A New Perceptron Algorithm for Sequence Labeling with Non-local Features

Jun'ichi Kazama and Kentaro Torisawa
Japan Advanced Institute of Science and Technology (JAIST)
Asahidai 1-1, Nomi, Ishikawa, 923-1292 Japan
{kazama, torisawa}@jaist.ac.jp

Abstract

We cannot use non-local features with current major methods of sequence labeling such as CRFs due to concerns about complexity. We propose a new perceptron algorithm that can use non-local features. Our algorithm allows the use of all types of non-local features whose values are determined from the sequence and the labels. The weights of local and non-local features are learned together in the training process with guaranteed convergence. We present experimental results from the CoNLL 2003 named entity recognition (NER) task to demonstrate the performance of the proposed algorithm.

1 Introduction

Many NLP tasks such as POS tagging and named entity recognition have recently been solved as sequence labeling. Discriminative methods such as Conditional Random Fields (CRFs) (Lafferty et al., 2001), Semi-Markov Random Fields (Sarawagi and Cohen, 2004), and perceptrons (Collins, 2002a) have been popular approaches for sequence labeling because of their excellent performance, which is mainly due to their ability to incorporate many kinds of overlapping and non-independent features.

However, the common limitation of these methods is that the features are limited to "local" features, which only depend on a very small number of labels (usually two: the previous and the current). Although this limitation makes training and inference tractable, it also excludes the use of possibly useful "non-local" features that are accessible after all labels are determined. For example, non-local features such as "same phrases in a document do not have different entity classes" were shown to be useful in named entity recognition (Sutton and McCallum, 2004; Bunescu and Mooney, 2004; Finkel et al., 2005; Krishnan and Manning, 2006).

We propose a new perceptron algorithm in this paper that can use non-local features along with local features. Although several methods have already been proposed to incorporate non-local features (Sutton and McCallum, 2004; Bunescu and Mooney, 2004; Finkel et al., 2005; Roth and Yih, 2005; Krishnan and Manning, 2006; Nakagawa and Matsumoto, 2006), these present a problem that the types of non-local features are somewhat constrained. For example, Finkel et al. (2005) enabled the use of non-local features by using Gibbs sampling. However, it is unclear how to apply their method of determining the parameters of a non-local model to other types of non-local features, which they did not used. Roth and Yih (2005) enabled the use of hard constraints on labels by using integer linear programming. However, this is equivalent to only allowing non-local features whose weights are fixed to negative infinity. Krishnan and Manning (2006) divided the model into two CRFs, where the second model uses the output of the first as a kind of non-local information. However, it is not possible to use non-local features that depend on the labels of the very candidate to be scored. Nakagawa and Matsumoto (2006) used a Bolzmann distribution to model the correlation of the POS of words having the same lexical form in a document. However, their method can only be applied when there are convenient links such as the same lexical form.

Since non-local features have not yet been extensively investigated, it is possible for us to find new useful non-local features. Therefore, our objective in this study was to establish a framework, where all

Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 315–324, Prague, June 2007. ©2007 Association for Computational Linguistics

types of non-local features are allowed.

With non-local features, we cannot use efficient procedures such as forward-backward procedures and the Viterbi algorithm that are required in training CRFs (Lafferty et al., 2001) and perceptrons (Collins, 2002a). Recently, several methods (Collins and Roark, 2004; Daumé III and Marcu, 2005; McDonald and Pereira, 2006) have been proposed with similar motivation to ours. These methods alleviate this problem by using some approximation in perceptron-type learning.

In this paper, we follow this line of research and try to solve the problem by extending Collins' perceptron algorithm (Collins, 2002a). We exploited the not-so-familiar fact that we can design a perceptron algorithm with guaranteed convergence if we can find at least one wrong labeling candidate even if we cannot perform exact inference. We first ran the A* search only using local features to generate n-best candidates (this can be efficiently performed), and then we only calculated the true score with non-local features for these candidates to find a wrong labeling candidate. The second key idea was to update the weights of local features during training if this was necessary to generate sufficiently good candidates. The proposed algorithm combined these ideas to achieve guaranteed convergence and effective learning with non-local features.

The remainder of the paper is organized as follows. Section 2 introduces the Collins' perceptron algorithm. Although this algorithm is the starting point for our algorithm, its baseline performance is not outstanding. Therefore, we present a margin extension to the Collins' perceptron in Section 3. This margin perceptron became the direct basis of our algorithm. We then explain our algorithm for non-local features in Section 4. We report the experimental results using the CoNLL 2003 shared task dataset in Section 6.

2 Perceptron Algorithm for Sequence Labeling

Collins (2002a) proposed an extension of the perceptron algorithm (Rosenblatt, 1958) to sequence labeling. Our aim in sequence labeling is to assign label $y_i \in \mathcal{Y}$ to each word $x_i \in \mathcal{X}$ in a sequence. We denote sequence $x_1, \ldots, x_T$ as $\boldsymbol{x}$ and the corresponding labels as $\boldsymbol{y}$. We assume weight vector $\boldsymbol{\alpha} \in R^d$ and feature mapping Φ that maps each $(\boldsymbol{x}, \boldsymbol{y})$ to feature vector $\Phi(\boldsymbol{x}, \boldsymbol{y}) = (\Phi_1(\boldsymbol{x}, \boldsymbol{y}), \cdots, \Phi_d(\boldsymbol{x}, \boldsymbol{y})) \in R^d$. The model determines the labels by:

$$\boldsymbol{y}' = \operatorname{argmax}_{\boldsymbol{y} \in \mathcal{Y}^{|\boldsymbol{x}|}} \Phi(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{\alpha},$$

where $\cdot$ denotes the inner product. The aim of the learning algorithm is to obtain an appropriate weight vector, $\boldsymbol{\alpha}$, given training set $\{(\boldsymbol{x}_1, \boldsymbol{y}_1^*), \cdots, (\boldsymbol{x}_L, \boldsymbol{y}_L^*)\}$.

