情報処理学会論文誌 Vol. 50 No. 5 1477–1492 (May 2009)

Boostingに基づく規則学習における部分候補を用いた高速化手法

岩倉友哉[†1] 岡本青史[†1]

本稿では，Boosting に基づく規則学習の高速化手法を提案する．Boosting とは，学習事例の重みを変化させることで，複数の弱仮説を作成し，それらを組み合わせて，最終的な仮説を作成する手法である．Boosting に基づく学習アルゴリズムは，自然言語処理，OCR などの様々なタスクにおいて高い精度を出せることが示されている．しかし，自然言語処理タスクのように，数十万の異なる素性および学習事例から構成される学習データを扱う場合においては，学習時間が問題となる．本稿では，素性の組合せを規則として学習する Boosting に基づく学習アルゴリズムの高速化のための手法を提案する．まず，素性の組合せで表現される規則候補の生成において，重複なくかつ枝刈りに適した生成方法を提案する．次に，各 Boosting ラウンドにおいて素性の部分集合から生成される候補を対象に規則を学習する方法を提案する．自然言語処理タスクである English Syntactic Chunking および日本語係り受け解析において本手法の評価を行った．その結果，本手法により，精度を保持したまま，100 倍以上の学習時間の改善が行えることを示せた．

Fast Boosting-based Rule Learning Using Subsets of Candidates

Tomoya Iwakura[†1] and Seishi Okamoto[†1]

This paper proposes techniques to improve training speed of boosting-based algorithms for learning rules represented by combination of features. Boosting is a method to crate a final hypothesis by repeatedly generating a weak hypothesis in each training iteration with a given weak learner. Boosting-based algorithms are successfully applied to several tasks such as Natural Language Processing, OCR, and so on. However, learning on the training data consisting of large number of samples and features requires long training time. We propose two techniques for improving training time of boosting based algorithms. The first one is generating candidate rules suited for pruning. The other is limiting search space by distributing features to buckets. Our algorithms repeatedly select a bucket and find a rule from candidate rules generated from the selected bucket. We evaluate our methods with English syntactic chunking and Japanese Dependency Parsing. The experimental results show that our methods improve training time by over 100 times while maintaining competitive accuracy obtained with boosting algorithms without our techniques.

1. Introduction

近年，Boosting[19] に基づくアルゴリズム，Support Vector Machines（SVM）[24] といった教師あり機械学習アルゴリズムが様々な分野に適用され成功を収めている．たとえば，Boosting であれば，自然言語処理の文書分類[20]，Named Entity 抽出[2]，構文解析[5]，English Syntactic Chunking[12]，文の分類[11]，OCR における手書き文字認識[18] など様々な分野に適用され，高い精度が残せることが示されている．また，Boosting に基づく規則学習手法は，精度だけでなく，分類速度の面でも良い性能を出せることが示されている[11),12)]．

しかし，Boosting に基づく学習アルゴリズムでは，学習時間が問題となる．Boosting では，学習事例の重みを変化させることで，与えられた学習アルゴリズムにて複数の弱仮説を作成し，それらを組み合わせた最終仮説を生成する．そのため，同一の学習データに対して，複数回の学習を行う必要がある．さらに，自然言語処理のタスクのような，数十万の事例および素性で構成される学習データ上で，素性の組合せを考慮することを考えると，学習時間がより深刻な問題となる．

本稿では，素性の組合せを規則として学習する Boosting に基づく学習アルゴリズムの高速化手法を提案する．まず，2 章で，本提案手法が用いる Boosting アルゴリズムについて説明する．続いて，3 章で，Boosting における規則学習の高速化のための手法として，枝刈りに適した規則候補の生成方法，素性の部分集合を用いた規則発見方法の 2 つについて説明する．4 章で，評価に用いるタスクである English Syntactic Chunking と日本語係り受け解析について述べ，実験結果を 5 章で述べる．6 章で先行研究について述べ，7 章で本稿をまとめる．

†1 株式会社富士通研究所
Fujitsu Laboratories Ltd.

2. Boosting アルゴリズム

2.1 前提条件

本稿における Boosting アルゴリズムが扱う問題について述べる．$\mathcal{X}$ を事例集合とし，扱うラベル集合を，$\mathcal{Y} = \{-1, +1\}$ とする．学習の目的は，学習データ，$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ から，マッピング，

$F : \mathcal{X} \to \mathcal{Y}$

を導出することである．

本稿では，各事例 $\mathbf{x}_i \in \mathcal{X}$（$1 \leq i \leq m$）は，素性の集合とし，素性集合と呼ぶことにする．また，$y_i \in \mathcal{Y}$ は S 中の i 番目の素性集合のクラスラベルである．$|\mathbf{x}_i|$ を，i 番目の事例 $\mathbf{x}_i$ に含まれる素性の個数とし，$\mathbf{x}_i$ の j 番目の素性は，$x_{i,j} \in \mathbf{FT}$（$1 \leq j \leq |\mathbf{x}_i|$）と表記する．ここで，$\mathbf{FT} = \{f_1, f_2, \ldots, f_M\}$ は，Boosting アルゴリズムが扱う M 個の素性とする．本稿では，k 個の素性から構成される素性集合を k-素性集合と記載する．本稿では，各素性は，文字列で表現されるとする[⋆1]．

また，ある素性集合が他の素性集合を包含する場合を次に定義する．

定義 1. 2 つの素性集合 $\mathbf{x}$，$\mathbf{x}'$ において，$\mathbf{x}$ が持つすべての素性を $\mathbf{x}'$ が持つ場合に，$\mathbf{x}$ は $\mathbf{x}'$ の部分素性集合と呼び，次のように記す．

$\mathbf{x} \subseteq \mathbf{x}'$

2.2 弱仮説

本稿では，BoosTexter[20] で用いられている real-valued predictions and abstaining（RVPA）の考えを基に規則（弱仮説）を定義する．RVPA では，入力の素性集合が条件に合う場合は，実数で表現される確信度を返し，条件に合わない場合は 0 を返す．素性集合を分類するための弱仮説を次に定義する[⋆2]．

定義 2. 素性集合分類のための弱仮説

素性集合 $\mathbf{f}$ を規則，$\mathbf{x}$ を入力の素性集合とする．また，実数 c を規則 $\mathbf{f}$ の確信度とした場合，規則の適用を以下に定義する．

$$h_{\langle \mathbf{f}, c \rangle}(\mathbf{x}) = \begin{cases} c & \mathbf{f} \subseteq \mathbf{x} \\ 0 & \text{otherwise} \end{cases}$$

2.3 Boosting に基づく規則学習

本稿の Boosting に基づく規則学習は，T 種類の規則素性集合とその確信度（$\langle \mathbf{f}_1, c_1 \rangle, \ldots, \langle \mathbf{f}_T, c_T \rangle$）を T 回の Boosting ラウンドでの弱学習器による学習で獲得し，以下に定義される F を構築する．

$$F(\mathbf{x}) = sign\left(\sum_{t=1}^{T} h_{\langle \mathbf{f}_t, c_t \rangle}(\mathbf{x})\right).$$

本稿では，$sign(x)$ を，x が 0 以上なら 1，それ以外は -1 と定義する．

弱学習器は，学習データ $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ と，t 回目の Boosting ラウンドの時点での各学習事例の重み $\{w_{t,1}, \ldots, w_{t,m}\}$ を用いて，規則 $(\mathbf{f}_t, c_t)$ を導出する．$w_{t,i}$（$0 < w_{t,i}$）とは，i 番目（$1 \leq i \leq m$）の事例 $(\mathbf{x}_i, y_i)$ の t 回目（$1 \leq t \leq T$）の Boosting ラウンドの重みである．

弱学習器は，与えられた学習データと学習事例の重みを基に，次の式を最小にする素性集合 $\mathbf{f}$ を規則として選択し，その確信度 c を計算する．

$$\sum_{y \in \{-1, +1\}} W_{t,y}(\mathbf{f}) * \exp(-y * h_{\langle \mathbf{f}, c \rangle}(\mathbf{x}_i)) + W_t(\neg \mathbf{f}). \tag{1}$$

$[[\pi]]$ を，ある命題 π が成り立つ場合に 1，それ以外に 0 とする．また，$W_{t,y}(\mathbf{f})$，$W_t(\neg \mathbf{f})$ は，

$$W_{t,y}(\mathbf{f}) = \sum_{i=1}^{m} w_{t,i}[[\mathbf{f} \subseteq \mathbf{x_i} \wedge y_i = y]],$$

$$W_t(\neg \mathbf{f}) = \sum_{i=1}^{m} w_{t,i} - W_{t,+1}(\mathbf{f}) - W_{t,-1}(\mathbf{f}),$$

とする．式 (1) を，ある規則 $\mathbf{f}$ によって最小化する場合，その時の確信度は次となる[19]．

$$c = \frac{1}{2} \log\left(\frac{W_{t,+1}(\mathbf{f})}{W_{t,-1}(\mathbf{f})}\right). \tag{2}$$

各ラウンドで，式 (1) を最小にする規則として選択するのは，各ラウンドでの事例の重みの和に関連して決まる Boosting に基づく学習アルゴリズムのトレーニングエラーの上限値

⋆1 今回は，素性を文字列と定義したが，本稿の学習器は Morishita[15] の手法と同様，二値で表現されるバイナリのベクトルを扱うことが可能である．M 次元のバイナリのベクトル集合 $\{\mathbf{X_1}, \ldots, \mathbf{X}_m\}$ を扱う際は，ベクトルの値が 1，0 で表現されているとすると，$\mathbf{x}_i \leftarrow \{f_i | X_{i,j} \in \mathbf{X}_i \wedge X_{i,j} = 1\}$（$1 \leq i \leq m$，$1 \leq j \leq M$）のようにベクトルの 1 の値を持つ属性を対応する文字列で表現される素性へとマッピングすることで実現できる．

⋆2 BoosTexter[20] では，real-valued predictions（RVP）という弱仮説も提案されている．RVP では，条件を満たした場合だけでなく，条件を満たさない場合にも確信度を返す．RVPA は，RVP と比較し，学習が速く済み，ほぼ同等の精度を示しているとことから，RVPA を採用した．

を，各ラウンドで最小化する規則を選択することと等価であるからである[19]．

実際の規則学習では，式 (1) と等価である別の式を用いる．式 (2) を式 (1) に代入することで，次の式が得られる．

$$\sum_{i=1}^{m} w_{t,i} - \left(\sqrt{W_{t,+1}(\mathbf{f})} - \sqrt{W_{t,-1}(\mathbf{f})}\right)^2. \tag{3}$$

式 (3) から，式 (1) を最小化することは，次に定義される *gain* を最大化する素性集合 $\mathbf{f}$ を選択することと等価であることが分かる．

$$gain(\mathbf{f}) \stackrel{\text{def}}{=} \left|\sqrt{W_{t,+1}(\mathbf{f})} - \sqrt{W_{t,-1}(\mathbf{f})}\right|. \tag{4}$$

