A Boosted Semi-Markov Perceptron

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 a boosting algorithm that uses a semi-Markov perceptron. The training algorithm repeats the training of a semi-Markov model and the update of the weights of training samples. In the boosting, training samples that are incorrectly segmented or labeled have large weights. Such training samples are aggressively learned in the training of the semi-Markov perceptron because the weights are used as the learning ratios. We evaluate our training method with Noun Phrase Chunking, Text Chunking and Extended Named Entity Recognition. The experimental results show that our method achieves better accuracy than a semi-Markov perceptron and a semi-Markov Conditional Random Fields.

1 Introduction

Natural Language Processing (NLP) basic tasks, such as Noun Phrase Chunking, Text Chunking, and Named Entity Recognition, are realized by segmenting words and labeling to the segmented words. To realize these tasks, supervised learning algorithms have been applied successfully. In the early stages, algorithms for training classifiers, including Maximum Entropy Models (Tsuruoka and Tsujii, 2005), AdaBoost-based learning algorithms (Carreras et al., 2002), and Support Vector Machines (SVMs) (Kudo and Matsumoto, 2001) were widely used. Recently, learning algorithms for structured prediction, such as linear-chain structured predictions, and semi-Markov model-based ones, have been widely used. The examples of linear-chain structured predictions include Conditional Random Fields (CRFs) (Lafferty et al., 2001) and structured perceptron (Collins, 2002). The examples of semi-Markov model-based ones include semi-Markov model perceptron (Cohen and Sarawagi, 2004), and semi-Markov CRFs (Sarawagi and Cohen, 2005). Among these methods, semi-Markov-based ones have shown good performance in terms of accuracy (Cohen and Sarawagi, 2004; Sarawagi and Cohen, 2005; Okanohara et al., 2006; Iwakura et al., 2011). One of the reasons is that a semi-Markov learner trains models that assign labels to hypothesized segments (i.e., word chunks) instead of labeling to individual words. This enables use of features that cannot be easily used in word level processing such as the beginning word of a segment, the end word of a segment, and so on.

To obtain higher accuracy, boosting methods have been applied to learning methods for training classifiers. Boosting is a method to create a final hypothesis by repeatedly generating a weak hypothesis and changing the weights of training samples in each training iteration with a given weak learner such as a decision stump learner (Schapire and Singer, 2000) and a decision tree learner (Carreras et al., 2002). However, to the best of our knowledge, there are no approaches that apply boosting to learning algorithms for structured prediction. In other words, if we can successful apply boosting to learning algorithms for structured prediction, we expect to obtain higher accuracy.

This paper proposes a boosting algorithm for a semi-Markov perceptron. Our learning method uses a semi-Markov perceptron as a weak learner, and AdaBoost is used as the boosting algorithm. To apply boosting to the semi-Markov perceptron, the following methods are proposed; 1) Use the weights of training samples decided by AdaBoost as the learning ratios of the semi-Markov perceptron, and 2) Training on AdaBoost with the loss between the correct output of a training sample and the incorrect output that has the highest score. By the first method, the semi-Markov perceptron can aggressively learn training samples that are in-

Proceedings of the Seventeenth Conference on Computational Natural Language Learning, pages 47–55,
Sofia, Bulgaria, August 8-9 2013.

correctly classified at previous iteration because such training samples have large weights. The second method is a technique to apply AdaBoost to learning algorithms for structured prediction that generate negative samples from N-best outputs (Cohen and Sarawagi, 2004), or consider all possible candidates (Sarawagi and Cohen, 2005). We also prove the convergence of our training method.

This paper is organized as follows: In Section 2, we describe AdaBoost and Semi-Markov perceptron is described in Section 3. Our proposed method is described in Section 4, and the experimental setting, the experimetal results and related work are described in Section 5, 6, and 7.

2 AdaBoost

Let $\mathcal{X}$ be a domain or sample space and $\mathcal{Y}$ be a set of labels $\{-1, +1\}$. The goal is to induce a mapping $F : \mathcal{X} \rightarrow \mathcal{Y}$. Let S be $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)\}$, which is a set of training samples, where $\mathbf{x}_i$ is a sample in $\mathcal{X}$, and each y_i belongs to $\mathcal{Y}$. Each boosting learner learns T types of weak hypothesis with a given weak learner to produce a final hypothesis F:

$$F(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})).$$

where $sign(x)$ is 1 if x is positive, otherwise, it returns -1.

The h_t ($1 \leq t \leq T$) is the t-th weak hypothesis learned by the weak learner. $h_t(\mathbf{x})$ is the prediction to $\mathbf{x} \in \mathcal{X}$ with h_t, and α_t is the confidence value of h_t that is calculated by the boosting learner.

The given weak learner learns a weak hypothesis h_t from training samples $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ and weights over samples $\{w_{t,1}, ..., w_{t,m}\}$ at round t. $w_{t,i}$ is the weight of sample number i at round t for $1 \leq i \leq m$. We set $w_{1,i}$ to $1/m$.

