Chinese Segmentation with a Word-Based Perceptron Algorithm

Yue Zhang and **Stephen Clark**
Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD, UK
{yue.zhang,stephen.clark}@comlab.ox.ac.uk

Abstract

Standard approaches to Chinese word segmentation treat the problem as a tagging task, assigning labels to the characters in the sequence indicating whether the character marks a word boundary. Discriminatively trained models based on local character features are used to make the tagging decisions, with Viterbi decoding finding the highest scoring segmentation. In this paper we propose an alternative, word-based segmentor, which uses features based on complete words and word sequences. The generalized perceptron algorithm is used for discriminative training, and we use a beam-search decoder. Closed tests on the first and second SIGHAN bakeoffs show that our system is competitive with the best in the literature, achieving the highest reported F-scores for a number of corpora.

1 Introduction

Words are the basic units to process for most NLP tasks. The problem of Chinese word segmentation (CWS) is to find these basic units for a given sentence, which is written as a continuous sequence of characters. It is the initial step for most Chinese processing applications.

Chinese character sequences are ambiguous, often requiring knowledge from a variety of sources for disambiguation. Out-of-vocabulary (OOV) words are a major source of ambiguity. For example, a difficult case occurs when an OOV word consists of characters which have themselves been seen as words; here an automatic segmentor may split the OOV word into individual single-character words. Typical examples of unseen words include Chinese names, translated foreign names and idioms.

The segmentation of known words can also be ambiguous. For example, "这里面" should be "这里 (here) 面 (flour)" in the sentence "这里面和米很贵" (flour and rice are expensive here) or "这 (here) 里面 (inside)" in the sentence "这里面很冷" (it's cold inside here). The ambiguity can be resolved with information about the neighboring words. In comparison, for the sentences "洽谈会很成功", possible segmentations include "洽谈 (the discussion) 会 (will) 很 (very) 成功 (be successful)" and "洽谈会 (the discussion meeting) 很 (very) 成功 (be successful)". The ambiguity can only be resolved with contextual information outside the sentence. Human readers often use semantics, contextual information about the document and world knowledge to resolve segmentation ambiguities.

There is no fixed standard for Chinese word segmentation. Experiments have shown that there is only about 75% agreement among native speakers regarding the correct word segmentation (Sproat et al., 1996). Also, specific NLP tasks may require different segmentation criteria. For example, "北京银行" could be treated as a single word (Bank of Beijing) for machine translation, while it is more naturally segmented into "北京 (Beijing) 银行 (bank)" for tasks such as text-to-speech synthesis. Therefore, supervised learning with specifically defined training data has become the dominant approach.

Following Xue (2003), the standard approach to supervised learning for CWS is to treat it as a tagging task. Tags are assigned to each character in the sentence, indicating whether the character is a single-character word or the start, middle or end of a multi-character word. The features are usually confined to a five-character window with the current character in the middle. In this way, dynamic programming algorithms such as the Viterbi algorithm can be used for decoding.

Several discriminatively trained models have recently been applied to the CWS problem. Examples include Xue (2003), Peng et al. (2004) and Shi and Wang (2007); these use maximum entropy (ME) and conditional random field (CRF) models (Ratnaparkhi, 1998; Lafferty et al., 2001). An advantage of these models is their flexibility in allowing knowledge from various sources to be encoded as features.

Contextual information plays an important role in word segmentation decisions; especially useful is information about surrounding words. Consider the sentence "中国外企业", which can be from "其中 (among which) 国外 (foreign) 企业 (companies)", or "中国 (in China) 外企 (foreign companies) 业务 (business)". Note that the five-character window surrounding "外" is the same in both cases, making the tagging decision for that character difficult given the local window. However, the correct decision can be made by comparison of the two three-word windows containing this character.

In order to explore the potential of word-based models, we adapt the perceptron discriminative learning algorithm to the CWS problem. Collins (2002) proposed the perceptron as an alternative to the CRF method for HMM-style taggers. However, our model does not map the segmentation problem to a tag sequence learning problem, but defines features on segmented sentences directly. Hence we use a beam-search decoder during training and testing; our idea is similar to that of Collins and Roark (2004) who used a beam-search decoder as part of a perceptron parsing model. Our work can also be seen as part of the recent move towards *search-based* learning methods which do not rely on dynamic programming and are thus able to exploit larger parts of the context for making decisions (Daume III, 2006).