本稿における弱学習器は，式 (1) の代わりに，*gain* を最大化する素性集合 $\mathbf{f}$ の t 番目の規則 $\mathbf{f}_t$ とし，t 番目の規則の確信度を c_t とする．

続いて，$(\mathbf{f}_t, c_t)$ を用いて，各事例の重みを更新する．本稿では，2 種類の AdaBoost と呼ばれる Boosting アルゴリズムを用いる．

まず，1 つ目は，文献 19) で提案された AdaBoost を用いる．本稿では，この AdaBoost を *AdaBoost-normalized* と呼ぶことにする．

AdaBoost-normalized では，t 回目のラウンドでの更新後の事例の重み

$\{w_{t+1,1}, \ldots, w_{t+1,m}\}$

の和が 1 になるように正規化されるため，重み更新は，式 (5) になる．

$$w_{t+1,i} = \frac{w_{t,i}\exp(-y_i h_{\langle \mathbf{f}_t, c_t\rangle}(\mathbf{x}_i))}{Z_t}. \tag{5}$$

ここで，Z_t は正規化のための値となり，

$$Z_t = \sum_{i=1}^{m} w_{t,i}\exp(-y_i h_{\langle \mathbf{f}_t, c_t\rangle}(\mathbf{x}_i))).$$

である．

もう 1 つは，ADTrees learning algorithm [7] で用いられている AdaBoost を用いる．本稿では，この AdaBoost を *AdaBoost-unnormalized* と呼ぶことにする．*AdaBoost-unnormalized* では，事例の重み更新後に，重みの和が正規化されない．*AdaBoost-unnormalized* の更新式は式 (6) となる．

$$w_{t+1,i} = w_{t,i}\exp(-y_i h_{\langle \mathbf{f}_t, c_t\rangle}(\mathbf{x}_i)). \tag{6}$$

AdaBoost-unnormalized では，各ラウンドにて重みの和が正規化されない分だけ *AdaBoost-normalized* と比較し計算量が少なくなり，より高速な学習が行えると予想される．よって，本稿では，これら 2 つのアルゴリズムを比較することにする．

また，別の違いとして，事例の重みの初期値が異なる．*AdaBoost-normalized* の事例の重みの初期値は，$w_{1,i} = 1/m$（$1 \leq i \leq m$）である．m は，学習データ S 中に含まれる学習事例の数である．これに対し，*AdaBoost-unnormalized* の事例の重みの初期値は，$w_{1,i} = 1$（$1 \leq i \leq m$）である．

実際の計算においては，クラスの偏りと素性のスパースネスから起きる問題を考慮する必要がある．クラスの偏りを考慮するために，以下に定義されるデフォルト規則を用いて初期の事例の重みを更新する[16]．

$$\frac{1}{2}\log\left(\frac{W_{+1}}{W_{-1}}\right)$$

ここで，$W_y = \sum_{i=1}^{m}[[y_i = y]]$（$y \in \mathcal{Y}$）である．

また，素性の出現がスパースであるような場合は，$W_{t,+1}(\mathbf{f})$ あるいは $W_{t,-1}(\mathbf{f})$ が非常に小さい値あるいは 0 になることがおきる．これをさけるために，スムージングのための値 ε を導入する[19]．

本稿では，式 (7) を確信度の計算および事例の重みの更新に用いる．

$$c = \frac{1}{2}\log\left(\frac{W_{t,+1}(\mathbf{f}) + \varepsilon}{W_{t,-1}(\mathbf{f}) + \varepsilon}\right) \tag{7}$$

AdaBoost-normalized には $\varepsilon = 1/m$，*AdaBoost-unnormalized* には $\varepsilon = 1$ を用いる．

3. 高速規則学習方法

本章では，Boosting 上での規則の高速学習方法について述べる．まず，先行研究にて用いられている枝刈り手法について説明する．続いて，枝刈りに適した規則候補の生成方法と，規則候補の部分集合からの規則生成について説明する．

3.1 候補の枝刈り

本手法では，先行研究で用いられている次の 3 種類の枝刈り手法を用いる[11),12),15)]．

- **頻度に基づく枝刈り**：頻度に関する閾値 ξ を導入し，規則候補のうち，異なる ξ 種類の事例に出現する候補だけを対象とする．
- **サイズ制約に基づく枝刈り**：素性集合のサイズに関する閾値 ζ を導入し，規則候補のうち，含まれる素性の種類が ζ 以下の候補だけを対象とする．

- **$gain$ の上限値による枝刈り**：次に定義される $gain$ の上限値を用いて枝刈りを行う[15]．

$$u(\mathbf{f}) \stackrel{\text{def}}{=} max\left(\sqrt{W_{t,+1}(\mathbf{f})}, \sqrt{W_{t,-1}(\mathbf{f})}\right)$$

現在までに見つかった最適な規則の $gain$ を τ とする．この方法では，$u(\mathbf{f})$ が τ 以下であれば，$\mathbf{f}$ を含む規則の候補を枝刈りしても，最適な解が発見できる．このことは，$0 \leq W_{t,+1}(\mathbf{f}') \leq W_{t,+1}(\mathbf{f})$，$0 \leq W_{t,-1}(\mathbf{f}') \leq W_{t,-1}(\mathbf{f})$ という 2 つの条件から，素性集合 $\mathbf{f}$ を含むすべての素性集合 $\mathbf{f}'$（i.e. $\mathbf{f} \subseteq \mathbf{f}'$）の $gain(\mathbf{f}')$ の上限値は，$u(\mathbf{f})$ 以下になることから分かる．

3.2　規則候補の生成

2 つ以上の素性から構成される素性集合を規則として学習する場合，規則の候補の生成方法が学習時間に影響を与える．たとえば，ある素性集合 $\{a, b\}$ は，素性集合 $\{a\}$ あるいは $\{b\}$ から生成される可能性がある．したがって，学習時間を改善するために，重複なくかつ枝刈りに適した形で規則の候補を生成する方法が必要であることが分かる．

以下に規則候補の生成方法について説明する．まず，本稿では，$\mathbf{f}' = \mathbf{f} + f$ を，サイズ k の素性集合 $\mathbf{f}$ にある素性 f を加えて，サイズ $k+1$ の素性集合の $\mathbf{f}'$ を生成する意味とする．

また，$ID(f)$ を，素性 f を識別する整数値 id を返す関数とする．また，gen を新たな素性集合を生成するための関数とし，次に定義する．

$$gen(\mathbf{f}, f) = \begin{cases} \mathbf{f} + f & \text{if } ID(f) > \max\limits_{f' \in \mathbf{f}} ID(f') \\ \{\} & \text{otherwise} \end{cases}$$

ここで，$\{\}$ はサイズ 0 の素性集合とする．gen では，新たに追加される素性の id が追加先の素性集合内の素性と比較し最大であれば，新たな素性集合を生成する．これにより，素性に一意に id を付与することで，重複なく規則候補を生成できる．

続いて，規則候補の生成順を調節するために，関数 ID を定義し，素性の組合せの生成順を決定する．

たとえば，a，b，$c \in \mathbf{FT}$ から構成されるサイズ 2 の素性集合を

$$u(\{c\}) < \tau = gain(\{a\}) < u(\{b\})$$

という条件下で生成する場合を考える．

もし，$ID(a) < ID(b) < ID(c)$ という ID 定義である場合は，$\{a, b\} = gen(\{a\}, b)$，$\{a, c\} = gen(\{a\}, c)$，$\{b, c\} = gen(\{b\}, c)$ がサイズ 2 の素性集合候補として生成される．$gen(\{b\}, a)$ からは候補生成 gen の定義によって，$gen(\{c\}, a)$ と $gen(\{c\}, b)$ からは $gain$ の上限値による枝刈りから，候補は生成されない．

もし，$ID(c) < ID(b) < ID(a)$ という ID 定義である場合は，$\{c\}$ から生成される候補は $u(\{\{c\}\}) < \tau$ という関係から枝刈りされる．残りについては，候補生成 gen の定義から，$\{a, b\} = gen(\{b\}, a)$ だけがサイズ 2 の素性集合候補として生成される．

これらの例から分かるように，異なる id の付与方法によって，$gain$ の上限値のよる枝刈りにより，異なる枝刈り結果になる．

これらの枝刈り結果の違いは，式 (4) から分かるように，低頻度の素性集合ほどその $gain$ が小さい値になるという傾向に関係する．この $gain$ の特性から，サイズ 2 以上の規則候補となる素性集合を効率的に枝刈りするために，組合せ生成時には，低頻度の素性集合に高頻度な素性を追加する候補生成方法を考える．この候補生成手法では，高頻度の素性集合からは，規則候補が生成されないため，低頻度の素性集合から組合せが生成されるようになる．その結果，低頻度の素性集合は，$gain$ の上限値 τ の制約によって効率的に枝刈りされると期待される．

上記条件を満たす候補生成のために，各素性への id の割当ては，頻度順に小さい id を割り当てることする．もし，同頻度の素性が出現する場合，今回は，素性名の辞書式順で id の割当てを行う．$fq(\mathbf{f})$ を，素性集合 $\mathbf{f}$ を部分素性集合とする S 中の事例数とする．

$$fq(\mathbf{f}) = \sum_{i=1}^{m} [[\mathbf{f} \subseteq \mathbf{x_i}]].$$

各素性への id 付与で満たすべき条件は，次となる．

- If $(fq(\{a\}) < fq(\{b\}))$ then $(ID(a) < ID(b))$
- If $(fq(\{a\}) = fq(\{b\})$ & $lexo(a, b) = 1)$ then $(ID(a) < ID(b))$

$lexo(a, b)$ は，素性 a，$b \in \mathbf{FT}$ の素性名を辞書式順に比較し，$a < b$ であれば 1，それ以外は 0 を返す[⋆1]．この頻度に基づく id の割当て方法を **freq-numbering** と呼ぶことにする．

id を素性 $\{a, b, c\}$ に付与する例を考える．学習データ中での出現頻度が

$$fq(\{a\}) = 2,\ fq(\{b\}) = 2,\ fq(\{c\}) = 1$$

である場合，id の付与結果は，

$$ID(a) = 2,\ ID(b) = 3,\ ID(c) = 1$$

となる．

⋆1 文字列比較には，C++の標準ライブラリを利用した．

また，freq-numbering との比較のために，id を学習データ中での素性の出現順で付与する方法も用いる．これを，**app-numbering** と呼ぶことにする．app-numbering では，学習データ中に新規の素性が出現した場合は，ある整数 I（$0 \leq I$）を id としてその素性に割り当てる．続いて，次の新しい素性が出現した場合は，$I+1$ を割り当てる．