After obtaining t-th weak hypothesis h_t, the boosting learner calculates the confidence-value α_t for h_t. Then, the boosting learner updates the weight of each sample. We use the AdaBoost framework (Freund and Schapire, 1997; Schapire and Singer, 1999). The update of the sample weights in AdaBoost is defined as follows:

$$w_{t+1,i} = w_{t,i} \frac{e^{-y_i \ \alpha_t h_t(\mathbf{x}_i)}}{Z_t(\alpha_t),} \quad (1)$$

where e is Napier's constant and

$$Z_t(\alpha_t) = \sum_{i=1}^{m} w_{t,i} e^{-y_i \ \alpha_t h_t(\mathbf{x}_i)} \quad (2)$$

```
# Training data: S = {(X_i, Y_i)}_{i=1}^m
# The learning rations of S: {ε_i}_{i=1}^m
# The maximum iteration of perceptron: P
SemiMarkovPerceptron(S, P, {ε_i}_{i=1}^m)
w = ⟨0, ..., 0⟩ # Weight vector
a = ⟨0, ..., 0⟩ # For averaged perceptron
c = 1 # The total number of iteration
For p = 1...P
 For i = 1...m
  Y*_i = argmax_{Y∈Y(X_i)} w · Φ(X_i, Y)
  If Y*_i ≠ Y_i
   w = w + ε_i(Φ(X_i, Y_i) − Φ(X_i, Y*_i))
   a = w + cε_i(Φ(X_i, Y_i) − Φ(X_i, Y*_i))
  endIf
  c++
 endFor
endFor
return (w - a / c)
```

Figure 1: A pseudo code of a semi-Markov perceptron.

is the normalization factor for $\sum_{i=1}^{m} w_{t+1,i} = 1$.

Let π be any predicate and $[[\pi]]$ be 1 if π holds and 0 otherwise. The following upper bound holds for the training error of F consisting of T weak hypotheses (Schapire and Singer, 1999):

$$\frac{1}{m} \sum_{i=1}^{m} [[F(\mathbf{x}_i) \neq y_i]] \leq \prod_{t=1}^{T} Z_t(\alpha_t). \quad (3)$$

Eq. (1) and Eq. (3) suggest AdaBoost-based learning algorithms will converge by repeatedly selecting a confidence-value of α_t for h_t at each round, that satisfies the following Eq. (4) at each round:

$$Z_t(\alpha_t) < 1. \quad (4)$$

3 Semi-Markov Perceptron

In a semi-Markov learner, instead of labeling individual words, hypothesized segments are labeled. For example, if a training with an input 'I win' and a label set $\{_{NP}, _{VP}\}$ is conducted, considered segments with their labels are the following: "[I]$_{(NP)}$ [win]$_{(NP)}$", "[I]$_{(NP)}$ [win]$_{(VP)}$", "[I]$_{(VP)}$ [win]$_{(NP)}$", "[I]$_{(VP)}$ [win]$_{(VP)}$", "[I win]$_{(NP)}$", and "[I win]$_{(VP)}$".

Figure 1 shows a pseudo code of a semi-Markov perceptron (Semi-PER) (Cohen and Sarawagi, 2004). We used the averaged perceptron (Collins,

2002) based on the efficient implementation described in (Daumé III, 2006). Let $S = \{(\mathbf{X}_i, \mathbf{Y}_i)\}_{i=1}^m$ be a set of m training data, $\mathbf{X}_i$ be i-th training sample represented by a word sequence, and $\mathbf{Y}_i$ be the correct segments and the correct labeling of $\mathbf{X}_i$. $\mathbf{Y}_i$ consists of $|\mathbf{Y}_i|$ segments. $\mathbf{Y}_{i(j)}$ means the j-th segment of $\mathbf{Y}_i$, and $l(\mathbf{Y}_{i(j)})$ means the label of $\mathbf{Y}_{i(j)}$.

$\Phi(\mathbf{X}, \mathbf{Y})$ is a mapping to a D-dimensional feature vector defined as

$$\Phi(\mathbf{X}, \mathbf{Y}) = \sum_{d=1}^{D} \sum_{j=1}^{|\mathbf{Y}|} \phi_d(\mathbf{X}, \mathbf{Y}_{(j)}),$$

where ϕ_d is a feature represented by an indicator function that maps an input $\mathbf{X}$ and a segment with its label $\mathbf{Y}_{(j)}$ to a D-dimensional vector. For example, $\phi_{100}(\mathbf{X}, \mathbf{Y}_{(j)})$ might be the 100-th dimension's value is 1 if the beginning word of $\mathbf{Y}_{(j)}$ is "Mr." and the label $l(\mathbf{Y}_{(j)})$ is "NP".

$\mathbf{w}$ is a weight vector trained with a semi-Markov perceptron. $\mathbf{w} \cdot \Phi(\mathbf{X}, \mathbf{Y})$ is the score given to segments with their labels $\mathbf{Y}$ of $\mathbf{X}$, and $\mathcal{Y}(\mathbf{X})$ is the all possible segments with their labels for $\mathbf{X}$. The learning ratios of the training samples are $\{\epsilon_i\}_{i=1}^m$, and the ratios are set to 1 in a usual semi-Markov perceptron training.