We study several factors that influence the performance of the perceptron word segmentor, including the averaged perceptron method, the size of the beam and the importance of word-based features. We compare the accuracy of our final system to the state-of-the-art CWS systems in the literature using the first and second SIGHAN bakeoff data. Our system is competitive with the best systems, obtaining the highest reported F-scores on a number of the bakeoff corpora. These results demonstrate the importance of word-based features for CWS. Furthermore, our approach provides an example of the potential of search-based discriminative training methods for NLP tasks.

2 The Perceptron Training Algorithm

We formulate the CWS problem as finding a mapping from an input sentence $x \in X$ to an output sentence $y \in Y$, where X is the set of possible raw sentences and Y is the set of possible segmented sentences. Given an input sentence x, the correct output segmentation $F(x)$ satisfies:

$$F(x) = \operatorname*{arg\,max}_{y \in \text{GEN}(x)} \text{Score}(y)$$

where $\text{GEN}(x)$ denotes the set of possible segmentations for an input sentence x, consistent with notation from Collins (2002).

The score for a segmented sentence is computed by first mapping it into a set of features. A feature is an indicator of the occurrence of a certain pattern in a segmented sentence. For example, it can be the occurrence of "里面" as a single word, or the occurrence of "里" separated from "面" in two adjacent words. By defining features, a segmented sentence is mapped into a global feature vector, in which each dimension represents the count of a particular feature in the sentence. The term "global" feature vector is used by Collins (2002) to distinguish between feature count vectors for whole sequences and the "local" feature vectors in ME tagging models, which are Boolean valued vectors containing the indicator features for one element in the sequence.

Denote the global feature vector for segmented sentence y with $\Phi(y) \in R^d$, where d is the total number of features in the model; then $\text{Score}(y)$ is computed by the dot product of vector $\Phi(y)$ and a parameter vector $\overline{\alpha} \in R^d$, where α_i is the weight for the ith feature:

$$\text{Score}(y) = \Phi(y) \cdot \overline{\alpha}$$

```
Inputs: training examples $(x_i, y_i)$
Initialization: set $\overline{\alpha} = 0$
Algorithm:
  for $t = 1..T$, $i = 1..N$
    calculate $z_i = \arg\max_{y \in \text{GEN}(x_i)} \Phi(y) \cdot \overline{\alpha}$
    if $z_i \neq y_i$
      $\overline{\alpha} = \overline{\alpha} + \Phi(y_i) - \Phi(z_i)$
Outputs: $\overline{\alpha}$
```

Figure 1: the perceptron learning algorithm, adapted from Collins (2002)

The perceptron training algorithm is used to determine the weight values $\overline{\alpha}$.

The training algorithm initializes the parameter vector as all zeros, and updates the vector by decoding the training examples. Each training sentence is turned into the raw input form, and then decoded with the current parameter vector. The output segmented sentence is compared with the original training example. If the output is incorrect, the parameter vector is updated by adding the global feature vector of the training example and subtracting the global feature vector of the decoder output. The algorithm can perform multiple passes over the same training sentences. Figure 1 gives the algorithm, where N is the number of training sentences and T is the number of passes over the data.

Note that the algorithm from Collins (2002) was designed for discriminatively training an HMM-style tagger. Features are extracted from an input sequence x and its corresponding tag sequence y:

$$\text{Score}(x, y) = \Phi(x, y) \cdot \overline{\alpha}$$

Our algorithm is not based on an HMM. For a given input sequence x, even the length of different candidates y (the number of words) is not fixed. Because the output sequence y (the segmented sentence) contains all the information from the input sequence x (the raw sentence), the global feature vector $\Phi(x, y)$ is replaced with $\Phi(y)$, which is extracted from the candidate segmented sentences directly.

Despite the above differences, since the theorems of convergence and their proof (Collins, 2002) are only dependent on the feature vectors, and not on the source of the feature definitions, the perceptron algorithm is applicable to the training of our CWS model.