3.3　部分素性集合を用いた規則学習

多くの Boosting に基づく学習アルゴリズムでは，各 Boosting のラウンドで，すべての素性を試し，規則を選択する方法を用いている[5),7),20)]．しかし，全種類の素性を試すような方法では，素性の種類が膨大である場合に，規則候補の評価回数が Boosting のラウンド数に比例して増加するため，学習時間が問題となる．また，素性の組合せまでを考慮する場合は，さらに学習時間が問題となる．

Boosting に基づく学習の速度を改善するために，素性全体を N 個の部分集合に分割し，各 Boosting のラウンドである部分集合を選択し，そこから生成される規則候補から規則を学習する方法を提案する．本稿では，これら部分集合を Bucket と呼ぶことにする．

規則をある Bucket から学習する場合は，その Bucket から生成される素性集合のうち $gain$ を最大にする素性集合を規則として選択する．

この方法では，各ラウンドで N 個に分割されたある部分集合だけを用いるので，各ラウンドでの規則候補の評価の計算量が約 $1/N$ となり，学習速度が大幅に改善されると期待される．

N 個の Bucket を作成するために，次の 3 つの手法を用いる．

- Entropy-based distribution（*E-dist*）：*E-dist* では素性を Bucket に分配するために，素性のエントロピーを用いる．この分配方法では，素性をエントロピーに基づきソートし，その後，ソート結果を順番に Bucket に分配する．
- Frequency based distribution（*F-dist*）：*F-dist* では素性を Bucket に分配するために，素性の学習データ中での頻度を用いる．この分配方法では，素性は頻度に基づきソートする．その後，ソート結果を順番に Bucket に分配する．
- Random Distribution（*R-dist*）：*R-dist* では各 Bucket に作成する際に，素性をランダムに選択する．*R-dist* においても，各素性は 1 つの Bucket にしか属さないように選択する．

素性の分配の例として，**図 1** の左に E-dist に基づく Bucket の作成方法を載せる．E-dist に基づく分配では，まず，学習データ中での各素性のエントロピーを計算する．続いて，各素性をそれぞれのエントロピーを基にソートする．最後に，ソート結果に基づきバケットに分配を行う．今回は，各バケットに属する素性のエントロピーの和ができるだけ均一になるように，ソート結果の先頭から順番にバケットに分配する．

```
## エントロピーに基づく素性の部分集合生成##
## S = {(x_i, y_i)}_{i=1}^m :  x_i ∈ X,  y_i ∈ Y
## f : ある素性 (f ∈ FT = {f_1, ..., f_M})
## fq(f, y) :Σ_{i=1}^m [[{f} ⊆ x_i ∧ y_i = y]]
## fq(f) :fq(f, +1) + fq(f, −1)
## ent[f]: 素性 f のエントロピー
## sortByEnt(F, ent): 素性 f ∈ FT を
## エントロピーに基づきソート.
## (a % b): a を b で割った余りを返す.
procedure distributeFtIntoBuckets(S, N)
## N 個の Bucket を準備
 B = {B[1], ..., B[N]}
## FT 中の M 個の素性について
## エントロピーを計算
 Foreach (f ∈ FT)
  p_(+1) = fq(f, +1)/fq(f);
  p_(−1) = fq(f, −1)/fq(f);
  ent[f] = −p_(+) log2 p_(+)  −  p_(−) log2 p_(−) ;
 end Foreach
##素性をエントロピーに基きソート
 Fs ← sortByEnt(F, ent)
 For i=1 . . . M :  B[(i % N) + 1]←Fs[i];
 return B
end
```

```
## Boosting ラウンド t における##
## 素性の部分集合からの規則発見##
## F_k : k-素性集合の集合
## f_t : 現在の最適な素性集合
## τ : 最適な素性集合 f_t の gain
procedure findConj(F_k)
 Foreach (f ∈ F_k)
  ## 規則候補の頻度に基づく枝刈り
  if (fq(f) < ξ) continue;
  ## 規則候補の gain 計算
  if (τ < gain(f))
   τ = gain(f);    f_t = f
  end if
  ## サイズに基づく枝刈り
  if (ζ < k) continue;
  ## 規則候補の gain の上限値に基づく枝刈り
  if ( u(f) < τ) continue;
  ## 規則候補の生成 - 3.2 節
  Foreach( f ∈ FT)
   F_{k+1} ← {F_{k+1} ∪ gen(f, f)}
  end Foreach
 end Foreach
 if (k < ζ)  findConj(F_{k+1})
 else  return f_t
end
```

図 1　エントロピーに基づく素性の分配（左）と Boosting の t 回目のラウンドにおける素性の部分集合からの規則発見（右）

Fig. 1　The figure in left side shows an overview of the method for distribute features into buckets based on entropy. The figure in right side shows an overview of the procedure for finding an optimal feature-set at boosting round t.

これら 3 つの分配方法に加えて，文献 6) で用いられている各ラウンドで素性の部分集合をランダムに選択する手法も評価する．この手法を Random-selection（Rand-S）と呼ぶことにする．Rand-S は，各ラウンドで，M/N 種類の素性を乱数に基づき選択し，学習する．

Rand-S でも本手法と同様に，各ラウンドで素性の部分集合だけを用いる．しかし，Rand-S ではすべての素性が学習中に試されるという保証がない．これに対して，本提案手法ではすべての素性が試されるという違いがある．また，ランダム選択に基づく R-dist と Rand-S

```
## S = {(x_i, y_i)}_{i=1}^m :  x_i ∈ X, y_i ∈ Y
##   ε : AdaBoost-normalized には 1/m ,  AdaBoost-unnormalized には 1 を使用.
## N : Bucket size
## Z_t : AdaBoost-normalized において重み更新後に Σ_{i=1}^m w_{t+1,i} = 1 とするための値
procedure AdaBoost.DF()
## 事例の重みを初期化
  c_0 = (1/2) log((W_{+1})/(W_{-1}))
 For i = 1, ..., m:
  if (AdaBoost-normalized)   w_{1,i} = exp(c_0)/(m Z_0)
  else if (AdaBoost-unnormalized)  w_{1,i} = exp(c_0)
 end For
 ## 素性を Buckets B = {B[1], ..., B[N]} に分配
 B = distributeFtIntoBuckets(S, N);
 ## 学習開始
 For t = 1, ..., T:
  ## B[(t%N) + 1] から生成される規則候補から 素性集合 f_t を選択
  f_t = findConj(B[(t%N) + 1])
  ## f_t の確信度を計算
     c_t = (1/2) log((W_{t,+1}(f_t)+ε)/(W_{t,-1}(f_t)+ε))
  ## 事例の重みを更新
  For i = 1, ..., m:
   if (AdaBoost-normalized)      w_{t+1,i} = w_{t,i} exp(−y_i h_{⟨f_t,c_t⟩}(x_i)) / Z_t
   else if (AdaBoost-unnormalized)      w_{t+1,i} = w_{t,i} exp(−y_i h_{⟨f_t,c_t⟩}(x_i))
  end For
 end For
 return F(x) = sign(c_0 + Σ_{t=1}^T h_{⟨f_t,c_t⟩}(x))
end
```

図 2　AdaBoost.DF の概要．*AdaBoost-normalized* あるいは *AdaBoost-unnormalized* の Boosting アルゴリズムに基づく
Fig. 2　An overview of AdaBoost.DF based on *AdaBoost-normalized* or *AdaBoost-unnormalized*.

は，ランダムに Bucket の作成あるいは素性部分集合の選択が行われるため，同じ学習データ，同じパラメータで再度学習を行った際に，同じモデルを再現できないという問題がある．たとえば，学習の再現性がないために起きる問題の例としては，素性設計を行うために素性を変更して実験を行う場合があげられる．適切な素性選択のために，素性を変更し実験を行うのは一般的であるが，ランダム選択に基づく手法では，精度の変化が，素性選択の結果なのか，ランダムに選択した部分素性集合の影響なのかという判断がしにくいという問題がある．これに対し，E-dist と F-dist は，学習の再現性があるため，素性設計のための実験において R-dist，Rand-S でおきる問題を回避することができる．

図 1 の右に Bucket を用いた規則学習の概要を載せる．各ラウンドでは，N 個の Bcuket のうちの 1 つが，サイズ 1 の素性集合から構成される初期 Bucket の F_1 として選択される．学習時は，F_1 および F_1 から生成されるサイズ ζ までの素性集合を規則の候補とし，候補の中で，*gain* を最大にする素性集合を規則として選択する．

図 2 に本提案手法の概要を載せる．この手法を，AdaBoost for Distributed Features（AdaBoost.DF）と呼ぶことにする．AdaBoost.DF で，Bucket のサイズ N を 1 とした場合は，多くの Boosting に基づく学習アルゴリズム同様[7),20)]，すべての素性が各ラウンドで試される．本提案手法では，各ラウンドですべての規則候補を試さないが，AdaBoost における事例の重み更新の形式を保持しているため，先行研究の規則学習[7),20)] と同様に，収束が保証される．付録 A，B に収束性の説明を載せる．

4. 実験データ

本稿では，English Syntactic Chunking および日本語係り受け解析，2 つのタスクで評価を行った．

4.1 English Syntactic Chunking

本手法を，English Syntactic Chunking（ESC）タスクを用いて評価を行う．ESC のデータとして，CoNLL-2000 の shared task のデータを用いる[22),⋆1]．

このタスクは，NP，VP，PP，ADJP，ADVP，CONJP，INITJ，LST，PTR，SBAR という 10 種類の基本フレーズを判別するタスクである．

このデータは，Penn Treebank の Wall Street Journal の部分からなる．トレーニングデータは，15–18 の 4 セクションで構成され，211,727 の単語を含む．テストデータはセクション 20 から構成され，47,377 の単語を含む．

基本フレーズは，2 つ以上の単語から構成される可能性がある．複数の単語から構成される基本フレーズを表現するためには **IOE2** を用いる[⋆2]．

IOE2 では，複数単語から構成される基本フレーズを表現するために，E，I，O の 3 種類のタグを用いる．E は基本フレーズの最後の単語に付与される．I は 2 単語以上から構成される基本フレーズの最後以外の単語に付与される．O タグは，基本フレーズを構成しない単語に付与される．

⋆1 http://lcg-www.uia.ac.be/conll2000/chunking/
⋆2 文献 9) で，IOE2 に基づく基本フレーズを判別するパーザが良い精度を示していることから採用した．

たとえば，

[He] (NP) [reckons] (VP) [the current account deficit] (NP) . . .