In the training of the Semi-PER, for a given $\mathbf{X}_i$, the learner finds $\mathbf{Y}_i^*$ with the Viterbi decoding as described in (Cohen and Sarawagi, 2004):

$$\mathbf{Y}_i^* = \arg\max_{\mathbf{Y} \in \mathcal{Y}(\mathbf{X}_i)} \mathbf{w} \cdot \Phi(\mathbf{X}, \mathbf{Y}).$$

If $\mathbf{Y}_i^*$ is not equivalent to $\mathbf{Y}_i$ (i.e. $\mathbf{Y}_i^* \neq \mathbf{Y}_i$), the weight $\mathbf{w}$ is updated as follows:

$$\mathbf{w} = \mathbf{w} + \epsilon_i(\Phi(\mathbf{X}_i, \mathbf{Y}_i) - \Phi(\mathbf{X}_i, \mathbf{Y}_i^*)).$$

The algorithm takes P passes over the training samples.

4 A Boosted Semi-Markov Perceptron

This section describes how we apply AdaBoost to a semi-Markov perceptron training.

4.1 Applying Boosting

Figure 2 shows a pseudo code for our boosting-based Semi-PER. To train the Semi-PER within an AdaBoost framework, we used the weights of samples decided by AdaBoost as learning ratios. The initial weight value of i-th sample at boosting round 1 is $w_{1,i} = 1/m$. In the first iteration, Semi-PER is trained with the initial weights of samples.

```
# Training data: S = {(X_i, Y_i)}_{i=1}^m
# A weight vector at boosting round t: W_t
# The weights of S at round t: {w_{t,i}}_{i=1}^m
# The iteration of perceptron training: P
# The iteration of boosting training: T
SemiBoost(S, T, P)
W_0 = <0, ..., 0>
Set initial value: w_{1,i} = 1/m (for 1 ≤ i ≤ m)
While t ≤ T
 w_t=SemiMarkovPerceptron(S,P,{w_{t,i}}_{i=1}^m)
 Find α_t that satisfies Z̃_t(α_t) < 1.
 Update :W_t = W_{t-1} + α_t w_t
 For i = 1...m
 w_{t+1,i} = w_{t,i} * e^{- α_t d_t(X_i)} / Z̃_t(α_t)
 t++
endWhile
return W_T
```

Figure 2: A pseudo code of a boosting for a semi-Markov perceptron.

Then, we update the weights of training samples. Our boosting algorithm assigns larger weights to training samples incorrectly segmented or labeled. To realize this, we first define a loss for $\mathbf{X}_i$ at boosting round t as follows:

$$d_t(\mathbf{X}_i) = s_t(\mathbf{X}_i, \mathbf{Y}_i) - s_t(\mathbf{X}_i, \mathbf{Y}_i^t),$$

where,

$$\mathbf{Y}_i^t = \arg\max_{\mathbf{Y} \in \mathcal{Y}(\mathbf{X}_i) \wedge \mathbf{Y} \neq \mathbf{Y}_i} s_t(\mathbf{X}_i, \mathbf{Y}),$$

and

$$s_t(\mathbf{X}, \mathbf{Y}) = \mathbf{w}_t \cdot \Phi(\mathbf{X}, \mathbf{Y}).$$

$s_t(\mathbf{X}, \mathbf{Y})$ is a score of a word sequence $\mathbf{X}$ that is segmented and labeled as $\mathbf{Y}$, and $\mathbf{w}_t$ is a weight vector trained by Semi-PER at boosting round t. When a given input is correctly segmented and labeled, the second best output is generated with a forward-DP backward-A* N-best search algorithm (Nagata, 1994). Then we find a confidence-value α_t that satisfies $\tilde{Z}_t(\alpha_t) < 1$:

$$\tilde{Z}_t(\alpha_t) = \sum_{i=1}^{m} w_{t,i} e^{-\alpha_t d_t(\mathbf{X}_i)}. \quad (5)$$

After obtaining α_t, the weight of each sample is updated as follows:

$$w_{t+1,i} = w_{t,i} * e^{-\alpha_t d_t(\mathbf{X}_i)} / \tilde{Z}_t(\alpha_t). \quad (6)$$

If $s_t(\mathbf{X}_i, \mathbf{Y}_i)$ is greater than $s_t(\mathbf{X}_i, \mathbf{Y}_i^t)$ (i.e., $0 < d_t(\mathbf{X}_i)$), the weight of $\mathbf{X}_i$ is decreased because $\mathbf{X}_i$ is correctly segmented and labeled. Otherwise ($d_t(\mathbf{X}_i) < 0$), $\mathbf{X}_i$ has a larger weight value. The updated weights are used as the learning ratios in the training of Semi-PER at the next boosting round. Finally, we update the weight vector $\mathbf{W}_t$ trained with boosting as follows:

$$\mathbf{W}_t = \mathbf{W}_{t-1} + \alpha_t \mathbf{w}_t$$

This process is repeated T times, and a model $\mathbf{W}_T$, which consists of T types of Semi-PER-based models, is obtained.

In test phase, the segments and labels of a word sequence $\mathbf{X}$ is decided as follows:

$$\mathbf{Y}^* = \arg\max_{\mathbf{Y} \in \mathcal{Y}(\mathbf{X})} \mathbf{W}_T \cdot \Phi(\mathbf{X}, \mathbf{Y}).$$

4.2 Learning a Confidence Value

Since our algorithm handles real valued scores of samples given by Semi-PER on the exponential loss of AdaBoost, it's difficult to analytically determine a confidence-value α_t that satisfies $\tilde{Z}_t(\alpha_t) < 1$ at boosting round t.

