{j=1}^{k} \alpha_j$
$\rho_i^{(k)} = \sum_{j=1}^{k} \alpha_j h_{\langle h_j, y_j \rangle}(\mathbf{x}_i)/\beta_k$
$d_i^{(k+1)} = \exp(-y_i \rho_i^{(k)} \beta_k)/Z$ (Z は正規化項)
end
returns $f(\mathbf{x}) = sgn(\sum_{k=1}^{K} \alpha_k h_{\langle t_k, y_k \rangle}(\mathbf{x})/\beta_K)$
end

図 2 アルゴリズム：Arc-GV
Fig. 2 Algorithm: Arc-GV.

本稿では，エラー率最小のかわりに，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[3] 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．$\mathbf{d}^{(k)} = (d_i^{(k)}, \ldots, d_L^{(k)})$，（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[3]．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．

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

AdaBoost[3] は Boosting の代表的なアルゴリズムであるが，本稿では Breiman らによる Arc-GV[4] を用いる（Arc-GV を用いる理由は，4 章で説明する）．Arc-GV の擬似コードを**図 2** に示す．Arc-GV では，Boosting が算出する弱学習器の重み α_k が分類器の**マージン** $\rho^{(k-1)}$ に依存する．以下断りがない限り

Boosting は Arc-GV のことを指すこととする．

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）が与えらえたとき，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[5]．

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．基本的には，分枝限定法（branch-and-bound）の考え方と同一である．

(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いることと変わりない．そこで，gain の上限値を見積もり，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[6),7)]．最右拡張は，部分集合を列挙するアルゴリズム Set Enumeration Tree[8)] の部分木への自然な拡張となっている．最右拡張では，まず，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．

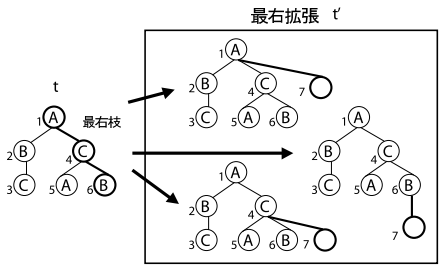

図 3　最右拡張
Fig. 3　Right-most-expansion.

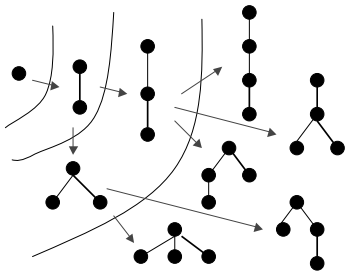

図 4　最右拡張の再帰（探索木）
Fig. 4　Recursion of right-most-expansion.

定義 4　最右拡張[6)]

木 t に 1 つのノード s を追加して得られる順序木 t' を，木 t の最右拡張と定義する．ただし，ノードの追加には以下の制約がある．(1) ノードは，t の最右枝（ルートからつねに右端のノードを葉まで選ぶ）に追加される．(2) 追加されるノードは，最右の兄弟（末弟）である．□

図 3 に最右拡張の例を示す．便宜上，木 t は前順序（pre-order）でノードに順序が付与されているものとする．図 3 の例では，位置 7 のノードが最右拡張で追加されるノードとなる．また，追加される位置の候補は 3 種類（位置 1，4，6）あるが，ラベルの候補集合を考えると，ラベルの種類数 × 位置数（図 3 では，$|\{A, B, C\}| \times 3 = 9$）通りの木 t' が，最右拡張より生成される．

さらに，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 4** に，その探索木の例を示す．図 4 中では，便宜的にラベルの種類を 1 種類のみとしている．この探索木を探索することで，全部分木を，完全に重複なく列挙できる．

最右拡張は木の列挙法であるため，木の集合 T から部分木を列挙する目的として単純に用いると，T の部分木にはならない木を列挙してしまう可能性がある．単純には，部分木になるものだけを最右拡張の候補にすればよい．詳細は，文献 6), 7) に譲る．