という基本フレーズ情報付き文を IOE2 で表現する場合は，

He/E-NP reckons/E-VP the/I-NP current/I-NP account/I-NP deficit/E-NP

と表現する．E-NP，E-VP は NP および VP の最後の単語，I-NP は 2 つ以上の単語から構成される NP の中の単語という意味である．各基本フレーズの種類において 2 種類のタグが IOE2 表現によって生成されるので，今回のタスクでは，21（$= 10 * 2 + 1$）種類のタグが生成される．

n 個の単語から構成される文 $\{w_1, \ldots, w_n\}$ の基本フレーズを判別する場合は，各単語を 21 種類のいずれかのタグに，次の素性を使って分類する．

- **単語および品詞タグ**：現在位置の単語 w_j（$1 \leq j \leq n$）およびその品詞タグ p_j に加えて，前後 2 単語（$\{w_{j-2}, w_{j-1}, w_{j+1}, w_{j+2}\}$）およびそれらの品詞タグ $\{p_{j-2}, p_{j-1}, p_{j+1}, p_{j+2}\}$ を用いる．
- **先行する単語に付与されたタグ情報**：今回は，各単語を文末から文頭の方向で分類する．j 番目の単語を分類する際は，単語 w_{j+1} および w_{j+2} に対し，最も高い確信度で付与されたタグ t_{j+1}，t_{j+2} を素性として用いる．

この素性設計に基づき作成された ESC タスクの学習データでは，92,825 種類の素性を含む．

二値分類を扱う今回の Boosting を，複数のクラスを扱うようにするために，one-vs-the-rest を用いる．この方法では，ESC パーザは 21 種類の分類器から構成されることになる．

最終的な解は，Viterbi アルゴリズムを用いて求める．まず，ESC パーザからの各タグへの分類結果の出力値を 0 から 1 の範囲に，次に定義される sigmoid 関数を用いて変換する．

$$s(X) = 1/(1 + \exp(-\beta X))$$

X はある分類器の出力値，$(X = c_0 + \sum_{t=1}^{T} h_{\langle \mathbf{f}_t, c_t \rangle})$ である．今回は，$\beta = 5$ を用いた．

その後，Viterbi アルゴリズムを用いて，IOE2 の接続条件を満したタグ列のうち，sigmoid 関数で変換後の確信度の log 値の和を最大とするタグ列をパーザの出力とする．

4.2　日本語係り受け解析

日本語係り受け解析とは，文中において，「係り元文節」と「係り先文節」という修飾関係にを判別するタスクである．以下，係り元文節と係り先文節の候補をそれぞれ「係り元」，「候補」と記す．

素性は，文献 8) の Standard Feature を参考に決定した．

- 係り元と候補の文節情報：ここでの，文節情報とは，各文節の主辞の形態素の表層形・基本形・品詞・品詞細分類・活用形，および，各文節内部に出現する句読点，開き括弧，閉じ括弧，助詞である．今回，主辞として，文節形態素のうち記号・助詞以外の形態素のうち最も右側の形態素とした．
- 係り元と候補間の情報：係り元と候補の間の距離（1，2–5，6 以上），係り元と候補の間の文節に出現する，句読点，開き括弧，閉じ括弧，助詞を素性として用いた．

日本語係り受け解析手法としては，文献 17) のアルゴリズムを用いた．このアルゴリズムの詳細については，文献 17) を参照願いたい．

日本語係り受け解析の実験では，京大コーパス Version 4.0 を用いた．この実験では，文献 8) と同様に，学習データに，1 月 1 日〜1 月 8 日の記事，評価には，1 月 9 日，1 月 10 日，1 月 15 日の記事を用いた．精度として，係り受け正解率と文正解率の評価結果を用いた．

5. 実 験 結 果

5.1　ESC における実験結果

5.1.1　候補生成方法および Bucket サイズによる比較

ESC パーザの平均学習時間を，**表 1** に載せる．表 1 の結果は，Boosting のラウンド回数 $T = 10{,}000$，頻度制約 ξ は {0, 5, 10}，サイズ制約 ζ は {1, 2, 3}，Bucket サイズ N は {1, 10, 100, 1,000} を用いて得られた結果である[⋆1]．

表 1 から **freq-numbering**（**freq.**）に基づく学習は，**app-numbering**（**app.**）に基づく学習よりすべてのパラメータの平均において短い学習時間となっていることが分かる．この結果から，freq-numbering に基づく規則の候補生成が学習時間の改善に貢献することが分かる．さらに，Bucket サイズ N を 10，100，1,000 と変更することで，すべての種類の素性を試す Bucket サイズ $N = 1$ の場合と比較し，大幅な学習時間の改善が得られることが分かる．

各 ESC パーザにて得られる平均精度を，**表 2** に載せる．ここでの精度は F-measure（F 値）である．表 2 から，Bucket サイズを変更しても，精度に大きな変化がないことが分かる．このことから，本提案手法である，Boosting における素性の部分集合からの規則学習は，精度を保持したまま，学習時間を大幅に改善できることが分かる．

また，表 2 から，素性の組合せを考慮するパラメータに $\zeta = \{2, 3\}$ を指定して学習を行っ

⋆1 今回の実験は 3.0 Ghz Xeon processor および 6 Gbyte のメインメモリ上の Linux にて行った．すべてのシステムは C++で実装した．

表 1　English Syntactic Chunking におけるにおける 2 種類の候補生成方法と Bucket サイズ別の平均学習時間（単位は時間）．頻度制約（$\xi = 0, 5, 10$）で得られた学習時間の平均

Table 1　Comparison of average training time. These results are average training time obtained with parameters $\xi = 0, 5, 10$.

	AdaBoost-normalized				*AdaBoost-unnormalized*			
	E-dist	F-dist	R-dist	Rand-S	E-dist	F-dist	R-dist	Rand-S
$\zeta = 1$								
$N = 1$	22.51				21.74			
$N = 10$	3.12	3.02	3.15	3.56	2.50	2.40	2.52	2.96
$N = 100$	0.78	0.78	0.78	0.81	0.24	0.24	0.24	0.26
$N = 1{,}000$	0.55	0.55	0.55	0.55	0.02	0.02	0.02	0.02
$\zeta = 2$								
$N = 1$ (**app.**)	96.91				110.00			
(**freq.**)	47.80				51.74			
$N = 10$ (**app.**)	9.74	9.60	9.76	9.67	10.22	10.05	10.28	10.24
(**freq.**)	5.33	5.34	5.51	5.77	5.40	5.27	5.42	5.77
$N = 100$ (**app.**)	1.43	1.44	1.43	1.44	0.97	0.97	0.97	0.95
(**freq.**)	1.01	1.02	1.02	1.04	0.51	0.52	0.52	0.53
$N = 1{,}000$ (**app.**)	0.66	0.66	0.66	0.66	0.14	0.14	0.14	0.14
(**freq.**)	0.60	0.60	0.60	0.60	0.07	0.07	0.07	0.07
$\zeta = 3$								
$N = 1$ (**app.**)	180.25				223.96			
(**freq.**)	70.66				87.60			
$N = 10$ (**app.**)	16.76	16.65	16.80	16.30	21.53	21.26	21.65	20.29
(**freq.**)	7.27	7.20	7.33	7.59	8.21	8.12	8.29	8.41
$N = 100$ (**app.**)	2.15	2.16	2.14	2.13	2.09	2.10	2.07	1.99
(**freq.**)	1.21	1.22	1.21	1.23	0.79	0.80	0.79	0.80
$N = 1{,}000$ (**app.**)	0.77	0.77	0.77	0.77	0.28	0.28	0.28	0.28
(**freq.**)	0.65	0.64	0.64	0.64	0.12	0.12	0.12	0.13

表 2　English Syntactic Chunking における 2 種類の候補生成方法と Bucket サイズ別の精度の比較．頻度制約（$\xi = 0, 5, 10$）で得られた F 値の平均

Table 2　Comparison of average accuracy ($F_{\beta=1}$) in English Syntactic Chunking. These results are average training time obtained with parameters $\xi = 0, 5, 10$.

	AdaBoost-normalized				*AdaBoost-unnormalized*			
	E-dist	F-dist	R-dist	Rand-S	E-dist	F-dist	R-dist	Rand-S
$\zeta = 1$								
$N = 1$	91.41				92.25			
$N = 10$	91.36	91.43	91.43	91.42	92.26	92.26	92.26	92.24
$N = 100$	91.32	91.38	91.39	91.40	92.25	92.29	92.27	92.3
$N = 1{,}000$	91.35	91.30	91.49	91.5	92.13	92.01	92.22	92.31
$\zeta = 2$								
$N = 1$ (**app.**)	93.34				93.53			
(**freq.**)	93.29				93.48			
$N = 10$ (**app.**)	93.37	93.35	93.31	93.38	93.58	93.53	93.55	93.56
(**freq.**)	93.32	93.32	93.30	93.31	93.55	93.53	93.51	93.52
$N = 100$ (**app.**)	93.18	93.22	93.18	93.25	93.54	93.56	93.53	93.58
(**freq.**)	93.26	93.23	93.29	93.29	93.56	93.49	93.54	93.59
$N = 1{,}000$ (**app.**)	92.72	92.52	92.66	92.76	93.36	93.15	93.44	93.37
(**freq.**)	92.93	92.82	93.00	92.97	93.43	93.39	93.41	93.48
$\zeta = 3$								
$N = 1$ (**app.**)	93.29				93.49			
(**freq.**)	93.3				93.43			
$N = 10$ (**app.**)	93.37	93.41	93.36	93.34	93.49	93.43	93.52	93.49
(**freq.**)	93.27	93.29	93.28	93.30	93.46	93.43	93.46	93.51
$N = 100$ (**app.**)	93.23	93.31	93.28	93.40	93.52	93.55	93.55	93.56
(**freq.**)	93.16	93.43	93.30	93.28	93.51	93.48	93.55	93.49
$N = 1{,}000$ (**app.**)	92.67	92.45	92.67	92.75	93.35	93.18	93.31	93.34
(**freq.**)	92.92	92.86	93.04	93.05	93.47	93.42	93.45	93.48

た ESC パーザの平均精度は，組合せを考慮しない $\zeta = 1$ で学習を行った ESC パーザの平均精度と比較して，高いことが分かる．これらの結果から，素性の組合せの発見が精度改善に貢献することが分かる．

5.1.2　Boosting アルゴリズム別の比較

表 3 に，*AdaBoost-normalized* および *AdaBoost-unnormalized* に基づく ESC パーザで得られる平均精度を載せる．各 Bucket サイズの平均精度とは，各 Bucket サイズにおいて，パラメータ $\zeta = \{1, 2, 3\}$，$\xi = \{0, 5, 10\}$ で作成された ESC パーザで得られたテスト結果の平均である．

表 3 から，*AdaBoost-unnormalized* に基づく ESC パーザのほうが，*AdaBoost-normalized* に基づく ESC パーザより，平均的に良い精度を残していることがことが分かる．

また，表 3 の結果から，両方の Boosting において，freq-numbering が app-numbering と比較しても同等の精度を残せていることが分かる．

5.1.3　Boosting ラウンド数の影響調査

Boosting のラウンド数の影響を調査するために，$N = \{1, 10, 100, 1{,}000\}$，$\xi = 0$，$\zeta = 2$ というパラメータにおいて，Boosting ラウンド数を 10 倍の $T = 100{,}000$ に設定し実験を行った．**表 4** に各 ESC パーザでテストデータ上で得られた最も良い結果を載せる．この結果から，ほとんどのパーザが $N = 1$ で作成されたパーザの結果より，高い精度を示していることが分かる．この結果から，提案手法において，Boosting のラウンド数を増加するこ

表 3　English Syntactic Chunking における AdaBoost 別の比較．表 1 と同じ ESC パーザにで得られた結果の平均 F 値を記載．**app.** は app-numbering, **freq.** は freq-numbering

Table 3　Comparison of two types of boosting algorithms. We lists the average F-measures obtained with the same parsers used in Table 1.