Therefore, we use a bisection search to find a confidence-value. To detemin the range for the bisection search, we use a range between 0 and the confidence-value for a weak hypothesis h_t that returns its prediction as one of {-1,+1}. We define $h_t(\mathbf{X}_i)$ as $sign(d_t(\mathbf{X}_i))$. Schapire and Singer proposed an algorithm based on AdaBoost, called real AdaBoost (Schapire and Singer, 1999). The real AdaBoost analytically calculates the confidence-value that minimizes Eq. (2). The derivation of $Z_t(\alpha_t)$ with α_t is

$$Z_t'(\alpha_t) = \sum_{i=1}^{m} -h_t(\mathbf{X}_i) w_{t,i} e^{-\alpha_t h_t(\mathbf{X}_i)}.$$

By solving $Z_t'(\alpha_t) = 0$, we obtain

$$\tilde{\alpha_t} = \frac{1}{2} \log\left(\frac{\sum_{i=1}^{m} w_{t,i}[[h_t(\mathbf{X}_i) = 1]]}{\sum_{i=1}^{m} w_{t,i}[[h_t(\mathbf{X}_i) = -1]]}\right).$$

Finally, we select the value that minimizes Eq. (5) from the range between 0 and $2 \times \tilde{\alpha_t}$ with the bisection search as the confidence-value α_t. This is because we expect to find a better confidence-value from a wider range.

4.3 Convergence Analysis

If we repeatedly find a confidence-value ($0 < \alpha_t$) that satisfies $\tilde{Z}_t(\alpha_t) < 1$ at each boosting round, the training of the semi-Markov model will be converged as in the classification case described in Section 2.[1] The following bound on the training error can be proved:

$$\frac{1}{m} \sum_{i=1}^{m} [[\mathbf{Y}_i^* \neq \mathbf{Y}_i]] \leq \prod_{t=1}^{T} \tilde{Z}_t(\alpha_t)$$

where

$$\mathbf{Y}_i^* = \arg\max_{\mathbf{Y} \in \mathcal{Y}(\mathbf{X}_i)} \mathbf{W}_T \cdot \Phi(\mathbf{X}_i, \mathbf{Y}).$$

By unraveling Eq. (6), we have that

$$\begin{aligned} w_{T+1,i} &= w_{T,i} * e^{-\alpha_t d_t(\mathbf{X}_i)} / \tilde{Z}_t(\alpha_t) \\ &= \frac{e^{-\sum_{t=1}^{T} \alpha_t d_t(\mathbf{X}_i)}}{m \prod_{t=1}^{T} \tilde{Z}_t(\alpha_t)} \\ &= \frac{e^{-\sum_{t=1}^{T} \alpha_t \mathbf{w}_t \cdot (\Phi(\mathbf{X}_i, \mathbf{Y}_i) - \Phi(\mathbf{X}_i, \mathbf{Y}_i^t))}}{m \prod_{t=1}^{T} \tilde{Z}_t(\alpha_t)}. \end{aligned}$$

Therefore, if $\mathbf{Y}_i^* \neq \mathbf{Y}_i$,

$$\frac{e^{-\sum_{t=1}^{T} \alpha_t \mathbf{w}_t \cdot (\Phi(\mathbf{X}_i, \mathbf{Y}_i) - \Phi(\mathbf{X}_i, \mathbf{Y}_i^*))}}{m \prod_{t=1}^{T} \tilde{Z}_t(\alpha_t)} \leq w_{T+1,i},$$

since, for $1 \leq t \leq T$,

$$\mathbf{w}_t \cdot \Phi(\mathbf{X}_i, \mathbf{Y}_i^*) \leq \mathbf{w}_t \cdot \Phi(\mathbf{X}_i, \mathbf{Y}_i^t).$$

Moreover, when $\mathbf{Y}_i^* \neq \mathbf{Y}_i$, the following is satisfied.

$$\begin{aligned} 1 &\leq e^{-\sum_{t=1}^{T} \alpha_t \mathbf{w}_t \cdot (\Phi(\mathbf{X}_i, \mathbf{Y}_i) - \Phi(\mathbf{X}_i, \mathbf{Y}_i^*))} \\ &\leq e^{-\sum_{t=1}^{T} \alpha_t \mathbf{w}_t \cdot (\Phi(\mathbf{X}_i, \mathbf{Y}_i) - \Phi(\mathbf{X}_i, \mathbf{Y}_i^t))} \\ &= e^{-\sum_{t=1}^{T} \alpha_t d_t(\mathbf{X}_i)}. \end{aligned}$$

Therefore,

$$[[\mathbf{Y}_i^* \neq \mathbf{Y}_i]] \leq e^{-\sum_{t=1}^{T} \alpha_t d_t(\mathbf{X}_i)}.$$

These give the stated bound on training error;

$$\begin{aligned} \frac{1}{m} \sum_{i=1}^{m} [[\mathbf{Y}_i^* \neq \mathbf{Y}_i]] &\leq \frac{\sum_{i=1}^{m} e^{-\sum_{t=1}^{T} \alpha_t d_t(\mathbf{X}_i)}}{m} \\ &= \sum_{i=1}^{m} (\prod_{t=1}^{T} \tilde{Z}_t(\alpha_t)) w_{T+1,i} \\ &= \prod_{t=1}^{T} \tilde{Z}_t(\alpha_t). \end{aligned}$$

[1] $0 < \alpha_t$ means the weighted error of the current Semi-PER, $\sum_{i=1}^{m} [[\mathbf{Y}_i^t \neq \mathbf{Y}_i]] w_{i,t}$, is less than 0.5 on the training data. Fortunately, this condition was always satisfied with the training of Semi-PER in our experiments.