3.2 探索空間の枝刈り

前節で定義した探索空間を分枝限定法（branch-and-bound）の考えに従い枝刈りする．本節では，その具体的な手法について述べる．定理1（Morishita[9]の拡張）は，t の全上位木 $t', (\forall t' \supseteq t)$ 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'，$(\forall t' \supseteq t)$，$y \in \{\pm 1\}$ について，ルール $\langle t', y\rangle$ の gain は $\mu(t)$ を上限値とする（$gain(\langle t' y\rangle) \leq \mu(t)$）．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

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

ここで，$y = +1$ の場合に着目すると

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

（$\because$ t の全上位木 $t', (\forall t' \supseteq t)$ 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

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

□

Algorithm: Find Optimal Rule
argument: $T = \{\langle \mathbf{x}_1, y_1, d_1\rangle \ldots, \langle \mathbf{x}_L, y_L, d_L\rangle\}$
（$\mathbf{x}_i$ は木，$y_i \in \{\pm 1\}$ 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
    τ = 0  // 準最適 gain
    function project (t)
        if μ(t) ≤ τ then return
        y' = argmax_{y∈{±1}} gain(⟨t, y⟩)
        if gain(⟨t, y'⟩) > τ then
            ⟨t̂, ŷ⟩ = ⟨t, y'⟩
            τ = gain(⟨t̂, ŷ⟩)   // 準最適
        end

        RME = φ
        foreach t' ∈ {t の最右拡張 }
            s = 最右拡張で追加されるノード
            if μ(s) ≤ τ then continue
            RME = RME ∪ t'
        end

