# EXPERIMENTAL COMPARISON OF DISCRIMINATIVE LEARNING APPROACHES FOR CHINESE WORD SEGMENTATION

by

Dong Song
B.Sc., Simon Fraser University, 2005

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Summer 2008

# APPROVAL

**Name:** Dong Song

**Degree:** Master of Science

**Title of thesis:** Experimental Comparison of Discriminative Learning Approaches for Chinese Word Segmentation

**Examining Committee:** Dr. Ke Wang
Chair

Dr. Anoop Sarkar, Senior Supervisor

Dr. Fred Popowich, Supervisor

Dr. Ping Xue, External Examiner

**Date Approved:** April 15, 2008

# Abstract

Natural language processing tasks assume that the input is tokenized into individual words. In languages like Chinese, however, such tokens are not available in the written form. This thesis explores the use of machine learning to segment Chinese sentences into word tokens. We conduct a detailed experimental comparison between various methods for word segmentation. We have built two Chinese word segmentation systems and evaluated them on standard data sets.

The state of the art in this area involves the use of character-level features where the best segmentation is found using conditional random fields (CRF). The first system we implemented uses a majority voting approach among different CRF models and dictionary-based matching, and it outperforms the individual methods. The second system uses novel global features for word segmentation. Feature weights are trained using the averaged perceptron algorithm. By adding global features, performance is significantly improved compared to character-level CRF models.

**Key words:** word segmentation; machine learning; natural language processing.

*To my homeland*

# Acknowledgments

I would like to express my sincere gratitude and respect to my senior supervisor, Dr. Anoop Sarkar. His immense knowledge in natural language processing, his enthusiasm, support, and patience, taken together, make him an excellent mentor and advisor.

I would also like to thank my supervisor, Dr. Fred Popowich, and my thesis examiner, Dr. Ping Xue, for their great help towards my thesis. Also, I thank all members in the natural language lab at Simon Fraser University and all my other friends for their continuous support, and I thank Adam Lein for proofreading my thesis.

The most important, I am grateful to my parents for their endless love, encouragement and support.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Word segmentation refers to the process of demarcating blocks in a character sequence such that the produced output is composed of separated tokens and is meaningful. For example, “we live in an information age” is a segmented form of the unsegmented text “weliveinaninformationage”. Word segmentation is an important task that is prerequisite for various natural language processing applications. For instance, only if we have identified each word in a sentence, can part of speech tags (e.g. NNP or DT) then be assigned and the syntax tree for the whole sentence be built. In systems dealing with English or French, tokens are assumed to be already available since words have always been separated by spaces in these languages. While in Chinese, characters are written next to each other without marks identifying words. As an illustration, the character sequence “我们生活在信息时代” in Chinese written text has the same meaning as “we live in an information age” in English; nevertheless, no white-space is used between the noun “我们”(we), the verb “生活”(live), the preposition “在”(in) and the nouns “信息”(information), “时代”(age). To better understand Chinese and to achieve more accurate results for machine translation, named entity recognition, information extraction and other natural language tasks dealing with Chinese, segmentation has to be performed in advance so that words are isolated from each other.

## 1.1 Challenges of Chinese Word Segmentation

Chinese word segmentation is considered to be a significantly challenging task for the following reasons:

- First, it is challenging to produce the most plausible segmentation output. For example, given the Chinese character sequence "北京大学生比赛"(Competition among university students in Beijing), a plausible segmentation would be "北京(Beijing)/大学生(university students)/比赛(competition)". On the other hand, the character sequence "北京大学"(Beijing University) is a named entity word, representing an institution. If we recognize "北京大学" as the institution name, the segmentation for the above character sequence would become "北京大学(Beijing University)/生(give birth to)/比赛(competition)"(Beijing University gives birth to the competition), which is not plausible. Also, there may be other segmentation results, such as "北京(Beijing)/大学(university)/生(give birth to)/比赛(competition)", many of which, however, are not plausible, either. Thus, different technical approaches can produce different segmentations, and picking up the most plausible one is desirable while challenging.

- Second, out-of-vocabulary words are a major bottleneck in the segmentation process. Resources are scarce, and the Chinese word repository is huge. As a result, any piece of Chinese text may include character sequences that do not appear in dictionaries. This greatly complicates the task of segmentation. For example, suppose we encounter an unknown Chinese character, there are at least two possibilities: (1) this character itself is an out-of-vocabulary word; (2) this character combining with the preceding sequence of characters form an out-of-vocabulary word. In addition, various productive processes can derive words that are inevitably omitted by any dictionary, for instance, including morphologically derived words "学生们"(students), derived by appending the suffix "们" to the singular noun "学生"(student). Moreover, new words are created every day reflecting the political, social and cultural changes of the world. For example, before the year 2002, the word "非典" (Severe Acute Respiratory Syndrome) was never in the dictionary. As a consequence, it is expensive and unfeasible to frequently update dictionaries, collecting and inserting these newly appearing words; therefore, there are always going to be out-of-vocabulary words.

## 1.2 Motivation to Study This Problem

Due to the challenges it encounters, Chinese word segmentation is never considered a closed problem, and in spite of the above difficulties, this task becomes more attractive for the following reasons:

- Accurately segmenting text is an important preprocessing step for related applications, such as Chinese named entity recognition and machine translation. Before either of these tasks can take place, it is convenient to segment text into words [4, 5, 50]. Also, in speech synthesis applications, word boundary information is central to identify tone changes in Mandarin spoken language. To illustrate, the character "一"(one) is pronounced in its first-tone if it is a single-character word, but changes to second-tone when it is combined in front of a fourth-tone character to form a word, such as in "一片"(one piece). In order to make the correct tone modification, the speech generator must be aware of word boundaries in text.

- Similar to Chinese, certain other human languages, such as Thai and Japanese Kanji, don't contain spaces in their writing system, but display similar properties, such as word length distribution, as in Chinese. Therefore, by exploring approaches to Chinese word segmentation, given sufficient data, we can easily transfer segmentation algorithms onto these other unsegmented writing systems to achieve more accurate segmentation results, and better support various natural language processing tasks on these other languages as well.

- The Chinese word segmentation task has similarities with certain other sequence learning problems. If we understand the word segmentation problem, we will be able to apply the underlying technique to these other problems. For instance, in speech recognition, an important sub-problem is automatic speech segmentation, in which the sound signals are broken down and classified into a string of phonemes for phonetic segmentation. Also, another sub-problem, lexical segmentation, decomposes a complex spoken sentence into smaller lexical segments. Other sequence learning tasks include part-of-speech tagging, in which a word sequence is associated with a part-of-speech tag sequence, which corresponds to grammatical categories like noun, verb, and etc. Similar to tagging, another sequence learning task is finding non-recursive phrasal chunks in a word sequence. Research into sequence learning can be re-used in these different natural language processing tasks.