5 Experimental Settings

5.1 Noun Phrase Chunking

The Noun Phrase (NP) chunking task was chosen because it is a popular benchmark for testing a structured prediction. In this task, noun phrases called base NPs are identified. "[He] (NP) reckons [the current account deficit] (NP)..." is an example. The training set consists of 8,936 sentences, and the test set consists of 2,012 sentences.[2] To tune parameters for each algorithm, we used the 90% of the train data for the training of parameter tuning, and the 10% of the training data was used as a development data for measuring accuracy at parameter tuning. A final model was trained from all the training data with the parameters that showed the highest accuracy on the development data.

5.2 Text Chunking

We used a standard data set prepared for CoNLL-2000 shared task.[3] This task aims to identify 10 types of chunks, such as, NP, VP, PP, ADJP, ADVP, CONJP, INITJ, LST, PTR, and SBAR. "[He] (NP) [reckons] (VP) [the current account deficit] (NP)..." is an example of text chunking. The data consists of subsets of Penn Wall Street Journal treebank; training (sections 15-18) and test (section 20). To tune parameters for each algorithm, we used the same approach of the NP chunking one.

5.3 Japanese Extended NE Recognition

To evaluate our algorithm on tasks that include large number of classes, we used an extended NE recognition (ENER) task (Sekine et al., 2002). This Japanese corpus for ENER (Hashimoto et al., 2008) consists of about 8,500 articles from 2005 Mainichi newspaper. The corpus includes 240,337 tags for 191 types of NEs. To segment words from Japanese sentences, we used ChaSen.[4] Words may include partial NEs because words segmented with ChaSen do not always correspond with NE boundaries. If such problems occur when we segment the training data, we annotated a word chunk with the type of the NE included in the word chunk. The evaluations are performed based on the gold standard data for the test. We created the following sets for this experiment. Training data is news articles from January to October 2005 in the corpus, which includes 205,876 NEs. Development data is news articles in November 2005 in the corpus, which includes 15,405 NEs. Test data is news articles in December 2005 in the corpus, which includes 19,056 NEs.

Table 1: Features.

$[t_j, CL_j]$, $[t_j, WB_j]$, $[t_j, PB_j]$,
$[t_j, w_{bp}]$, $[t_j, p_{bp}]$,
$[t_j, w_{ep}]$, $[t_j, p_{ep}]$, $[t_j, w_{ip}]$,$[t_j, p_{ip}]$,
$[t_j, w_{bp}, w_{ep}]$, $[t_j, p_{bp}, p_{ep}]$,
$[t_j, w_{bp}, p_{ep}]$, $[t_j, p_{bp}, w_{ep}]$,
$[t_j, w_{bp-1}]$, $[t_j, p_{bp-1}]$, $[t_j, w_{bp-2}]$, $[t_j, p_{bp-2}]$,
$[t_j, w_{ep+1}]$, $[t_j, p_{ep+1}]$, $[t_j, w_{ep+2}]$, $[t_j, p_{ep+2}]$,
$[t_j, p_{bp-2}, p_{bp-1}]$, $[t_j, p_{ep+1}, p_{ep+2}]$,
$[t_j, p_{bp-2}, p_{bp-1}, p_{bp}]$, $[t_j, p_{ep}, p_{ep+1}, p_{ep+2}]$
% Features used for only Text Chunking and NP Chunking
$[t_j, w_{bp}, w_{ip}]$, $[t_j, w_{bp}, p_{ip}]$,
$[t_j, w_{bp}, p_{ip}]$, $[t_j, p_{bp}, p_{ip}]$,
$[t_j, w_{ep}, w_{ip}]$, $[t_j, w_{ep}, p_{ip}]$,
$[t_j, w_{ep}, p_{ip}]$, $[t_j, p_{ep}, p_{ip}]$,
$[t_j, w_{bp}, w_{ep}, w_{ip}]$, $[t_j, w_{bp}, w_{ep}, p_{ip}]$,
$[t_j, w_{bp}, w_{ep}, p_{ip}]$, $[t_j, w_{bp}, p_{ep}, p_{ip}]$

5.4 Evaluation Metrics

Our evaluation metrics are recall (RE), precision (PR), and F-measure (FM) defined as follows:

$$RE = C_{ok}/C_{all},\ PR = C_{ok}/C_{rec}$$

and

$$FM = 2 \times RE \times PR/(RE + PR),$$

where C_{ok} is the number of correctly recognized chunks with their correct labels, C_{all} is the number of all chunks in a gold standard data, and C_{rec} is the number of all recognized chunks.

