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 technique that enables us to merge similar rules that consist of the same set of words and attributes that only differ in order. We evaluate our methods by using features generated from the current word and its surrounding words. Thus similar rules, for example, that consist of the same set of words but differ only in positions, 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 competitive 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 SVMs and e-mails, processing speed of base technologies such as POS tagging and text chunking. 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 differ only in positions, 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 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 \( \mathcal{F} = \{f_1, f_2, \ldots, f_M\} \) be \( M \) types of features represented by strings. Let \( S = \{ (x_1, y_1), \ldots, (x_n, y_n) \} \), where each example \( x_i \in \mathcal{X} \) 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 \( |x_i| \) \( (0 < |x_i| \leq M) \) be the number of features included in a feature-set \( x_i \), which we call the size of \( x_i \), and \( x_{i,j} \in \mathcal{F} \) \( (1 \leq j \leq |x_i|) \) be a feature included in \( x_i \). We call a feature-set of size \( k \) a \( k \)-feature-set. We call \( x_i \) a subset of \( x_j \), if a feature-set \( x_i \) contains all the features in a feature-set \( x_j \). We denote subsets of feature-sets as \( x_i \subseteq x_j \).

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

\[
h_f(x, c) = \begin{cases} c & \text{if } f \subseteq x \\ 0 & \text{otherwise} \end{cases}.
\]

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

\[
F(x) = \text{sign}\left(\sum_{r=1}^{R} h_{f_r,c_r}(x)\right).
\]

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

\[
gain(f) \triangleq |\sqrt{W_{r+1}(f)} - \sqrt{W_{r-1}(f)}|,
\]

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

\[
W_{r,g}(f) = \sum_{i=1}^{m} w_{r,i}[f \subseteq x_i \land y_i = g],
\]

where \([\pi] \) is 1 if a proposition \( \pi \) 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 \( \nu \) types of feature-sets having gain in top \( \nu \) as \( \{f_1, \ldots, f_{\nu+1}\} \) at each iteration.

Then the boosting-based learner calculates the confidence value of each rule in the selected \( \nu \) rules and updates the weight of each sample. The confidence value \( c_r \) for the first rule \( f_1 \) in the selected \( \nu \) rules is defined as

\[
c_r \triangleq \frac{1}{\nu} \sum_{i=1}^{m} w_{r,i}[f_1 \subseteq x_i \land y_i = g]
\]

International Conference RANLP 2009 - Borovets, Bulgaria, pages 161–165
\[
\begin{align*}
\# F_k &: \text{A set of } k\text{-feature-sets} \\
\# R_o &: \nu \text{ optimal rules (feature-sets)} \\
\# R_k, \omega &: \omega k\text{-feature-sets for generating candidates} \\
\text{SelectNext}(R_o, n, S, W_i); \text{Select } n \text{ best rules in } R_o \\
\text{with gain } \{w_r | y_i \in \{ \pm 1\}\} \text{ and training samples } S \\
\# F_N, F_A &: \text{non-atomic, atomic features} \\
\text{procedure weak-learner}(F_k; S, W_i); \\
\# \nu \text{ best feature-sets as rules} \\
\text{SelectNext}(R_o, n, S, W_i); \\
\text{if } (c_r \leq \tau) \text{ return } R_o; \# \text{ Size constraint} \\
\# \omega \text{ best feature-sets in } F_k \text{ for generating candidates} \\
R_k, \omega = \text{SelectNext}(F_k, \omega, S, W_i); \\
\tau = \min \{ \text{gain}(f); \# \text{The gain of } \nu\text{-th optimal rule} \} \\
\text{Foreach } (f \in R_k, \omega) \# \text{Pruning candidates with upper bound of gain} \\
\text{if } u(f) \in \tau \text{ continue; } \\
\text{Foreach } (f \in F_N) \# \text{Generate candidates} \\
F_{k+1} = (F_k + 1) \text{gen}(f_k, f_i) ; \\
\text{end Foreach} \text{ } \text{end Foreach} \\
\text{return weak-learner}(F_k+1, S, W_i); \\
\end{align*}
\]

