Fast Boosting-based Part-of-Speech Tagging and Text Chunking with Efficient Rule Representation for Sequential Labeling

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

Abstract

This paper proposes two techniques for fast sequential labeling such as part-of-speech (POS) tagging and text chunking. The first technique is a boosting-based algorithm that learns rules represented by combination of features. To avoid time-consuming evaluation of combination, we divide features into not used ones and used ones for learning combination. The other is a rule representation. Usual POS taggers and text chunkers decide the tag of each word by using the features generated from the word and its surrounding words. Thus similar rules, for example, that consist of the same set of words but only differ in locations from current words, are generated. We use a rule representation that enables us to merge such rules. We evaluate our methods with POS tagging and text chunking. The experimental results show that our methods show faster processing speed than taggers and chunkers without our methods while maintaining accuracy.

1 Introduction

Several machine learning algorithms such as Support Vector Machines (SVMs) and boosting-based learning algorithms have been applied to Natural Language Processing (NLP) problems successfully. The cases of boosting include text categorization [11], POS tagging [5] and text chunking [7, 5], and so on. Furthermore, parsers based on boosting-based learners have shown fast processing speed [7, 5]. However, to process large data such as WEB data and e-mails, processing speed of base technologies such as POS tagging and text chunking will be important.

This paper proposes two techniques for improving processing speed of POS tagging and text chunking. The first technique is a boosting-based algorithm that learns rules. Instead of specifying combination of features manually, we specify features that are not used for the combination of features as atomic. Our boosting algorithm learns rules that consist of features or a feature from non-atomic features, and rules consisting of a feature from atomic features.

The other is a rule representation for sequential labeling such as POS tagging and text chunking. Usual POS taggers and text chunkers decide the tag of each word by using features generated from the current word and its surrounding words. Thus each word and its attributes, such as character-types, are evaluated several times in different relative locations from current word. We propose a representation that enables us to merge similar rules that consist of the same set of words and attributes that only differ in positions from current word.

The experimental results with English POS tagging and text chunking show the taggers and chunkers based on our methods show faster processing speed than without our methods while maintaining competitive accuracy.

2 Boosting-based Learner

2.1 Preliminaries

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 samples $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)\}$, 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 following mapping from S:

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

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 $\mathbf{x}_i$. We call a feature-set of size k as a k-feature-set. We call $\mathbf{x}_i$ is a subset of $\mathbf{x}_j$, if a feature-set $\mathbf{x}_j$ contains all the features in a feature-set $\mathbf{x}_i$. We denote subsets of feature-sets as

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

Then we define weak hypothesis based on the idea of the real-valued predictions and abstaining [11]. Let $\mathbf{f}$ be a feature-set, called a rule, c be a real number, called a confidence value, and $\mathbf{x}$ be an input feature-set, then a weak-hypothesis for feature-sets is defined as

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

2.2 Boosting-based Rule Learning

We use a boosting-based algorithm that has shown fast training speed by treating a weak learner that learns several rules at each iteration [5]. The learner learns a final hypothesis F consisting of R types of rules 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 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 with $gain$:

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

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

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

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

The weak learner selects a feature-set having the highest $gain$ as the r-th rule, and the weak learner selects ν types of feature-sets having $gain$ in top ν as $\{\mathbf{f}_r, ..., \mathbf{f}_{r+\nu-1}\}$ at each iteration.