Bucket sizes \	*AdaBoost-unnormalized*		*AdaBoost-normalized*	
Numbering	**app.**	**freq.**	**app.**	**freq.**
$N = 1$	92.68	92.67	93.09	93.05
$N = 10$	92.71	92.67	93.09	93.08
$N = 100$	92.63	92.63	93.13	93.10
$N = 1{,}000$	92.23	92.44	92.93	93.02

表 4　English Syntactic Chunking における 4 種類の Bucket 作成方法で得られる最高精度．学習のパラメータには，$T = 100{,}000$，$N = \{1, 10, 100, 1{,}000\}$，$\xi = 0$，$\zeta = 2$，freq-numbering を利用

Table 4　The best results for various bucketing methods and bucket sizes on test data. We list the results of ESC parsers trained with parameters of $T = 100{,}000$, $N = \{1, 10, 100, 1{,}000\}$, $\xi = 0$, $\zeta = 2$ and freq-numbering.

	AdaBoost-normalized				*AdaBoost-unnormalized*			
$N = 1$	93.58				93.66			
	E-dist	F-dist	R-dist	Rand-S	E-dist	F-dist	R-dist	Rand-S
$N = 10$	93.67	93.68	93.59	93.71	93.75	93.67	93.69	93.69
$N = 100$	93.65	93.68	93.62	93.71	93.72	93.68	93.68	93.72
$N = 1{,}000$	93.60	93.40	93.38	93.57	93.71	93.76	93.70	93.75

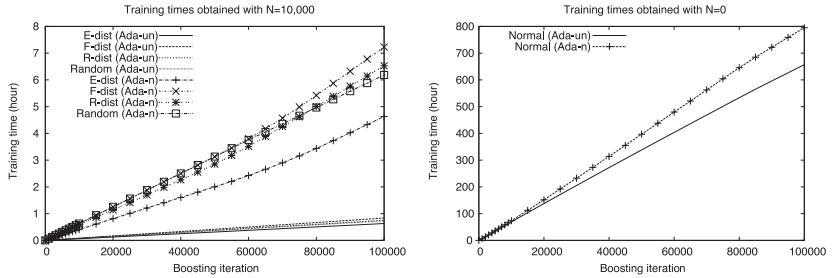


図 3　English Syntactic Chunking における Boosting アルゴリズムによる学習時間の違い．$N = 1{,}000$，$\zeta = 2$，左のグラフは $\xi = 0$，freq-numbering による結果．右のグラフ（“Normal”）は $N = 0$，$\zeta = 2$，$\xi = 0$，freq-numbering による結果．Ada-un，Ada-n は *AdaBoost-unnormalized*，*AdaBoost-normalized*

Fig. 3　Training time obtained with AdaBoost.DF: The results obtained with $N = 1{,}000$, $\zeta = 2$, $\xi = 0$ and freq-numbering are shown in the left graph. The other graph shows results obtained with “Normal” denoting a training with $N = 0$, $\zeta = 2$, $\xi = 0$ and freq-numbering. Ada-un and Ada-n denote *AdaBoost-unnormalized* and *AdaBoost-normalized*.

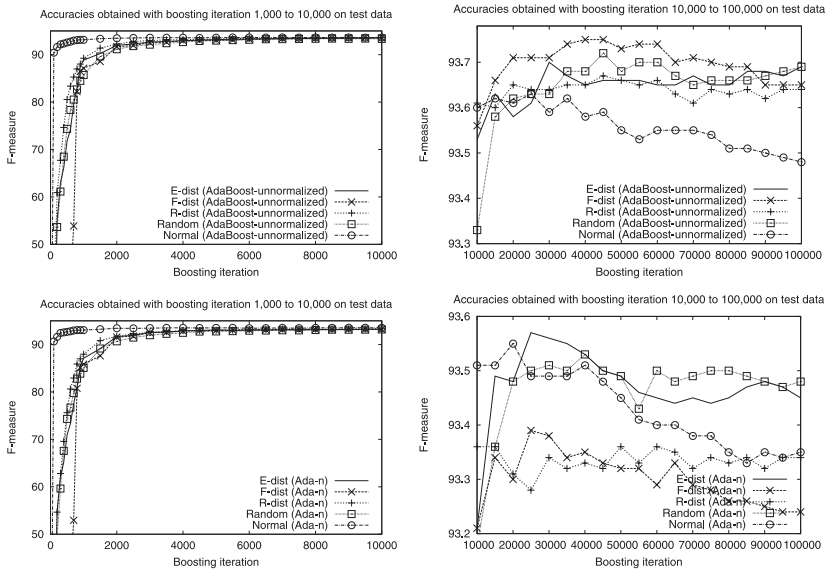


図 4　English Syntactic Chunking における Boosting 回数による評価データ上での F 値の変化．*AdaBoost-unnormalized*（上），*AdaBoost-normalized*（下），freq-numbering，$N = 1{,}000$，$\zeta = 2$，$\xi = 0$ で学習した規則を利用．“Normal” は $N = 1$，$\zeta = 2$，$\xi = 0$ で学習した規則を利用

Fig. 4　Accuracy ($F_{\beta=1}$) obtained with ESC parsers trained with freq-numbering over test data. Accuracy of *AdaBoost-unnormalized* is top and accuracy of *AdaBoost-normalized* is bottom. Parameters for training with AdaBoost.DF are $N = 1{,}000$, $\zeta = 2$, $\xi = 0$. “Normal” denotes ESC parsers trained with $N = 1$, $\zeta = 2$, $\xi = 0$.

とで，精度改善につながることも分かる．

図 3 に Boosting アルゴリズム別の学習時間を載せる．これらの結果から，*AdaBoost-normalized* に基づく学習より *AdaBoost-unnormalized* に基づく学習が速いことが分かる．これは，*AdaBoost-normalized* では事例に重みを各ラウンドで正規化するのに対し，*AdaBoost-unnormalized* では重みの正規化を行わないことに関係すると予想される．

図 4 に各 Boosting ラウンド数におけるテストデータでの精度を載せる．Bucket サイズ $N = 1{,}000$ で学習した規則に基づく ESC パーザは，Bucket サイズ $N = 1$ で学習した規則に基づく ESC パーザと比較し，Boosting ラウンドが 4,000 あたりまでは，低い精度となっている．しかし，Boosting ラウンド 4,000 以降では，Bucket サイズ $N = 1{,}000$ で学習した規則に基づく ESC パーザが良い精度を示していることが分かる．

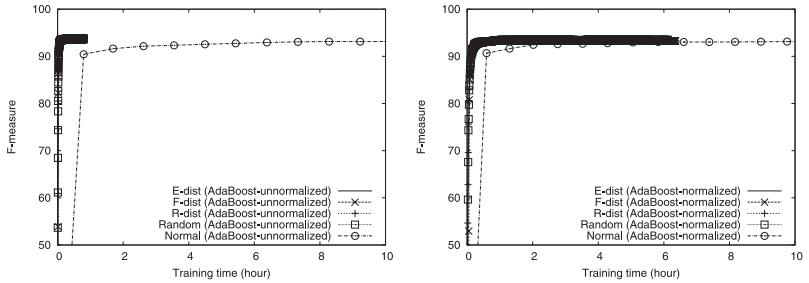


図 5　English Syntactic Chunking で各学習時間に得られる規則で得られる評価データ上での精度変化．*AdaBoost-unnormalized*（左），*AdaBoost-normalized*（右）．横軸は時間（hour），縦軸は F 値

Fig. 5　Accuracy obtained with rules at each training time on test data. The left graph is for *AdaBoost-unnormalized* and the right grpah is for *AdaBoost-normalized*. X-axis is for training time (hour). Y-axis is for F-measure.

図 5 に各学習時間で得られる規則に基づくテストデータ上での抽出精度を載せる．図 5 から，提案手法に基づく ESC パーザは，短時間で高精度を達成していることが分かる．これらの結果から，提案手法においては，必要な Boosting ラウンド数は多くなるが，短時間で，同等以上の精度を達成できることが分かる．

5.2　日本語係り受け解析における実験結果

5.2.1　学 習 時 間

表 5 に学習時間を載せる．表 5 の結果は，Boosting のラウンド回数 $T = 10{,}000$，頻度制約 ξ は $\{0\}$，サイズ制約 ζ は $\{1, 2, 3\}$，Bucket サイズ N は $\{1, 10, 100, 1{,}000\}$ をによる結果である．

これらの結果から，日本語係り受け解析においても，freq-numbering が app-numbering と比較し，より短い時間で学習可能であることが分かる．また，バケットサイズを大きくすることで，学習時間を大幅に改善できることが分かる．この実験では，*AdaBoost-normalized* と *AdaBoost-unnormalized* との比較では，全体的にみてほぼ同等の学習時間となった．ESC では，*AdaBoost-unnormalized* が *AdaBoost-normalized* より短い学習時間であったことから，*AdaBoost-normalized* と *AdaBoost-unnormalized* による学習時間は，タスクに依存すると考えられる．

5.2.2　精　　度

表 6 に 1 月 9 日の記事での係り受け精度の評価結果を載せる．表 6 から，素性の組合せを発見することで，精度が大幅に改善されることが分かる．たとえば，($N = 1$，$\zeta = 1$，*AdaBoost-unnormalized*) による結果と ($N = 1$，$\zeta = 3$，**app.**，*AdaBoost-unnormalized*) による結果では，約 4 ポイントの差がある．これらの結果から，日本語係り受け解析においても，本手法による素性の自動組合せ発見は有効であるとことが分かる．また，freq-numbering が app-numbering と同等の精度を出している点，バケットサイズを大きくしても同等の精度を達成していることから，本提案手法で，精度を保持しつつ，学習時間を改善できることが分かる．

図 6 に，各学習時間で得られる規則による係り受け精度を載せる．図 6 から，本手法で，日本語係り受け解析においても，短時間で高い精度が得られることが分かる．

表 5　日本語係り受け解析の学習時間（単位は時間）．N はバケットサイズ，ζ はサイズ制約

Table 5　Training time for Japanese Dependency Parsing (hour). N is a bucket size. ζ is a size constraint.