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

c_r = \frac{1}{2} \log \left( \frac{w_{r+1}(f_{r+1})}{w_{r+1}(f_{r+1})} \right),

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

\[
W_{r+1} = w_r \cdot \exp(-h(f_r(x_i))).
\]

Then the learner adds \( f_r \) to \( F_r \) as the \( r \)-th rule and its confidence value. When we calculate the confidence value \( c_{r+1} \) for \( f_{r+1} \), we use \( \{w_{r+1,1}, \ldots, w_{r+1,m}\} \) 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 \( W_r(f) = \sum_{\forall x \in \mathcal{X}} w_r(f \mid \{f \in F_r \}) \). Then \( W-dist \) sorts features based on the weight of each feature, and inserts each feature to one of the buckets. The weak learner finds \( \nu \) 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 \( \omega \) best \((k-1)\)-feature-sets in \( F_{k-1} \) with \( \text{gain} \).

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

For example, if we use features \( F_A = \{A, B, C\} \) and \( F_N = \{a, b, c\} \), we examine followings as candidates: \( \{A, B\}, \{C\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\} \) and \( \{a, b, c\} \).

Then \( \text{gain} \) is a function to generate combination of features. We denote \( f^* = f + f \) as the generation of \( k+1 \)-feature-set \( f^* \) that consists of a feature \( f \) and a \( k \)-feature-set \( f \). Let \( ID(f) \) be the integer corresponding to \( f \), called \( id \), and \( \phi \) be 0-feature-set. Then the \( \text{gen} \) is defined as follows.

\[
\text{gen}(f, f) = \begin{cases} 
\phi & \text{if } (f \subseteq F_A) \\
(1 + f) & \text{if } ID(f) > \max_{f' \in F_A} ID(f'). \\
\text{otherwise} & \end{cases}
\]

\[
\begin{align*}
\# S &= \{\{x_i, y_i\}_{i=1}^{m} \}; \quad x_i \subseteq X, y_i \in \{ \pm 1\} \\
\# W_r &= \{w_r|y_i = y\}_{i=1}^{m} \text{ Weights of samples after learning} \\
\# \nu \text{ types of rules} \\
\# [B] &= \text{The size of bucket } B = \{B[0], \ldots, B[B - 1]\} \\
\# b, r &= \text{The current bucket and rule number} \\
\# \text{distFT} &: \text{distribute features to buckets} \\
\text{procedure AdaBoost.SDFAN} \\
B &= \text{distFT}(S, [B]); \# \text{Distributing features into } B \\
\text{Initialize values and weights:} \\
r = 1; \quad b = 0; \quad c_0 = \frac{1}{2} \log \left( \frac{1}{2} \right); \\
\text{For } i = 1, \ldots, m; \quad w_{r+1} = \exp(c_i) \\
\text{While } (r < B) \# \text{Learning } B \text{ types of rules} \\
\text{Select } \nu \text{ rules and increment bucket id } b \\
R &= \text{weak-learner}(B[0], S, W_r); b++; \\
\text{Foreach } f \in R_i \# \text{Update weights with each rule} \\
c = \frac{1}{2} \log \left( \frac{1}{2} \right) (1 + f) \\
\text{For } i = 1, \ldots, m; \quad w_{r+1,i} = w_r \cdot \exp(-y_ib(f)(x_i)); \\
f_r = f; \quad c_r = c + r; +; \\
\text{end Foreach} \text{ } \text{end While} \\
\text{return } f(x) = \text{sign}(c_0 + \sum_{i=1}^{R} b(f, c_i)(x_i)) \\
\end{align*}
\]