Then the boosting-based learner calculates the confidence value of each rule in the selected ν rules and updates the weight of each sample. The confidence value c_r for the first rule $\mathbf{f}_r$ in the selected ν rules is defined as

```
## $F_k$ : A set of k-feature-sets
## $\mathcal{R}_o$ : $\nu$ optimal rules (feature-sets)
## $R_{k,\omega}$ : $\omega$ k-feature-sets for generating candidates
## selectNBest($\mathcal{R}$, $n$, $S$, $W_r$): Select $n$ best rules in $\mathcal{R}$
##   with $gain$ on $\{w_{i,r}\}_{i=1}^{m}$ and training samples $S$
## $\mathcal{FN}$, $\mathcal{FA}$ : non-atomic, atomic features
procedure weak-learner($F_k$,$S$, $W_r$)
 ## $\nu$ best feature-sets as rules
 $\mathcal{R}_o$ = selectNBest( $\mathcal{R}_o \cup F_k$, $\nu$, $S$, $W_r$);
 if ($\zeta \leq k$) return $\mathcal{R}_o$;  ## Size constraint
 ## $\omega$ best feature-sets in $F_k$ for generating candidates
 $R_{k,\omega}$ = selectNBest($F_k$, $\omega$, $S$, $W_r$);
 $\tau = \min_{\mathbf{f}\in\mathcal{R}_o} gain(\mathbf{f})$;  ## The $gain$ of $\nu$-th optimal rule
 Foreach ( $\mathbf{f_k} \in R_{k,\omega}$)
  ## Pruning candidates with upper bound of gain
   if ( $u(\mathbf{f_k}) < \tau$) continue;
   Foreach ($f \in \mathcal{FN}$)  ## Generate candidates
    $F_{k+1} = (F_{k+1} \cup gen(\mathbf{f_k}, f))$;
   end Foreach
 end Foreach
 return weak-learner($F_{k+1}, S, W_r$);
```

Fig. 1: *Find rules with given weights.*

$$c_r = \frac{1}{2}\log(\frac{W_{r,+1}(\mathbf{f}_r)+\varepsilon}{W_{r,-1}(\mathbf{f}_r)+\varepsilon}),$$

where ε is a value to avoid to happen that $W_{r,+1}(\mathbf{f})$ or $W_{r,-1}(\mathbf{f})$ is very small or even zero [10]. We set ε to 1. After the calculation of c_r for $\mathbf{f}_r$, the learner updates the weight of each sample with

$$w_{r+1,i} \quad = \quad w_{r,i} exp(-y_i h_{\langle \mathbf{f}_r, c_r \rangle}(\mathbf{x}_i)). \qquad (1)$$

Then the learner adds ($\mathbf{f_r}$, c_r) to F as the r-th rule and its confidence value. When we calculate the confidence value c_{r+1} for $\mathbf{f}_{r+1}$, we use $\{w_{r+1,1}, ..., w_{r+1,m}\}$ as the weights of samples. After processing all the selected rules, the learner starts the next iteration. The learner continues training until obtaining R rules.

2.3 Learning Rules

We extend a weak learner that learns several rules from a small portion of candidate rules called a bucket used in [5]. Figure 1 describes an overview of the weak learner.

At each iteration, one of the $|B|$ types of buckets is given as an initial 1-feature-sets F_1 to the weak learner. We use *W-dist* that is a method to distributes features to $|B|$-buckets. To distribute features to buckets, *W-dist* calculates the weight of each feature that is defined as $W_r(f) = \sum_{i=1}^{m} w_{r,i}[[\{f\} \subseteq \mathbf{x}_i]]$ ($f \in \mathcal{F}$). Then *W-dist* sorts features based on the weight of each feature, and insert each feature to one of the buckets.

The weak learner finds ν best feature-sets as rules from feature-sets that include one of the features in F_1. The weak learner generates candidate k-feature-sets ($1 < k$) from ω best (k-1)-feature-sets in F_{k-1} with *gain*.

We define two types of features, $\mathcal{FA}$ and $\mathcal{FN}$ (i.e $\mathcal{F} = \mathcal{FA} \cup \mathcal{FN}$). $\mathcal{FA}$ and $\mathcal{FN}$ are a set of atomic features and a set of non-atomic features. When we generate candidate rules that consist of more than a feature, we only use non-atomic features in $\mathcal{FN}$.