2.1 The averaged perceptron

The averaged perceptron algorithm (Collins, 2002) was proposed as a way of reducing overfitting on the training data. It was motivated by the voted-perceptron algorithm (Freund and Schapire, 1999) and has been shown to give improved accuracy over the non-averaged perceptron on a number of tasks. Let N be the number of training sentences, T the number of training iterations, and $\overline{\alpha}^{n,t}$ the parameter vector immediately after the nth sentence in the tth iteration. The averaged parameter vector $\overline{\gamma} \in R^d$ is defined as:

$$\overline{\gamma} = \frac{1}{NT} \sum_{n=1..N,t=1..T} \overline{\alpha}^{n,t}$$

To compute the averaged parameters $\overline{\gamma}$, the training algorithm in Figure 1 can be modified by keeping a total parameter vector $\overline{\sigma}^{n,t} = \sum \overline{\alpha}^{n,t}$, which is updated using $\overline{\alpha}$ after each training example. After the final iteration, $\overline{\gamma}$ is computed as $\overline{\sigma}^{n,t}/NT$. In the averaged perceptron algorithm, $\overline{\gamma}$ is used instead of $\overline{\alpha}$ as the final parameter vector.

With a large number of features, calculating the total parameter vector $\overline{\sigma}^{n,t}$ after each training example is expensive. Since the number of changed dimensions in the parameter vector $\overline{\alpha}$ after each training example is a small proportion of the total vector, we use a lazy update optimization for the training process.[1] Define an update vector $\overline{\tau}$ to record the number of the training sentence n and iteration t when each dimension of the averaged parameter vector was last updated. Then after each training sentence is processed, only update the dimensions of the total parameter vector corresponding to the features in the sentence. (Except for the last example in the last iteration, when each dimension of $\overline{\tau}$ is updated, no matter whether the decoder output is correct or not).

Denote the sth dimension in each vector before processing the nth example in the tth iteration as $\alpha_s^{n-1,t}$, $\sigma_s^{n-1,t}$ and $\tau_s^{n-1,t} = (n_{\tau,s}, t_{\tau,s})$. Suppose that the decoder output $z_{n,t}$ is different from the training example y_n. Now $\alpha_s^{n,t}$, $\sigma_s^{n,t}$ and $\tau_s^{n,t}$ can

[1] Daume III (2006) describes a similar algorithm.

be updated in the following way:

$$\sigma_s^{n,t} = \sigma_s^{n-1,t} + \alpha_s^{n-1,t} \times (tN{+}n - t_{\tau,s}N - n_{\tau,s})$$
$$\alpha_s^{n,t} = \alpha_s^{n-1,t} + \Phi(y_n) - \Phi(z_{n,t})$$
$$\sigma_s^{n,t} = \sigma_s^{n,t} + \Phi(y_n) - \Phi(z_{n,t})$$
$$\tau_s^{n,t} = (n, t)$$

We found that this lazy update method was significantly faster than the naive method.

3 The Beam-Search Decoder

The decoder reads characters from the input sentence one at a time, and generates candidate segmentations incrementally. At each stage, the next incoming character is combined with an existing candidate in two different ways to generate new candidates: it is either appended to the last word in the candidate, or taken as the start of a new word. This method guarantees exhaustive generation of possible segmentations for any input sentence.

Two agendas are used: the source agenda and the target agenda. Initially the source agenda contains an empty sentence and the target agenda is empty. At each processing stage, the decoder reads in a character from the input sentence, combines it with each candidate in the source agenda and puts the generated candidates onto the target agenda. After each character is processed, the items in the target agenda are copied to the source agenda, and then the target agenda is cleaned, so that the newly generated candidates can be combined with the next incoming character to generate new candidates. After the last character is processed, the decoder returns the candidate with the best score in the source agenda. Figure 2 gives the decoding algorithm.

For a sentence with length l, there are 2^{l-1} different possible segmentations. To guarantee reasonable running speed, the size of the target agenda is limited, keeping only the B best candidates.

