A Fast Boosting-based Learner for Feature-Rich Tagging and Chunking

Tomoya Iwakura Seishi Okamoto
Fujitsu Laboratories Ltd.
1-1, Kamikodanaka 4-chome, Nakahara-ku, Kawasaki 211-8588, Japan
{iwakura.tomoya,seishi}@jp.fujitsu.com

Abstract

Combination of features contributes to a significant improvement in accuracy on tasks such as part-of-speech (POS) tagging and text chunking, compared with using atomic features. However, selecting combination of features on learning with large-scale and feature-rich training data requires long training time. We propose a fast boosting-based algorithm for learning rules represented by combination of features. Our algorithm constructs a set of rules by repeating the process to select several rules from a small proportion of candidate rules. The candidate rules are generated from a subset of all the features with a technique similar to beam search. Then we propose POS tagging and text chunking based on our learning algorithm. Our tagger and chunker use candidate POS tags or chunk tags of each word collected from automatically tagged data. We evaluate our methods with English POS tagging and text chunking. The experimental results show that the training time of our algorithm are about 50 times faster than Support Vector Machines with polynomial kernel on the average while maintaining state-of-the-art accuracy and faster classification speed.

1 Introduction

Several boosting-based learning algorithms have been applied to Natural Language Processing problems successfully. These include text categorization (Schapire and Singer, 2000), Natural Language Parsing (Collins and Koo, 2005), English syntactic chunking (Kudo et al., 2005) and so on.

Furthermore, classifiers based on boosting-based learners have shown fast classification speed (Kudo et al., 2005).

However, boosting-based learning algorithms require long training time. One of the reasons is that boosting is a method to create a final hypothesis by repeatedly generating a weak hypothesis in each training iteration with a given weak learner. These weak hypotheses are combined as the final hypothesis. Furthermore, the training speed of boosting-based algorithms becomes more of a problem when considering combination of features that contributes to improvement in accuracy.

This paper proposes a fast boosting-based algorithm for learning rules represented by combination of features. Our learning algorithm uses the following methods to learn rules from large-scale training samples in a short time while maintaining accuracy; 1) Using a rule learner that learns several rules as our weak learner while ensuring a reduction in the theoretical upper bound of the training error of a boosting algorithm, 2) Repeating to learn rules from a small proportion of candidate rules that are generated from a subset of all the features with a technique similar to beam search, 3) Changing subsets of features used by weak learner dynamically for alleviating overfitting.

We also propose feature-rich POS tagging and text chunking based on our learning algorithm. Our POS tagger and text chunker use candidate tags of each word obtained from automatically tagged data as features.

The experimental results with English POS tagging and text chunking show drastically improvement of training speeds while maintaining competitive accuracy compared with previous best results and fast classification speeds.

2 Boosting-based Learner

2.1 Preliminaries

We describe the problem treated by our boosting-based learner as follows. Let $\mathcal{X}$ be the set of examples and $\mathcal{Y}$ be a set of labels $\{-1, +1\}$. Let $\mathcal{F} = \{f_1, f_2, ..., f_M\}$ be M types of features represented by strings. Let S be a set of training sam-

CoNLL 2008: Proceedings of the 12th Conference on Computational Natural Language Learning, pages 17–24
Manchester, August 2008

$S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m : \mathbf{x}_i \subseteq \mathcal{X}, y_i \in \{\pm 1\}$
a smoothing value ε =1
rule number r: the initial value is 1.
Initialize: **For** i=1,...,m: $w_{1,i} = exp(\frac{1}{2}\log(\frac{W_{+1}}{W_{-1}}))$;
While $(r \leq R)$
Train weak-learner using $(S, \{w_{r,i}\}_{i=1}^m)$
Get ν types of rules: $\{\mathbf{f}_j\}_{j=1}^{\nu}$
$\{\mathbf{f}_j\}_{j=1}^{\nu} \leftarrow$ **weak-learner**$(S, \{w_{r,i}\}_{i=1}^m)$;
Update weights with confidence value
Foreach $\mathbf{f} \in \{\mathbf{f}_j\}_{j=1}^{\nu}$
$c = \frac{1}{2}\log(\frac{W_{r,+1}(\mathbf{f})+\varepsilon}{W_{r,-1}(\mathbf{f})+\varepsilon})$
For i=1,...,m: $w_{r+1,i} = w_{r,i}\, exp(-y_i h_{\langle \mathbf{f}, c\rangle})$
$\mathbf{f}_r = \mathbf{f}$; $c_r = c$; r++;
endForeach
endWhile
Output: $F(\mathbf{x}) = sign(\log(\frac{W_{+1}}{W_{-1}}) + \sum_{r=1}^R h_{\langle \mathbf{f}_r, c_r\rangle}(\mathbf{x}))$

Figure 1: A generalized version of our learner

ples $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)\}$, where each example $\mathbf{x}_i \in \mathcal{X}$ consists of features in $\mathcal{F}$, which we call a feature-set, and $y_i \in \mathcal{Y}$ is a class label. The goal is to induce a mapping

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

from S.

Let $|\mathbf{x}_i|$ $(0 < |\mathbf{x}_i| \leq M)$ be the number of features included in a feature-set $\mathbf{x}_i$, which we call the size of $\mathbf{x_i}$, and $x_{i,j} \in \mathcal{F}$ $(1 \leq j \leq |\mathbf{x}_i|)$ be a feature included in $\mathbf{x}_i$. [1] We call a feature-set of size k a k-feature-set. Then we define subsets of feature-sets as follows.