## 1.3 Introduction to SIGHAN Bakeoff

To encourage and to promote better research in Chinese word segmentation, SIGHAN, the Association for Computational Linguistics (ACL) Special Interest Group on Chinese Language Processing, has been holding the International Chinese Word Segmentation Bakeoff for several years.

The first bakeoff was held in 2003, and the results were presented at the second SIGHAN Workshop at ACL 2003 in Sapporo, Japan [39]. The second bakeoff was held in 2005, and the results were presented at the fourth SIGHAN Workshop at IJCNLP-05 on Jeju Island, Korea [12]. The third bakeoff was held in 2006, and the results were presented at the fifth SIGHAN Workshop at ACL 2006 in Sydney, Australia [25].

In each bakeoff, several corpora are available for the word segmentation task. For example, in the third bakeoff, four corpora, one from Academia Sinica (CKIP), one from City University of Hong Kong (CityU), one from Microsoft Research Asia (MSRA) and the other one from University of Pennsylvania/University of Colorado (UPUC), were evaluated. The participating teams may return results on any subset of these corpora. The only constraint is that they are not allowed to select a corpus where they have previous access to the testing portion of the corpus. Each training corpus is provided in the format of one sentence per line, separating words and punctuation by spaces; while the corresponding test data is in the same format, except that the spaces are absent.

Each bakeoff consists of an open test and a closed test. In the open test, the participants are allowed to train on the training set for a particular corpus, and in addition, they may use any other material including material from other training corpora, proprietary dictionaries, material from the world wide web and so forth. In the closed test, however, they may only use training material from the training data for the particular corpus they are testing on. No other material or knowledge is allowed, including, but not limited to, part-of-speech information, externally generated word-frequency counts, Arabic and Chinese numbers, feature characters for place names, and common Chinese surnames.

Each submitted output is compared with the gold standard segmentation for that test set, and is evaluated in terms of precision ($P$), recall ($R$), evenly-weighted F-score($F$), out-of-vocabulary recall rate ($R_{OOV}$), and in-vocabulary recall rate ($R_{IV}$). In each year's bakeoff, a scoring script, implementing the standard evaluation methodology, is officially provided.

Precision is defined as the number of correctly segmented words divided by the total number of words in the segmentation result, where the correctness of the segmented words is determined by matching the segmentation with the gold standard test set. Recall is defined as the number of correctly segmented words divided by the total number of words in the gold standard test set. Evenly-weighted F-score is calculated by the following formula:

$$\text{F-score} = \frac{\text{Precision} \times \text{Recall} \times 2}{\text{Precision} + \text{Recall}}$$

The out-of-vocabulary recall rate is defined as the number of correctly segmented words that are not in the dictionary, divided by the total number of words which are in the gold standard test set but not in the dictionary. The in-vocabulary recall rate is defined as the number of correctly segmented words that are in the dictionary, divided by the total number of words which are in the gold standard test set and also in the dictionary.

## 1.4 Approaches and Contributions

In this thesis, two approaches solving the Chinese word segmentation problem are proposed. The first approach adapts character-level majority voting among outputs from three different methods to obtain performance higher than any of the individual methods. The second approach uses global features, such as the sentence language model score, along with local features in a discriminative learning approach to word segmentation. We use the averaged perceptron as a global linear model over the N-best output of a character-based conditional random field. Both systems are evaluated on the CityU, MSRA and UPUC corpora from the third SIGHAN Bakeoff.

The main contributions of this thesis are as follows:

- We show that the majority voting approach improves the segmentation performance over the individual methods. The voting procedure combines advantages from each of its individual methods and produces results with high in-vocabulary recall rate and high out-of-vocabulary recall rate.

- We discover that by including global features and combining them with local features in averaged perceptron learning, the segmentation F-score is improved significantly, compared to the character-based one-best CRF tagger and the perceptron algorithm merely applying local features.

## 1.5 Thesis Outline

The rest of the thesis is organized as follows. In Chapter 2, we provide an overview of general approaches in Chinese word segmentation problem. In Chapter 3, the majority voting method and its experimental results on the three corpora from the third SIGHAN bakeoff are described in detail, and in Chapter 4, training a perceptron with global and local features for Chinese word segmentation is explored. Finally, in Chapter 5, we summarize the methods of this thesis and point out possible future work.

# Chapter 2

# General Approaches

In the literature, different methods have been proposed to deal with Chinese word segmentation problem. In this chapter, we provide a general review of various types of approaches, classified into three main categories: the *dictionary-based matching* approach, the *character-based or subword-based sequence learning* approach, and the *global linear model* approach.

The dictionary-based matching approach is simple and efficient. It uses a machine-readable lexical dictionary, which can be prepared beforehand, and it maps possible words in sentences to entries in the dictionary. One major difficulty of this kind of approach is that, in order to get a high-quality result, the dictionary has to be as complete as possible. Also, dictionary matching using the greedy longest match can ignore plausible word sequences in favor of implausible ones.

In the sequence learning approach, each character is assigned a particular tag, indicating the position of that character within a word. Patterns or statistical information are obtained from the tagged training examples with machine learning techniques, and this information is then used to predict tags for unseen test data so that the optimal tag sequence is produced.

A global linear model, on the other hand, attempts not to attach probabilities to decisions, but instead to compute scores for the entire word segmentation sentence using a variety of features from the training set. It tends to choose the highest scoring candidate $y^*$ as the most plausible output from $GEN(x)$, a set of possible outputs for a given input $x$. That is,

$$y^* = \operatorname*{argmax}_{y \in GEN(x)} \Phi(x, y) \cdot \mathbf{w}$$

where $\Phi(x, y)$ represents the set of features, and $\mathbf{w}$ is the parameter vector assigning a weight to each feature in $\Phi(x, y)$.

Having briefly introduced these three general categories, in the subsequent sections of this chapter, a few common methods from each category will be explained in detail.

## 2.1 Dictionary-based Matching

### 2.1.1 Greedy longest matching algorithm

The simplest and easiest-to-implement algorithm for Chinese word segmentation is the greedy longest matching method. It is a dictionary-based approach, by traversing from left to right in the current test sentence, greedily taking the longest match based on the words in the lexical dictionary.