4 Feature templates

The feature templates are shown in Table 1. Features 1 and 2 contain only word information, 3 to 5 contain character and length information, 6 and 7 contain only character information, 8 to 12 contain word and character information, while 13 and 14 contain word and length information. Any segmented sentence is mapped to a global feature vector according to these templates. There are $356,337$ features with non-zero values after 6 training iterations using the development data.

```
Input: raw sentence $sent$ – a list of characters
Initialization: set agendas $src = [[]]$, $tgt = []$
Variables: candidate sentence $item$ – a list of words
Algorithm:
  for $index = 0..sent$.length$-1$:
    var $char = sent[index]$
    foreach $item$ in $src$:
      // append as a new word to the candidate
      var $item_1 = item$
      $item_1$.append($char$.toWord())
      $tgt$.insert($item_1$)
      // append the character to the last word
      if $item$.length $> 1$:
        var $item_2 = item$
        $item_2[item_2$.length$-1]$.append($char$)
        $tgt$.insert($item_2$)
    $src = tgt$
    $tgt = []$
Outputs: $src$.best_item
```

Figure 2: The decoding algorithm

For this particular feature set, the longest range features are word bigrams. Therefore, among partial candidates ending with the same bigram, the best one will also be in the best final candidate. The decoder can be optimized accordingly: when an incoming character is combined with candidate items as a new word, only the best candidate is kept among those having the same last word.

5 Comparison with Previous Work

Among the character-tagging CWS models, Li et al. (2005) uses an uneven margin alteration of the traditional perceptron classifier (Li et al., 2002). Each character is classified independently, using information in the neighboring five-character window. Liang (2005) uses the discriminative perceptron algorithm (Collins, 2002) to score whole character tag sequences, finding the best candidate by the global score. It can be seen as an alternative to the ME and CRF models (Xue, 2003; Peng et al., 2004), which

1	word w
2	word bigram w_1w_2
3	single-character word w
4	a word starting with character c and having length l
5	a word ending with character c and having length l
6	space-separated characters c_1 and c_2
7	character bigram c_1c_2 in any word
8	the first and last characters c_1 and c_2 of any word
9	word w immediately before character c
10	character c immediately before word w
11	the starting characters c_1 and c_2 of two consecutive words
12	the ending characters c_1 and c_2 of two consecutive words
13	a word of length l and the previous word w
14	a word of length l and the next word w

Table 1: feature templates

do not involve word information. Wang et al. (2006) incorporates an N-gram language model in ME tagging, making use of word information to improve the character tagging model. The key difference between our model and the above models is the word-based nature of our system.

One existing method that is based on sub-word information, Zhang et al. (2006), combines a CRF and a rule-based model. Unlike the character-tagging models, the CRF submodel assigns tags to sub-words, which include single-character words and the most frequent multiple-character words from the training corpus. Thus it can be seen as a step towards a word-based model. However, sub-words do not necessarily contain full word information. Moreover, sub-word extraction is performed separately from feature extraction. Another difference from our model is the rule-based submodel, which uses a dictionary-based forward maximum match method described by Sproat et al. (1996).

6 Experiments

Two sets of experiments were conducted. The first, used for development, was based on the part of Chinese Treebank 4 that is not in Chinese Treebank 3 (since CTB3 was used as part of the first bake-off). This corpus contains 240K characters (150K words and 4798 sentences). 80% of the sentences (3813) were randomly chosen for training and the rest (985 sentences) were used as development testing data. The accuracies and learning curves for the non-averaged and averaged perceptron were compared. The influence of particular features and the agenda size were also studied.

The second set of experiments used training and testing sets from the first and second international Chinese word segmentation bakeoffs (Sproat and Emerson, 2003; Emerson, 2005). The accuracies are compared to other models in the literature.

F-measure is used as the accuracy measure. Define precision p as the percentage of words in the decoder output that are segmented correctly, and recall r as the percentage of gold standard output words that are correctly segmented by the decoder. The (balanced) F-measure is $2pr/(p+r)$.

CWS systems are evaluated by two types of tests. The *closed* tests require that the system is trained only with a designated training corpus. Any extra knowledge is not allowed, including common surnames, Chinese and Arabic numbers, European letters, lexicons, part-of-speech, semantics and so on. The *open* tests do not impose such restrictions.