Definition 1 *Subsets of feature-sets*

If a feature-set $\mathbf{x}_j$ contains all the features in a feature-set $\mathbf{x}_i$, then we call $\mathbf{x}_i$ is a subset of $\mathbf{x}_j$ and denote it as

$$\mathbf{x_i} \subseteq \mathbf{x_j}.$$

Then we define weak hypothesis based on the idea of the real-valued predictions and abstaining (RVPA, for short) (Schapire and Singer, 2000). [2]

Definition 2 *Weak hypothesis for feature-sets*

Let $\mathbf{f}$ be a feature-set, called a rule, $\mathbf{x}$ be a feature-set, and c be a real number, called a confidence value, then a weak-hypothesis for feature-sets is defined as

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

[1] Our learner can handle binary vectors as in (Morishita, 2002). When our learner treats binary vectors for M attributes $\{\mathbf{X}_1, ..., \mathbf{X}_m\}$, the learner converts each vector to the corresponding feature-set as $\mathbf{x}_i \leftarrow \{f_i | X_{i,j} \in \mathbf{X}_i \wedge X_{i,j} = 1\}$ $(1 \leq i \leq m, 1 \leq j \leq M)$.

[2] We use the RVPA because training with RVPA is faster than training with Real-valued-predictions (RVP) while maintaining competitive accuracy (Schapire and Singer, 2000). The idea of RVP is to output a confidence value for samples which do not satisfy the given condition too.

2.2 Boosting-based Rule Learning

Our boosting-based learner selects R types of rules for creating a final hypothesis F on several training iterations. The F is defined as

$$F(\mathbf{x}) = sign(\textstyle\sum_{r=1}^R h_{\langle \mathbf{f}_r, c_r\rangle}(\mathbf{x})).$$

We use a learning algorithm that generates several rules from a given training samples $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ and weights over samples $\{w_{r,1}, ..., w_{r,m}\}$ as input of our weak learner. $w_{r,i}$ is the weight of sample number i after selecting $r-1$ types of rules, where $0 < w_{r,i}$, $1 \leq i \leq m$ and $1 \leq r \leq R$.

Given such input, the weak learner selects ν types of rules $\{\mathbf{f}_j\}_{j=1}^{\nu}$ $(\mathbf{f}_j \subseteq \mathcal{F})$ with $gain$:

$$gain(\mathbf{f}) \stackrel{\text{def}}{=} |\sqrt{W_{r,+1}(\mathbf{f})} - \sqrt{W_{r,-1}(\mathbf{f})}|,$$

where $\mathbf{f}$ is a feature-set, and $W_{r,y}(\mathbf{f})$ is

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

and $[[\pi]]$ is 1 if a proposition π holds and 0 otherwise.

The weak learner selects a feature-set having the highest $gain$ as the first rule, and the weak learner finally selects ν types of feature-sets having $gain$ in top ν as $\{\mathbf{f}_j\}_{j=1}^{\nu}$ at each boosting iteration.

Then the boosting-based learner calculates the confidence value of each $\mathbf{f}$ in $\{\mathbf{f}_j\}_{j=1}^{\nu}$ and updates the weight of each sample. The confidence value c_j for $\mathbf{f}_j$ is defined as

$$c_j = \tfrac{1}{2}\log(\tfrac{W_{r,+1}(\mathbf{f}_j)}{W_{r,-1}(\mathbf{f}_j)}).$$

After the calculation of c_j for $\mathbf{f}_j$, the learner updates the weight of each sample with

$$w_{r+1,i} = w_{r,i} exp(-y_i h_{\langle \mathbf{f}_j, c_j\rangle}). \quad (1)$$

Then the learner adds $(\mathbf{f_j}, c_j)$ to F as the r-th rule and its confidence value. [3] When we calculate the confidence value c_{j+1} for $\mathbf{f}_{j+1}$, we use $\{w_{r+1,1}, ..., w_{r+1,m}\}$. The learner adds $(\mathbf{f}_{j+1}, c_{j+1})$ to F as the $r+1$-th rule and confidence value.

After the updates of weights with $\{\mathbf{f}_j\}_{j=1}^{\nu}$, the learner starts the next boosting iteration. The learner continues training until obtaining R rules.

Our boosting-based algorithm differs from the other boosting algorithms in the number of rules learned at each iteration. The other boosting-based algorithms usually learn a rule at each iteration

[3] Eq. (1) is the update of the AdaBoost used in ADTrees learning algorithm (Freund and Mason, 1999). We use this AdaBoost by the following two reasons. 1) The paper (Iwakura and Okamoto, 2007) showed that the accuracy of text chunking with the AdaBoost of ADTrees is slightly higher than text chunking with the AdaBoost of BoosTexter for RVPA (Schapire and Singer, 2000), 2) We expect the AdaBoost of ADTrees can realize faster training because this AdaBoost does not normalize weights at each update compared with the AdaBoost of BoosTexter normalizes weights at each iteration.

$sortByW(\mathcal{F}, fq)$: Sort features ($f \in \mathcal{F}$)
in ascending order based on weights of features
$(a \,\%\, b)$: Return the reminder of $(a \div b)$
$|B|$-buckets: $B = \{B[0], ..., B[|B|-1]\}$
procedure **distFT**(S, $|B|$)
##Calculate the weight of each feature
Foreach ($f \in \mathcal{F}$) $W_r(f) = \sum_{i=1}^{m} w_{r,i}[[\{f\} \subseteq \mathbf{x}_i]]$
##Sort features based on thier weights and
store the results in Fs
$Fs \leftarrow sortByW(\mathcal{F}, W_r)$
Distribute features to buckets
For i=0...M : $B[(i \,\%\, |B|)] = (B[(i \,\%\, |B|)] \cup Fs[i])$
return B

Figure 2: Distribute features to buckets based on weights

(Schapire and Singer, 2000; Freund and Mason, 1999). Despite the difference, our boosting-based algorithm ensures a reduction in the theoretical upper bound of training error of the AdaBoost. We list the detailed explanation in Appendix.A.