The learning algorithm, which is illustrated in Collins (2002a), proceeds as follows. The weight vector is initialized to zero. The algorithm passes over the training examples, and each sequence is decoded using the current weights. If $\boldsymbol{y}'$ is not the correct answer $\boldsymbol{y}^*$, the weights are updated according to the following rule.

$$\boldsymbol{\alpha}_{new} = \boldsymbol{\alpha} + \Phi(\boldsymbol{x}, \boldsymbol{y}^*) - \Phi(\boldsymbol{x}, \boldsymbol{y}').$$

This algorithm is proved to converge (i.e., there are no more updates) in the separable case (Collins, 2002a).[1] That is, if there exist weight vector $\boldsymbol{U}$ (with $||\boldsymbol{U}|| = 1$), δ (> 0), and R (> 0) that satisfy:

$$\forall i, \forall \boldsymbol{y} \in \mathcal{Y}^{|\boldsymbol{x}_i|} \quad \Phi(\boldsymbol{x_i}, \boldsymbol{y_i}^*) \cdot \boldsymbol{U} - \Phi(\boldsymbol{x_i}, \boldsymbol{y}) \cdot \boldsymbol{U} \geq \delta,$$
$$\forall i, \forall \boldsymbol{y} \in \mathcal{Y}^{|\boldsymbol{x}_i|} \quad ||\Phi(\boldsymbol{x_i}, \boldsymbol{y_i}^*) - \Phi(\boldsymbol{x_i}, \boldsymbol{y})|| \leq R,$$

the number of updates is at most R^2/δ^2.

The perceptron algorithm only requires one candidate $\boldsymbol{y}'$ for each sequence $\boldsymbol{x_i}$, unlike the training of CRFs where all possible candidates need to be considered. This inherent property is the key to training with non-local features. However, note that the tractability of learning and inference relies on how efficiently $\boldsymbol{y}'$ can be found. In practice, we can find $\boldsymbol{y}'$ efficiently using a Viterbi-type algorithm only when the features are all local, i.e., $\Phi_s(\boldsymbol{x}, \boldsymbol{y})$ can be written as the sum of (two label) local features ϕ_s as $\Phi_s(\boldsymbol{x}, \boldsymbol{y}) = \sum_i^T \phi_s(\boldsymbol{x}, y_{i-1}, y_i)$. This locality constraint is also required to make the training of CRFs tractable (Lafferty et al., 2001).

One problem with the perceptron algorithm described so far is that it offers no treatment for overfitting. Thus, Collins (2002a) also proposed an averaged perceptron, where the final weight vector is

[1]Collins (2002a) also provided proof that guaranteed "good" learning for the non-separable case. However, we have only considered the separable case throughout the paper.

Algorithm 3.1: Perceptron with margin for sequence labeling (parameters: C)

```
$\boldsymbol{\alpha} \leftarrow \mathbf{0}$
until no more updates do
  for $i \leftarrow 1$ to $L$ do
    $\boldsymbol{y}' = \mathrm{argmax}_{\boldsymbol{y}} \Phi(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$
    $\boldsymbol{y}'' = \text{2nd-best}_{\boldsymbol{y}} \Phi(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$
    if $\boldsymbol{y}' \neq \boldsymbol{y}_i^*$ then
      $\boldsymbol{\alpha} = \boldsymbol{\alpha} + \Phi(\boldsymbol{x}_i, \boldsymbol{y}_i^*) - \Phi(\boldsymbol{x}_i, \boldsymbol{y}')$
    else if $\Phi(\boldsymbol{x}_i, \boldsymbol{y}_i^*) \cdot \boldsymbol{\alpha} - \Phi(\boldsymbol{x}_i, \boldsymbol{y}'') \cdot \boldsymbol{\alpha} \leq C$ then
      $\boldsymbol{\alpha} = \boldsymbol{\alpha} + \Phi(\boldsymbol{x}_i, \boldsymbol{y}_i^*) - \Phi(\boldsymbol{x}_i, \boldsymbol{y}'')$
```

the average of all weight vectors during training. However, we found in our experiments that the averaged perceptron performed poorly in our setting. We therefore tried to make the perceptron algorithm more robust to overfitting. We will describe our extension to the perceptron algorithm in the next section.

3 Margin Perceptron Algorithm for Sequence Labeling

We extended a perceptron with a margin (Krauth and Mézard, 1987) to sequence labeling in this study, as Collins (2002a) extended the perceptron algorithm to sequence labeling.

In the case of sequence labeling, the margin is defined as:

$$\gamma(\boldsymbol{\alpha}) = \min_{\boldsymbol{x}_i} \min_{\boldsymbol{y} \neq \boldsymbol{y}_i^*} \frac{\Phi(\boldsymbol{x_i}, \boldsymbol{y_i}^*) \cdot \boldsymbol{\alpha} - \Phi(\boldsymbol{x_i}, \boldsymbol{y}) \cdot \boldsymbol{\alpha}}{||\boldsymbol{\alpha}||}$$

Assuming that the best candidate, $\boldsymbol{y}'$, equals the correct answer, $\boldsymbol{y}^*$, the margin can be re-written as:

$$= \min_{\boldsymbol{x}_i} \frac{\Phi(\boldsymbol{x_i}, \boldsymbol{y_i}^*) \cdot \boldsymbol{\alpha} - \Phi(\boldsymbol{x_i}, \boldsymbol{y''}) \cdot \boldsymbol{\alpha}}{||\boldsymbol{\alpha}||},$$

where $\boldsymbol{y}'' = \text{2nd-best}_{\boldsymbol{y}} \Phi(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$. Using this relation, the resulting algorithm becomes Algorithm 3.1. The algorithm tries to enlarge the margin as much as possible, as well as make the best scoring candidate equal the correct answer.

Constant C in Algorithm 3.1 is a tunable parameter, which controls the trade-off between the margin and convergence time. Based on the proofs in Collins (2002a) and Li et al. (2002), we can prove that the algorithm converges within $(2C + R^2)/\delta^2$ updates and that $\gamma(\boldsymbol{\alpha}) \geq \delta C/(2C + R^2) = (\delta/2)(1 - (R^2/(2C + R^2)))$ after training. As can be seen, the margin approaches at least half of true margin δ (at the cost of infinite training time), as $C \to \infty$.

Note that if the features are all local, the second-best candidate (generally n-best candidates) can also be found efficiently by using an A* search that uses the best scores calculated during a Viterbi search as the heuristic estimation (Soong and Huang, 1991).