Open tests measure a model's capability to utilize extra information and domain knowledge, which can lead to improved performance, but since this extra information is not standardized, direct comparison between open test results is less informative.

In this paper, we focus only on the closed test. However, the perceptron model allows a wide range of features, and so future work will consider how to integrate open resources into our system.

6.1 Learning curve

In this experiment, the agenda size was set to 16, for both training and testing. Table 2 shows the precision, recall and F-measure for the development set after 1 to 10 training iterations, as well as the number of mistakes made in each iteration. The corresponding learning curves for both the non-averaged and averaged perceptron are given in Figure 3.

The table shows that the number of mistakes made in each iteration decreases, reflecting the convergence of the learning algorithm. The averaged per-

Iteration	1	2	3	4	5	6	7	8	9	10
P (non-avg)	89.0	91.6	92.0	92.3	92.5	92.5	92.5	92.7	92.6	92.6
R (non-avg)	88.3	91.4	92.2	92.6	92.7	92.8	93.0	93.0	93.1	93.2
F (non-avg)	88.6	91.5	92.1	92.5	92.6	92.6	92.7	92.8	92.8	92.9
P (avg)	91.7	92.8	93.1	92.2	93.1	93.2	93.2	93.2	93.2	93.2
R (avg)	91.6	92.9	93.3	93.4	93.4	93.5	93.5	93.5	93.6	93.6
F (avg)	91.6	92.9	93.2	93.3	93.3	93.4	93.3	93.3	93.4	93.4
#Wrong sentences	3401	1652	945	621	463	288	217	176	151	139

Table 2: accuracy using non-averaged and averaged perceptron.
P - precision (%), R - recall (%), F - F-measure.

B	2	4	8	16	32	64	128	256	512	1024
Tr	660	610	683	830	1111	1645	2545	4922	9104	15598
Seg	18.65	18.18	28.85	26.52	36.58	56.45	95.45	173.38	325.99	559.87
F	86.90	92.95	93.33	93.38	93.25	93.29	93.19	93.07	93.24	93.34

Table 3: the influence of agenda size.
B - agenda size, Tr - training time (seconds), Seg - testing time (seconds), F - F-measure.

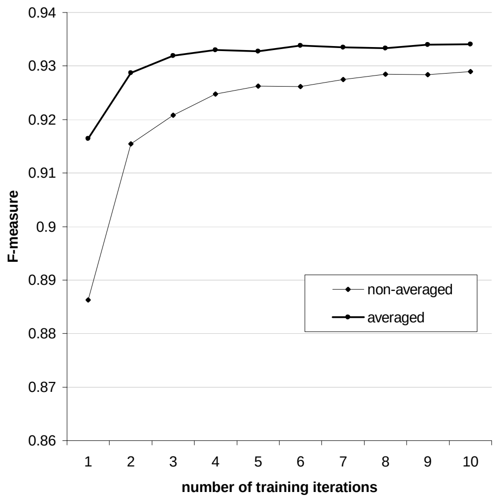

Figure 3: learning curves of the averaged and non-averaged perceptron algorithms

ceptron algorithm improves the segmentation accuracy at each iteration, compared with the non-averaged perceptron. The learning curve was used to fix the number of training iterations at 6 for the remaining experiments.

6.2 The influence of agenda size

Reducing the agenda size increases the decoding speed, but it could cause loss of accuracy by eliminating potentially good candidates. The agenda size also affects the training time, and resulting model, since the perceptron training algorithm uses the decoder output to adjust the model parameters. Table 3 shows the accuracies with ten different agenda sizes, each used for both training and testing.

Accuracy does not increase beyond $B = 16$. Moreover, the accuracy is quite competitive even with B as low as 4. This reflects the fact that the best segmentation is often within the current top few candidates in the agenda.[2] Since the training and testing time generally increases as N increases, the agenda size is fixed to 16 for the remaining experiments.

6.3 The influence of particular features

Our CWS model is highly dependent upon word information. Most of the features in Table 1 are related to words. Table 4 shows the accuracy with various features from the model removed.