Figure 1 shows an overview of our boosting-based rule learner. To avoid to happen that $W_{r,+1}(\mathbf{f})$ or $W_{r,-1}(\mathbf{f})$ is very small or even zero, we use the smoothed values ε (Schapire and Singer, 1999). Furthermore, to reflect imbalance class distribution, we use the default rule (Freund and Mason, 1999), defined as $\frac{1}{2}\log(\frac{W_{+1}}{W_{-1}})$, where $W_y = \sum_{i=1}^{m}[[y_i = y]]$ for $y \in \{\pm 1\}$. The initial weights are defined with the default rule.

3 Fast Rule Learner

3.1 Generating Candidate Rules

We use a method to generate candidate rules without duplication (Iwakura and Okamoto, 2007). We denote $\mathbf{f}' = \mathbf{f} + f$ as the generation of $k+1$-feature-set $\mathbf{f}'$ consisting of a feature f and a k-feature-set $\mathbf{f}$. Let $ID(f)$ be the integer corresponding to f, called id, and ϕ be 0-feature-set. Then we define gen generating a feature-set as

$$gen(\mathbf{f}, f) = \begin{cases} \mathbf{f} + f & if\ ID(f) > \max_{f' \in \mathbf{f}} ID(f') \\ \phi & otherwise \end{cases}.$$

We assign smaller integer to more infrequent features as id. If there are features having the same frequency, we assign id to each feature with lexicographic order of features. Training based on this candidate generation showed faster training speed than generating candidates by an arbitrary order (Iwakura and Okamoto, 2007).

3.2 Training with Redistributed Features

We propose a method for learning rules by repeating to select a rule from a small portion of candidate rules. We evaluated the effectiveness of four types of methods to learn a rule from a subset of features on boosting-based learners with a text chunking task (Iwakura and Okamoto, 2007). The results showed that Frequency-based distribution (*F-dist*) has shown the best accuracy. *F-dist* distributes features to subsets of features, called buckets, based on frequencies of features.

F_k : A set of k-feature-sets
$\mathcal{R}_o$: ν optimal rules (feature-sets)
$R_{k,\omega}$: ω k-feature-sets for generating candidates
selectNBest($\mathcal{R}$, n, S, W_r): n best rules from $\mathcal{R}$
with $gain$ on $\{w_{i,r}\}_{i=1}^{m}$ and training samples S
procedure **weak-learner**(F_k, S, W_r)
ν best feature-sets as rules
$\mathcal{R}_o$ = **selectNBest**($\mathcal{R}_o \cup F_k$, ν, S, W_r);
if ($\zeta \leq k$) **return** $\mathcal{R}_o$; ## Size constraint
ω best feature-sets in F_k for generating candidates
$R_{k,\omega}$ = **selectNBest**(F_k, ω, S, W_r);
$\tau = \min_{\mathbf{f} \in \mathcal{R}_o} gain(\mathbf{f})$; ## The $gain$ of ν-th optimal rule
Foreach ($\mathbf{f_k} \in R_{k,\omega}$)
if ($u(\mathbf{f_k}) < \tau$) **continue**; ## Upper bound of gain
Foreach ($f \in \mathcal{F}$) ## Generate candidates
$\mathbf{f_{k+1}} = gen(\mathbf{f_k}, f)$;
if ($\xi \leq \sum_{i=1}^{m}[[\mathbf{f_{k+1}} \subseteq \mathbf{x_i}]]$) $F_{k+1} = (F_{k+1} \cup \mathbf{f_{k+1}})$;
end Foreach
end Foreach
return weak-learner(F_{k+1}, S, W);

Figure 3: Find optimal feature-sets with given weights

However, we guess training using a subset of features depends on how to distribute features to buckets like online learning algorithms that generally depend on the order of the training examples (Kazama and Torisawa, 2007).

To alleviate the dependency on selected buckets, we propose a method that redistributes features, called Weight-based distribution (*W-dist*). *W-dist* redistributes features to buckets based on the weight of feature defined as

$$W_r(f) = \sum_{i=1}^{m} w_{r,i}[[\{f\} \subseteq \mathbf{x}_i]]$$

for each $f \in \mathcal{F}$ after examining all buckets. Figure 2 describes an overview of *W-dist*.

3.3 Weak Learner for Learning Several Rules

We propose a weak learner that learns several rules from a small portion of candidate rules.

Figure 3 describes an overview of the weak learner. At each iteration, one of the $|B|$-buckets is given as an initial 1-feature-sets F_1. The weak learner finds ν best feature-sets as rules from candidates consisting of F_1 and feature-sets generated from F_1. The weak learner generates candidates k-feature-sets ($1 < k$) from ω best (k-1)-feature-sets in F_{k-1} with $gain$.

We also use the following pruning techniques (Morishita, 2002; Kudo et al., 2005).

- **Frequency constraint**: We examine candidates seen on at least ξ different examples.
- **Size constraint**: We examine candidates whose size is no greater than a size threshold ζ.
- **Upper bound of gain**: We use the upper bound of gain defined as

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

For any feature-set $\mathbf{f}' \subseteq \mathcal{F}$, which contains $\mathbf{f}$ (i.e. $\mathbf{f} \subseteq \mathbf{f}'$), the $gain(\mathbf{f}')$ is bounded under $u(\mathbf{f})$, since $0 \leq W_{r,y}(\mathbf{f}') \leq W_{r,y}(\mathbf{f})$ for $y \in \{\pm 1\}$. Thus, if $u(\mathbf{f})$