The basic form is quite simple, and it has been officially used as the model to produce baseline scores in all SIGHAN bakeoffs [12, 25, 39]. It starts from the beginning of a sentence, matches a character sequence as long as possible with words in the dictionary, and then it continues the process, beginning from the next character after the identified word, until the entire sentence is segmented. Suppose we have a character sequence "$C_1C_2C_3\ldots$". First, we check to see whether "$C_1$" appears in the dictionary. Then, "$C_1C_2$" is matched with words in the dictionary, and "$C_1C_2C_3$" is examined, and so on. This process continues until a character sequence with length longer than the longest word in the dictionary is encountered. After that, the longest match will be considered as the most plausible one and be chosen as the first output word. Suppose "$C_1C_2$" is the most plausible word, then we start from the next character "$C_3$" and repeat the above process over again. If, however, none of the words in the dictionary starts with the character "$C_1$", then "$C_1$" is segmented as a single-character word, and the segmentation process continues from "$C_2$". This whole procedure is repeated until the character sequence is completely segmented.

As an example, suppose the dictionary, extracted from the training set, contains the following words: "我们"(we), "在"(in), "信息"(information), and "时代"(age). For the unsegmented test sentence "我们生活在信息时代"(we live in an information age), starting from the beginning of this sentence, we find the first longest character sequence "我们", matching entries in the dictionary, and we identify it as a word. Next, the character "生" doesn't have a corresponding entry in the dictionary; therefore, we output it as a single-character word. Similarly, "活" is determined to be a single-character word as well. Continuing this matching process from the character "在", we eventually produce the segmentation result as "我们/生/活/在/信息/时代". This example shows the simplicity of the greedy longest matching algorithm, and in addition, by comparing the segmentation result with the gold standard "我们/生活/在/信息/时代", produced by human experts, we observe that it is different from the gold standard and clearly realize the necessity of a large dictionary in order to achieve the correct segmentation.

### 2.1.2 Segmentation using unigram word frequencies

At the Linguistic Data Consortium (LDC), Zhibiao Wu implemented a Chinese word segmentation system[1], which uses the dictionary words together with their unigram frequencies, both of which are extracted from the training data set. Given an unsegmented sentence, its segmentation is no longer based on the greedy longest match alone. Instead, the sentence is segmented into a list of all possible candidates using the dictionary. Then, words from these candidates are connected to form different segmentation paths for the whole sentence, and dynamic programming is adapted to find the path which has the highest score. The score for a sentence is the product of the unigram probabilities of the words in that sentence:

$$y_1^*, \dots, y_n^* = \operatorname*{argmax}_{y_1,\dots,y_n} P(y_1, \dots, y_n) = \operatorname*{argmax}_{y_1,\dots,y_n} \prod_{i=1}^{n} P(y_i)$$

If two paths return the identical score, then the one which contains the least number of words is selected as the final path.

Here is a simple example. Suppose from the training data set, dictionary words and their frequency counts are summarized in Table 2.1. We can also easily calculate each word's probability.

To segment the sequence "abcd", we first produce all its possible segmentations using words in the dictionary. That is,

- a/b/c/d
- ab/c/d
- a/bc/d

Figure 2.1 shows all paths from the *start* towards the *end* of this sentence.

[1] http://projects.ldc.upenn.edu/Chinese/LDC_ch.htm

| Word | Frequency Count | Probability |
|---|---|---|
| a | 2 | 0.2 |
| b | 3 | 0.3 |
| c | 1 | 0.1 |
| d | 2 | 0.2 |
| ab | 1 | 0.1 |
| bc | 1 | 0.1 |
| TOTAL | 10 | 1.0 |

Table 2.1: An example for segmentation using unigram word frequencies

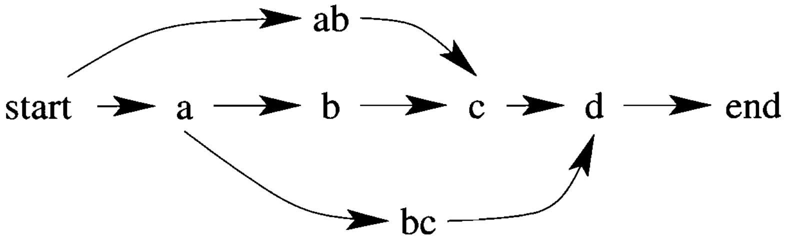

Figure 2.1: Graph with different paths for segmentation using unigram word frequencies

Given the words' probability information, the score for each path in Figure 2.1 is determined:

- a → b → c → d: $0.2 \times 0.3 \times 0.1 \times 0.2 = 0.0012$
- ab → c → d: $0.1 \times 0.1 \times 0.2 = 0.002$
- a → bc → d: $0.2 \times 0.1 \times 0.2 = 0.004$

Thus, the path "a → bc → d" has the highest score, and "abcd" is segmented as "a/bc/d". Comparing it with the greedy longest match which produces the result "ab/c/d", we can see that these two algorithms are different.

In our experiments for Chinese word segmentation, to evaluate the above algorithm, we uses the Perl program which implements the unigram word frequency segmenter, originally written by Zhibiao Wu. The version we used in our experiments was a revision, kindly provided to us by Michael Subotin[2] and David Chiang[3], that has been fixed to work with UTF-8 and to do full Viterbi for finding the best segmentation. However, as we shall see in the experimental results in Section 3.7, it does not perform better than the greedy longest matching algorithm.

## 2.2 Sequence Learning Algorithms

Chinese word segmentation can also be treated as a sequence learning task in which each character is assigned a particular tag, indicating the position of that character within a word. Patterns or statistical information is obtained from the tagged training examples with machine learning techniques, and this information is then used to predict tags for unseen test data so that the optimal tag sequence is produced.

Various tagsets have been explored for Chinese word segmentation [48]. Although different tagsets can be used to recognize different features inside a word, on the other hand, the choice of tagset has only a minor influence to the performance [35]. Combining the tagsets in a voting scheme can sometimes lead to an improvement in accuracy as shown in [37]. The two most typical types of tagset are the 3-tag "IOB" set and the 4-tag "BMES" set. In the "IOB" tagset, the first character of a multi-character word is assigned the "B" (**B**eginning)

[2] msubotin@umiacs.umd.edu

[3] chiang@isi.edu

tag, and each remaining character of the multi-character word is assigned the "I" (**I**nside) tag. For a single-character word, that character is assigned the "O" (**O**utside) tag, indicating that character is outside a multi-character word. While in the "BMES" tagset, for each multi-character word, its first character is given the "B" (**B**eginning) tag , its last character is given the "E" (**E**nd) tag, while each remaining character is given the "M" (**M**iddle) tag. In addition, for a single-character word, "S" (**S**ingle) is used as its tag (same as the "O" tag in "IOB" tagset). For instance, the sentence "新华社/上海/二月/十日/电" (Report from Xinhua News Agency on February 10th in Shanghai) is tagged as follows:

- With "IOB" Tagset: 新-B 华-I 社-I 上-B 海-I 二-B 月-I 十-B 日-I 电-O
- With "BMES" Tagset: 新-B 华-M 社-E 上-B 海-E 二-B 月-E 十-B 日-E 电-S

After assigning tags to the training data, generative modeling or discriminative modeling can be used to learn how to predict a sequence of character tags for the input unsegmented sentence.