There are other methods for improving robustness by making margin larger for the structural output problem. Such methods include ALMA (Gentile, 2001) used in (Daumé III and Marcu, 2005)[2], MIRA (Crammer et al., 2006) used in (McDonald et al., 2005), and Max-Margin Markov Networks (Taskar et al., 2003). However, to the best of our knowledge, there has been no prior work that has applied a perceptron with a margin (Krauth and Mézard, 1987) to structured output.[3] Our method described in this section is one of the easiest to implement, while guaranteeing a large margin. We found in the experiments that our method outperformed the Collins' averaged perceptron by a large margin.

4 Algorithm

4.1 Definition and Basic Idea

Having described the basic perceptron algorithms, we will know explain our algorithm that learns the weights of local and non-local features in a unified way.

Assume that we have local features and non-local features. We use the superscript, l, for local features as $\Phi_i^l(\boldsymbol{x}, \boldsymbol{y})$ and g for non-local features as $\Phi_i^g(\boldsymbol{x}, \boldsymbol{y})$. Then, feature mapping is written as $\Phi^a(\boldsymbol{x}, \boldsymbol{y}) = \Phi^l(\boldsymbol{x}, \boldsymbol{y}) + \Phi^g(\boldsymbol{x}, \boldsymbol{y}) = (\Phi_1^l(\boldsymbol{x}, \boldsymbol{y}), \cdots, \Phi_n^l(\boldsymbol{x}, \boldsymbol{y}), \Phi_{n+1}^g(\boldsymbol{x}, \boldsymbol{y}), \cdots, \Phi_d^g(\boldsymbol{x}, \boldsymbol{y}))$. Here, we define:

$$\begin{aligned} \Phi^l(\boldsymbol{x}, \boldsymbol{y}) &= (\Phi_1^l(\boldsymbol{x}, \boldsymbol{y}), \cdots, \Phi_n^l(\boldsymbol{x}, \boldsymbol{y}), 0, \cdots, 0) \\ \Phi^g(\boldsymbol{x}, \boldsymbol{y}) &= (0, \cdots, 0, \Phi_{n+1}^g(\boldsymbol{x}, \boldsymbol{y}), \cdots, \Phi_d^g(\boldsymbol{x}, \boldsymbol{y})) \end{aligned}$$

Ideally, we want to determine the labels using the whole feature set as:

$$\boldsymbol{y}' = \mathrm{argmax}_{\boldsymbol{y} \in \mathcal{Y}^{|\boldsymbol{x}|}} \Phi^a(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{\alpha}.$$

[2](Daumé III and Marcu, 2005) also presents the method using the averaged perceptron (Collins, 2002a)

[3]For re-ranking problems, Shen and Joshi (2004) proposed a perceptron algorithm that also uses margins. The difference is that our algorithm trains the sequence labeler itself and is much simpler because it only aims at labeling.

Algorithm 4.1: Candidate algorithm (parameters: n, C)

```
$\boldsymbol{\alpha} \leftarrow \mathbf{0}$
until no more updates do
  for $i \leftarrow 1$ to $L$ do
    $\{\boldsymbol{y}^n\} = \text{n-best}_{\boldsymbol{y}} \Phi^l(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$
    $\boldsymbol{y}' = \text{argmax}_{\boldsymbol{y} \in \{\boldsymbol{y}^n\}} \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$
    $\boldsymbol{y}'' = \text{2nd-best}_{\boldsymbol{y} \in \{\boldsymbol{y}^n\}} \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$
    if $\boldsymbol{y}' \neq \boldsymbol{y_i}^*$
        & $\Phi^a(\boldsymbol{x}_i, \boldsymbol{y}_i^*) \cdot \boldsymbol{\alpha} - \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}') \cdot \boldsymbol{\alpha} \leq C$ then
      $\boldsymbol{\alpha} = \boldsymbol{\alpha} + \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}_i^*) - \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}')$
    else if $\Phi^a(\boldsymbol{x}_i, \boldsymbol{y}_i^*) \cdot \boldsymbol{\alpha} - \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}'') \cdot \boldsymbol{\alpha} \leq C$ then
      $\boldsymbol{\alpha} = \boldsymbol{\alpha} + \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}_i^*) - \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}'')$
```

However, if there are non-local features, it is impossible to find the highest scoring candidate efficiently, since we cannot use the Viterbi algorithm. Thus, we cannot use the perceptron algorithms described in the previous sections. The training of CRFs is also intractable for the same reason.

To deal with this problem, we first relaxed our objective. The modified objective was to find a good model from those with the form:

$$\begin{aligned} \{\boldsymbol{y}^n\} &= \text{n-best}_{\boldsymbol{y}} \Phi^l(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{\alpha} \\ \boldsymbol{y}' &= \text{argmax}_{\boldsymbol{y} \in \{\boldsymbol{y}^n\}} \Phi^a(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{\alpha}, \end{aligned} \quad (1)$$

That is, we first generate n-best candidates $\{\boldsymbol{y}^n\}$ under the local model, $\Phi^l(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$. This can be done efficiently using the A* algorithm. We then find the best scoring candidate under the total model, $\Phi^a(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$, only from these n-best candidates. If n is moderately small, this can also be done in a practical amount of time.

This resembles the re-ranking approach (Collins and Duffy, 2002; Collins, 2002b). However, unlike the re-ranking approach, the local model, $\Phi^l(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$, and the total model, $\Phi^a(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$, correlate since they share a part of the vector and are trained at the same time in our algorithm. The re-ranking approach has the disadvantage that it is necessary to use different training corpora for the first model and for the second, or to use cross validation type training, to make the training for the second meaningful. This reduces the effective size of training data or increases training time substantially. On the other hand, our algorithm has no such disadvantage.