$S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$: $\mathbf{x}_i \subseteq \mathcal{X}$, $y_i \in \{\pm 1\}$
$W_r = \{w_{r,i}\}_{i=1}^{m}$: Weights of samples after learning
r types of rules. $w_{1,i} = 1$ $(1 \leq i \leq m)$
$|B|$: The size of bucket $B = \{B[0], ..., B[|B|-1]\}$
b, r : The current bucket and rule number
procedure **AdaBoost.SDF**()
B = **distFT**$(S, |B|)$; ## *Distributing features into* B
Initialize values and weights:
$r = 1$; $b = 0$; $c_0 = \frac{1}{2} log(\frac{W_{+1}}{W_{-1}})$;
For i = 1,...,m: $w_{1,i} = exp(c_0)$;
While $(r \leq R)$ ## *Learning* R *types of rules*
##Select ν rules and increment bucket id b
$\mathcal{R}$ = **weak-learner**$(B[b], S, W_r)$; b++;
Foreach $(\mathbf{f} \in \mathcal{R})$ ##Update weights with each rule
$c = \frac{1}{2}\log(\frac{W_{r,+1}(\mathbf{f})+1}{W_{r,-1}(\mathbf{f})+1})$;
For i=1,..,m $w_{r+1,i} = w_{r,i}\, exp(-y_i h_{\langle \mathbf{f}, c \rangle})$;
$\mathbf{f}_r = \mathbf{f}$; $c_r = c$; r++;
end Foreach
if $(b == |B|)$ ## Redistribution
B = **distFT**$(S, |B|)$; b=0;
end if
end While
return $F(\mathbf{x}) = sign(c_0 + \sum_{r=1}^{R} h_{\langle \mathbf{f}_r, c_r \rangle}(\mathbf{x}))$

Figure 4: An overview of AdaBoost.SDF

· words, words that are turned into all capitalized, prefixes and suffixes (up to 4) in a 7-word window.
· labels assigned to three words on the right.
· whether the current word has a hyphen, a number, a capital letter
· whether the current word is all capital, all small
· candidate POS tags of words in a 7-word window

Figure 5: Feature types for POS tagging

is less than the $gain$ of the current optimal rule τ, candidates containing $\mathbf{f}$ are safely pruned.

Figure 4 describes an overview of our algorithm, which we call AdaBoost for a weak learner learning *S*everal rules from *D*istributed *F*eatures (*AdaBoost.SDF*, for short).

The training of AdaBoost.SDF with ($\nu = 1, \omega = \infty, 1 < |B|$) is equivalent to the approach of AdaBoost.DF (Iwakura and Okamoto, 2007). If we use ($|B| = 1$,$\nu = 1$), AdaBoost.SDF examines all features on every iteration like (Freund and Mason, 1999; Schapire and Singer, 2000).

4 POS tagging and Text Chunking

4.1 English POS Tagging

We used the Penn Wall Street Journal treebank (Marcus et al., 1994). We split the treebank into training (sections 0-18), development (sections 19-21) and test (sections 22-24) as in (Collins, 2002). We used the following candidate POS tags, called candidate feature, in addition to commonly used features (Giménez and Màrquez, 2003; Toutanova et al., 2003) shown in Figure 5.

We collect candidate POS tags of each word from the automatically tagged corpus provided for the shared task of English Named Entity recognition in CoNLL 2003. [4] The corpus includes 17,003,926 words with POS tags and chunk tags

[4]http://www.cnts.ua.ac.be/conll2003 /ner/

· words and POS tags in a 5-word window.
· labels assigned to two words on the right.
· candidate chunk tags of words in a 5-word window

Figure 6: Feature types for text chunking

annotated by a POS tagger and a text chunker. Thus, the corpus includes wrong POS tags and chunk tags.

We collected candidate POS tags of words that appear more than 9 times in the corpus. We express these candidates with one of the following ranges decided by their frequency fq; $10 \leq fq < 100$, $100 \leq fq < 1000$ and $1000 \leq fq$.

For example, we express 'work' annotated as NN 2000 times like "1000≤NN". If 'work' is current word, we add 1000≤NN as a candidate POS tag feature of the current word. If 'work' appears the next of the current word, we add 1000≤NN as a candidate POS tag of the next word.

4.2 Text Chunking

We used the data prepared for CoNLL-2000 shared tasks. [5] This task aims to identify 10 types of chunks, such as, NP, VP and PP, and so on.

The data consists of subsets of Penn Wall Street Journal treebank; training (sections 15-18) and test (section 20). We prepared the development set from section 21 of the treebank as in (Tsuruoka and Tsujii, 2005). [6]

Each base phrase consists of one word or more. To identify word chunks, we use **IOE2** representation. The chunks are represented by the following tags: E-X is used for end word of a chunk of class X. I-X is used for non-end word in an X chunk. O is used for word outside of any chunk.

For instance, "[He] (NP) [reckons] (VP) [the current account deficit] (NP)..." is represented by IOE2 as follows; "He/E-NP reckons/E-VP the/I-NP current/I-NP account/I-NP deficit/E-NP".

We used features shown in Figure 6. We collected the followings as candidate chunk tags from the same automatically tagged corpus used in POS tagging.

- Candidate tags expressed with frequency information as in POS tagging
- The ranking of each candidate decided by frequencies in the automatically tagged data
- Candidate tags of each word

For example, if we collect "work" annotated as I-NP 2000 times and as E-VP 100 time, we generate the following candidate features for "work"; 1000≤I-NP, 100≤E-VP<1000, rank:I-NP=1 rank:E-NP=2, candidate=I-NP and candidate=E-VP.

[5]http://lcg-www.uia.ac.be/conll2000/chunking/

[6]We used http://ilk.uvt.nl/~sabine/chunklink/chunklink_2-2-2000_for_conll.pl for creating development data.

Table 1: Training data for experiments: ♯ of S, M, ♯ of cl and av. ♯ of ft indicate the number samples, the distinct number of feature types, the number of class in each data set, and the average number of features, respectively. POS and ETC indicate POS-tagging and text chunking. The "-c" indicates using candidate features collected from parsed unlabeled data.

data	♯ of S	M	♯ of cl	av. ♯ of ft
POS	912,344	579,052	45	22.09
POS-c	912,344	579,793	45	35.39
ETC	211,727	92,825	22	11.37
ETC-c	211,727	93,333	22	45.49

We converted the chunk representation of the automatically tagged corpus to IOE2 and we collected chunk tags of each word appearing more than nine times.