        foreach t' ∈ RME
            project(t')
        end
    end

    foreach t ∈ { サイズ 1 の部分木 (全ノード)}
        project (t)
    end
    return ⟨t̂, ŷ⟩
end
```

図 5 アルゴリズム：Find Optimal Rule
Fig. 5 Algorithm: Find Optimal Rule.

定理1より，gain の上限値が与えられるので，分枝限定法を用い，探索空間を枝刈りすることができる．具体的には，まず，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（これまで探索した中で最大の gain）τ と，それに対応する準最適ルールを保持する．もし，$gain(\langle t, y\rangle) > \tau$ となるようなルールが探索中に見つかれば，準最適ルール，準最適解をそれぞれ更新する．

ここで，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，$gain(\langle t', y\rangle) > \tau$ 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 5 にアルゴリズム **Find Optimal Rule** の擬似コードを示す．

3.3 Boosting の高速化

Boosting の逐次計算のたびに，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

Boosting を用いた場合，最終的な分類器は K 個の仮説の線形結合となる．

$$f(\mathbf{x}) = sgn\Big(\sum_{k=1}^{K} \alpha_k h_{\langle t_k, y_k\rangle}(\mathbf{x})\Big). \qquad (3)$$

ただし，α_k は，Boosting が出力する k 番目の仮説に対する重みである．$h_{\langle t,y\rangle}(\mathbf{x})) = 2 \cdot y \cdot (I(t \subseteq \mathbf{x}) - 1)$ に注意すると，式 (3) は，以下のような線形分類器に変形できる．

$$\begin{aligned} f(\mathbf{x}) &= sgn\Big(\sum_{k=1}^{K} \alpha_k \cdot y_k (2I(t_k \subseteq \mathbf{x}) - 1)\Big) \\ &= sgn\Big(\sum_{t \in \mathcal{G}} \lambda_t \cdot I(t \subseteq \mathbf{x}) - b\Big), \qquad (4) \end{aligned}$$

ただし，

$$b = \sum_{k=1}^{K} y_k \alpha_k, \quad \lambda_t = \sum_{\{k | t = t_k\}} 2 \cdot y_k \cdot \alpha_k$$

である．$\mathcal{G}$ 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．λ_t は，各素性に対応する**重み**である．$-b$ はバイアス項であり，ルールが適用されなかったときのデフォルト値に対応する．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 $\mathbf{x}$ が与えられたとき，$\mathbf{x}$ 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（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[7)]

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． □

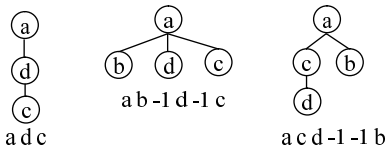

図 6 文字列表現
Fig. 6 String encoding.

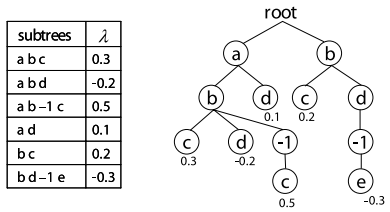

図 7 素性集合の TRIE
Fig. 7 Feature set in TRIE.

このような文字列表現の例を**図 6** 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 7**）．文字列表現中のノードの順番と，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）．

4. SVM との関連性

本章では，Tree Kernel[10),11)] を用いた Support Vector Machine（SVM）と提案手法の関連性を文献 12) の分析に基づき論じる．

AdaBoost および，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[4),12),13)]．

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

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

$$||\mathbf{w}||_1 = 1. \qquad (7)$$

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 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[4]．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．

$$\max_{\mathbf{w}\in\mathbb{R}^J,\rho\in\mathbb{R}^+} \quad \rho \tag{8}$$

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

$$||\mathbf{w}||_2 = 1. \tag{10}$$

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（$\Phi(\mathbf{x}) \in \mathbb{R}^J$）．$l_2$-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[10),11)] 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☆2．

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（$\Phi(\mathbf{x}) = (2I(t_1 \subseteq \mathbf{x}) - 1,\ldots,2I(t_{|\mathcal{F}|} \subseteq \mathbf{x}) - 1, -2I(t_1 \subseteq \mathbf{x}) + 1,\ldots,-2I(t_{|\mathcal{F}|} \subseteq \mathbf{x}) + 1)$）．全弱学習器を個々の次元とする素性空間（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☆3．このような再設計を与えると，Boosting の制約 (6) は SVM の制約 (9) と同一になる．つまり，部分木を素性とする Decision Stumps と Tree Kernel 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．換言すると，部分木を「素性」として表現するか，「弱学習器」として表現するかの違いといえる．