However, we are no longer able to find the highest scoring candidate under $\Phi^a(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$ exactly with this approach. We cannot thus use the perceptron algorithms directly. However, by examining the

Algorithm 4.2: Perceptron with local and non-local features (parameters: n, C^a, C^l)

```
$\boldsymbol{\alpha} \leftarrow \mathbf{0}$
until no more updates do
  for $i \leftarrow 1$ to $L$ do
    $\{\boldsymbol{y}^n\} = \text{n-best}_{\boldsymbol{y}} \Phi^l(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$
    $\boldsymbol{y}' = \text{argmax}_{\boldsymbol{y} \in \{\boldsymbol{y}^n\}} \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$
    $\boldsymbol{y}'' = \text{2nd-best}_{\boldsymbol{y} \in \{\boldsymbol{y}^n\}} \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$
    if $\boldsymbol{y}' \neq \boldsymbol{y}_i^*$
        & $\Phi^a(\boldsymbol{x}_i, \boldsymbol{y}_i^*) \cdot \boldsymbol{\alpha} - \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}') \cdot \boldsymbol{\alpha} \leq C^a$ then
      $\boldsymbol{\alpha} = \boldsymbol{\alpha} + \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}_i^*) - \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}')$          (A)
    else if $\Phi^a(\boldsymbol{x}_i, \boldsymbol{y}_i^*) \cdot \boldsymbol{\alpha} - \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}'') \cdot \boldsymbol{\alpha} \leq C^a$ then
      $\boldsymbol{\alpha} = \boldsymbol{\alpha} + \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}_i^*) - \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}'')$          (A)
    else
  (B) if $\boldsymbol{y}^1 \neq \boldsymbol{y_i}^*$ then ($\boldsymbol{y}^1$ represents the best in $\{\boldsymbol{y}^n\}$)
        $\boldsymbol{\alpha} = \boldsymbol{\alpha} + \Phi^l(\boldsymbol{x}_i, \boldsymbol{y}_i^*) - \Phi^l(\boldsymbol{x}_i, \boldsymbol{y}^1)$
      else if $\Phi^l(\boldsymbol{x}_i, \boldsymbol{y}_i^*) \cdot \boldsymbol{\alpha} - \Phi^l(\boldsymbol{x}_i, \boldsymbol{y}^2) \cdot \boldsymbol{\alpha} \leq C^l$ then
        $\boldsymbol{\alpha} = \boldsymbol{\alpha} + \Phi^l(\boldsymbol{x}_i, \boldsymbol{y}_i^*) - \Phi^l(\boldsymbol{x}_i, \boldsymbol{y}^2)$
```

proofs in Collins (2002a), we can see that the essential condition for convergence is that the weights are always updated using some $\boldsymbol{y}$ ($\neq \boldsymbol{y}^*$) that satisfies:

$$\Phi(\boldsymbol{x}_i, \boldsymbol{y}_i^*) \cdot \boldsymbol{\alpha} - \Phi(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \boldsymbol{\alpha} \leq 0$$
$$(\leq C \text{ in the case of a perceptron with a margin}). \quad (2)$$

That is, $\boldsymbol{y}$ does not necessarily need to be the exact best candidate or the exact second-best candidate. The algorithm also converges in a finite number of iterations even with Eq. (1) as long as Eq. (2) is satisfied.

4.2 Candidate Algorithm

The algorithm we came up with first based on the above idea, is Algorithm 4.1. We first find the n-best candidates using the local model, $\Phi^l(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$. At this point, we can determine the value of the non-local features, $\Phi^g(\boldsymbol{x}, \boldsymbol{y})$, to form the whole feature vector, $\Phi^a(\boldsymbol{x}, \boldsymbol{y})$, for the n-best candidates. Next, we re-score and sort them using the total model, $\Phi^a(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$, to find a candidate that violates the margin condition. We call this algorithm the "candidate algorithm". After the training has finished, $\Phi^a(\boldsymbol{x}_i, \boldsymbol{y}_i^*) \cdot \boldsymbol{\alpha} - \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \boldsymbol{\alpha} > C$ is guaranteed for all $(\boldsymbol{x}_i, \boldsymbol{y})$ where $\boldsymbol{y} \in \{\boldsymbol{y}^n\}, \boldsymbol{y} \neq \boldsymbol{y}^*$. At first glance, this seems sufficient condition for good models. However, this is not true because if $\boldsymbol{y}^* \notin \{\boldsymbol{y}^n\}$, the inference defined by Eq. (1) is not guaranteed to find the correct answer, $\boldsymbol{y}^*$. In fact, this algorithm does not work well with non-local features as we found in the experiments.

4.3 Final Algorithm

Our idea for improving the above algorithm is that the local model, $\Phi^l(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{\alpha}$, must at least be so good that $\boldsymbol{y}^* \in \{\boldsymbol{y}^n\}$. To achieve this, we added a modification term that was intended to improve the local model when the local model was not good enough even when the total model was good enough.

The final algorithm resulted in Algorithm 4.2. As can be seen, the part marked (B) has been added. We call this algorithm the "proposed algorithm". Note that the algorithm prioritizes the update of the total model, (A), over that of the local model, (B), although the opposite is also possible. Also note that the update of the local model in (B) is "aggressive" since it updates the weights until the best candidate output by the local model becomes the correct answer and satisfies the margin condition. A "conservative" updating, where we cease the update when the n-best candidates contain the correct answer, is also possible from our idea above. We made these choices since they worked better than the other alternatives.

The tunable parameters are the local margin parameter, C^l, the total margin parameter, C^a, and n for the n-best search. We used $C = C^l = C^a$ in this study to reduce the search space.

We can prove that the algorithm in Algorithm 4.2 also converges in a finite number of iterations. It converges within $(2C + R^2)/\delta^2$ updates, assuming that there exist weight vector $\boldsymbol{U}^l$ (with $||\boldsymbol{U}^l|| = 1$ and $U_i^l = 0 \ \ (n+1 \leq i \leq d)$), δ (> 0), and R (> 0) that satisfy:

$$\forall i, \forall \boldsymbol{y} \in \mathcal{Y}^{|\boldsymbol{x}_i|} \ \ \Phi^l(\boldsymbol{x_i}, \boldsymbol{y_i}^*) \cdot \boldsymbol{U}^l - \Phi^l(\boldsymbol{x_i}, \boldsymbol{y}) \cdot \boldsymbol{U}^l \geq \delta,$$

$$\forall i, \forall \boldsymbol{y} \in \mathcal{Y}^{|\boldsymbol{x}_i|} \ \ ||\Phi^a(\boldsymbol{x_i}, \boldsymbol{y_i}^*) - \Phi^a(\boldsymbol{x_i}, \boldsymbol{y})|| \leq R.$$

In addition, we can prove that $\gamma'(\boldsymbol{\alpha}) \geq \delta C/(2C + R^2)$ for the margin after convergence, where $\gamma'(\boldsymbol{\alpha})$ is defined as:

$$\min_{\boldsymbol{x}_i} \min_{\boldsymbol{y} \in \{\boldsymbol{y}^n\}, \neq \boldsymbol{y}_i^*} \frac{\Phi^a(\boldsymbol{x_i}, \boldsymbol{y_i}^*) \cdot \boldsymbol{\alpha} - \Phi^a(\boldsymbol{x_i}, \boldsymbol{y}) \cdot \boldsymbol{\alpha}}{||\boldsymbol{\alpha}||}$$

See Appendix A for the proofs.