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

The \( \text{gen} \) 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 (\( \zeta \)):** We examine candidates whose size is no greater than a threshold \( \zeta \).
- **Upper bound of gain:** The upper bound is defined as

\[
u(f)_{\text{det}} = \max \{\sqrt{W_{r+1}(f)}, \sqrt{W_{r+1}(f)}\}.
\]

For any feature-set \( F' \subseteq F \), which contains \( f \) (i.e. \( f \subseteq F' \)), the \( \text{gain}(F') \) is bounded under \( u(f) \), since \( 0 \leq W_{r+1}(f) \leq W_{r+1}(f) \) for \( y \in \{ \pm 1\} \). Thus if \( u(f) \) is less than \( r \), the gain of the current optimal rule, candidates that contain \( f \) are safely pruned.

Figure 2 describes an overview of our algorithm, which we call AdaBoost for a weak learner learning. Several rules from \( \text{Distributed Features} \) consist of Atomic and Non-Atomic (AdaBoost.SDFAN, for short).

#### 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 “T,” “am” and “happy” as features. To distinguish words that appear different locations, we usually express words with relative locations from current word like “-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 \( \frac{1}{2} \log \left( \frac{w_{r+1}(f)}{w_{r+1}(f)} \right) \), where \( W_r = \sum_{y \in \{ \pm 1\}} |y_i| \) for \( y \in \{ \pm 1\} \).
short), to merge similar rules. To use CSLR-rep, we convert
weak-hypotheses (WHs, for short) generated by AdaBoost.SDFAN
to CSLR-rep. A CSLR-rep-based WH is represented as
\{(rule, \{p_i, cl_i, c_i\}, \ldots, \{p_q, cl_q, c_q\})\}.

The rule is a rule generated by merging rules learned
by AdaBoost.SDFAN. \(p_i\), 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 \{((I_{-2}, am_{-1}), JJ, c_0),
((I_{-1}, am_0), VBP, c_1)\} and \{\((I_{0}, am_{1}), PRP, c_2)\}\}
be WHs generated by AdaBoost.SDFAN, and JJ, VBP and PRP be class tags.
These WHs are converted to the following CSLR-rep-based rule:
\{\{I_{0}, am_{1}\}, \{JJ, c_0\}, \{VBP, c_1\}, \{0, PRP, c_2\}\}\}.

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 “T”, respectively.

When we use the three original WHs 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 \((bp \times -1)\) as the scoring-position of the
current score.

3.3 Rule Application
We describe an overview of the application of rules represen-
ted by CSLR-rep. We consider two types of features,
dynamic-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 \(W\)

Figure 4 shows an overview of the application. Let \{\(wd_1, \ldots, wd_y\)\}
be an input that consists of \(N\) \((1 \leq N)\) words.
Each word \(wd_i\) \((1 \leq i \leq N)\) has \(w_{d_j}\) types
of attributes. We denote \(j\)-th attribute of \(wd_i\) as \(w_{d_j,i}\).

is a set of rules represented by CSLR-rep and RC[rc] is the set of
\{(scoring-position, class, score)\} of \(rc\).

The application has two stages for static-features and
dynamic-features. Our algorithm first 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. 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 Rs[i].

We define subsets of rules as follows:

**Definition 1** Subsets of rules
If there exists rule in \(\{rule, \mbox{scores}\} \in RC \mbox{ that satisfies}
\(rc \subseteq \mbox{rule} \wedge \mbox{rc} \neq \mbox{rule}\), we call \(rc\) a subset of rules of \(RC\)
and denote it as

\(rc \subset RC\).

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

We describe an example. Let \(RC = \{\{E_{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” first. “I” is inserted to \(Rs[1]\) because of \(\{I_{0}\} \subset RC\).
We check “am:1” with “(I)“ in \(Rs[1]\), and \(\{I_{0}, am_{1}\}\) is found.
Finally we check “happy:2” with \(Rs[1]\). 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 di-
rection. If the dynamic-features of “am” and “happy” are
VBP and JJ, we check VBP and JJ with 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)“ in \(Rs[1]\), \(\{I_{0}, VBP_{-1}\}\) is found and
inserted to \(Rs[1]\). Then we check “JJ:2” with “I” and
\(\{I_{0}, VBP_{-1}\}\) in \(Rs[1]\). Then we check these dynamic-
features with \(Rs[2]\).