2 つのアルゴリズムの違いは，$||\mathbf{w}||$ のノルムである（Boosting は l_1，式 (7)，SVMs は l_2，式 (10)）．文献 12) 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．詳細は文献 12) 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，スパース素性の性質から，以下のような「現実的な」利点がある．

- 解釈のしやすさ
  できるだけ少数の素性で分類し，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 2 種類のタスクで行った．

- 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[15)] 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」，カテゴリは，「意見」「断定」「叙述」の 3 つ，それぞれの文数は 160，989，561 である．

3 つのタスクのデータの一例を**表 1** に示す．

文の表現方法として，以下の 3 つを用いた．

- bag-of-words（**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．具体的には，まず，ChaSen☆4 を用いて単語を切り出

☆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

☆2 Tree Kernel の正確な定義に従えば，各部分木の頻度を考える必要があるが，テキスト分類のようにラベルの種類が多いとき（スパースのとき）は，このような二値ベクトルで近似できる．

☆3 元の空間をアフィン変換しているので，素性として表現される情報そのものは変わらない．

☆4 http://chasen.naist.jp/

表 1 データの例
Table 1 Examples of data set.

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	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。
	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。
	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。

表 2 実験結果，PHS，MOD（F 値，精度 %，再現率 %）
Table 2 Results of experiments on PHS and MOD (F-measure, precision %, and recall %).

		PHS	MOD		
			意見	断定	叙述
Boosting	bow	76.0 (76.1 / 75.9)	59.6 (59.4 / 60.0)	70.0 (70.4 / 69.9)	82.2 (81.0 / 83.5)
	dep	78.7 (79.1 / 78.4)	78.7* (90.2 / 70.0)	86.7* (88.0 / 85.6)	91.7* (91.1 / 92.4)
	n-gram	79.3 (79.8 / 78.5)	76.7* (87.2 / 68.6)	87.2 (86.9 / 87.4)	91.6 (91.0 / 92.2)
SVMs	bow	76.8 (78.3 / 75.4)	57.2 (79.0 / 48.4)	71.3 (64.3 / 80.0)	82.1 (82.7 / 81.5)
	dep	77.0 (80.7 / 73.6)	24.2 (95.7 / 13.8)	81.7 (86.7 / 77.2)	87.6 (86.1 / 89.2)
	n-gram	78.9 (80.4 / 77.5)	57.5 (98.0 / 40.9)	84.1 (90.1 / 78.9)	90.1 (88.2 / 92.0)

す．ただし，単語の表層形ではなく，原形を用いた．提案手法では，サイズ 1 の構造のみが素性集合となる．これは Boostexter[2] と同一である．

- 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．具体的には，CaboCha☆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，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るものとする．また，文頭，文末にはダミーのノードを置く．
- 可変長 N-グラム（**n-gram**）
  各単語（原形）の係り先が，直後の単語であると見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．bi-gram や tri-gram といった固定長の N-グラムとは異なり，長さの制限はないことに注意されたい．

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 11) で提案されている部分木のサイズに対する事前分布や，親子間の伸縮などは考慮せず，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．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 値の平均によって行う．SVM のソフトマージンパラメータ，および Boosting の繰返し回数は，それぞれ適当に変化させながら，テストデータに対する精度を最良にする値を採用した☆☆．すなわち，それぞれ最良かつ公平な条件での比較を行う．システムは C++で実装し，全実験は，XEON 2.4 Ghz，主記憶 4.0 Gbyte の Linux 上で行った．

5.2 実験結果

表 2 に，PHS，MOD の実験結果を示す．ただし，下線の結果は，ベースライン（bow）に対し有意差があることを，星印の結果は，同一素性を用いた SVM に対し有意差があることを示す．検定には，対応がとれている場合の母比率の差を検定する手法であるマクネマー検定[16] を用い，有意水準は 1%としている．

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の精度が高いことが確認できる．係り受け（dep）と N-グラム（n-gram）を比較すると，MOD タスクの「意見」カテゴリにおいて，係り受け構造を用いるほうが若干精度が良いが，全般的に顕著な差は確認できなかった．

5.3.2 Tree Kernel を用いた SVM との比較

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．dep，n-gram を用いた場合でも，提案手法が同一かそれ以上の精度を示している．しかし，SVM の場合，カテゴリによっては著しく精度が悪い．このような顕著な精度低下の要因として，Tree