We also incorporated the idea behind Bayes point machines (BPMs) (Herbrich and Graepel, 2000) to improve the robustness of our method further. BPMs try to cancel out overfitting caused by the order of examples, by training several models by shuffling the training examples.[4] However, it is very time consuming to run the complete training process several times. We thus ran the training in only one pass over the shuffled examples several times, and used the averaged output weight vectors as a new initial weight vector, because we thought that the early part of training would be more seriously affected by the order of examples. We call this "BPM initialization". [5]

5 Named Entity Recognition and Non-Local Features

We evaluated the performance of the proposed algorithm using the named entity recognition task. We adopted IOB (IOB2) labeling (Ramshaw and Marcus, 1995), where the first word of an entity of class "C" is labeled "B-C", the words in the entity are labeled "I-C", and other words are labeled "O".

We used non-local features based on Finkel et al. (2005). These features are based on observations such as "same phrases in a document tend to have the same entity class" (phrase consistency) and "a sub-phrase of a phrase tends to have the same entity class as the phrase" (sub-phrase consistency). We also implemented the "majority" version of these features as used in Krishnan and Manning (2006). In addition, we used non-local features, which are based on the observation that "entities tend to have the same entity class if they are in the same conjunctive or disjunctive expression" as in "$\cdots$ in U.S., EU, and Japan" (conjunction consistency). This type of non-local feature was not used by Finkel et al. (2005) or Krishnan and Manning (2006).

6 Experiments

6.1 Data and Setting

We used the English dataset of the CoNLL 2003 named entity shared task (Tjong et al., 2003) for the experiments. It is a corpus of English newspaper articles, where four entity classes, PER, LOC, ORG, and MISC are annotated. It consists of training, development, and testing sets (14,987, 3,466,

[4]The results for the perceptron algorithms generally depend on the order of the training examples.

[5]Note that we can prove that the perceptron algorithms converge even though the weight vector is not initialized as $\boldsymbol{\alpha} = \boldsymbol{0}$.

and 3,684 sentences, respectively). Automatically assigned POS tags and chunk tags are also provided. The CoNLL 2003 dataset contains document boundary markers. We concatenated the sentences in the same document according to these markers.[6] This generated 964 documents for the training set, 216 documents for the development set, and 231 documents for the testing set. The documents generated as above become the sequence, $\boldsymbol{x}$, in the learning algorithms.

We first evaluated the baseline performance of a CRF model, the Collins' perceptron, and the Collins' averaged perceptron, as well as the margin perceptron, with only local features. We next evaluated the performance of our perceptron algorithm proposed for non-local features.

We used the local features summarized in Table 1, which are similar to those used in other studies on named entity recognition. We omitted features whose surface part listed in Table 1 occurred less than twice in the training corpus.

We used CRF++ (ver. 0.44)[7] as the basis of our implementation. We implemented scaling, which is similar to that for HMMs (see such as (Rabiner, 1989)), in the forward-backward phase of CRF training to deal with very long sequences due to sentence concatenation.[8]

We used Gaussian regularization (Chen and Rosenfeld, 2000) for CRF training to avoid overfitting. The parameter of the Gaussian, σ^2, was tuned using the development set. We also tuned the margin parameter, C, for the margin perceptron algorithm.[9] The convergence of CRF training was determined by checking the log-likelihood of the model. The convergence of perceptron algorithms was determined by checking the per-word labeling error, since the

Table 1: Local features used. The value of a node feature is determined from the current label, y_0, and a surface feature determined only from $\boldsymbol{x}$. The value of an edge feature is determined by the previous label, y_{-1}, the current label, y_0, and a surface feature. Used surface features are the word (w), the downcased word (wl), the POS tag (pos), the chunk tag (chk), the prefix of the word of length n (pn), the suffix (sn), the word form features: 2d - cp (these are based on (Bikel et al., 1999)), and the gazetteer features: go for ORG, gp for PER, and gm for MISC. These represent the (longest) match with an entry in the gazetteer by using IOB2 tags.

Node features: $\{\text{""}, \mathrm{x}_{-2}, \mathrm{x}_{-1}, \mathrm{x}_0, \mathrm{x}_{+1}, \mathrm{x}_{+2}\} \times y_0$
x =, w, wl, pos, chk, p1, p2, p3, p4, s1, s2, s3, s4, 2d, 4d, d&a, d&-, d&/, d&,, d&., n, ic, ac, l, cp, go, gp, gm
Edge features: $\{\text{""}, \mathrm{x}_{-2}, \mathrm{x}_{-1}, \mathrm{x}_0, \mathrm{x}_{+1}, \mathrm{x}_{+2}\} \times y_{-1} \times y_0$
x =, w, wl, pos, chk, p1, p2, p3, p4, s1, s2, s3, s4, 2d, 4d, d&a, d&-, d&/, d&,, d&., n, ic, ac, l, cp, go, gp, gm
Bigram node features: $\{\mathrm{x}_{-2}\mathrm{x}_{-1}, \mathrm{x}_{-1}\mathrm{x}_0, \mathrm{x}_0\mathrm{x}_{+1}\} \times y_0$
x = wl, pos, chk, go, gp, gm
Bigram edge features: $\{\mathrm{x}_{-2}\mathrm{x}_{-1}, \mathrm{x}_{-1}\mathrm{x}_0, \mathrm{x}_0\mathrm{x}_{+1}\} \times y_{-1} \times y_0$
x = wl, pos, chk, go, gp, gm

number of updates was not zero even after a large number of iterations in practice. We stopped training when the relative change in these values became less than a pre-defined threshold (0.0001) for at least three iterations.