### 2.2.1 HMM - A Generative Model

A generative model is one which explicitly states how the observations are assumed to have been generated. It defines the joint probability $\Pr(\mathbf{x}, \mathbf{y})$, given the input $\mathbf{x}$ and the label $\mathbf{y}$, and it makes predictions by calculating $\Pr(\mathbf{y} \mid \mathbf{x})$ and $\Pr(\mathbf{x})$, and then picks the most likely label $y \in \mathbf{y}$. The Hidden Markov Model [32] is the typical model for $\Pr(\mathbf{x}, \mathbf{y})$.

The Hidden Markov Model (HMM) defines a set of states. Suppose $N$ is the number of states in the model so that we can denote the individual states as $\mathbf{s} = \{s_1, s_2, \ldots, s_N\}$, and the state at time t as $q_t$. Also, $M$, the number of distinct observation symbols per state, is known, and the individual symbols are denoted as $\mathbf{v} = \{v_1, v_2, \ldots, v_M\}$. In addition, the transition probability $\mathbf{a} = \{a_{ij}\}$ where $a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)$, and the emission probability $\mathbf{b} = \{b_j(k)\}$ where $b_j(k) = P(v_k \text{ at t} \mid q_t = s_j)$ ($1 \leq j \leq$ N and $1 \leq k \leq$ M) are given. Moreover, the initial state distribution $\pi = \{\pi_i\}$, where $\pi_i = P(q_1 = s_i)$ ($1 \leq i \leq$ N), is defined.

For example, the sentence "我-B 们-I 生-B 活-I 在-O 信-B 息-I 时-B 代-I" can be described as follows in Figure 2.2. Inside this figure, there are 3 states: B, I, and O. For this sentence, the state sequence is (B, I, B, I, O, B, I, B, I), and (我, 们, 生, 活, 在, 信, 息, 时, 代) represents its observation sequence $(o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8, o_9)$, where each observation $o_t$ is one of the symbols from $\mathbf{v}$. Every arrow from one state to another state introduces

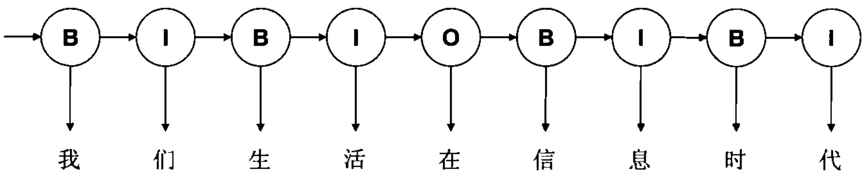

Figure 2.2: An HMM example

a transition probability, and every arrow from one state to its observation introduces an emission probability.

In HMM, only the observation, not the state, is directly observable, and the data itself does not tell us which state $x_i$ is linked with a particular observation. An HMM computes $\Pr(x_1, x_2, \dots, x_T, o_1, o_2, \dots, o_T)$ where the state sequence is hidden. Once we have an HMM $\lambda$ and an observation sequence $\mathbf{o} = o_1, o_2, \dots, o_T$, there are three problems of interest as originally stated in [32]:

- **The Evaluation Problem:** What is the probability that the observations are generated by the model? In other words, what is $\Pr(\mathbf{o} \mid \lambda)$?
- **The Decoding Problem:** What is the most likely state sequence $x_1^*, x_2^*, \dots, x_T^*$ in the model that produced the observation? In other words, we want to find the state sequence that satisfies $\underset{x_1, x_2, \dots, x_T}{\operatorname{argmax}} \Pr(x_1, x_2, \dots, x_T, o_1, o_2, \dots, o_T)$.
- **The Learning Problem:** How should we adjust the model parameters in order to maximize $\Pr(\mathbf{o} \mid \lambda)$?

There are some crucial assumptions made in HMMs. First, in first order HMMs, the next state is dependent only on the current state. That is, $a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)$. Even though the next state may depend on the past $k$ states in a $k^{th}$ order HMM, due to the high computational complexity, first order HMMs are the most commonly applied model. Second, it is assumed that state transition probabilities are independent of the actual time at which the transitions take place. That is, $P(q_{t_1+1} = s_j \mid q_{t_1} = s_i) = P(q_{t_2+1} = s_j \mid q_{t_2} = s_i)$ for any $t_1$ and $t_2$. Third, the current observation is statistically independent of the previous observations. Mathematically, $\Pr(o_1, o_2, \dots, o_t \mid q_1, q_2, \dots, q_t) = \prod_{i=1}^{t} P(o_i \mid q_i)$.

The Chinese word segmentation task can be treated as an example of the decoding problem, finding the most likely state(tag) sequence given the observed character sequence:

$$x_1^*, x_2^*, \ldots, x_T^* = \operatorname*{argmax}_{x_1, x_2, \ldots, x_T} \Pr(x_1, x_2, \ldots, x_T, o_1, o_2, \ldots, o_T) \tag{2.1}$$

where $\Pr(x_1, x_2, \ldots, x_T, o_1, o_2, \ldots, o_T) = \prod_{i=1}^{t} P(x_{i+1} \mid x_i) \times P(o_i \mid x_i)$ by the above Markov assumptions.