For example, if we use features $\mathcal{FA} = \{A, B, C\}$ and $\mathcal{FN} = \{a, b, c\}$, we examine followings as candidates; $\{A\}$,$\{B\}$,$\{C\}$,$\{a\}$,$\{b\}$, $\{c\}$, $\{a, b\}$, $\{b, c\}$ and $\{a, b, c\}$.

The gen is a function to generate combination of features. We denote $\mathbf{f}' = \mathbf{f} + f$ as the generation of $k+1$-feature-set $\mathbf{f}'$ that consists 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 the gen is defined as follows.

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

```
## $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ : $\mathbf{x}_i \subseteq \mathcal{X}$, $y_i \in \{\underline{+}1\}$
## $W_r = \{w_{r,i}\}_{i=1}^{m}$: Weights of samples after learning
## r types of rules.
## $|B|$ : The size of bucket $B = \{B[0], ..., B[|B|-1]\}$
## $b$, $r$ : The current bucket and rule number
## distFT: distribute features to buckets
procedure AdaBoost.SDFAN()
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 $\nu$ 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  $w_{r+1,i} = w_{r,i}\ exp(-y_i h_{\langle \mathbf{f}, c \rangle}(\mathbf{x}_i))$;
   $\mathbf{f}_r = \mathbf{f}$; $c_r = c$; $r$++;
 end Foreach
  if ($b$ == $|B|$) ## Redistribution of features
   $B$ = distFT($S$, $|B|$); $b$=0;
  end if
end While
return $F(\mathbf{x}) = sign(c_0 + \sum_{r=1}^{R} h_{\langle \mathbf{f}_r, c_r \rangle}(\mathbf{x}))$
```

Fig. 2: *An overview of AdaBoost.SDFAN.*

The gen excludes the generation of candidates that include an atomic feature. 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 as in [4].

We also use the following pruning techniques.

• **Size constraint** (ζ): We examine candidates whose size is no greater than a threshold ζ.

• **Upper bound of gain**: The upper bound is defined as

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

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 $u(\mathbf{f})$ is less than τ, the $gain$ of the current optimal rule, candidates that contain $\mathbf{f}$ are safely pruned.

Figure 2 describes an overview of our algorithm, which we call AdaBoost for a weak learner learning *S*everal rules from *D*istributed *F*eatures consist of Atomic and Non-atomic (*AdaBoost.SDFAN*, for short). [1]

3 Efficient Rule Representation

3.1 A Problem of Conventional Methods

When identifying the POS tags of words and chunks of words in usual parsers, we firstly generate features from current word and its surrounding words.

Let "I am happy ." be a sequence of words. If we identify a tag of "am" with 3-word window, we use "I", "am" and "happy" as features. To distinguish words that appear different locations, we usually express words with relative locations from current word like "I:-1", "am:0" and "happy:1", where the -1, 0 and 1 after ":" are location-markers for relative locations. When "happy" is a current word, we have to express "am" as "am:-1". Thus similar rules that differ in relative locations are generated.

3.2 Efficient Rule Representation

We propose a rule representation, called *Compressed Sequential Labeling Rule Representation* (*CSLR-rep*, for

[1] To reflect imbalance class distribution, we use the default rule defined as $\frac{1}{2}\log(\frac{W_{+1}}{W_{-1}})$, where $W_y = \sum_{i=1}^{m}[[y_i = y]]$ for $y \in \{\pm 1\}$.

```
## $\mathbf{f}$: a rule generated by AdaBoost.SDFAN
## $sc$ : the score of $\mathbf{f}$
## $cl$ : the class of $\mathbf{f}$
## $s(f)$: the feature-stem of a feature $f$
## $p(f)$: the location-marker of a feature $f$
## $\mathbf{fn}$: the conversion result of $\mathbf{f}$
## $\mathcal{RC}[\mathbf{fn}]$: scores for $\mathbf{fn}$
procedure ruleConv( $\mathbf{f}$, $sc$, $cl$)
  $bp = \min_{f \in \mathbf{f}} p(f)$ ## select the base position
  Foreach $f \in \mathbf{f}$ ## generate new rule
    $lm = p(f) - bp$ ## new location-marker of $f$
    ## append new representation of $f$
    $\mathbf{fn} = \mathbf{fn}$ + "$s(f)$:$lm$"
  endForeach
  $\mathcal{RC}[\mathbf{fn}] = \mathcal{RC}[\mathbf{fn}] \cup (-bp, cl, sc)$
```