5.5 Features

Table 1 lists features used in our experiments. For NP Chunking and Text Chunking, we added features derived from segments in addition to ENER features.[5]

w_k is the k-th word, and p_k is the Part-Of-Speech (POS) tag of k-th word. bp is the position of the first word of the current segment in a given

[2] We used the data obtained from ftp://ftp.cis.upenn.edu/pub/chunker/ .

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

[4] We used ChaSen-2.4.2 with Ipadic-2.7.0. ChaSen's web page is http://chasen-legacy.sourceforge.jp/.

[5] We did not use the additional features for ENER because the features did not contribute to accuracy.

word sequence. ep indicates the position of the last word of the current segment. ip is the position of words inside the current segment ($bp < ip < ep$). If the length of the current segment is 2, we use features that indicate there is no inside word as the features of ip-th words. t_j is the NE class label of j-th segment. CL_j is the length of the current segment, whether it be 1, 2, 3, 4, or longer than 4. WB_j indicates word bigrams, and PB_j indicates POS bigrams inside the current segment.

5.6 Algorithms to be Compared

The following algorithms are compared with our method.

- **Semi-Markov perceptron** (**Semi-PER**) (Cohen and Sarawagi, 2004): We used one-best output for training. This Semi-PER is also used as the weak learner of our boosting algorithm.

- **Semi-Markov CRF** (**Semi-CRF**) (Sarawagi and Cohen, 2005): To train Semi-CRF, a stochastic gradient descent (SGD) training for L1-regularized with cumulative penalty (Tsuruoka et al., 2009) was used. The batch size of SGD was set to 1.

These algorithms are based on sequentially classifying segments of several adjacent words, rather than single words. Ideally, all the possible word segments of each input should be considered for this algorithm. However, the training of these algorithms requires a great deal of memory. Therefore, we limit the maximum size of the word-segments. We use word segments consisting of up to ten words due to the memory limitation.

We set the maximum iteration for Semi-PER to 100, and the iteration number for Semi-CRF trained with SGD to $100 \times m$, where m is the number of training samples. The regularization parameter C of Semi-CRF and the number of iteration for Semi-PER are tuned on development data.[6] For our boosting algorithm, the number of boosting iteration is tuned on development data with the number of iteration for Semi-PER tuned on development data. We set the maximum iteration number for boosting to 50.

[6]For C of Semi-CRF, $\{1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}, 10^{-9}\}$ were examined.

Table 2: Results of NP Chunking.

Learner	F-measure	Recall	Precision
Semi-PER	94.32	94.53	94.11
Semi-CRF	94.32	94.52	94.13
Semi-Boost	**94.60**	**94.85**	**94.35**

Table 3: Results of Text Chunking.

Learner	F-measure	Recall	Precision
Semi-PER	94.10	94.15	94.05
Semi-CRF	93.79	93.96	93.62
Semi-Boost	**94.15**	**94.27**	**94.03**

6 Experimental Results

We used a machine with Intel(R) Xeon(R) CPU X5680 @ 3.33GHz and 72 GB memory. In the following, our proposed method is referred as **Semi-Boost**.

6.1 NP Chunking

Table 2 shows the experimental results on NP Chunking. Semi-Boost showed the best accuracy. Semi-Boost showed 0.28 higher F-measure than Semi-PER and Semi-CRF. To compare the results, we employed a McNemar paired test on the labeling disagreements as was done in (Sha and Pereira, 2003). All the results indicate that there is a significant difference ($p < 0.01$). This result shows that Semi-Boost showed high accuracy.

6.2 Text Chunking

Table 3 shows the experimental results on Text Chunking. Semi-Boost showed 0.36 higher F-measure than Semi-CRF, and 0.05 higher F-measure than Semi-PER. The result of McNemar test indicates that there is a significant difference ($p < 0.01$) between Semi-Boost and Semi-CRF. However, there is no significant difference between Semi-Boost and Semi-PER.

6.3 Extended Named Entity Recognition

Table 4 shows the experimental results on ENER. We could not train Semi-CRF because of the lack of memory for this task. Semi-Boost showed 0.24 higher F-measure than that of Semi-PER. The results indicate there is a significant difference ($p <$

Table 4: Experimental results for ENER.

Learner	F-measure	Recall	Precision
Semi-PER	81.86	79.06	84.87
Semi-CRF	N/A		
Semi-Boost	**82.10**	**79.36**	**85.03**

Table 5: Training time of each learner (second) for NP Chunking (NP), Text Chunking (TC) and ENER. The number of Semi-Boost iteration is only one time. The +20 cores means training of Semi-Boost with 20 cores.

Learner	NP	TC	ENER
Semi-PER	475	559	13,559
Semi-CRF	2,120	8,228	N/A
Semi-Boost	499	619	32,370
+20 cores	487	650	19,598

0.01).[7]

6.4 Training Speed

We compared training speed under the following condition; The iteration for Semi-PER is 100, the iteration number for Semi-CRF trained with SGD is $100 \times m$, where m is the number of training samples, and the one time iteration of boosting with the perceptron iteration 100. Therefore, all training methods attempted $100 \times m$ times estimation.