We used $n = 20$ (n of the n-best) for training since we could not use too a large n because it would have slowed down training. However, we could examine a larger n during testing, since the testing time did not dominate the time for the experiment. We found an interesting property for n in our preliminary experiment. We found that an even larger n in testing (written as n') achieved higher accuracy, although it is natural to assume that the same n that was used in training would also be appropriate for testing. We thus used $n' = 100$ to evaluate performance during parameter tuning. After finding the best C with $n' = 100$, we varied n' to investigate its

[6]We used sentence concatenation even when only using local features, since we found it does not degrade accuracy (rather we observed a slight increase).

[7]http://chasen.org/˜taku/software/CRF++

[8]We also replaced the optimization module in the original package with that used in the Amis maximum entropy estimator (http://www-tsujii.is.s.u-tokyo.ac.jp/amis) since we encountered problems with the provided module in some cases.

[9]For the Gaussian parameter, we tested {13, 25, 50, 100, 200, 400, 800} (the accuracy did not change drastically among these values and it seems that there is no accuracy hump even if we use smaller values). We tested {500, 1000, 1414, 2000, 2828, 4000, 5657, 8000, 11313, 16000, 32000} for the margin parameters.

Table 2: Summary of performance (F_1).

Method	dev	test	C (or σ^2)
local features			
CRF	**91.10**	**86.26**	100
Perceptron	89.01	84.03	-
Averaged perceptron	89.32	84.08	-
Margin perceptron	**90.98**	**85.64**	11313
+ non-local features			
Candidate ($n' = 100$)	90.71	84.90	4000
Proposed ($n' = 100$)	**91.95**	**86.30**	5657

Table 3: Effect of n'.

Method	dev	test	C
Proposed ($n' = 20$)	91.76	86.19	5657
Proposed ($n' = 100$)	91.95	86.30	5657
Proposed ($n' = 400$)	92.13	86.39	5657
Proposed ($n' = 800$)	92.09	86.39	5657
Proposed ($n' = 1600$)	92.13	**86.46**	5657
Proposed ($n' = 6400$)	**92.19**	86.38	5657

effects further.

6.2 Results

Table 2 compares the results. CRF outperformed the perceptron by a large margin. Although the averaged perceptron outperformed the perceptron, the improvement was slight. However, the margin perceptron greatly outperformed compared to the averaged perceptron. Yet, CRF still had the best baseline performance with only local features.

The proposed algorithm with non-local features improved the performance on the test set by 0.66 points over that of the margin perceptron without non-local features. The row "Candidate" refers to the candidate algorithm (Algorithm 4.1). From the results for the candidate algorithm, we can see that the modification part, (B), in Algorithm 4.2 was essential to make learning with non-local features effective.

We next examined the effect of n'. As can be seen from Table 3, an n' larger than that for training yields higher performance. The highest performance with the proposed algorithm was achieved when $n' = 6400$, where the improvement due to non-local features became 0.74 points.

The performance of the related work (Finkel et al., 2005; Krishnan and Manning, 2006) is listed in Table 4. We can see that the final performance of our algorithm was worse than that of the related work.

We changed the experimental setting slightly to investigate our algorithm further. Instead of

Table 4: The performance of the related work.

Method	dev	test
Finkel et al., 2005 (Finkel et al., 2005)		
baseline CRF	-	85.51
+ non-local features	-	86.86
Krishnan and Manning, 2006 (Krishnan and Manning, 2006)		
baseline CRF	-	85.29
+ non-local features	-	87.24

Table 5: Summary of performance with POS/chunk tags by TagChunk.

Method	dev	test	C (or σ^2)
local features			
CRF	**91.39**	**86.30**	200
Perceptron	89.36	84.35	-
Averaged perceptron	89.76	84.50	-
Margin perceptron	**91.06**	**86.24**	32000
+ non-local features			
Proposed ($n' = 100$)	92.23	87.04	5657
Proposed ($n' = 6400$)	**92.54**	**87.17**	5657

the POS/chunk tags provided in the CoNLL 2003 dataset, we used the tags assigned by TagChunk (Daumé III and Marcu, 2005)[10] with the intention of using more accurate tags. The results with this setting are summarized in Table 5. Performance was better than that in the previous experiment for all algorithms. We think this was due to the quality of the POS/chunk tags. It is interesting that the effect of non-local features rose to 0.93 points with $n' = 6400$, even though the baseline performance was also improved. The resulting performance of the proposed algorithm with non-local features is higher than that of Finkel et al. (2005) and comparable with that of Krishnan and Manning (2006). This comparison, of course, is not fair because the setting was different. However, we think the results demonstrate a potential of our new algorithm.

The effect of BPM initialization was also examined. The number of BPM runs was 10 in this experiment. The performance of the proposed algorithm dropped from 91.95/86.30 to 91.89/86.03 without BPM initialization as expected in the setting of the experiment of Table 2. The performance of the margin perceptron, on the other hand, changed from 90.98/85.64 to 90.98/85.90 without BPM initialization. This result was unexpected from the result of our preliminary experiment. However, the performance was changed from 91.06/86.24 to

[10]http://www.cs.utah.edu/~hal/TagChunk/

Table 6: Comparison with re-ranking approach.

Method	dev	test	C
local features			
Margin Perceptron	91.06	86.24	32000
+ non-local features			
Re-ranking 1 ($n' = 100$)	91.62	86.57	4000
Re-ranking 1 ($n' = 80$)	**91.71**	**86.58**	4000
Re-ranking 2 ($n' = 100$)	92.08	86.86	16000
Re-ranking 2 ($n' = 800$)	**92.26**	**86.95**	16000
Proposed ($n' = 100$)	92.23	87.04	5657
Proposed ($n' = 6400$)	**92.54**	**87.17**	5657

Table 7: Comparison of training time ($C = 5657$).

Method	dev	test	time (sec.)
local features			
Margin Perceptron	91.04	86.28	15,977
+ non-local features			
Re-ranking 1 ($n' = 100$)	91.48	86.53	86,742
Re-ranking 2 ($n' = 100$)	92.02	86.85	112,138
Proposed ($n' = 100$)	92.23	87.04	28,880

$91.17/86.08$ (i.e., dropped for the evaluation set as expected), in the setting of the experiment of Table 5. Since the effect of BPM initialization is not conclusive only from these results, we need more experiments on this.