Fig. 3: *Generating CSLR-rep based rules.*

short), to merge similar rules. To use *CSLR-rep*, we convert weak-hypotheses (*WH*s, for short) generated by AdaBoost.SDFAN to *CSLR-rep*. A *CSLR-rep*-based *WH* is represented as

$$\langle \mathbf{rule}, \{(p_1, cl_1, c_1), ..., (p_q, cl_q, c_q)\}\rangle.$$

The $\mathbf{rule}$ is a rule generated by merging rules learned by AdaBoost.SDFAN. p_p, called scoring-position, denotes the position of a word to assign a score c_p of a class cl_p ($1 \leq p \leq q$) from current word.

We describe an example. Let $\langle\{I{:}{-2}, am{:}{-1}\}, JJ, c_0\rangle$, $\langle\{I{:}-1, am{:}0\}, VBP, c_1\rangle$ and $\langle\{I{:}0, am{:}1\}, PRP, c_2\rangle$ be *WH*s generated by AdaBoost.SDFAN, and JJ, VBP and PRP be class tags. These *WH*s are converted to the following *CSLR-rep*-based rule; $\langle\{I{:}0, am{:}1\}, \{(2, JJ, c_0), (1, VBP, c_1), (0, PRP, c_2)\}\rangle$,

When the converted *WH* in the example is applied to a word sequence "I am happy .", we can assign scores to all the three words by just checking $\{I{:}0, am{:}1\}$. The scores for "JJ", "VBP" and "PRP" are assigned to "happy", "am" and "I", respectively.

When we use the three original *WH*s in the example, we have to check three rules to assign scores to the words.

Figure 3 shows an overview for the rule conversion. We assume each feature is divided into a location-marker and a feature-stem. A location-marker is the relative location from a current word. A feature-stem is a word or one of its attributes such as character-types without a location-marker.

We use the relative location of a feature appeared in left-most word in each rule as base-position (*bp*, for short). Then we convert each feature to a new feature that consists of its feature-stem and new location-marker. The new location-marker means a relative location from the *bp*. We add the value of (*bp* × -1) as the scoring-position of the current score.

3.3 Rule Application

We describe an overview of the application of rules represented by *CSLR-rep*. We consider two types of features, **static-features** and **dynamic-features**, in this application. Static-features are generated from input word sequences. Dynamic-features are dynamically generated from the tag of each word assigned with the highest score. We define W as a word window size that means using a current word and its surrounding words appearing $\frac{W-1}{2}$ left and $\frac{W-1}{2}$ right of the current word.

Figure 4 shows an overview of the application. Let $\{\mathbf{wd}_1, .., \mathbf{wd}_N\}$ be an input that consists of N ($1 \leq N$) words. Each word $\mathbf{wd}_i$ ($1 \leq i \leq N$) has $|\mathbf{wd}_i|$ types of attributes. We denote j-th attribute of $\mathbf{wd}_i$ as $wd_{i,j}$. $\mathcal{RC}$ is a set of rules represented by *CSLR-rep* and $\mathcal{RC}[\mathbf{rc}]$ is the set of $\langle$ scoring-position, class, score $\rangle$ of $\mathbf{rc}$.