Table 5 shows the training time of each learner. In NP Chunking, the training time of Semi-PER, Semi-CRF, and Semi-Boost were 475 seconds, 2,120 seconds, and 499 seconds. In Text Chunking, the training time of Semi-PER, Semi-CRF, and our method were 559 seconds, 8,228 seconds, and 619 seconds. Semi-Boost shows competitive training speed with Semi-PER and 4 to 13 times faster training speed in terms of the total number of parameter estimations The difference of time between Semi-PER and our method is the time for calculating confidence-value of boosting.

When Semi-Boost trained a model for ENER, the training speed was degraded. The training time of Semi-Boost was 32,370 and the training time of Semi-PER was 13,559. One of the reasons is the generation of an incorrect output of each training sample. In our observation, when the number of classes is increased, the generation speed of incorrect outputs with N-best search is degraded. To improve training speed, we used 20 cores for generating incorrect outputs. When the training with 20 cores was conducted, the training data was split to 20 portions, and each portion was processed with one of each core. The training time with the 20 cores was 19,598 for ENER. However, the training time of NP Chunking was marginally improved and that of Text Chunking was slightly increased. This result implies that multi-core processing is effective for the training of large classes like ENER in Semi-Boost.

In fact, since Semi-Boost requires additional boosting iterations, the training time of Semi-Boost increases. However, the training time increases linearly by the number of boosting iteration. Therefore, Semi-Boost learned models from the large training data of ENER.

[7] The results on the test data were compared by character units as in Japanese morphological analysis (Iwakura et al., 2011). This is because the ends or beginnings of Japanese NEs do not always correspond with word boundaries.

Table 6: The best results for NP Chunking (FM).

(Kudo and Matsumoto, 2001)	94.22
(Sun et al., 2009)	94.37
This paper	**94.60**

6.5 Memory Usage

Semi-Boost consumed more memory than Semi-PER. This is because our learning method maintains a weight vector for boosting in addition to the weight vector of Semi-PER. Compared with Semi-CRF, Semi-Boost showed lower memory consumption. On the training data for Text Chunking, the memory size of Semi-Boost, Semi-PER, and Semi-CRF are 4.4 GB, 4.1 GB, and 18.0 GB. When we trained models for ENER, Semi-PER consumed 32 GB and Semi-Boost consumed 33 GB. However, Semi-CRF could not train models because of the lack of memory. This is because Semi-CRF maintains a weight vector and a parameter vector for L1-norm regularization and Semi-CRF considers all possible patterns generated from given sequences in training. In contrast, Semi-PER and Semi-Boost only consider features that appeared in correct ones and incorrectly recognized ones. These results indicate that Semi-Boost can learn models from large training data.

Table 7: The best results for Text Chunking (FM).

Semi-supervised learning	
(Ando and Zhang, 2005)	94.39
(Iwakura and Okamoto, 2008)	94.32
(Suzuki and Isozaki, 2008)	95.15
With additional resources	
(Zhang et al., 2001)	94.17
(Daumé III and Marcu, 2005)	94.4
Without lexical resources	
(Kudo and Matsumoto, 2001)	93.91
(Kudo et al., 2005)	94.12
(Tsuruoka and Tsujii, 2005)	93.70
(Tsuruoka et al., 2009)	93.68
This paper	**94.15**

7 Related Work

7.1 NP Chunking

Table 6 shows the previous best results for NP Chunking. The F-measure of Semi-Boost is 94.60 that is 0.23 higher than that of (Sun et al., 2009) and 0.38 higher than that of (Kudo and Matsumoto, 2001).

7.2 Text Chunking

Table 7 shows the previous best results for Text Chunking. We see that our method attained a higher accuracy than the previous best results obtained without any additional lexical resources such as chunking methods based on SVM (Kudo and Matsumoto, 2001), CRF with reranking (Kudo et al., 2005), Maximum Entropy (Tsuruoka and Tsujii, 2005), and CRF (Tsuruoka et al., 2009). This result indicates that our method performs well in terms of accuracy.

The previous results with lexical resources or semi-supervised ones showed higher accuracy than that of our method. For example, lexical resources such as lists of names, locations, abbreviations and stop words were used (Daumé III and Marcu, 2005), and a full parser output was used in (Zhang et al., 2001). Semi-supervised ones used a generative model trained from automatically labeled data (Suzuki and Isozaki, 2008), the candidate tags of words collected from automatically labeled data (Iwakura and Okamoto, 2008), or automatically created classifiers by learning from thousands of automatically generated auxiliary classification problems from unlabeled data (Ando and Zhang, 2005). Our algorithm can also incorporate the lexical resources and the semi-supervised approaches. Future work should evaluate the effectiveness of the incorporation of them.

7.3 Extended Named Entity Recognition

For ENER, the best result was the Semi-PER one (Iwakura et al., 2011). The F-measure of Semi-PER was 81.95, and the result was higher than NE chunker based on structured perceptron (Collins, 2002), and NE chunkers based on shift-reduce-parsers (Iwakura et al., 2011). Our method showed 0.15 higher F-measure than that of the Semi-PER one. This result is also evidence that our method performs well in terms of accuracy.

7.4 Training Methods

There have been methods proposed to improve the training speed for semi-Markov-based learners. With regard to reducing the space of lattices built into the semi-Markov-based algorithms, a method was proposed to filter nodes in the lattices with a naive Bayes classifier (Okanohara et al., 2006). To improve training speed of Semi-CRF, a succinct representation of potentials common across overlapping segments of semi-Markov model was proposed (Sarawagi, 2006). These methods can also be applied to Semi-PER. Therefore, we can expect improved training speed with these methods.