Among the features, vocabulary words (feature 1) and length prediction by characters (features 3 to 5) showed strong influence on the accuracy, while word bigrams (feature 2) and special characters in them (features 11 and 12) showed comparatively weak influence.

[2]The optimization in Section 4, which has a pruning effect, was applied to this experiment. Similar observations were made in separate experiments without such optimization.

Features	F	Features	F
All	93.38	w/o 1	92.88
w/o 2	93.36	w/o 3, 4, 5	92.72
w/o 6	93.13	w/o 7	93.13
w/o 8	93.14	w/o 9, 10	93.31
w/o 11, 12	93.38	w/o 13, 14	93.23

Table 4: the influence of features. (F: F-measure. Feature numbers are from Table 1)

6.4 Closed test on the SIGHAN bakeoffs

Four training and testing corpora were used in the first bakeoff (Sproat and Emerson, 2003), including the Academia Sinica Corpus (AS), the Penn Chinese Treebank Corpus (CTB), the Hong Kong City University Corpus (CU) and the Peking University Corpus (PU). However, because the testing data from the Penn Chinese Treebank Corpus is currently unavailable, we excluded this corpus. The corpora are encoded in GB (PU, CTB) and BIG5 (AS, CU). In order to test them consistently in our system, they are all converted to UTF8 without loss of information.

The results are shown in Table 5. We follow the format from Peng et al. (2004). Each row represents a CWS model. The first eight rows represent models from Sproat and Emerson (2003) that participated in at least one closed test from the table, row "Peng" represents the CRF model from Peng et al. (2004), and the last row represents our model. The first three columns represent tests with the AS, CU and PU corpora, respectively. The best score in each column is shown in bold. The last two columns represent the average accuracy of each model over the tests it participated in (SAV), and our average over the same tests (OAV), respectively. For each row the best average is shown in bold.

We achieved the best accuracy in two of the three corpora, and better overall accuracy than the majority of the other models. The average score of S10 is 0.7% higher than our model, but S10 only participated in the HK test.

Four training and testing corpora were used in the second bakeoff (Emerson, 2005), including the Academia Sinica corpus (AS), the Hong Kong City University Corpus (CU), the Peking University Corpus (PK) and the Microsoft Research Corpus (MR).

	AS	CU	PU	SAV	OAV
S01	93.8	90.1	**95.1**	93.0	**95.0**
S04			93.9	93.9	**94.0**
S05	94.2		89.4	91.8	**95.3**
S06	94.5	92.4	92.4	93.1	**95.0**
S08		90.4	93.6	92.0	**94.3**
S09	96.1		94.6	**95.4**	95.3
S10			94.7	**94.7**	94.0
S12	95.9	91.6		93.8	**95.6**
Peng	95.6	92.8	94.1	94.2	**95.0**
	96.5	**94.6**	94.0		

Table 5: the accuracies over the first SIGHAN bakeoff data.

	AS	CU	PK	MR	SAV	OAV
S14	94.7	94.3	95.0	96.4	95.1	**95.4**
S15b	**95.2**	94.1	94.1	95.8	94.8	**95.4**
S27	94.5	94.0	95.0	96.0	94.9	**95.4**
Zh-a	94.7	94.6	94.5	96.4	95.1	**95.4**
Zh-b	95.1	**95.1**	**95.1**	97.1	**95.6**	95.4
	94.6	**95.1**	94.5	**97.2**		

Table 6: the accuracies over the second SIGHAN bakeoff data.

Different encodings were provided, and the UTF8 data for all four corpora were used in this experiment.

Following the format of Table 5, the results for this bakeoff are shown in Table 6. We chose the three models that achieved at least one best score in the closed tests from Emerson (2005), as well as the sub-word-based model of Zhang et al. (2006) for comparison. Row "Zh-a" and "Zh-b" represent the pure sub-word CRF model and the confidence-based combination of the CRF and rule-based models, respectively.

Again, our model achieved better overall accuracy than the majority of the other models. One system to achieve comparable accuracy with our system is Zh-b, which improves upon the sub-word CRF model (Zh-a) by combining it with an independent dictionary-based submodel and improving the accuracy of known words. In comparison, our system is based on a single perceptron model.