	AdaBoost-normalized				*AdaBoost-unnormalized*			
	E-dist	F-dist	R-dist	Rand-S	E-dist	F-dist	R-dist	Rand-S
$\zeta = 1$								
$N = 1$	0.67				0.67			
$N = 10$	0.07	0.06	0.07	0.08	0.06	0.05	0.06	0.07
$N = 100$	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.00
$N = 1{,}000$	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.00
$\zeta = 2$								
$N = 1$ (**app.**)	18.87				20.13			
(**freq.**)	8.82				9.48			
$N = 10$ (**app.**)	1.43	1.41	1.48	1.41	1.45	1.49	1.44	1.37
(**freq.**)	0.73	0.72	0.73	0.69	0.79	0.81	0.78	0.76
$N = 100$ (**app.**)	0.12	0.12	0.13	0.11	0.12	0.12	0.11	0.10
(**freq.**)	0.07	0.07	0.07	0.06	0.06	0.06	0.06	0.05
$N = 1{,}000$ (**app.**)	0.02	0.02	0.02	0.02	0.01	0.01	0.01	0.01
(**freq.**)	0.02	0.02	0.02	0.02	0.01	0.01	0.01	0.01
$\zeta = 3$								
$N = 1$ (**app.**)	692.81				566.98			
(**freq.**)	658.35				562.63			
$N = 10$ (**app.**)	65.95	65.89	65.20	51.58	69.57	70.84	70.31	55.27
(**freq.**)	57.60	58.33	59.00	41.17	63.55	63.08	64.71	44.37
$N = 100$ (**app.**)	5.30	5.33	5.29	4.44	5.49	5.56	5.56	4.61
(**freq.**)	4.76	4.89	4.76	3.56	4.97	5.02	5.13	3.89
$N = 1{,}000$ (**app.**)	0.44	0.46	0.45	0.37	0.44	0.46	0.45	0.37
(**freq.**)	0.43	0.46	0.43	0.15	0.43	0.44	0.43	0.14

表 6　日本語係り受け解析の係り受け精度（1 月 9 日の記事）．N はバケットサイズ，ζ はサイズ制約
Table 6　Accuracy for Japanese Dependency Parsing on Jan. 9th. N is a bucket size. ζ is a size constraint.

	AdaBoost-normalized				*AdaBoost-unnormalized*			
	E-dist	F-dist	R-dist	Rand-S	E-dist	F-dist	R-dist	Rand-S
$\zeta = 1$								
$N = 1$	83.61				84.16			
$N = 10$	83.58	83.67	83.45	83.63	84.17	84.15	84.01	84.02
$N = 100$	83.60	83.49	83.56	83.70	84.09	84.13	84.13	83.89
$N = 1{,}000$	83.22	83.63	83.41	83.32	83.78	84.10	83.97	83.73
$\zeta = 2$								
$N = 1$ (**app.**)	88.23				88.13			
(**freq.**)	87.54				88.01			
$N = 10$ (**app.**)	88.40	88.28	88.26	88.21	88.55	88.36	88.54	88.49
(**freq.**)	87.99	88.10	88.48	88.23	88.23	87.97	88.27	88.37
$N = 100$ (**app.**)	88.26	87.93	88.31	88.17	88.33	88.34	88.53	88.24
(**freq.**)	88.49	88.07	88.22	88.12	88.38	88.48	88.25	88.59
$N = 1{,}000$ (**app.**)	87.42	87.65	87.50	87.64	87.69	87.90	87.82	87.76
(**freq.**)	87.71	87.85	87.78	87.88	88.06	88.17	87.86	88.03
$\zeta = 3$								
$N = 1$ (**app.**)	88.13				88.32			
(**freq.**)	88.14				88.43			
$N = 10$ (**app.**)	88.58	88.34	88.33	88.23	88.44	88.62	88.63	88.72
(**freq.**)	88.35	88.12	88.23	88.23	88.47	88.44	88.48	88.43
$N = 100$ (**app.**)	88.26	88.29	88.32	88.05	88.39	88.26	88.10	88.57
(**freq.**)	88.24	88.03	88.10	87.72	88.75	88.28	88.23	88.03
$N = 1{,}000$ (**app.**)	87.60	87.62	87.55	87.42	87.74	88.10	87.91	88.08
(**freq.**)	87.95	87.69	88.00	87.71	88.46	88.20	88.13	88.23

5.3　学習された規則の例

日本語係り受け解析において学習された規則の一例をあげる．係り受け関係にあると判別する規則としては，(d1)「動詞_D_H 基本形_D_H を_M_C」(d2)「から_D_C まで_M」(d3)「早く_D_H 実現_M_H」などがあった．ここで，“_D” は，係り元の文節に出現する素性，“_M” は，係り先の文節に出現する素性，“_H” は主辞の素性，“_C” は格助詞の素性である．(d1) の例からは，「動詞の基本形は” ヲ格” を含む名詞を修飾する」という知識を，(d2), (d3) の例からは，「〜から ··· まで」，「早く ··· 実現」のようによく使われる言い回しを素性の組合せで表現できていることが分かる．

係り受け関係にないと判別される規則の例としては，(n1)「基本連用形_D_H 名詞_M_H 格助詞_M_C」，(n2)「ごめんなさい_M_H」，というものがあった．(n1) の例のように，直感的に係り受け関係になりにくいと分かる素性の組合せが，人手で組合せを指定しなくても学習されていることが分かる．また，(n2) のように慣用表現のようなものが係り先になりにくいという規則も学習されている．

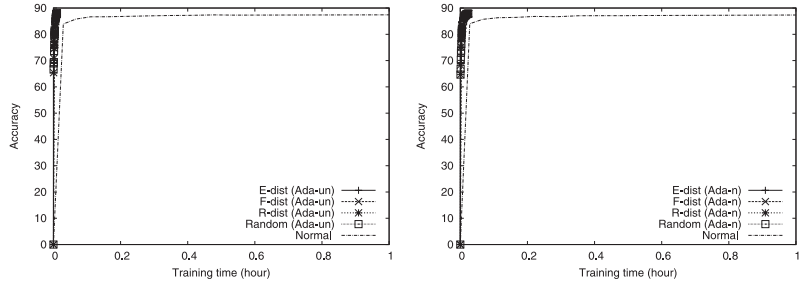


図 6　日本語係り受け解析における各学習時間で得られる規則での係り精度変化．評価には 1 月 9 日のデータを使用．Ada-un（*AdaBoost-unnormalized*）および Ada-n（*AdaBoost-normalized*），は，$N = 1{,}000$, $\zeta = 2$, $\xi = 0$, app-numbering による結果．Normal は $N = 0$, $\zeta = 2$, $\xi = 0$, freq-numbering による結果．横軸は時間（hour），縦軸は係り受け精度（accuracy）

Fig. 6　Accuracy obtained with rules at each training time on test data. Ada-un (*AdaBoost-unnormalized*) and Ada-n (*AdaBoost-normalized*) mean parameters of $N = 1{,}000$, $\zeta = 2$, $\xi = 0$, *AdaBoost-unnormalized* and app-numbering. Normal means parameters $N = 0$, $\zeta = 2$, $\xi = 0$ and freq-numbering. X-axis is for training time (hour). Y-axis is for dependency accuracy.

6. 先 行 研 究

6.1　English Syntactic Chunking における先行研究との比較

表 7 に ESC における先行研究との比較を載せる．本学習手法のパラメータ（$N = 1{,}000$, $\zeta = 2$, *AdaBoost-unnormalized*）で学習した規則に基づく ESC パーザは，多くの先行研究と比較し，同等あるいは良い精度を残せていることが分かる．さらに，これらのパーザを作るために用いたパラメータ（$N = 1{,}000$, $T = 10{,}000$, freq-numbering, *AdaBoost-unnormalized*）での規則学習時間は，1 時間以下であった．この結果から，本手法において，短時間で，高精度な ESC パーザが作成できることが分かる．

また，本手法では，自動で素性の組合せを発見することで，高い精度を得ている．SVMs に基づく手法[9),13)]，Re-ranking に基づく手法[12)] を除き，先行研究では人手で素性組合せを決定している．

表 7　English Syntactic Chunking における先行研究との比較．本手法の結果は，$\xi = 0$，$\zeta = 2$，$N = 1{,}000$，*AdaBoost-unnormalized* で学習した規則による

Table 7　Comparison with previous best results. We list best results obtained by ESC parsers trained with $\xi = 0$, $\zeta = 2$ and *AdaBoost-unnormalized*. $N = 1{,}000$ means bucket size for creating classifiers.

先行研究	$F_{\beta=1}$
SVMs based on IOB2 rep. [13]	93.48
Regularized Winnow (RW) [25]	93.57
RW+ linguistic features [25]	94.17
SVMs based on IOE2 rep. [9]	93.85
SVMs-voting [9]	93.91
Perceptron in two layers [3]	93.74
Alternating Structure Optimization (ASO) [1]	93.60
ASO + unlabaled data [1]	94.39
CRF [12]	93.76
CRF+Re-ranking [12]	94.12
ME based a bidirectional inference [23]	93.70

本稿手法	$F_{\beta=1}$
Boosting without bucketing	93.66
Boosting with E-dist	93.71
Boosting with F-dist	93.76
Boosting with R-dist	93.70
Boosting with Random-selection	93.75

表 8　日本語係り受け解析における先行研究との比較．本手法の結果は，$\xi = 0$，$\zeta = 3$，$N = 1{,}000$，$T = 10{,}000$，app-numbering，*AdaBoost-unnormalized* で学習した規則による．ST は Standard features，All は All features の意味

Table 8　Comparison with previous best results. We list results obtained by parasers trained with $N = 1{,}000$, $\xi = 0$, $\zeta = 3$, $T = 10{,}000$, app-numbering and *AdaBoost-unnormalized*.