6.3 Comparison with re-ranking approach

Finally, we compared our algorithm with the re-ranking approach (Collins and Duffy, 2002; Collins, 2002b), where we first generate the n-best candidates using a model with only local features (the first model) and then re-rank the candidates using a model with non-local features (the second model).

We implemented two re-ranking models, "re-ranking 1" and "re-ranking 2". These models differ in how to incorporate the local information in the second model. "re-ranking 1" uses the score of the first model as a feature in addition to the non-local features as in Collins (2002b). "re-ranking 2" uses the same local features as the first model[11] in addition to the non-local features. The first models were trained using the margin perceptron algorithm in Algorithm 3.1. The second models were trained using the algorithm, which is obtained by replacing $\{\boldsymbol{y}^n\}$ with the n-best candidates by the first model. The first model used to generate n-best candidates for the development set and the test set was trained using the whole training data. However, CRFs or perceptrons generally have nearly zero error on the training data, although the first model should mis-label to some extent to make the training of the second model meaningful. To avoid this problem, we adopt cross-validation training as used in Collins (2002b). We split the training data into 5 sets. We then trained five first models using 4/5 of the data, each of which was used to generate n-best candidates for the remaining 1/5 of the data.

As in the previous experiments, we tuned C using the development set with $n' = 100$ and then tested other values for n'. Table 6 shows the results. As can be seen, re-ranking models were outperformed by our proposed algorithm, although they also outperformed the margin perceptron with only local features ("re-ranking 2" seems better than "re-ranking 1"). Table 7 shows the training time of each algorithm.[12] Our algorithm is much faster than the re-ranking approach that uses cross-validation training, while achieving the same or higher level of performance.

7 Discussion

As we mentioned, there are some algorithms similar to ours (Collins and Roark, 2004; Daumé III and Marcu, 2005; McDonald and Pereira, 2006; Liang et al., 2006). The differences of our algorithm from these algorithms are as follows.

Daumé III and Marcu (2005) presented the method called LaSO (Learning as Search Optimization), in which intractable exact inference is approximated by optimizing the behavior of the search process. The method can access non-local features at each search point, if their values can be determined from the search decisions already made. They provided robust training algorithms with guaranteed convergence for this framework. However, a difference is that our method can use non-local features whose value depends on all labels throughout training, and it is unclear whether the features whose values can only be determined at the end of the search (e.g., majority features) can be learned effectively with such an incremental manner of LaSO.

The algorithm proposed by McDonald and Pereira (2006) is also similar to ours. Their target was non-projective dependency parsing, where exact inference is intractable. Instead of using

[11] The weights were re-trained for the second model.

[12] Training time was measured on a machine with 2.33 GHz QuadCore Intel Xeons and 8 GB of memory. C was fixed to 5657.

n-best/re-scoring approach as ours, their method modifies the single best projective parse, which can be found efficiently, to find a candidate with higher score under non-local features. Liang et al. (2006) used n candidates of a beam search in the Collins' perceptron algorithm for machine translation. Collins and Roark (2004) proposed an approximate incremental method for parsing. Their method can be used for sequence labeling as well. These studies, however, did not explain the validity of their updating methods in terms of convergence.

To achieve robust training, Daumé III and Marcu (2005) employed the averaged perceptron (Collins, 2002a) and ALMA (Gentile, 2001). Collins and Roark (2004) used the averaged perceptron (Collins, 2002a). McDonald and Pereira (2006) used MIRA (Crammer et al., 2006). On the other hand, we employed the margin perceptron (Krauth and Mézard, 1987), extending it to sequence labeling. We demonstrated that this greatly improved robustness.

With regard to the local update, (B), in Algorithm 4.2, "early updates" (Collins and Roark, 2004) and "y-good" requirement in (Daumé III and Marcu, 2005) resemble our local update in that they tried to avoid the situation where the correct answer cannot be output. Considering such commonality, the way of combining the local update and the non-local update might be one important key for further improvement.

It is still open whether these differences are advantages or disadvantages. However, we think our algorithm can be a contribution to the study for incorporating non-local features. The convergence guarantee is important for the confidence in the training results, although it does not mean high performance directly. Our algorithm could at least improve the accuracy of NER with non-local features and it was indicated that our algorithm was superior to the re-ranking approach in terms of accuracy and training cost. However, the achieved accuracy was not better than that of related work (Finkel et al., 2005; Krishnan and Manning, 2006) based on CRFs. Although this might indicate the limitation of perceptron-based methods, it has also been shown that there is still room for improvement in perceptron-based algorithms as our margin perceptron algorithm demonstrated.

8 Conclusion

In this paper, we presented a new perceptron algorithm for learning with non-local features. We think the proposed algorithm is an important step towards achieving our final objective. We would like to investigate various types of new non-local features using the proposed algorithm in future work.

Appendix A: Convergence of Algorithm 4.2

Let $\boldsymbol{\alpha}^k$ be a weight vector before the kth update and ϵ_k be a variable that takes 1 when the kth update is done in (A) and 0 when done in (B). The update rule can then be written as $\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \epsilon_k(\Phi^{a*} - \Phi^a + (1-\epsilon_k)(\Phi^{l*} - \Phi^l)$.[13] First, we obtain

$$\begin{aligned}\boldsymbol{\alpha}^{k+1} \cdot \boldsymbol{U}^l &= \boldsymbol{\alpha}^k \cdot \boldsymbol{U}^l + \epsilon_k(\Phi^{a*} \cdot \boldsymbol{U}^l - \Phi^a \cdot \boldsymbol{U}^l) \\ &\quad +(1-\epsilon_k)(\Phi^{l*} \cdot \boldsymbol{U}^l - \Phi^l \cdot \boldsymbol{U}^l) \\ &\geq \boldsymbol{\alpha}^k \cdot \boldsymbol{U}^l + \epsilon_k\delta + (1-\epsilon_k)\delta \\ &= \boldsymbol{\alpha}^k \cdot \boldsymbol{U}^l + \delta \geq \boldsymbol{\alpha}^1 \cdot \boldsymbol{U}^l + k\delta = k\delta\end{aligned}$$