4.3 Applying AdaBoost.SDF

AdaBoost.SDF treats the binary classification problem. To extend AdaBoost.SDF to multi-class, we used the one-vs-the-rest method.

To identify proper tag sequences, we use Viterbi search. We map the confidence value of each classifier into the range of 0 to 1 with sigmoid function [7], and select a tag sequence which maximizes the sum of those log values by Viterbi search.

5 Experiments

5.1 Experimental Settings

We compared AdaBoost.SDF with Support Vector Machines (SVM). SVM has shown good performance on POS tagging (Giménez and Màrquez, 2003) and Text Chunking (Kudo and Matsumoto, 2001). Furthermore, SVM with polynomial kernel implicitly expands all feature combinations without increasing the computational costs. Thus, we compared AdaBoost.SDF with SVM. [8]

To evaluate the effectiveness of candidate features, we examined two types of experiments with candidate features and without them. We list the statics of training sets in Table 1.

We tested R=100,000, $|B|$=1,000, ν = {1,10,100}, ω={1,10,100,∞}, ζ={1,2,3}, and ξ={1,5} for AdaBoost.SDF. We tested the soft margin parameter C={0.1,1,10} and the kernel degree d={1,2,3} for SVM. [9]

We used the followings for comparison; *Training time* is time to learn 100,000 rules. *Best training time* is time for generating rules to show the best F-measure ($F_{\beta=1}$) on development data. *Accuracy* is $F_{\beta=1}$ on a test data with the rules at *best training time*.

[7] $s(X) = 1/(1 + exp(-\beta X))$, where $X = F(\mathbf{x})$ is a output of a classifier. We used β=5 in this experiment.

[8] We used TinySVM (http://chasen.org/~taku/software/TinySVM/).

[9] We used machines with 2.66 GHz QuadCore Intel Xeon and 10 GB of memory for all the experiments.

Table 2: Experimental results of POS tagging and Text Chunking (TC) with candidate features. F and time indicate the average $F_{\beta=1}$ of test data and time (hour) to learn 100,000 rules for all classes with *F-dist*. These results are listed separately with respect to each $\xi = \{1, 5\}$.

ν	POS($\xi = 1$)		POS ($\xi = 5$)		TC ($\xi = 1$)		TC ($\xi = 5$)	
	F	time	F	time	F	time	F	time
1	97.27	196.3	97.23	195.7	93.98	145.3	93.95	155.8
10	97.23	23.05	97.17	22.35	93.96	2.69	93.88	2.70
100	96.82	2.99	96.83	2.91	93.16	0.74	93.14	0.56

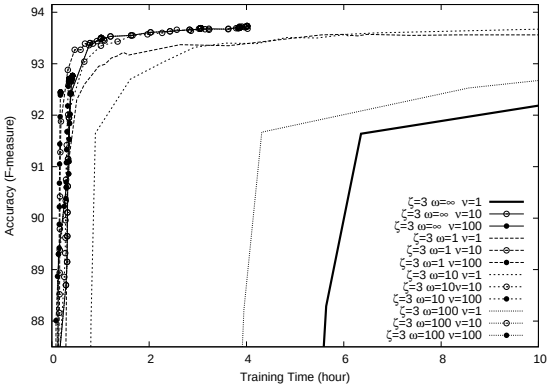

Figure 7: Accuracy on development data of Text Chunking ($\zeta = 3$) obtained with parsers based on *F-dist*. We measured accuracy obtained with rules at each training time. The widest line is AdaBoost.SDF (ν=1,ω=∞). The others are AdaBoost.SDF (ν=10 (○), ν=100(●), ν=1&ω={1,10,100}).

5.2 Effectiveness of Several Rule Learning

Table 2 shows average accuracy and training time. We used *F-dist* as the distribution method. These average accuracy obtained with rules learned by AdaBoost.SDF (ν=10) on both tasks are competitive with the average accuracy obtained with rules learned by AdaBoost.SDF (ν=1). These results have shown that learning several rules at each iteration contributes significant improvement of training time. These results have also shown that the learning several rule at each iteration methods are more efficient than training by just using the frequency constraint ξ.

Figure 7 shows a snapshot for accuracy obtained with chunkers using different number of rules. This graph shows that chunkers based on AdaBoost.SDF (ν=10,100) and AdaBoost.SDF (ν=1,ω={1,10,100}) have shown better accuracy than chunkers based on AdaBoost.SDF (ν=1,ω=∞) at each training time. These result have shown that learning several rules at each iteration and learning combination of features as rules with a technique similar to beam search are effective in improving training time while giving a better convergence.

Figure 7 also implies that taggers and chunkers based on AdaBoost.SDF (ν=100) will show better or competitive accuracy than accuracy of the others by increasing numbers of rules to be learned while maintaining faster convergence speed.

Table 3: Experimental results on POS tagging and Text Chunking. *Accuracies* ($F_{\beta=1}$) on test data and training time (hour) of AdaBoost.SDF are averages of ω={1,10,100,∞} for each ζ with *F-dist* and $\xi = 1$. $F_{\beta=1}$ and time (hour) of SVMs are averages of C={0.1,1,10} for each kernel parameter d.