☆ http://chasen.naist.jp/~ taku/software/cabocha/

☆☆ これらのパラメータは，正則化の強さをコントロールするという意味で同種のものである．

表 3 枝刈り率（PHS）
Table 3 Pruning rates in PHS dataset.

	枝刈り ($\times 10^6$)	呼び出し ($\times 10^6$)	枝刈り率
dep	3809	4018	0.94
dep (NB)	3811	4020	0.94
n-gram	3108	3308	0.93
n-gram (NB)	3110	3311	0.93

Kernel をはじめとする Convolution Kernel 全体が持つ欠点があげられる．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きわ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[11]，2) 正規分布の分散と同形の平滑化を与える[17]，3) カイ 2 乗値により素性選択する[18]，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，提案手法より高い精度を得る可能性がある．しかし，これらの手法の目的は，あくまでも精度向上であり，提案手法の持つ利点「高速分類」，「解釈のしやすさ」の実現ではない．また，学習の素性空間が大きく変わってしまうため，素性空間が同一という条件での公平な比較ができない．さらに，減衰率といった設定すべきパラメータの数が増えるという意味で問題が複雑になってしまう．

5.3.3 高速化の効果

分枝限定法に基づく学習アルゴリズムの効果をここで検証する．

枝刈りをまったく行わない場合，探索空間が指数的に増加するため，現実問題として実行不可能である．そのため枝刈りを行わないアルゴリズムとの直接的な比較は行えない．そこで，実際にどれぐらいの割合で枝刈りが実施されるかを調査することで，本手法の有効性を検証する．枝刈り率は（枝刈りが行われた回数/関数 project が呼ばれる回数）として計算できる．

表 3 に Boosting の繰返し回数を 10 万回に固定した場合の PHS データにおける枝刈り率をまとめる．ただし，NB（=No Boosting）は，3.3 節で述べた Boosting の高速化を行わなかった場合の枝刈り率である．結果，枝刈り率は 93〜94%程度で探索空間の大部分を効果的に枝刈りしていることが分かる．一方，事前に高い準最適解を求めておく Boosting の高速化手法は，呼び出し回数を若干低減することはできたが，その効果は期待していたほど大きくなかった．これは，最適値に近いルールが多数あり，事前に準最適解を求めなくとも探索の比較的早い段階で準最適解が更新されることを示唆している．

表 4 サポート素性の一部
Table 4 Examples of support features.

キーワード	重み λ_t	部分木 t（サポート素性）
A. にくい	0.00402	切れる にくい
	−0.00055	にくい なる た
	−0.00056	にくい 。
	−0.00069	読む にくい
	−0.00073	にくい なる
	−0.00076	使う にくい
	−0.00170	にくい
B. 使う	0.00273	使う たい
	0.00015	使う
	0.00013	使う てる
	0.00007	使う やすい
	−0.00010	使う やすい た
	−0.00076	使う にくい
	−0.00085	は 使う づらい
	−0.00188	方 が 使う やすい
	−0.00233	を 使う てる た
C. 充電	0.00280	充電 時間 が 短い
	−0.00410	充電 時間 が 長い

学習にかかった時間は 30 分程度であり，Boosting の 1 回の繰返し計算時間を算出すると 0.02 秒程度となる．これらより，本手法は実用に十分耐えうる手法だと考える．

5.3.4 提案手法の利点

4 章で，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 + dep）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いるかが分かる．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，**表 4** 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

表 5 分類の実行例
Table 5 A running example of actual classification.
Input: 液晶が大きくて綺麗，見やすい.
Output: 0.014（正例，良い点）

重み λ_t	分類に適用されたルール t
0.00368	やすい
0.00352	綺麗
0.00237	見る やすい
0.00174	が 大きい
0.00107	液晶 が 大きい
0.00074	液晶 が
0.00058	液晶
0.00027	て
0.00036	見る
−0.00001	大きい
−0.00033	、
−0.00052	が

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，多くの素性は負（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」）になったりすると，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．これらは係り受け構造を考慮した素性である☆.

また，「方 が 使い やすい（サイズ 4)」，「充電時間が長い（サイズ 4)」以外にも，比較的サイズの大きいルールが本手法により発見できた．たとえば MOD タスクでは，「意見」カテゴリに顕著なルールとして「〜 で は ある まい 。(サイズ 5)」「〜 み て は ならない 。(サイズ 6)」などが得られた．これらは，意見性を表現するための特徴的な文末表現である．このような長いルールは，固定長 N-グラム，固定長の部分木といった手法ではその上限サイズを十分に大きくしない限り取得できない．本手法では，ルールに対する明示的なサイズの上限はなく，分類に必要であれば自動的にサイズの大きいルールが取得される．これは従来手法にはない利点だと考える．

さらに，**表 5** 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．

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で，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，以下があげられる．

6.1 他のタスク

本タスクでは N-グラムと係り受けに顕著な差は見られなかった．その点を考えると，少なくとも本タスクでは N-グラムをラベル付き順序木の部分木として定式化することは冗長であったかもしれない．しかし，単語そのものが木構造で表現される場合，たとえば品詞やシソーラスといった情報が単語に付随される場合，単語の並びを単純なシーケンスとして定式化できない．このように複数の言語資源を統合したい場合，木構造で定式化する必要があり，本手法の有効性が期待できる．今後は，複数の構文解析結果のリランキング[19]といった品詞情報が有効に働くタスクに本手法を適用したいと考える