A formal technique to solve this decoding problem is the Viterbi algorithm [45], a dynamic programming method. The Viterbi algorithm operates on a finite number of states. At any time, the system is in some state, represented as a node. Multiple sequences of states can lead to a particular state. In any stage, the algorithm examines all possible paths leading to a state and only the most likely path is kept and used in exploring the most likely path toward the next state. At the end of the algorithm, by traversing backward along the most likely path, the corresponding state sequence can be found. Figure 2.3 shows the pseudo-code for the Viterbi algorithm.

```
Initialization:
  for $i$ = 1, ..., N do
    $\phi_1(i) = \pi_i \cdot b_i(o_1)$
    $s_1(i) = i$
  end for
Recursion:
  for $t$ = 1, ..., T-1 and $j$ = 1, ..., N do
    $\phi_{t+1}(j) = max_{i=1,\ldots,N}(\phi_t(i) \cdot a_{ij} \cdot b_j(o_t))$
    $s_{t+1}(j) = s_t(\tilde{i})$.append($j$), where $\tilde{i} = \text{argmax}_{i=1,\ldots,N}(\phi_t(i) \cdot a_{ij} \cdot b_j(o_t))$
  end for
Termination:
  $p^* = max_{i=1,\ldots,N}(\phi_T(i))$
  $s^* = s_T(\tilde{i})$, where $\tilde{i} = \text{argmax}_{i=1,\ldots,N}(\phi_T(i))$, and $s^*$ is the optimal state sequence.
```

Figure 2.3: The Viterbi algorithm

For example, suppose we have the character sequence “我(I)在(at)这(here)里(in)” (I am here). Table 2.2 shows the transition matrix, and Table 2.3 shows the emission matrix. Suppose the initial state distribution is $\{\pi_B = 0.5, \pi_I = 0.0, \pi_O = 0.5\}$.

The Viterbi algorithm to find the most likely tag sequence for this sentence proceeds as

| | B | I | O |
|---|---|---|---|
| B | 0.0 | 1.0 | 0.0 |
| I | 0.3 | 0.5 | 0.2 |
| O | 0.8 | 0.0 | 0.2 |

Table 2.2: Transition matrix

| | B | I | O |
|---|---|---|---|
| 我(I) | 0.4 | 0.2 | 0.4 |
| 在(at) | 0.1 | 0.1 | 0.8 |
| 这(here) | 0.6 | 0.2 | 0.2 |
| 里(in) | 0.2 | 0.7 | 0.1 |

Table 2.3: Emission matrix

follows:

1. Suppose we have an initial start state $s$. For the first observation character “我(I)”, the score $Score(\{o_1^B\})$ for leading to the state “B” equals

$$\pi_B \times p(o_1 = \text{我} \mid x_1 = B) = 0.5 \times 0.4 = 0.2$$

Similarly, the score $Score(\{o_1^O\})$ for leading to the state “O” can be calculated, and it equals 0.2. Since there is no transition from the start state $s$ to the state “I”, its corresponding path is ignored. This step is shown in Figure 2.4, in which the bold arrows represent the most likely paths towards each of the two possible states “B” and “O”.

2. For the second observation character “在(at)”, the path score from each of the previous paths to each of the current possible states is examined. For example, the path score $Score(\{o_1^B, o_2^I\})$ is calculated as

$$Score(\{o_1^B\}) \times p(x_2 = I \mid x_1 = B) \times p(o_2 = \text{在} \mid x_2 = I) = 0.2 \times 1.0 \times 0.1 = 0.02$$

Similarly, $Score(\{o_1^B, o_2^B\})$, $Score(\{o_1^B, o_2^O\})$, $Score(\{o_1^O, o_2^B\})$, $Score(\{o_1^O, o_2^I\})$, and $Score(\{o_1^O, o_2^O\})$ are calculated as well, and the most likely paths towards each of the three possible states “B”, “I” and “O” are recorded (See Figure 2.5).

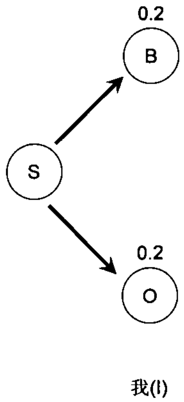

Figure 2.4: Step 1 of the Viterbi algorithm for the example

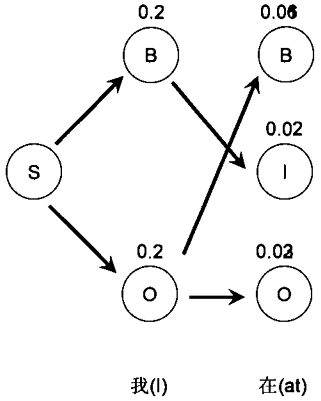

Figure 2.5: Step 2 of the Viterbi algorithm for the example

3. Continuing this procedure for each of the remaining observations "这(here)" and "里(in)", we eventually reach the final state *f*, and the best path to each intermediate state is produced (See Figure 2.6).

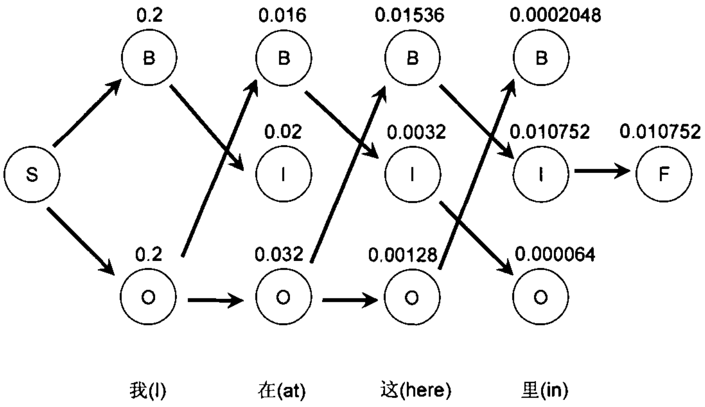

Figure 2.6: Step 3.2 of the Viterbi algorithm for the example

4. Then, starting from this final state *f*, we traverse back along the arrows in bold so that the optimal path, which is the tag sequence we would like to get, is generated (See Figure 2.7). In our example, it is the tag sequence "我-O 在-O 这-B 里-I", representing the segmentation result "我/在/这里"(I am here).

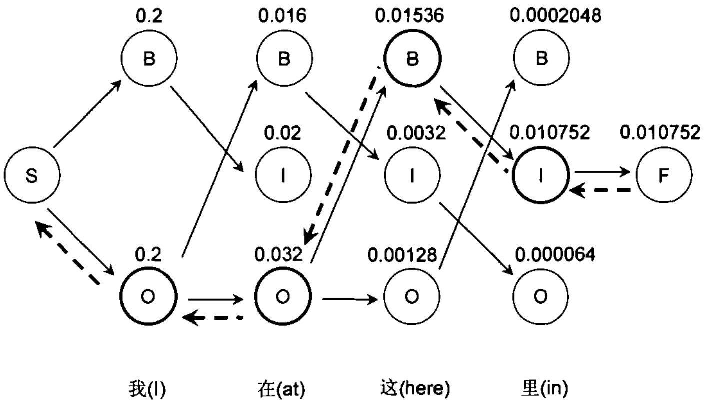

Figure 2.7: Back Traversing Step of the Viterbi algorithm for the example

Although the generative model has been employed in a wide range of applications [1, 17, 29, 44], the model itself, especially a higher order HMM, has some limitations due to the complexity in modeling $\Pr(\mathbf{x})$ which may contain many highly dependent features which are difficult to model while retaining tractability. To reduce such complexity, in the first order HMM, the next state is dependent only on the current state, and the current observation is independent of previous observations. These independence assumptions, on the other hand, seriously hurt the performance [2].