In summary, closed tests for both the first and the second bakeoff showed competitive results for our system compared with the best results in the literature. Our word-based system achieved the best F-measures over the AS (96.5%) and CU (94.6%) corpora in the first bakeoff, and the CU (95.1%) and MR (97.2%) corpora in the second bakeoff.

7 Conclusions and Future Work

We proposed a word-based CWS model using the discriminative perceptron learning algorithm. This model is an alternative to the existing character-based tagging models, and allows word information to be used as features. One attractive feature of the perceptron training algorithm is its simplicity, consisting of only a decoder and a trivial update process. We use a beam-search decoder, which places our work in the context of recent proposals for search-based discriminative learning algorithms. Closed tests using the first and second SIGHAN CWS bakeoff data demonstrated our system to be competitive with the best in the literature.

Open features, such as knowledge of numbers and European letters, and relationships from semantic networks (Shi and Wang, 2007), have been reported to improve accuracy. Therefore, given the flexibility of the feature-based perceptron model, an obvious next step is the study of open features in the segmentor.

Also, we wish to explore the possibility of incorporating POS tagging and parsing features into the discriminative model, leading to joint decoding. The advantage is two-fold: higher level syntactic information can be used in word segmentation, while joint decoding helps to prevent bottom-up error propagation among the different processing steps.

Acknowledgements

This work is supported by the ORS and Clarendon Fund. We thank the anonymous reviewers for their insightful comments.

References

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of ACL'04*, pages 111–118, Barcelona, Spain, July.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8, Philadelphia, USA, July.

Hal Daume III. 2006. *Practical Structured Learning for Natural Language Processing*. Ph.D. thesis, USC.

Thomas Emerson. 2005. The second international Chinese word segmentation bakeoff. In *Proceedings of The Fourth SIGHAN Workshop*, Jeju, Korea.

Y. Freund and R. Schapire. 1999. Large margin classification using the perceptron algorithm. In *Machine Learning*, pages 277–296.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th ICML*, pages 282–289, Massachusetts, USA.

Y. Li, Zaragoza, R. H., Herbrich, J. Shawe-Taylor, and J. Kandola. 2002. The perceptron algorithm with uneven margins. In *Proceedings of the 9th ICML*, pages 379–386, Sydney, Australia.

Yaoyong Li, Chuanjiang Miao, Kalina Bontcheva, and Hamish Cunningham. 2005. Perceptron learning for Chinese word segmentation. In *Proceedings of the Fourth SIGHAN Workshop*, Jeju, Korea.

Percy Liang. 2005. Semi-supervised learning for natural language. Master's thesis, MIT.

F. Peng, F. Feng, , and A. McCallum. 2004. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of COLING*, Geneva, Switzerland.

Adwait Ratnaparkhi. 1998. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. Ph.D. thesis, UPenn.

Yanxin Shi and Mengqiu Wang. 2007. A dual-layer CRF based joint decoding method for cascade segmentation and labelling tasks. In *Proceedings of IJCAI*, Hyderabad, India.

Richard Sproat and Thomas Emerson. 2003. The first international Chinese word segmentation bakeoff. In *Proceedings of The Second SIGHAN Workshop*, pages 282–289, Sapporo, Japan, July.

R. Sproat, C. Shih, W. Gail, and N. Chang. 1996. A stochastic finite-state word-segmentation algorithm for Chinese. In *Computational Linguistics*, volume 22(3), pages 377–404.

Xinhao Wang, Xiaojun Lin, Dianhai Yu, Hao Tian, and Xihong Wu. 2006. Chinese word segmentation with maximum entropy and n-gram language model. In *Proceedings of the Fifth SIGHAN Workshop*, pages 138–141, Sydney, Australia, July.

N. Xue. 2003. Chinese word segmentation as character tagging. In *International Journal of Computational Linguistics and Chinese Language Processing*, volume 8(1).

Ruiqiang Zhang, Genichiro Kikui, and Eiichiro Sumita. 2006. Subword-based tagging by conditional random fields for Chinese word segmentation. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion*, volume Short Papers, pages 193–196, New York City, USA, June.