Method	係り受け正解率 / 文正解率		
	1 月 9 日	1 月 10 日	1 月 15 日
CC algorithm [10] (ST)	88.17/45.92	88.80/44.76	88.00/47.24
(All)	89.22/47.90	89.80/44.94	89.55/49.79
SR algorithm [17] (ST)	88.18/45.92	88.80/44.76	88.03/47.24
(All)	89.22/47.90	89.79/44.87	89.55/49.79
CLE algorithm [14] (ST)	88.64/45.34	88.16/43.14	88.07/45.12
(All)	89.21/46.83	80.05/45.03	88.90/48.43
Tournament Model [8] (ST)	89.89/49.63	89.63/48.34	89.40/49.70
(All)	90.09/49.71	90.11/49.02	90.35/52.59
Boosting without bucketing	88.14/45.32	88.80/45.54	88.25/46.95
Boosting with E-dist	88.46/46.39	88.43/44.87	88.03/47.61
Boosting with F-dist	88.20/45.73	88.70/45.81	88.20/47.11
Boosting with R-dist	88.13/46.06	88.04/42.93	88.01/47.53
Boosting with Random-selection	88.23/44.83	88.22/44.07	88.13/47.61

6.2　日本語係り受け解析における先行研究との比較

表 8 に，日本語係り受け解析における先行研究との比較を載せる．この表の先行研究の結果は，文献 8) の結果から抜粋したものである．表 8 から，本手法より多くの素性を使っている（All features）の結果と比較では，多少精度が劣るものの，本手法とほぼ同じ素性を使っている（Standard features）の結果との比較では，同等の結果が得られていることが分かる．この表中の本手法の結果は，($\xi = 0$，$\zeta = 3$，$N = 1{,}000$，$T = 10{,}000$，app-numbering，*AdaBoost-unnormalized*）で学習した規則によるものであり，学習時間は，30 分未満であった．これらの先行研究と比較から，日本語係り受け解析においても，本手法を用いることで，短時間で，高い精度の日本語係り受け解析器が作成できることが分かる．

6.3　先行研究における Boosting に基づく学習の高速化

Pfahringer ら[16] は，組合せを発見する素性を絞り込むことで高速化を行う手法を提案している．Pfahringer らの手法では，現在の最適な規則を含む組合せだけを考慮する，そのパスに属する事例の重み和が最も大きいパスだけを探索するなどの尺度で，組合せ発見の絞り込みを行う．この手法と比較し，AdaBoost.DF は，素性の組合せを発見する際に，各素性の学習データ内での頻度に基づき組合せの生成を決定することで，*gain* の上限値による枝刈りを効率的に行う．

LazyBoosting[6] も各ラウンドで，本手法同様に，各ラウンドで計算する規則の候補を制限するという手法を導入している．LazyBoosting では，学習する規則の候補を各ラウンドでランダムに選択する．しかし，このランダムによる選択方法では，同じ学習データおよび同じ学習パラメータを用いたとしても，各ラウンドで選択される素性が同じであるという保証がないため，再現性がないといえる．これに対し，本手法における E-dist および F-dist は学習の再現性がある．また，AdaBoost.DF では，組合せを高速に発見するために規則候補の部分集合からの規則選択手法を用いている点も異なる．

Collins ら[5] は不要な *gain* の計算を行わないために，重みが更新された事例に出現する素性の *gain* だけを更新する差分計算方法を提案しており，すべての素性から 1 つの素性からなる規則を学習する場合において有効性が示されている．この手法では，ある素性が規則として選択されるような場合において，すべての規則候補を評価して最適な規則を選択するという条件下で，高速化が行える点が本提案手法と異なる．この差分計算による手法と本手法による効果は，タスクに依存すると考えられる．しかし，この差分計算手法を，本手法のように，枝刈りを入れながら，素性の組合せからなる規則を学習に適用することを考えると，各ラウンドで計算されていない候補が存在するという問題のため単純に適用ができ

ない．また，差分計算および差分計算による高速化では，素性の組合せを考慮した場合は，膨大な記憶領域が必要になるという点への対処が必要になると考えられる．

Boosting に基づく規則学習の高速化においては，次のような手法も提案されている．Kudo らは，同じ規則が複数回選択されるという Boosting の傾向から，短時間での収束のために，過去に見付けた規則から規則を探索する Boosting ラウンドを導入している[12]．また，Sebastiani らは，各ラウンドで複数の規則を選択し，それらを組み合わせて 1 つの弱仮説とする AdaBoost.MH^{KR} というアルゴリズムを提案している[21]．今後は，本提案手法に加えて，先行研究で用いられている手法を利用することでさらなる高速化が期待される．

7. ま と め

本稿では，素性の組合せを規則として学習する Boosting に基づく学習手法の高速化を提案した．本提案手法では，素性の組合せで表現される規則候補の生成において，重複なくかつ枝刈りに適した生成を行うことで，学習時間の短縮を行う．さらに，各 Boosting ラウンドにおいて素性の部分集合から生成される候補を対象に規則を学習することで，各ラウンドでの規則候補の評価時間を削減する．

自然言語処理タスクである English Syntactic Chunking と日本語係り受け解析で評価を行った．その結果，本提案手法を用いない場合と比較し，提案手法では精度を保持したまま，100 倍以上の学習時間の改善が行えることを確認した．

今後の課題としては，本提案手法を，Decision Tree を弱学習器として扱う Boosting アルゴリズム[2]，木構造分類のための部分木を規則として学習する Boosting アルゴリズム[11] など，異なる弱学習器を扱う Boosting アルゴリズムへの適用および評価があげられる．また，本提案手法の様々なタスクでの評価，オンライン学習アルゴリズム[4] などより低い計算量で学習可能なアルゴリズムとの比較があげられる．

参 考 文 献

1) Ando, R. and Zhang, T.: A high-performance semi-supervised learning method for text chunking, *Proc. 43rd ACL*, pp.1–9 (June 2005).
2) Carreras, X., Màrques, L. and Padró, L.: Named entity extraction using adaboost, *Proc. CoNLL-2002*, pp.167–170 (2002).
3) Carreras, X. and Màrquez, L.: Phrase recognition by filtering and ranking with perceptrons, *RANLP*, pp.205–216 (2003).
4) Collins, M.: Discriminative training methods for Hidden Markov Models: Theory and experiments with perceptron algorithms, *Proc. EMNLP'02*, pp.1–8 (2002).
5) Collins, M. and Koo, T.: Discriminative reranking for natural language parsing, *Comput. Linguist.*, Vol.31, No.1, pp.25–70 (2005).
6) Escudero, G., Màrquez, L. and Rigau, G.: Boosting applied to word sense disambiguation, *Proc. 11th ECML*, pp.129–141 (2000).
7) Freund, Y. and Mason, L.: The alternating decision tree learning algorithm, *Proc. 16th ICML*, pp.124–133 (1999).
8) Iwatate, M., Asahara, M. and Matsumoto, Y.: Japanese dependency parsing using a tournament model, *Coling 2008*, pp.361–368 (2008).
9) Kudo, T. and Matsumoto, Y.: Chunking with Support Vector Machines, *Proc. NAACL 2001*, pp.192–199 (2001).
10) Kudo, T. and Matsumoto, Y.: Japanese dependency analysis using cascaded chunking, *Proc. CoNLL-2002*, pp.63–69 (2002).
11) Kudo, T. and Matsumoto, Y.: A boosting algorithm for classification of semi-structured text, *Proc. EMNLP 2004*, pp.301–308 (July 2004).
12) Kudo, T., Suzuki, J. and Isozaki, H.: Boosting-based parse reranking with subtree features, *Proc. 43rd ACL*, pp.189–196, Association for Computational Linguistics (June 2005).
13) Kudo, T. and Matsumoto, Y.: Use of support vector learning for chunk identification, *Proc. CoNLL-2000 and LLL-2000*, pp.142–144 (2000).
14) McDonald, R., Crammer, K. and Pereira, F.: Online large-margin training of dependency parsers, *Proc. ACL'05*, pp.91–98 (2005).
15) Morishita, S.: Computing optimal hypotheses efficiently for boosting, *Progress in Discovery Science*, pp.471–481, Springer (2002).
16) Pfahringer, B., Holmes, G. and Kirkby, R.: Optimizing the induction of alternating decision trees, *Proc. PAKDD*, pp.477–487 (2001).
17) Sassano, M.: Linear-time dependency analysis for Japanese, *Proc. COLING 2004*, pp.8–14 (2004).
18) Schapire, R.E.: Theoretical views of boosting and applications, *ALT*, pp.13–25 (1999).
19) Schapire, R.E. and Singer, Y.: Improved boosting algorithms using confidence-rated predictions, *Machine Learning*, Vol.37, No.3, pp.297–336 (1999).
20) Schapire, R.E. and Singer, Y.: Boostexter: A boosting-based system for text categorization, *Machine Learning*, Vol.39, No.2/3, pp.135–168 (2000).
21) Sebastiani, F., Sperduti, A. and Valdambrini, N.: An improved boosting algorithm and its application to text categorization, *Proc. 9th CIKM*, pp.78–85 (2000).
22) Tjong, E.F., Sang, K. and Buchholz, S.: Introduction to the conll-2000 shared task: Chunking, *Proc. CoNLL-2000 and LLL-2000*, pp.127–132, Lisbon, Portugal

(2000).
23) Tsuruoka, Y. and Tsujii, J.: Bidirectional inference with the easiest-first strategy for tagging sequence data, *HLT/EMNLP* (2005).
24) Vapnik, V.: *Statistical Learning Theory*, John Wiley & Sons (1998).
25) Zhang, T., Damerau, F. and Johnson, D.: Text chunking using regularized winnow, *ACL*, pp.539–546 (2001).

付　　録

A. *AdaBoost-normalized* に基づく学習アルゴリズムの収束性

まず，文献 19) の Theorem 1 で証明されている AdaBoost（本稿の *AdaBoost-normalized*）のエラーの上限値について紹介する．このトレーニングエラーの上限値は各ラウンドでの事例の重みの積となることを示す．続いて，このトレーニングエラーがいかなる規則が追加された場合においても，トレーニングエラーの上限値は，1 つ前のトレーニングエラーの上限値より減少するかあるいは同一となることを示す．

まず，Schapire ら[19] により提案されている AdaBoost に基づき導出される T 個の規則から構成される F はトレーニングエラーの上限値が

$$\frac{1}{m}\sum_{i=1}^{m}[[F(\mathbf{x}_i)\neq y_i]]\leq\prod_{t=1}^{T}Z_t$$

となることが示されている[19]．まず，このトレーニングエラーの上限値について説明する．

$w_{1,i}=1/m$ とすると，重み更新規則式 (5) を展開することで，式 (8) が得られる．

$$w_{T+1,i}=\frac{\exp(-y_i\sum_{t=1}^{T}h_{\langle\mathbf{f}_t,c_t\rangle}(\mathbf{x}_i))}{m\prod_{t=1}^{T}Z_t}. \tag{8}$$

さらに，$F(\mathbf{x}_i)\neq y_i$ の場合，$F(\mathbf{x}_i)y_i=sign(\sum_{t=1}^{T}h_{\langle\mathbf{f}_t\rangle}(\mathbf{x}_i))y_i<0$ なので，

$$\exp\left(-y_i\sum_{t=1}^{T}h_{\langle\mathbf{f}_t,c_t\rangle}(\mathbf{x}_i)\right)\geq 1.$$