POS tagging without candidate features

Alg. /	1		2		3	
ζ (d)	$F_{\beta=1}$	time	$F_{\beta=1}$	time	$F_{\beta=1}$	time
ν=1	96.96	5.09	97.10	27.90	97.10	30.92
ν=10	96.89	0.79	97.12	4.56	97.07	4.74
ν=100	96.57	0.10	96.82	0.81	96.73	0.81
SVMs	96.60	101.63	97.15	166.76	96.93	625.32

POS tagging with candidate features

Alg. /	1		2		3	
ζ (d)	$F_{\beta=1}$	time	$F_{\beta=1}$	time	$F_{\beta=1}$	time
ν=1	97.06	6.65	97.30	109.20	97.29	330.82
ν=10	96.98	1.27	97.29	13.26	97.23	38.27
ν=100	96.61	0.14	96.93	1.64	96.76	5.05
SVMs	96.76	170.24	97.31	206.39	97.23	1346.04

Text Chunking without candidate features

Alg. /	1		2		3	
ζ (d)	$F_{\beta=1}$	time	$F_{\beta=1}$	time	$F_{\beta=1}$	time
ν=1	92.50	0.12	93.60	0.26	93.47	0.41
ν=10	92.34	0.02	93.50	0.05	93.39	0.07
ν=100	89.70	0.008	92.31	0.02	92.03	0.02
SVMs	92.14	8.55	93.91	7.38	93.49	9.82

Text Chunking with candidate features

Alg. /	1		2		3	
ζ (d)	$F_{\beta=1}$	time	$F_{\beta=1}$	time	$F_{\beta=1}$	time
ν=1	92.89	0.25	94.19	26.10	94.04	300.77
ν=10	92.85	0.04	94.11	2.97	94.08	3.06
ν=100	91.99	0.01	93.37	0.32	93.24	0.34
SVMs	92.77	12.74	94.31	9.63	94.20	49.27

5.3 Comparison with SVM

Table 3 lists average accuracy and training time on POS tagging and text chunking with respect to each (ν, ζ) for AdaBoost.SDF and d for SVM. AdaBoost.SDF with $_{\nu=10}$ and $_{\nu=100}$ have shown much faster training speeds than SVM and AdaBoost.SDF ($_{\nu=1,\omega=\infty}$) that is equivalent to the AdaBoost.DF (Iwakura and Okamoto, 2007).

Furthermore, the accuracy of taggers and chunkers based on AdaBoost.SDF ($_{\nu=10}$) have shown competitive accuracy with those of SVM-based and AdaBoost.DF-based taggers and chunkers. AdaBoost.SDF ($_{\nu=10}$) showed about 6 and 54 times faster training speeds than those of AdaBoost.DF on the average in POS tagging and text chunking. AdaBoost.SDF ($_{\nu=10}$) showed about 147 and 9 times faster training speeds than the training speeds of SVM on the average of POS tagging and text chunking. On the average of the both tasks, AdaBoost.SDF ($_{\nu=10}$) showed about 25 and 50 times faster training speed than AdaBoost.DF and SVM. These results have shown that AdaBoost.SDF with a moderate parameter ν can improve training time drastically while maintaining accuracy.

These results in Table 3 have also shown that rules represented by combination of features and the candidate features collected from automatically tagged data contribute to improved accuracy.

5.4 Effectiveness of Redistribution

We compared $F_{\beta=1}$ and *best training time* of *F-dist* and *W-dist*. We used $\zeta = 2$ that has shown better average accuracy than $\zeta = \{1, 3\}$ in both tasks. Table 4 lists comparison of *F-dist* and *W-dist* on POS tagging and text chunking. Most of accuracy obtained with *W-dist*-based taggers and parsers better than accuracy obtained with *F-dist*-based taggers and parsers. These results have shown that *W-dist* improves accuracy without drastically increasing training time. The text chunker and the tagger trained with AdaBoost.SDF ($\nu = 10$, $\omega = 10$ and *W-dist*) has shown competitive accuracy with that of the chunker trained with AdaBoost.SDF ($\nu = 1, \omega = \infty$ and *F-dist*) while maintaining about 7.5 times faster training speed.

Table 4: Results obtained with taggers and chunkers based on *F-dist* and *W-dist*. These results obtained with taggers and chunkers trained with $\omega = \{1, 10, 100, \infty\}$ and $\zeta = 2$. F and time indicate average $F_{\beta=1}$ on test data and average *best training time*.

POS tagging with *F-dist*

ν	ω=1		ω=10		ω=100		ω=∞	
	F	time	F	time	F	time	F	time
1	97.31	30.03	97.31	64.25	97.32	142.9	97.26	89.59
10	97.26	3.21	97.32	9.57	97.30	15.54	97.30	19.64
100	96.86	0.62	96.95	1.32	96.95	2.13	96.96	2.43

POS tagging with *W-dist*

ν	ω=1		ω=10		ω=100		ω=∞	
	F	time	F	time	F	time	F	time
1	97.32	29.96	97.31	57.05	97.31	163.2	97.32	98.71
10	97.24	2.66	97.30	25.70	97.28	16.20	97.29	20.49
100	97.00	0.54	97.02	1.31	97.07	2.22	97.08	2.58

Text Chunking with *F-dist*

ν	ω=1		ω=10		ω=100		ω=∞	
	F	time	F	time	F	time	F	time
1	93.95	7.42	94.30	23.30	94.22	34.74	94.31	21.26
10	93.99	0.98	94.08	2.44	94.19	3.11	94.18	3.18
100	93.32	0.16	93.33	0.32	93.42	0.40	93.42	0.40

Text Chunking with *W-dist*

ν	ω=1		ω=10		ω=100		ω=∞	
	F	time	F	time	F	time	F	time
1	93.99	2.93	94.24	24.77	94.32	35.72	94.32	35.61
10	93.98	0.71	94.30	2.82	94.29	3.60	94.30	4.05
100	93.66	0.17	93.65	0.36	93.50	0.42	93.50	0.42

5.5 Tagging and Chunking Speeds

We measured testing speeds of taggers and chunkers based on rules or models listed in Table 5. [10]

We examined two types of fast classification algorithms for polynomial kernel: Polynomial Kernel Inverted (PKI) and Polynomial Kernel Expanded (PKE). The PKI leads to about 2 to 12 times improvements, and the PKE leads to 30 to 300 compared with normal classification approach of SVM (Kudo and Matsumoto, 2003). [11]

The POS-taggers based on AdaBoost.SDF, SVM with PKI, and SVM with PKE processed 4,052 words, 159 words, and 1,676 words per second, respectively. The chunkers based on these three methods processed 2,732 words, 113 words, and 1,718 words per second, respectively.