Therefore, $(k\delta)^2 \leq (\boldsymbol{\alpha}^{k+1} \cdot \boldsymbol{U}^l)^2 \leq (||\boldsymbol{\alpha}^{k+1}||||\boldsymbol{U}^l||)^2 = ||\boldsymbol{\alpha}^{k+1}||^2$ — (1). On the other hand, we also obtain

$$\begin{aligned}||\boldsymbol{\alpha}^{k+1}||^2 &\leq ||\boldsymbol{\alpha}^k||^2 + 2\epsilon_k\boldsymbol{\alpha}^k(\Phi^{a*} - \Phi^a) \\ &\quad +2(1-\epsilon_k)\boldsymbol{\alpha}^k(\Phi^{l*} - \Phi^l) \\ &\quad +\{\epsilon_k(\Phi^{a*} - \Phi^a) + (1-\epsilon_k)(\Phi^{l*} - \Phi^l)\}^2 \\ &\leq ||\boldsymbol{\alpha}^k||^2 + 2C + R^2 \\ &\leq ||\boldsymbol{\alpha}^1||^2 + k(R^2 + 2C) = k(R^2 + 2C) \text{— (2)}\end{aligned}$$

We used $\boldsymbol{\alpha}^k(\Phi^{a*} - \Phi^a) \leq C^a$, $\boldsymbol{\alpha}^k(\Phi^{l*} - \Phi^l) \leq C^l$ and $C^l = C^a = C$ to derive $2C$ in the second inequality. We used $||\Phi^{l*} - \Phi^l|| \leq ||\Phi^{a*} - \Phi^a|| \leq R$ to derive R^2.

Combining (1) and (2), we obtain $k \leq (R^2 + 2C)/\delta^2$. Substituting this into (2) gives $||\boldsymbol{\alpha}^k|| \leq (R^2+2C)/\delta$. Since $\boldsymbol{y}* = \boldsymbol{y}'$ and $\Phi^{a*} \cdot \boldsymbol{\alpha} - \Phi^{a''} \cdot \boldsymbol{\alpha} > C$ after convergence, we obtain

$$\gamma'(\boldsymbol{\alpha}) = \min_{\boldsymbol{x}_i} \frac{\Phi^{a*} \cdot \boldsymbol{\alpha} - \Phi^{a''} \cdot \boldsymbol{\alpha}}{||\boldsymbol{\alpha}||} \geq C\delta/(2C + R^2).$$

[13]We use the shorthand $\Phi^{a*} = \Phi^a(\boldsymbol{x}_i, \boldsymbol{y}_i^*)$, $\Phi^a = \Phi^a(\boldsymbol{x}_i, \boldsymbol{y})$, $\Phi^{l*} = \Phi^l(\boldsymbol{x}_i, \boldsymbol{y}_i^*)$, and $\Phi^l = \Phi^l(\boldsymbol{x}_i, \boldsymbol{y})$ where $\boldsymbol{y}$ represents the candidate used to update ($\boldsymbol{y}'$, $\boldsymbol{y}''$, $\boldsymbol{y}^1$, or $\boldsymbol{y}^2$).

References

D. M. Bikel, R. L. Schwartz, and R. M. Weischedel. 1999. An algorithm that learns what's in a name. *Machine Learning*, 34(1-3):211–231.

R. Bunescu and R. J. Mooney. 2004. Collective information extraction with relational markov networks. In *ACL 2004*.

S. F. Chen and R. Rosenfeld. 2000. A survey of smoothing techniques for ME models. *IEEE Transactions on Speech and Audio Processing*, 8(1):37–50.

M. Collins and N. Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL 2002*, pages 263–270.

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *ACL 2004*.

M. Collins. 2002a. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *EMNLP 2002*.

M. Collins. 2002b. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *ACL 2002*.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*.

H. Daumé III and D. Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *ICML 2005*.

J. R. Finkel, T. Grenager, and C. Manning. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In *ACL 2005*.

C. Gentile. 2001. A new approximate maximal margin classification algorithm. *JMLR*, 3.

R. Herbrich and T. Graepel. 2000. Large scale Bayes point machines. In *NIPS 2000*.

W. Krauth and M. Mézard. 1987. Learning algorithms with optimal stability in neural networks. *Journal of Physics A 20*, pages 745–752.

V. Krishnan and C. D. Manning. 2006. An effective two-stage model for exploiting non-local dependencies in named entity recognitioin. In *ACL-COLING 2006*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML 2001*, pages 282–289.

Y. Li, H. Zaragoza, R. Herbrich, J. Shawe-Taylor, and J. Kandola. 2002. The perceptron algorithm with uneven margins. In *ICML 2002*.

P. Liang, A. Bouchard-Côté, D. Klein, and B. Taskar. 2006. An end-to-end discriminative approach to machine translation. In *ACL-COLING 2006*.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *EACL 2006*.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *ACL 2005*.

T. Nakagawa and Y. Matsumoto. 2006. Guessing parts-of-speech of unknown words using global information. In *ACL-COLING 2006*.

L. R. Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

L. A. Ramshaw and M. P. Marcus. 1995. Text chunking using transformation-based learning. In *third ACL Workshop on very large corpora*.

F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psycological Review*, pages 386–407.

D. Roth and W. Yih. 2005. Integer linear programming inference for conditional random fields. In *ICML 2005*.

S. Sarawagi and W. W. Cohen. 2004. Semi-Markov random fields for information extraction. In *NIPS 2004*.

L. Shen and A. K. Joshi. 2004. Flexible margin selection for reranking with full pairwise samples. In *IJCNLP 2004*.

F. K. Soong and E. Huang. 1991. A tree-trellis based fast search for finding the n best sentence hypotheses in continuous speech recognition. In *ICASSP-91*.

C. Sutton and A. McCallum. 2004. Collective segmenation and labeling of distant entitites in information extraction. University of Massachusetts Rechnical Report TR 04-49.

B. Taskar, C. Guestrin, and D. Koller. 2003. Max-margin Markov networks. In *NIPS 2003*.

E. F. Tjong, K. Sang, and F. De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *CoNLL 2003*.