### 2.2.2 CRF - A Discriminative Model

Different from generative models that are used to represent the joint probability distribution $\Pr(\mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ is a random variable over data sequences to be labeled, and where $\mathbf{y}$ is a random variable over corresponding label sequences, a discriminative model directly models $\Pr(\mathbf{y} \mid \mathbf{x})$, the conditional probability of a label sequence given an observation sequence, and it aims to select the label sequence that maximizes this conditional probability. For many NLP tasks, the current most popular method to model this conditional probability is using the Conditional Random Field (CRF) [24] framework. The prime advantage of CRF over the generative model HMM is that it is no longer necessary to retain the independence assumptions. Therefore, rich and overlapping features can be included in this discriminative model.

Here is the formal definition of CRF, described by Lafferty et al. in [24]:

**Definition** Let $\mathbf{g} = (\mathbf{v}, \mathbf{e})$ be a graph such that $\mathbf{y} = (\mathbf{y}_v)_{v \in \mathbf{v}}$, so that $\mathbf{y}$ is indexed by the vertices of $\mathbf{g}$. Then $(\mathbf{x}, \mathbf{y})$ is a conditional random field in case, when conditioned on $\mathbf{x}$, the random variable $\mathbf{y}_v$ obey the Markov property with respect to the graph: $P(\mathbf{y}_v \mid \mathbf{x}, \mathbf{y}_w, w \neq v) = P(\mathbf{y}_v \mid \mathbf{x}, \mathbf{y}_w, w \sim v)$, where $w{\sim}v$ means that $w$ and $v$ are neighbors in $\mathbf{g}$.

As seen from the definition, CRF globally conditions on the observation $\mathbf{x}$, and thus, arbitrary features can be incorporated freely. In the usual case of sequence modeling, $\mathbf{g}$ is a simple linear chain, and its graphical structure is shown in Figure 2.8.

The conditional distribution $\Pr(\mathbf{y} \mid \mathbf{x})$ of a CRF follows from the joint distribution $\Pr(\mathbf{x}, \mathbf{y})$ of an HMM. This explanation of CRF is taken from [43]. Consider the HMM joint probability equation:

$$\Pr(\mathbf{x}, \mathbf{y}) = \prod_{t=1}^{T} \Pr(y_t \mid y_{t-1}) \Pr(x_t \mid y_t) \tag{2.2}$$

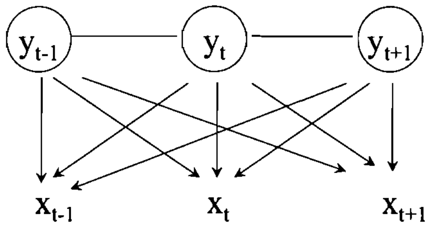

Figure 2.8: Graphical structure of a chained CRF

We rewrite Equation 2.2 as

$$\Pr(\mathbf{x},\mathbf{y}) = \frac{1}{Z}\exp\left\{\sum_t \sum_{i,j\in \mathrm{s}} \lambda_{ij}\delta(y_t = i)\delta(y_{t-1} = j) + \sum_t \sum_{i\in \mathrm{s}} \sum_{o\in \mathrm{o}} \mu_{oi}\delta(y_t = i)\delta(x_t = o)\right\} \tag{2.3}$$

where $\theta = \{\lambda_{ij},\ \mu_{oi}\}$ are the parameters of the distribution, and they can be any real numbers. $\delta$ represents a set of features. For instance,

$$\delta(y_t = B) = \begin{cases} 1 & \text{If } y_t \text{ is assigned the tag B} \\ 0 & \text{Otherwise} \end{cases}$$

Every HMM can be written in Equation 2.3 by setting $\lambda_{ij} = \log P(y_t = i \mid y_{t-1} = j)$ and $\mu_{oi} = \log P(x_t = o \mid y_t = i)$. Z is a normalization constant to guarantee that the distribution sums to one.

If we introduce the concept of feature functions:

$$f_k(y_t, y_{t-1}, x_t) = \begin{cases} f_{ij}(y, y', x) = \delta(y = i)\delta(y' = j) & \text{for each transition } (i,j) \\ f_{io}(y, y', x) = \delta(y = i)\delta(x = o) & \text{for each state-observation pair } (i,o) \end{cases}$$

then Equation 2.3 can be rewritten more compactly as follows:

$$P(\mathbf{x},\mathbf{y}) = \frac{1}{Z}\exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right\}. \tag{2.4}$$

Equation 2.4 defines exactly the same family of distributions as Equation 2.3, and therefore as the original HMM Equation 2.2.

To derive the conditional distribution $P(\mathbf{y} \mid \mathbf{x})$ from the HMM Equation 2.4, we write

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{P(\mathbf{x},\mathbf{y})}{\sum_{y'} P(\mathbf{x},\mathbf{y}')} = \frac{\exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right\}}{\sum_{y'} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y'_t, y'_{t-1}, x_t)\right\}} \tag{2.5}$$

This conditional distribution in Equation 2.5 is a linear chain, in particular one that includes features only for the current word's identity. At this point, the graphical structure for the CRF is almost identical to the HMM, allowing features that condition only on the current word. However, the graphical structure can be generalized slightly to allow each feature function to optionally condition on the entire input sequence. This new graphical structure is shown in Figure 2.9. This leads to the general definition of linear chain CRFs:

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, \mathbf{x})\right\}, \tag{2.6}$$

where the subscript $t$ in $y_{t-1}$ and $y_t$ refers to the graphical structure of the linear-chain CRF as in Figure 2.9, and $Z(\mathbf{x})$ is an instance-specific normalization function

$$Z(\mathbf{x}) = \sum_y \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, \mathbf{x})\right\}.$$