[10] We list average speeds of three times tests measured with a machine with Xeon 3.8 GHz CPU and 4 GB of memory.

[11] We use a chunker YamCha for evaluating classification speeds based on PKI or PKE (http://www.chasen.org/~taku/software/yamcha/). We list the average speeds of SVM-based tagger and chunker with PKE of a threshold parameter $\sigma = 0.0005$ for rule selection in both task. The accuracy obtained with models converted by PKE are slightly lower than the accuracy obtained with their original models in our experiments.

Table 5: Comparison with previous best results: (Top : POS tagging, Bottom: Text Chunking)

POS tagging	$F_{\beta=1}$
Perceptron (Collins, 2002)	97.11
Dep. Networks (Toutanova et al., 2003)	97.24
SVM (Giménez and Màrquez, 2003)	97.05
ME based a bidirectional inference (Tsuruoka and Tsujii, 2005)	97.15
Guided learning for bidirectional sequence classification (Shen et al., 2007)	97.33
AdaBoost.SDF with candidate features (ζ=2,ν=1,ω=100, *W-dist*)	97.32
AdaBoost.SDF with candidate features (ζ=2,ν=10,ω=10, *F-dist*)	97.32
SVM with candidate features (C=0.1, d=2)	97.32

Text Chunking	$F_{\beta=1}$
Regularized Winnow + full parser output (Zhang et al., 2001)	94.17
SVM-voting (Kudo and Matsumoto, 2001)	93.91
ASO + unlabeled data (Ando and Zhang, 2005)	94.39
CRF+Reranking(Kudo et al., 2005)	94.12
ME based a bidirectional inference (Tsuruoka and Tsujii, 2005)	93.70
LaSo (Approximate Large Margin Update) (Daumé III and Marcu, 2005)	94.4
HySOL (Suzuki et al., 2007)	94.36
AdaBoost.SDF with candidate featuers (ζ=2,ν=1,ω=∞, *W-dist*)	94.32
AdaBoost.SDF with candidate featuers (ζ=2,ν=10,ω=10,*W-dist*)	94.30
SVM with candidate features (C=1, d=2)	94.31

One of the reasons that boosting-based classifiers realize faster classification speed is sparseness of rules. SVM learns a final hypothesis as a linear combination of the training examples using some coefficients. In contrast, this boosting-based rule learner learns a final hypothesis that is a subset of candidate rules (Kudo and Matsumoto, 2004).

6 Related Works

6.1 Comparison with Previous Best Results

We list previous best results on English POS tagging and Text chunking in Table 5. These results obtained with the taggers and chunkers based on AdaBoost.SDF and SVM showed competitive F-measure with previous best results. These show that candidate features contribute to create state-of-the-art taggers and chunkers.

These results have also shown that AdaBoost.SDF-based taggers and chunkers show competitive accuracy by learning combination of features automatically. Most of these previous works manually selected combination of features except for SVM with polynomial kernel and (Kudo and Matsumoto, 2001) a boosting-based re-ranking (Kudo et al., 2005).

6.2 Comparison with Boosting-based Learners

LazyBoosting randomly selects a small proportion of features and selects a rule represented by a feature from the selected features at each iteration (Escudero et al., 2000).

Collins and Koo proposed a method only updates values of features co-occurring with a rule feature on examples at each iteration (Collins and Koo, 2005).

Kudo et al. proposed to perform several pseudo iterations for converging fast (Kudo et al., 2005) with features in the cache that maintains the features explored in the previous iterations.

AdaBoost.MH^{KR} learns a weak-hypothesis represented by a set of rules at each boosting iteration (Sebastiani et al., 2000).

AdaBoost.SDF differs from previous works in the followings. AdaBoost.SDF learns several rules at each boosting iteration like AdaBoost.MH^{KR}. However, the confidence value of each hypothesis in AdaBoost.MH^{KR} does not always minimize the upper bound of training error for AdaBoost because the value of each hypothesis consists of the sum of the confidence value of each rule. Compared with AdaBoost.MH^{KR}, AdaBoost.SDF computes the confidence value of each rule to minimize the upper bound of training error on given weights of samples at each update.

Furthermore, AdaBoost.SDF learns several rules represented by combination of features from limited search spaces at each boosting iteration. The creation of subsets of features in AdaBoost.SDF enables us to recreate the same classifier with same parameters and training data. Recreation is not ensured in the random selection of subsets in LazyBoosting.

7 Conclusion

We have proposed a fast boosting-based learner, which we call AdaBoost.SDF. AdaBoost.SDF repeats to learn several rules represented by combination of features from a small proportion of candidate rules. We have also proposed methods to use candidate POS tags and chunk tags of each word obtained from automatically tagged data as features in POS tagging and text chunking.

The experimental results have shown drastically improvement of training speed while maintaining competitive accuracy compared with previous best results.

Future work should examine our approach on several tasks. Future work should also compare our algorithm with other learning algorithms.

Appendix A: Convergence

The upper bound of the training error for AdaBoost of (Freund and Mason, 1999), which is used in AdaBoost.SDF, is induced by adopting THEOREM 1 presented in (Schapire and Singer, 1999). Let Z_R be $\sum_{i=1}^{m} w_{R+1,i}$ that is a sum of weights updated with R rules. The bound holds on the training error after selecting R rules,

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

is induced as follows.