6.2 グラフへの拡張

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[20]，今後は，それらを適用したいと考える．

☆「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，N-グラムだとできない.

参 考 文 献

1) Joachims, T.: Text categorization with support vector machines: learning with many relevant features, *Proc. ECML*, pp.137–142 (1998).
2) Schapire, R.E. and Singer, Y.: BoosTexter: A Boosting-based System for Text Catego-

rization, *Machine Learning*, Vol.39, No.2/3, pp.135–168 (2000).
3) Freund, Y. and Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting, *Journal of Computer and System Sciences*, Vol.55, No.1, pp.119–139 (1996).
4) Breiman, L.: Prediction games and arching algoritms, *Neural Computation*, Vol.11, No.7, pp.1493–1518 (1999).
5) Morishita, S. and Sese, J.: Traversing Itemset Lattices with Statistical Metric Pruning, *Proc. ACM SIGACT-SIGMOD-SIGART Symp. on Database Systems* (*PODS*), pp.226–236 (2000).
6) Abe, K., Kawasoe, S., Asai, T., Arimura, H. and Arikawa, S.: Optimized Substructure Discovery for Semi-structured Data, *Proc. PKDD* (2002).
7) Zaki, M.: Efficiently Mining Frequent Trees in a Forest, *Proc. KDD*, pp.71–80 (2002).
8) Bayardo, R.J.: Efficiently Mining Long Patterns from Databases, *SIGMOD 1998, Proc. ACM SIGMOD International Conference on Management of Data*, ACM Press (1998).
9) Morhishita, S.: Computing Optimal Hypotheses Efficiently for Boosting, *Progress in Discovery Science*, pp.471–481, Springer (2002).
10) Collins, M. and Duffy, N.: Convolution Kernels for Natural Language, *Proc. Neural Information Processing Systems* (*NIPS*) (2001).
11) Kashima, H. and Koyanagi, T.: SVM Kernels for Semi-Structured Data, *Proc.ICML*, pp.291–298 (2002).
12) Rätsch, G.: Robust Boosting via Convex Optimization, Ph.D. Thesis, Department of Computer Science, University of Potsdam (2001).
13) Schapire, R.E., Freund, Y., Bartlett, P. and Lee, W.S.: Boosting the margin: a new explanation for the effectiveness of voting methods, *Proc. 14th International Conference on Machine Learning*, pp.322–330, Morgan Kaufmann (1997).
14) Vapnik, V.N.: *The Nature of Statistical Learning Theory*, Springer (1995).
15) 田村直良，和田啓二：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，Vol.5, No.1 (1996).
16) Gillick, L. and Cox, S.: Some Statistical Issues in the Comparison of Speech Recognition Algorithms, *Proc. ICASSP*, pp.532–535 (1989).
17) Haussler, D.: Convolution Kernels on Discrete Structures, Technical report, UC Santa Cruz (UCS-CRL-99-10) (1999).
18) Suzuki, J. and Isozaki, H.: Convolution Kernels with feature selection for natural language processing tasks, *Proc. ACL* (2004).
19) Collins, M.: Discriminative Reranking for Natural Language Parsing, *Proc. ICML*, pp.175–182 (2000).
20) Yan, X. and Han, J.: gSpan: Graph-based substructure pattern mining, *Proc. ICDM*, pp.721–724 (2002).

(平成 15 年 10 月 27 日受付)
(平成 16 年 7 月 1 日採録)

工藤　拓（正会員）

1999 年京都大学工学部電気電子工学科卒業．2001 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．2004 年同大学院博士後期課程修了．同年より NTT コミュニケーション科学基礎研究所リサーチアソシエイト．2001 年度本学会山下記念研究賞受賞．工学博士．統計的自然言語処理，テキストマイニング，機械学習に興味を持つ．

松本　裕治（正会員）

1977 年京都大学工学部情報工学科卒業．1979 年同大学大学院工学研究科修士課程情報工学専攻修了．同年電子技術総合研究所入所．1984 年〜1985 年英国インペリアルカレッジ客員研究員．1985 年〜1987 年（財）新世代コンピュータ技術開発機構に出向．京都大学助教授を経て，1993 年より奈良先端科学技術大学院大学教授，現在に至る．工学博士．専門は自然言語処理．人工知能学会，日本ソフトウェア科学会，言語処理学会，認知科学会，AAAI，ACL，ACM 各会員．