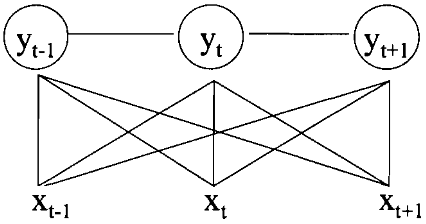

Figure 2.9: Graphical structure of a general chained CRF

For example, in Chinese word segmentation, given an input sentence $\mathbf{x}$=都至关重要, the CRF calculates conditional probabilities of different tagging sequences such as

$$p(\text{tagging} = SBMME \mid \text{input} = \mathbf{x}) = \frac{\exp\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, \mathbf{x})\}}{Z(\text{all possible taggings for } \mathbf{x})},$$

and picks the tag sequence that gives the highest probability. During this calculation, various features are applied. For instance, one possible feature $f_k$ might be

$$f_{100}(y_t, y_{t-1}, \mathbf{x}) = \begin{cases} 1 & \text{if } y_{t-1}\text{=B, } y_t\text{=M, } x_{-2}\text{=至(to), } x_0\text{=重(heavy)} \\ 0 & \text{otherwise} \end{cases}$$

where "$x_{-2}$=至(to)" represents that the character two positions to the left of the current character is the character "至(to)", and where "$x_0$=重(heavy)" represents that the current character is "重(heavy)". The estimation of the parameters $\lambda$ is typically performed by penalized maximum likelihood. For a conditional distribution and training data $D = \{\mathbf{x}^i, \mathbf{y}^i\}_{i=1}^N$, where each $\mathbf{x}^i$ is a sequence of inputs, and each $\mathbf{y}^i$ is a sequence of the desired predictions, we want to maximize the following log likelihood, sometimes called the conditional log likelihood:

$$L(\lambda) = \sum_{i=1}^{N} \log p(\mathbf{y}^i \mid \mathbf{x}^i). \tag{2.7}$$

After substituting the CRF model (Equation 2.6) into this likelihood (Equation 2.7) and applying regularization to avoid over-fitting, we get the expression:

$$L(\lambda) = \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}^{(i)}) - \sum_{t=1}^{T} \log Z(\mathbf{x}^{(i)}) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2}. \tag{2.8}$$

Regularization in Equation 2.8 is given by the last term $\sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2}$. To optimize $L(\lambda)$, approaches such as the steepest ascent along the gradient, Newton's method, or BFGS algorithm, can be applied.

CRF has been widely adopted in natural language processing tasks. For example, part-of-speech tagging with CRF such as in [11], base noun-phrase chunking with CRF such as in [36], or named entity extraction with CRF such as in [21, 55]. Moreover, the toolkit, *CRF++*[4] [23], coded in C++ programming language, has successfully implemented the CRF framework for sequence learning, and it is used extensively in our experiments.

## 2.3 Global Linear Models

For sequence learning approaches, tagged training sentences are broken into series of decisions, each associated with a probability. Parameter values are estimated, and tags for test sentences are chosen based on related probabilities and parameter values. While a global linear model [9] computes a global score based on various features.

A global linear model is defined as follows: Let $\mathbf{x}$ be a set of inputs, and $\mathbf{y}$ be a set of possible outputs. For instance, $\mathbf{x}$ could be unsegmented Chinese sentences, and $\mathbf{y}$ could be the set of possible word segmentation corresponding to $\mathbf{x}$.

- Each $y \in \mathbf{y}$ is mapped to a $d$-dimensional feature vector $\Phi(x,y)$, with each dimension being a real number, summarizing partial information contained in $(x,y)$.

[4] available from http://crfpp.sourceforge.net/

- A weight parameter vector $\mathbf{w} \in \Re^d$ assigns a weight to each feature in $\Phi(x,y)$, representing the importance of that feature. The value of $\Phi(x,y) \cdot \mathbf{w}$ is the score of $(x,y)$. The higher the score, the more plausible it is that $y$ is the output for $x$.
- In addition, we have a function $GEN(x)$, generating the set of possible outputs $y$ for a given $x$.

Having $\Phi(x,y)$, $\mathbf{w}$, and $GEN(x)$ specified, we would like to choose the highest scoring candidate $y^*$ from $GEN(x)$ as the most plausible output. That is,

$$F(x) = \operatorname*{argmax}_{y \in GEN(x)} \Phi(x,y) \cdot \mathbf{w} \tag{2.9}$$

where $F(x)$ returns the highest scoring output $y^*$ from $GEN(x)$.

To set the weight parameter vector $\mathbf{w}$, different kinds of learning methods have been applied. Here, we describe two general types of approaches for training $\mathbf{w}$: the perceptron learning approach and the exponentiated gradient approach.

### 2.3.1 Perceptron Learning Approach

A perceptron [34] is a single-layered neural network. It is trained using online learning, that is, processing examples one at a time, during which it adjusts a weight parameter vector that can then be applied on input data to produce the corresponding output. The weight adjustment process awards features appearing in the truth and penalizes features not contained in the truth. After the update, the perceptron ensures that the current weight parameter vector is able to correctly classify the present training example.

Suppose we have $m$ examples in the training set. The original perceptron learning algorithm [34] is shown in Figure 2.10.

The weight parameter vector $\mathbf{w}$ is initialized to $\mathbf{0}$. Then the algorithm iterates through those $m$ training examples. For each example $x$, it generates a set of candidates $GEN(x)$, and picks the most plausible candidate, which has the highest score according to the current $\mathbf{w}$. After that, the algorithm compares the selected candidate with the truth, and if they are different from each other, $\mathbf{w}$ is updated by increasing the weight values for features appearing in the truth and by decreasing the weight values for features appearing in this top candidate. If the training data is linearly separable, meaning that it can be discriminated by a function which is a linear combination of features, the learning is proven to converge in a finite number of iterations [13].

```
Inputs: Training Data $\langle(x_1, y_1), \ldots, (x_m, y_m)\rangle$; number of iterations T
Initialization: Set $\mathbf{w} = \mathbf{0}$
Algorithm:
  for $t = 1, \ldots, T$ do
    for $i = 1, \ldots, m$ do
      Calculate $y_i'$, where $y_i' = \operatorname*{argmax}_{y \in GEN(x)} \Phi(x_i, y) \cdot \mathbf{w}$
      if $y_i' \neq y_i$ then
        $\mathbf{w} = \mathbf{w} + \Phi(x_i, y_i) - \Phi(x_i, y_i')$
      end if
    end for
  end for
Output: The updated weight parameter vector $\mathbf{w}$
```

Figure 2.10: The original perceptron learning algorithm

This original perceptron learning algorithm is simple to understand and to analyze. However, the incremental weight updating suffers from over-fitting, which tends to classify the training data better, at the cost of classifying the unseen data worse. Also, the algorithm is not capable to deal with training data that is linearly inseparable.

Freund and Schapire [13] proposed a variant of the perceptron learning approach — the voted perceptron algorithm. Instead of storing and updating parameter values inside one weight vector, its learning process keeps track of all intermediate weight vectors, and these intermediate vectors are used in the classification phase to vote for the answer. The intuition is that good prediction vectors tend to survive for a long time and thus have larger weight in the vote. Figure 2.11 shows the voted perceptron training and prediction phases from [13], with slightly modified representation.