The application has two stages for static-features and dynamic-features. Our algorithm firstly assigns scores with rules consisting of only Static-features to each word in the direction of beginning of sentence (BOS) to end of sentence (EOS) direction. $\mathbf{Rs}[i]$ keeps the status of rule applications for i-th word. If the algorithm finds a subset of rules while applying rules from i-th word, the algorithm adds the subset of rules to $\mathbf{Rs}[i]$. [2] We define subsets of rules as follows:

Definition 1 *Subsets of rules*
If there exists $\mathbf{rule}$ *in* $\langle \mathbf{rule}, scores\rangle \in \mathcal{RC}$ *that satisfies* $\mathbf{rc} \subseteq \mathbf{rule} \wedge \mathbf{rc} \neq \mathbf{rule}$, *we call* $\mathbf{rc}$ *is a subset of rules of* $\mathcal{RC}$ *and denote it as*

$$\mathbf{rc} \subset \mathcal{RC}$$

Then we apply rules that include dynamic-features. All the subsets of rules are kept in $\mathbf{Rs}$ after examining all the Static-features, we can assign scores to words by just checking dynamic-feature of each word with $\mathbf{Rs}$. When checking rules that include the dynamic-feature of i-th word we check subsets of rules of words in ($i - \frac{W-1}{2} - \Delta$) to (i + $max(\frac{W-1}{2}, \Delta)$ - 1). We use the tags of words with in Δ in the direction of EOS.

We describe an example. Let $\mathcal{RC}$ ={ {I:0, am:1}, {I:0, VBP:1}, {I:0, VBP:1, JJ:2} } be a set of rules. When applying the rules to "I am happy ." with $(W, \Delta) = (3, 2)$, we check "I:0" first. "I:0" is inserted to $\mathbf{Rs}[1]$ because of {I:0} $\subset \mathcal{RC}$. Then we check "am:1" with "{I:0}" in $\mathbf{Rs}[1]$, and {I:0, am:1} is found. Finally we check "happy:2" with $\mathbf{Rs}[1]$. We check the other words like this. After checking all the words from BOS to EOS direction, we start to check rules that include dynamic-features from EOS to BOS direction. If the dynamic-features of "am" and "happy" are VBP and JJ, we check VBP and JJ with $\mathbf{Rs}$. For example, VBP is treated as "VBP:1" from the position of "I" and "VBP:0" from the position of "am". When we check "VBP:1" with "{I:0}" in $\mathbf{Rs}[1]$, {I:0, VBP:1} is found and inserted to $\mathbf{Rs}[1]$. Then we check "JJ:2" with "I:0" and {I:0, VBP:1} in $\mathbf{Rs}[1]$. Then we check these dynamic-features with $\mathbf{Rs}[2]$.

Unfortunately, the *CSLR-rep* has some drawbacks. One of the drawbacks is the increase of dynamic-features. When we convert rules that consist of more than a feature to *CSLR-rep*, the number of types of dynamic-features increases. Since original rule representation only handles dynamic-features within Δ, the total number of types of dynamic-features is up to "Δ × CL", where CL is the number of classes in each task. However, the total number of dynamic-features in *CSLR-rep* is up to " ($\frac{W-1}{2}$ + Δ + $max(\frac{W-1}{2}, \Delta)$ -1) × CL " because we express each feature with the relative location from the base-position of each rule.

4 POS tagging and Text Chunking

4.1 English POS Tagging

We used the Penn Wall Street Journal treebank [8]. We split the treebank into training (sections 0-18), development (sections 19-21) and test (sections 22-24) as in [5]. We used the following features:

[2] We use a TRIE structure called double array for representing rules [1]. To keep the statuses of rule applications, we store the last position in a TRIE where each subset of rules reached.

```
## $\mathcal{RC}[\mathbf{rc}]$: pairs of score-positions and scores of $\mathbf{rc}$
## $\mathbf{Rs}[i]$: subset of rules of $i$-th word
##         Initial value for each word is 0-feature-set
procedure ruleApplication( $\{\mathbf{wd}_1, .., \mathbf{wd}_N\}$, $\mathcal{FN}$)
## For Static-feature
For $i' = 1; i' \leq N; i'$++ # beginning position
 For $i = i'; i < i + W; i$++ # combination position
  For $j = 1; j \leq |\mathbf{wd}_i|; j$++# attributes
   Foreach $\mathbf{rc} \in \mathbf{Rs}[i']$
    $lm = i - i'$ ## current location-marker
   $\mathbf{rc}' = \mathbf{rc}$ + "$wd_{i,j}$:$lm$"
    # If $\mathcal{RC}[\mathbf{rc}']$ is applied,
    # assign the scores with base position $i$'
    assignScores($\mathcal{RC}[\mathbf{rc}']$, $i'$)
    If $\mathbf{rc}' \subset \mathcal{RC}$  $\mathbf{Rs}[i'] = \mathbf{Rs}[i'] \cup \mathbf{rc}'$
  endForeach
 # If no subset of rules for $i'$, go to $i' + 1$-th word
 If $\mathbf{Rs}[i'] = \{\phi\}$ break
 endFor
endFor
## For Dynamic-feature : EOS to BOS direction
For $i' = N; 1 \leq i'; i'--$ # beginning position
 # Checking rules including Dynamic-feature
 $db = i' - \frac{W-1}{2} - \Delta; de = i' + max(\frac{W-1}{2}, \Delta);$
  For $i = db; i < de; i$++
  Foreach $\mathbf{rc} \in \mathbf{Rs}[i]$
    $lm = j - i'$ ## current location-marker
    $\mathbf{rc}' = \mathbf{rc}$ + "$dft_{i':lm}$" # $dft_j$ is the tag of $i$'-th word
    assignScores($\mathcal{RC}[\mathbf{rc}']$, $i$)
    If $\mathbf{rc}' \subset \mathcal{RC}$  $\mathbf{Rs}[i] = \mathbf{Rs}[i] \cup \mathbf{rc}'$
  endForeach
 endFor
endFor
```

Fig. 4: *Application of CSLR-rep based rules.*

· words, words that are turned into all capitalized, in a W-word window size, tags assigned to Δ words on the right.
· whether the current word has a hyphen, a number, a capital letter, the current word is all capital, all small
· prefixes and suffixes of current word (up to 4)
· candidate-tags of words in a W-word window
We collect candidate POS tags of each word, called candidate feature, from the automatically tagged corpus provided for the shared task of English Named Entity recognition in CoNLL 2003 as in [5]. [3] [4] We express these candidates with one of the following ranges decided by their frequency fq: $10 \leq fq < 100$, $100 \leq fq < 1000$ and $1000 \leq fq$.

If 'work' is annotated as NN 2000 times, we express it 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 [5]. [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.

Table 1: *Training data for experiments. POS and ETC indicate POS tagging and text chunking. ♯ of S, ♯ of cl and M indicate the number samples, the number of class in each data set and the distinct number of feature types for each pair of (W, Δ).*

			M (W, Δ)		
data	♯ of S	♯ of cl	(3, 1)	(5, 2)	(7, 3)
POS	912,344	45	283,979	440,725	593,065
ETC	211,727	22	56,917	93,333	128,651

Table 2: *Accuracy on Test Data.*

POS tagging

		-Atomic		+Atomic	
$(W,\Delta)/\ \zeta$	1	2	3	2	3
(3,1)	96.81	97.09	97.05	97.00	97.04
(5,2)	96.96	97.30	97.30	97.25	97.28
(7,3)	96.99	**97.36**	97.30	97.31	**97.34**

text chunking

		-Atomic		+Atomic	
$(W,\Delta)/\ \zeta$	1	2	3	2	3
(3,1)	92.40	93.87	93.69	93.91	93.82
(5,2)	92.87	94.31	94.14	**94.34**	94.31
(7,3)	93.09	**94.32**	94.11	94.12	94.11