したがって，

$$[[F(\mathbf{x}_i)\neq y_i]]\leq\exp\left(-y_i\sum_{t=1}^{T}h_{\langle\mathbf{f}_t,c_t\rangle}(\mathbf{x}_i)\right). \tag{9}$$

よって，式 (8)，(9) から上述のトレーニングエラーの上限値が得られる．

$$\begin{aligned}\frac{1}{m}\sum_{i=1}^{m}[[F(\mathbf{x}_i)\neq y_i]]&\leq\frac{1}{m}\sum_{i=1}^{m}\exp\left(-y_i\sum_{t=1}^{T}h_{\langle\mathbf{f}_t,c_t\rangle}(\mathbf{x}_i)\right)\\&=\sum_{i=1}^{m}\left(\prod_{t=1}^{T}Z_t\right)w_{T+1,i}\\&=\prod_{t=1}^{T}Z_t\end{aligned}$$

続いて，*AdaBoost-normalized* により導出される $T-1$ 個の規則に対し，新たな規則が追加される場合に，いかなる規則が追加されたとしても，$T-1$ 個の規則と T 個の規則から得られるトレーニングエラーの上限値は次の関係

$$\prod_{t=1}^{T}Z_t\leq\prod_{t=1}^{T-1}Z_t$$

を持つことを示す．

まず，T 個の規則から得られるトレーニングエラーの上限値 $\prod_{t=1}^{T}Z_T$ は次のように書きかえられる．

$$\prod_{t=1}^{T}Z_t=\left(\prod_{t=1}^{T-1}Z_t\right)Z_T \tag{10}$$

ここで，Z_T を，定義 2 を基に書きかえると

$$\begin{aligned}Z_T&=\sum_{i=1}^{m}w_{T-1,i}\ \exp(-y_ih_{\langle\mathbf{f}_T,c_T\rangle}(\mathbf{x}_i))\\&=W_{T-1}(\neg\mathbf{f}_T)-W_{T-1,+1}(\mathbf{f}_T)\exp(-c_T)+W_{T-1,-1}(\mathbf{f}_T)\exp(c_T),\end{aligned}$$

となる．ここで，$W_{T-1}(\neg\mathbf{f})=\sum_{i=1}^{m}w_{T-1,i}-W_{T-1,+1}(\mathbf{f})-W_{T-1,-1}(\mathbf{f})$ である．

よって，$c_T=\frac{1}{2}\log(\frac{W_{T,+1}(\mathbf{f_T})}{W_{T,-1}(\mathbf{f_T})})$ から，Z_T は，最終的に，次のように書きかえられる．

$$\begin{aligned}Z_T&=W_{T-1}(\neg\mathbf{f}_T)-W_{T-1,+1}(\mathbf{f}_T)\exp(-c_T)+W_{T-1,-1}(\mathbf{f}_T)\exp(c_T)\\&=\sum_{i=1}^{m}w_{T-1,i}-W_{T-1,+1}(\mathbf{f}_T)-W_{T-1,-1}(\mathbf{f}_T)\\&\quad-W_{T-1,+1}(\mathbf{f}_T)\exp(-c_T)+W_{T-1,-1}(\mathbf{f}_T)\exp(c_T)\\&=\sum_{i=1}^{m}w_{T-1,i}-W_{T-1,+1}(\mathbf{f}_T)-W_{T-1,-1}(\mathbf{f}_T)+2\sqrt{W_{T-1,+1}(\mathbf{f}_T)W_{T-1,-1}(\mathbf{f}_T)}\end{aligned}$$

$$= \sum_{i=1}^{m} w_{T-1,i} - \left(\sqrt{W_{T-1,+1}(\mathbf{f}_T)} - \sqrt{W_{T-1,-1}(\mathbf{f}_T)}\right)^2.$$

$\sum_{i=1}^{m} w_{T-1,i} = 1$ なので，次の式が得られる．

$$Z_T = 1 - \left(\sqrt{W_{t,+1}(\mathbf{f}_T)} - \sqrt{W_{t,-1}(\mathbf{f}_T)}\right)^2 \leq 1. \tag{11}$$

式 (10) および式 (11) から，

$$\begin{aligned}\prod_{t=1}^{T} Z_t &= \left(\prod_{t=1}^{T-1} Z_t\right) Z_T \\ &= \left(\prod_{t=1}^{T-1} Z_t\right) \left(1 - \left(\sqrt{W_{T,+1}(\mathbf{f}_T)} - \sqrt{W_{T,-1}(\mathbf{f}_T)}\right)^2\right) \\ &\leq \left(\prod_{t=1}^{T-1} Z_t\right),\end{aligned}$$

が得られる．よって，いかなる規則が追加された場合においても，トレーニングエラーの上限値は，1 つ前のトレーニングエラーの上限値より減少するかあるいは同一となる．

B. *AdaBoost-unnormalized* に基づく学習アルゴリズムの収束性

AdaTrees 学習アルゴリズム[7] で用いられている AdaBoost（本稿の *AdaBoost-unnormalized*）のトレーニングエラーの上限値は各ラウンドでの更新後の事例の重みの和となる．このことを，文献 19) の Theorem 1 に基づき導出する．また，AdaTrees 学習アルゴリズム[7] で用いられている AdaBoost において，いかなる規則が追加された場合においても，トレーニングエラーの上限値は，1 つ前のトレーニングエラーの上限値より減少するかあるいは，最悪の場合でも同一となることを示す．

まず，AdaTrees 学習アルゴリズム[7] で用いられている AdaBoost により導出される T 個の規則から構成される F はトレーニングエラーの上限値

$$\sum_{i=1}^{m} [[F(\mathbf{x}_i) \neq y_i]] \leq Z'_T$$

を持つことを示す．ここで，

$$Z'_T = \sum_{i=1}^{m} w_{T+1,i}$$

である．

重み更新規則式 (6) を展開することで，式 (12) が得られる．

$$w_{T+1,i} = \exp\left(-y_i \sum_{t=1}^{T} h_{\langle \mathbf{f}_t, c_t \rangle}(\mathbf{x}_i)\right). \tag{12}$$

さらに，$F(\mathbf{x}_i) \neq y_i$ の場合，$F(\mathbf{x}_i) y_i = sign(\sum_{t=1}^{T} h_{\langle \mathbf{f}_t \rangle}(\mathbf{x}_i)) y_i < 0$ なので，

$$\exp\left(-y_i \sum_{t=1}^{T} h_{\langle \mathbf{f}_t, c_t \rangle}(\mathbf{x}_i)\right) \geq 1.$$

したがって，

$$[[F(\mathbf{x}_i) \neq y_i]] \leq \exp\left(-y_i \sum_{t=1}^{T} h_{\langle \mathbf{f}_t, c_t \rangle}(\mathbf{x}_i)\right). \tag{13}$$

よって，式 (12)，(13) からトレーニングエラーの上限値が得られる．

$$\begin{aligned}\sum_{i=1}^{m} [[F(\mathbf{x}_i) \neq y_i]] &\leq \sum_{i=1}^{m} \exp\left(-y_i \sum_{t=1}^{T} h_{\langle \mathbf{f}_t, c_t \rangle}(\mathbf{x}_i)\right) \\ &= \sum_{i=1}^{m} w_{T+1,i} = Z'_T\end{aligned}$$

続いて，AdaTrees 学習アルゴリズム[7] で用いられている AdaBoost により導出される $T-1$ 個の規則に対し，新たな規則が追加される場合に，いかなる規則が追加されたとしても $T-1$ 個の規則と T 個の規則から得られるトレーニングエラーの上限値は次の関係を持つことを示す．

$Z'_T \leq Z'_{T-1}$

定義 2 から，T 個の規則から得られるトレーニングエラーの上限値 $Z'_T = \sum_{i=1}^{m} w_{T+1,i}$ は次のように書きかえられる．

$$\begin{aligned}Z'_T &= \sum_{i=1}^{m} w_{T+1,i} \\ &= \sum_{i=1}^{m} w_{T,i} \exp(-y_i h_{\langle \mathbf{f}_T, c_T \rangle}(\mathbf{x}_i)) \\ &= W_T(\neg \mathbf{f}) - W_{T,+1}(\mathbf{f}_T) \exp(-c_T) + W_{T,-1}(\mathbf{f}_T) \exp(c_T).\end{aligned}$$

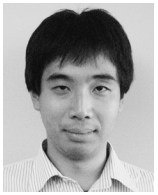
ここで，$W_T(\neg\mathbf{f}) = \sum_{i=1}^{m} w_{T,i} - W_{T,+1}(\mathbf{f}) - W_{T,-1}(\mathbf{f})$ である．

$c_T = \frac{1}{2}\log(\frac{W_{T,+1}(\mathbf{f_T})}{W_{T,-1}(\mathbf{f_T})})$ から，T 個の規則から得られるトレーニングエラーの上限値に対して，

$$\begin{aligned}
Z'_T &= W_T(\neg\mathbf{f}_T) - W_{T,+1}(\mathbf{f}_T)\exp(-c_T) + W_{T,-1}(\mathbf{f}_T)\exp(c_T) \\
&= \sum_{i=1}^{m} w_{T,i} - W_{T,+1}(\mathbf{f}_T) - W_{T,-1}(\mathbf{f}_T) - W_{T,+1}(\mathbf{f}_T)\exp(-c_T) + W_{T,-1}(\mathbf{f}_T)\exp(c_T) \\
&= \sum_{i=1}^{m} w_{T,i} - W_{T,+1}(\mathbf{f}_T) - W_{T,-1}(\mathbf{f}_T) + 2\sqrt{W_{T,+1}(\mathbf{f}_T)W_{T,-1}(\mathbf{f}_T)} \\
&= \sum_{i=1}^{m} w_{T,i} - \left(\sqrt{W_{T,+1}(\mathbf{f}_T)} - \sqrt{W_{T,-1}(\mathbf{f}_T)}\right)^2 \\
&= Z'_{T-1} - \left(\sqrt{W_{T,+1}(\mathbf{f}_T)} - \sqrt{W_{T,-1}(\mathbf{f}_T)}\right)^2 \leq Z'_{T-1}. \qquad (14)
\end{aligned}$$

が得られる．よって，いかなる規則が追加された場合においても，トレーニングエラーの上限値は，1 つ前のトレーニングエラーの上限値より減少するかあるいは同一となる．

(平成 20 年 6 月 19 日受付)
(平成 21 年 2 月 3 日採録)



岩倉　友哉（正会員）

2003 年 3 月九州工業大学大学院情報工学研究科博士前期課程修了．同年株式会社富士通研究所入社．自然言語処理・機械学習の研究開発に従事．

岡本　青史

1991 年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．1998 年学位（理学（博士））取得．株式会社富士通研究所勤務．東京大学大学院客員教授，北陸先端科学技術大学院大学客員教授．推論・機械学習・情報検索・知識発見の研究開発に従事．人工知能学会会員．