By unraveling the Eq. (1), we obtain $w_{R+1,i} = exp(-y_i \sum_{r=1}^{R} h_{\langle \mathbf{f}_r, c_r \rangle}(\mathbf{x}_i))$. Thus, we obtain $[[F(\mathbf{x}_i) \neq y_i]] \leq exp(-y_i \sum_{t=1}^{R} h_{\langle \mathbf{f}_r, c_r \rangle}(\mathbf{x}_i))$, since if $F(\mathbf{x}_i) \neq y_i$, then $exp(-y_i \sum_{r=1}^{R} h_{\langle \mathbf{f}_r \rangle}(\mathbf{x}_i)) \geq 1$. Combining these equations gives the stated bound on training error

$$\sum_{i=1}^{m}[[F(\mathbf{x}_i) \neq y_i]] \leq \sum_{i=1}^{m} exp(-y_i \sum_{t=1}^{R} h_{\langle \mathbf{f}_r, c_r \rangle}(\mathbf{x}_i)) = \sum_{i=1}^{m} w_{R+1,i} = Z_R. \quad (2)$$

Then we show that the upper bound of training error Z_R for R rules shown in Eq. (2) is less than or equal to the upper bound of the training error Z_{R-1} for R-1 rules. By unraveling the (2) and plugging the confidence values $c_R = \{\frac{1}{2}\log(\frac{W_{r,+1}(\mathbf{f}_R)}{W_{r,-1}(\mathbf{f}_R)}), 0\}$ given by the weak hypothesis into the unraveled equation, we obtain $Z_R \leq Z_{R-1}$, since

$$\begin{aligned} Z_R &= \sum_{i=1}^{m} w_{R+1,i} = \sum_{i=1}^{m} w_{R,i} exp(-y_i h_{\langle \mathbf{f}_R, c_R \rangle}) \\ &= \sum_{i=1}^{m} w_{R,i} - W_{r,+1}(\mathbf{f}_R) - W_{r,+1}(\mathbf{f}_R) + \\ &\quad W_{r,+1}(\mathbf{f}_R) exp(-c_R) + W_{r,-1}(\mathbf{f}_R) exp(c_R) \\ &= Z_{R-1} - (\sqrt{W_{R,+1}(\mathbf{f}_R)} - \sqrt{W_{R,-1}(\mathbf{f}_R)})^2 \end{aligned}$$

References

Ando, Rie and Tong Zhang. 2005. A high-performance semi-supervised learning method for text chunking. In *Proc. of 43rd Meeting of Association for Computational Linguistics*, pages 1–9.

Collins, Michael and Terry Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70.

Collins, Michael. 2002. Discriminative training methods for Hidden Markov Models: theory and experiments with perceptron algorithms. In *Proc. of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8.

Daumé III, Hal and Daniel Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proc. of 22th International Conference on Machine Learning*, pages 169–176.

Escudero, Gerard, Lluís Màrquez, and German Rigau. 2000. Boosting applied to word sense disambiguation. In *Proc. of 11th European Conference on Machine Learning*, pages 129–141.

Freund, Yoav and Llew Mason. 1999. The alternating decision tree learning algorithm,. In *Proc. of 16th International Conference on Machine Learning*, pages 124–133.

Giménez, Jesús and Lluís Màrquez. 2003. Fast and accurate part-of-speech tagging: The SVM approach revisited. In *Proc. of International Conference Recent Advances in Natural Language Processing 2003*, pages 153–163.

Iwakura, Tomoya and Seishi Okamoto. 2007. Fast training methods of boosting algorithms for text analysis. In *Proc. of International Conference Recent Advances in Natural Language Processing 2007*, pages 274–279.

Kazama, Jun'ichi and Kentaro Torisawa. 2007. A new perceptron algorithm for sequence labeling with non-local features. In *Proc. of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 315–324.

Kudo, Taku and Yuji Matsumoto. 2001. Chunking with Support Vector Machines. In *Proc. of The Conference of the North American Chapter of the Association for Computational Linguistics*, pages 192–199.

Kudo, Taku and Yuji Matsumoto. 2003. Fast methods for kernel-based text analysis. In *Proc. of 41st Meeting of Association for Computational Linguistics*, pages 24–31.

Kudo, Taku and Yuji Matsumoto. 2004. A boosting algorithm for classification of semi-structured text. In *Proc. of the 2004 Conference on Empirical Methods in Natural Language Processing 2004*, pages 301–308, July.

Kudo, Taku, Jun Suzuki, and Hideki Isozaki. 2005. Boosting-based parse reranking with subtree features. In *Proc. of 43rd Meeting of Association for Computational Linguistics*, pages 189–196.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of english: The Penn Treebank. pages 313–330.

Morishita, Shinichi. 2002. Computing optimal hypotheses efficiently for boosting. *Proc. of 5th International Conference Discovery Science*, pages 471–481.

Schapire, Robert E. and Yoram Singer. 1999. Improved boosting using confidence-rated predictions. *Machine Learning*, 37(3):297–336.

Schapire, Robert E. and Yoram Singer. 2000. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168.

Sebastiani, Fabrizio, Alessandro Sperduti, and Nicola Valdambrini. 2000. An improved boosting algorithm and its application to text categorization. In *Proc. of International Conference on Information and Knowledge Management*, pages 78–85.

Shen, Libin, Giorgio Satta, and Aravind Joshi. 2007. Guided learning for bidirectional sequence classification. In *Proc. of 45th Meeting of Association for Computational Linguistics*, pages 760–767.

Suzuki, Jun, Akinori Fujino, and Hideki Isozaki. 2007. Semi-supervised structured output learning based on a hybrid generative and discriminative approach. In *Proc. of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 791–800.

Toutanova, Kristina, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proc. of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 173–180.

Tsuruoka, Yoshimasa and Junichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proc. of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 467–474.

Zhang, Tong, Fred Damerau, and David Johnson. 2001. Text chunking using regularized winnow. In *Proc. of 39th Meeting of Association for Computational Linguistics*, pages 539–546.