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 the following features:
· words and POS tags in a W-word window.
· tags assigned to Δ words on the right.
· candidate-tags of words in a W-word window.
We collected the followings as candidate-tags for chunking from the same 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
If we collect "work" annotated as I-NP 2000 times and as E-VP 100 times, we generate the following candidate-tags for "work"; 1000≤I-NP, 100≤E-VP<1000, rank:I-NP=1 rank:E-NP=2, candidate=I-NP and candidate=E-VP. [7]

5 Experiments

We tested R=200,000, $|B|$=1,000, ν = 10, ω=10, ζ={1,2,3} and (W,Δ)={(3,1), (5,2), (7,3)}. Table 1 shows that the number of training samples, classes, features.

We examine two types of training, "-Atomic " and " +Atomic ", in this experiment. "-Atomic " indicates training with all the features as non-atomic. " +Atomic " indicates training by using atomic features. We specify prefixes, suffixes and candidate-tags as atomic for POS tagging, and candidate-tags as atomic for text chunking.

To extend AdaBoost.SDFAN to handle multi-class problems, we used the one-vs-the-rest method. To identify proper tag sequences, we use Viterbi search. [8]

5.1 Tagging and Chunking Accuracy

Table 2 shows accuracy obtained with each rules on POS tagging and text chunking. We calculate label accuracy for

[3] http://www.cnts.ua.ac.be/conll2003 /ner/
[4] We collected POS tags for each word that are annotated to the word more than 9 times in the corpus as candidates.
[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.

[7] We converted the chunk representation in the corpus to IOE2 and we collected chunk tags of each word appearing more than 9 times.
[8] We map the confidence value of each classifier into the range of 0 to 1 with sigmoid function defined as $s(X) = 1/(1+exp(-\beta X))$, where $X = F(\mathbf{x})$ is a output of a classifier. We used β=5 in this experiment. We select a tag sequence which maximizes the sum of those log values by Viterbi search.

Table 3: *Tagging and Chunking Speed. Each number is average processed words per second. We examine three times measurements for each tagger or chunker. Each time is obtained with all rules.*

POS tagging

	-Atomic			+ Atomic	
without *CSLR-rep*					
$(W, \Delta)/\ \zeta$	1	2	3	2	3
(3,1)	9477	4023	2505	5450	5096
(5,2)	8118	2564	1445	3915	3389
(7,3)	6615	1842	1007	3033	2464
with *CSLR-rep*					
$(W, \Delta)/\ \zeta$	1	2	3	2	3
(3,1)	19467	4258	2013	10969	9644
(5,2)	18261	2807	1102	8212	5934
(7,3)	15658	2195	754	7474	4939

text chunking

	-Atomic			+Atomic	
without *CSLR-rep*					
$(W, \Delta)/\ \zeta$	1	2	3	2	3
(3,1)	14510	3995	1036	13975	12221
(5,2)	11266	1681	401	9571	7018
(7,3)	9434	961	230	6849	4595
with *CSLR-rep*					
$(W, \Delta)/\ \zeta$	1	2	3	2	3
(3,1)	27705	4282	863	19496	16169
(5,2)	25471	2477	352	13692	8475
(7,3)	23338	1758	206	10058	5701

POS tagging as accuracy. As for text chunking, we calculate F-measure ($F_{\beta=1}$) given by $2rp/(r+p)$ as accuracy, where r and p are recall and precision. Each accuracy on a test data is calculated with the number of rules that show the best accuracy on development data.

We obtain almost the same accuracy even if we use part of features as atomic.

5.2 Tagging and Chunking Speed

Table 3 shows tagging and chunking speed. We measure the number of words processed by per second.[9] We obtain faster processing speed by using *CSLR-rep*-based rules traind with $\zeta = \{1, 2\}$ and - Atomic. These show that *CSLR-rep* contributes to improved processing time. When we use rules trained with $\zeta = 1$, we can get more improvement than using rules trained with $\zeta = 2$.