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 fea-
ture to CSLR-rep, the number of types of dynamic-features
increases. Since original rule representation only handles
dynamic-features within \(\Delta\), the total number of types of
dynamic-features is up to “\(\Delta \times CL\)”, where \(CL\) is
the number of classes in each task. However, the total num-
ber of dynamic-features in CSLR-rep is up to “\(\frac{W_{-1}}{\Delta} + \Delta + \max(\frac{W_{-1}}{\Delta}, \Delta) - 1) \times CL\) " 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), develop-
ment (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.
For \(\Delta = 1; \leq N', \Delta + \) # beginning position
For \(i = i'; i < i + W', \Delta + \) # combination position
For \(j = 1; \leq |w_{di}|, \Delta + \) attributes
\[ \text{foreach } rc \in R_{i}[i'] \]
\[ \text{lm} = i - i' \quad \# \text{current location-marker} \]
\[ rc' = rc \cdot \text{"wd}_{i,j}, \text{loc}" \]
\[ \# \text{If } RC[rc'] \text{ is applied,} \]
\[ \text{assign the scores with base position } i' \]
\[ \text{assignScores}(RC[rc'][i], i') \]
\[ \text{if } rc' \subset RC \quad R_{i}[i'] = R_{i}[i'] \cup rc' \]
endForeach
\[ \# \text{if no subset of rules for } i', \text{ go to } i' + 1 \text{-th word} \]
\[ \text{if } R_{i}[i'] = \{ \} \text{ break} \]
endFor
endFor
\[ \text{for Dynamic-feature : EOS to BOS direction} \]
\[ \text{For } i' = N; 1 \leq i'; i' - \# \text{ beginning position} \]
\[ \text{checking rules including Dynamic-feature} \]
\[ db = i' \cdot \Delta - 1 - \Delta; \quad dc = i' + \max (L_{i'} - \Delta); \]
\[ \text{for } i = 1; db < dc; i++ \]
\[ \text{foreach } rc \in R_{i}[i] \]
\[ \text{lm} = j - i' \quad \# \text{current location-marker} \]
\[ rc' = rc \cdot dft_{i,j,lm} \quad \# \text{dft}_{i,j,lm} \text{ is the tag of } i'-\text{th word} \]
\[ \text{assignScores}(RC[rc'][i], i) \]
\[ \text{if } rc' \subset RC \quad R_{i}[i'] = R_{i}[i'] \cup rc' \]
endForeach
endFor
endFor

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 \(\Delta\) 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\(\leq\)I-NP, 100\(\leq\)E-VP\(\leq\)1000, rank:1-NP=1
rank:E-VP=2, candidate=I-NP, and candidate=E-VP.

5 Experiments
We tested \(R=200,000, |B|=1,000, \nu = 10, \omega=10, \{1,2,3\}\) and \((W, \Delta)=\{(3,1), (5,2), (7,3)\}\). 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.

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
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 | 1 2 3 | 1 2 3 |
| (3,1) | 3977 | 5023 | 5255 |
| (5,2) | 8118 | 2564 | 1445 |
| (7,3) | 6673 | 1842 | 1007 |
| with CSLR-REP | 1 2 3 | 1 2 3 |
| (3,1) | 19467 | 2009 | 2013 |
| (5,2) | 18201 | 2807 | 1102 |
| (7,3) | 15658 | 2195 | 754 |

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 linear 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. Dauzé III and D. Marcus. 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. Suzukib, 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(23):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 max-margin scale unlabeled data. In Proc. of ACL-08: HLT, pages 665–673, June 2008.