Recent online learners update both parameters and the estimate of their confidence (Dredze and Crammer, 2008; Crammer et al., 2009; Mejer and Crammer, 2010; Wang et al., 2012). In these algorithms, less confident parameters are updated more aggressively than more confident ones. These algorithms maintain the confidences of features. In contrast, our boosting approach maintains the weights of training samples. In future work, we'd like to consider the use of these algorithms in boosting of semi-Markov learners.

8 Conclusion

This paper has proposed a boosting algorithm with a semi-Markov perceptron. The experimental results on Noun Phrase Chunking, Text Chunking and Japanese Extended Named Entity Recognition have shown that our method achieved better accuracy than a semi-Markov perceptron and a semi-Markov CRF. In future work, we'd like to evaluate the boosting algorithm with structured prediction tasks such as POS tagging and parsing.

References

Rie Ando and Tong Zhang. 2005. A high-performance semi-supervised learning method for text chunking. In *Proc. of ACL'05*, pages 1–9.

Xavier Carreras, Lluís Màrques, and Lluís Padró. 2002. Named entity extraction using adaboost. In *Proc. of CoNLL'02*, pages 167–170.

William W. Cohen and Sunita Sarawagi. 2004. Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. In *Proc. of KDD'04*, pages 89–98.

Michael Collins. 2002. Discriminative training methods for Hidden Markov Models: theory and experiments with perceptron algorithms. In *Proc. of EMNLP'02*, pages 1–8.

Koby Crammer, Alex Kulesza, and Mark Dredze. 2009. Adaptive regularization of weight vectors. In *Proc. of NIPS'09*, pages 414–422.

Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proc. of ICML'05*, pages 169–176.

Hal Daumé III. 2006. *Practical Structured Learning Techniques for Natural Language Processing*. Ph.D. thesis, University of Southern California.

Mark Dredze and Koby Crammer. 2008. Online methods for multi-domain learning and adaptation. In *Proc. of EMNLP'08*, pages 689–697.

Yoav Freund and Robert E. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139.

Taiichi Hashimoto, Takashi Inui, and Koji Murakami. 2008. Constructing extended named entity annotated corpora. *IPSJ SIG Notes*, 2008(113):113–120.

Tomoya Iwakura and Seishi Okamoto. 2008. A fast boosting-based learner for feature-rich tagging and chunking. In *Proc. of CoNLL'08*, pages 17–24.

Tomoya Iwakura, Hiroya Takamura, and Manabu Okumura. 2011. A named entity recognition method based on decomposition and concatenation of word chunks. In *Proc. of IJCNLP'11*, pages 828–836.

Taku Kudo and Yuji Matsumoto. 2001. Chunking with Support Vector Machines. In *Proc. of NAACL'01*, pages 192–199.

Taku Kudo, Jun Suzuki, and Hideki Isozaki. 2005. Boosting-based parse reranking with subtree features. In *Proc. of ACL'05*, pages 189–196.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML'01*, pages 282–289.

Avihai Mejer and Koby Crammer. 2010. Confidence in structured-prediction using confidence-weighted models. In *Proc. of EMNLP'10*, pages 971–981.

Masaaki Nagata. 1994. A stochastic japanese morphological analyzer using a forward-dp backward-a* n-best search algorithm. In *Proc. of COLING'94*, pages 201–207.

Daisuke Okanohara, Yusuke Miyao, Yoshimasa Tsuruoka, and Jun'ichi Tsujii. 2006. Improving the scalability of semi-markov conditional random fields for named entity recognition. In *Proc. of ACL'06*, pages 465–472.

Sunita Sarawagi and William W. Cohen. 2005. Semi-markov conditional random fields for information extraction. In *Proc. of NIPS'04*, pages 1185–1192.

Sunita Sarawagi. 2006. Efficient inference on sequence segmentation models. In *Proc. of ICML'06*, pages 793–800.

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

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

Satoshi Sekine, Kiyoshi Sudo, and Chikashi Nobata. 2002. Extended named entity hierarchy. In *Proc. of LREC'02*.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proc. of NAACL HLT'03*, pages 134–141.

Xu Sun, Takuya Matsuzaki, Daisuke Okanohara, and Jun'ichi Tsujii. 2009. Latent variable perceptron algorithm for structured classification. In *Proc. of IJCAI'09*, pages 1236–1242.

Jun Suzuki and Hideki Isozaki. 2008. Semi-supervised sequential labeling and segmentation using giga-word scale unlabeled data. In *Proc. of ACL-08: HLT*, pages 665–673.

Yoshimasa Tsuruoka and Junichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proc. of HLT/EMNLP*, pages 467–474.

Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. 2009. Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proc. of ACL/IJCNLP*, pages 477–485.

Jialei Wang, Peilin Zhao, and Steven C.H. Hoi. 2012. Exact soft confidence-weighted learning. In *Proc. of ICML'12*, pages 121–128.

Tong Zhang, Fred Damerau, and David Johnson. 2001. Text chunking using regularized winnow. In *Proc. of ACL'01*, pages 539–546.