However, the performance obtained with *CSLR-rep*-based rules trained with ($\zeta = 3, -Atomic$) is slower than with the original rules. We guess this is caused due to the following two reasons. Our *CSLR-rep* reduces the number of times of rule evaluation up to $1/W$. Thus *CSLR-rep* reduces processing time linearly. However, the number of combination of features exponentially increases. The other reason is that the number of times to generate dynamic-features is increased as described in the end of section 3.3.

We obtain much improvement by using atomic features with *CSLR-rep*. For example, processing speed obtained with the text chunker using rules ($\zeta = 3, W = 7$, +Atomic) is about 28 times faster than the speed obtained with the chunker using rules ($\zeta = 3, W = 7$, -Atomic).

6 Related Work

We list previous best results on English POS tagging and Text chunking in Table 4. The tagger and chunker based on AdaBoost.SDFAN show competitive F-measure with previous best results.

[9] We used a machine with 3.6GHz DualCore Intel Xeon and 10 GB memory.

Table 4: *Comparison with previous best results.*

POS tagging		**Text Chunking**	
Guided learning [12]	97.33	LaSo [2]	94.4
Boosting [5]	97.32	Boosting [5]	94.30
CRF [13]	97.40	CRF [13]	95.15
This paper	97.34	**This paper**	94.34

As for fast classification methods, techniques for converting or pruning models or rules generated by machine learning algorithms are proposed. Model conversion techniques for SVMs with polynomial kernel that converts kernel-based classifier into a simple liner classifier are proposed in [3, 6]. For AdaBoost, a pruning method for hypotheses is proposed in [9].

Our method uses a rule conversion technique for sequential labeling problems. Although *CSLR-rep* can only be used in tasks that use each word as different features time and again, such as POS tagging and text, we obtain faster processing speed without loss in accuracy.

7 Conclusion and Future Work

We have proposed techniques for fast boosting-based POS tagging and text chunking. To reduce time-consuming rule evaluation, our method controls the generation of combination of features by specifying part of features that are not used for combination. We have also proposed a rule representation that enables us to merge similar rules. Experimental results have showed our techniques improve classification speed while maintaining accuracy.

References

[1] J. Aoe. An efficient digital search algorithm by using a double-array structure. In *IEEE Transactions on Software Engineering*, volume 15(9), 1989.

[2] H. Daumé III and D. Marcu. Learning as search optimization: approximate large margin methods for structured prediction. In *Proc. of ICML 2005*, pages 169–176, 2005.

[3] H. Isozaki and H. Kazawa. Efficient Support Vector classifiers for named entity recognition. In *Proc. of COLING 2002*, pages 390–396, 2002.

[4] T. Iwakura and S. Okamoto. Fast training methods of boosting algorithms for text analysis. In *Proc. of RANLP 2007*, pages 274–279, 2007.

[5] T. Iwakura and S. Okamoto. A fast boosting-based learner for feature-rich tagging and chunking. In *Proc. of CoNLL 2008*, pages 17–24, 2008.

[6] T. Kudo and Y. Matsumoto. Fast methods for kernel-based text analysis. In *Proc. of ACL-03*, pages 24–31, 2003.

[7] T. Kudo, J. Suzuki, and H. Isozaki. Boosting-based parse reranking with subtree features. In *Proc. of ACL 2005*, pages 189–196, 2005.

[8] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The Penn Treebank. pages 313–330, 1994.

[9] D. D. Margineantu and T. G. Dietterich. Pruning adaptive boosting. In *Proc. of ICML 1997*, pages 211–218, 1997.

[10] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

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

[12] L. Shen, G. Satta, and A. Joshi. Guided learning for bidirectional sequence classification. In *Proc. of ACL 2007*, pages 760–767, 2007.

[13] J. Suzuki and H. Isozaki. Semi-supervised sequential labeling and segmentation using giga-word scale unlabeled data. In *Proc. of ACL-08: HLT*, pages 665–673, June 2008.