The voted perceptron keeps a count $c_i$ to record the number of times a particular weight parameter vector $(\mathbf{w}_i, c_i)$ survives in the training. For a training example, if its selected top candidate is different from the truth, a new count $c_{i+1}$, being initialized to 1, is used, and an updated weight vector $(\mathbf{w}_{i+1}, c_{i+1})$ is produced; meanwhile, the original $c_i$ and weight vector $(\mathbf{w}_i, c_i)$ are stored.

Compared with the original perceptron, the voted perceptron is more stable, due to maintaining the list of intermediate weight vectors for voting. Nevertheless, to store those weight vectors is space inefficient. Also, the weight calculation, using all intermediate weight parameter vectors during the prediction phase, is time consuming.

```
Training Phase
Input: Training data $\langle(x_1, y_1), \ldots, (x_m, y_m)\rangle$, number of iterations T
Initialization: k = 0, $\mathbf{w}_0 = \mathbf{0}$, $c_1 = 0$
Algorithm:
  for $t = 1, \ldots, T$ do
    for $i = 1, \ldots, m$ do
      Calculate $y_i^{'}$, where $y_i^{'} = \underset{y \in GEN(x)}{\text{argmax}}\ \Phi(x_i, y) \cdot \mathbf{w}_k$
      if $y_i^{'} = y_i$ then
        $c_k = c_k + 1$
      else
        $\mathbf{w}_{k+1} = \mathbf{w}_k + \Phi(x_i, y_i) - \Phi(x_i, y_i^{'})$
        $c_{k+1} = 1$
        $k = k + 1$
      end if
    end for
  end for
Output: A list of weight vectors $\langle(\mathbf{w}_1, c_1), \ldots, (\mathbf{w}_k, c_k)\rangle$

Prediction Phase
Input: The list of weight vectors $\langle(\mathbf{w}_1, c_1), \ldots, (\mathbf{w}_k, c_k)\rangle$, an unsegmented sentence x
Calculate:
    $y^* = \underset{y \in GEN(x)}{\text{argmax}} \left( \sum_{i=1}^{k} c_i \Phi(x, y) \cdot \mathbf{w}_i \right)$
Output: The voted top ranked candidate $y^*$
```

Figure 2.11: The voted perceptron algorithm

The averaged perceptron algorithm [7], an approximation to the voted perceptron, on the other hand, maintains the stability of the voted perceptron algorithm, but significantly reduces space and time complexities. In an averaged version, rather than using $\mathbf{w}$, the averaged weight parameter vector $\gamma$ over the $m$ training examples is used for future predictions on unseen data:

$$\gamma = \frac{1}{\mathrm{mT}} \sum_{i=1\ldots m,t=1\ldots T} \mathbf{w}^{i,t}$$

In calculating $\gamma$, an accumulating parameter vector $\sigma$ is maintained and updated using $\mathbf{w}$ for each training example. After the last iteration, $\sigma/(\mathrm{mT})$ produces the final parameter vector $\gamma$. The entire algorithm is shown in Figure 2.12.

```
Inputs: Training Data $\langle(x_1, y_1), \ldots, (x_m, y_m)\rangle$; number of iterations T
Initialization: Set $\mathbf{w} = \mathbf{0}$, $\gamma = \mathbf{0}$, $\sigma = \mathbf{0}$
Algorithm:
  for $t = 1, \ldots, T$ do
    for $i = 1, \ldots, m$ do
      Calculate $y_i^{'}$, where $y_i^{'} = \operatorname*{argmax}_{y \in GEN(x)} \Phi(x_i, y) \cdot \mathbf{w}$
      if $y_i^{'} \neq y_i$ then
        $\mathbf{w} = \mathbf{w} + \Phi(x_i, y_i) - \Phi(x_i, y_i^{'})$
      end if
      $\sigma = \sigma + \mathbf{w}$
    end for
  end for
Output: The averaged weight parameter vector $\gamma = \sigma/(\mathrm{mT})$
```

Figure 2.12: The averaged perceptron learning algorithm

When the number of features is large, it is expensive to calculate the total parameter $\sigma$ for each training example. To further reduce the time complexity, Collins [8] proposed the lazy update procedure. After processing each training sentence, not all dimensions of $\sigma$ are updated. Instead, an update vector $\tau$ is used to store the exact location $(p,t)$ where each dimension of the averaged parameter vector was last updated, and only those dimensions corresponding to features appearing in the current sentence are updated. $p$ represents the training example index where this particular feature was last updated, and $t$ represents its corresponding iteration number. While for the last example in the final iteration, each dimension of $\tau$ is updated, no matter whether the candidate output is correct or not. Figure 2.13 shows the averaged perceptron with lazy update procedure.

### 2.3.2 Exponentiated Gradient Approach

Different from the perceptron learning approach, the exponentiated gradient (EG) method [22] formulates the problem directly as the margin maximization problem. A set of dual variables $\alpha_{i,y}$ is assigned to data points $\mathbf{x}$. Specifically, to every point $x_i \in \mathbf{x}$, there corresponds a distribution $\alpha_{i,y}$ such that $\alpha_{i,y} \geq 0$ and $\sum_y \alpha_{i,y} = 1$. The algorithm attempts to optimize these dual variables $\alpha_{i,y}$ for each $i$ separately. In the word segmentation case, $x_i$ is a training example, and $\alpha_{i,y}$ is the dual variable corresponding to each possible segmented output $y$ for $x_i$.

Similar to the perceptron, the goal in the EG approach is to find

$$F(x) = \underset{y \in GEN(x)}{\text{argmax}} \ \Phi(x, y) \cdot \mathbf{w}$$

as well, and the weight parameter vector $\mathbf{w}$ is expressed as

$$\mathbf{w} = \sum_{i,y} \alpha_{i,y} \left[\Phi(x_i, y_i) - \Phi(x_i, y)\right] \qquad (2.10)$$

where $\alpha_{i,y}$s are dual variables to be optimized during the EG update process.

Given a training set $\{(x_i, y_i)\}_{i=1}^n$ and the weight parameter vector $\mathbf{w}$, the margin on the segmentation candidate $y$ for the $i^{th}$ training example is defined as the difference in score between the true segmentation and the candidate $y$. That is,

$$M_{i,y} = \Phi(x_i, y_i) \cdot \mathbf{w} - \Phi(x_i, y) \cdot \mathbf{w} \qquad (2.11)$$

For each dual variable $\alpha_{i,y}$, a new $\alpha'_{i,y}$ is obtained as

$$\alpha'_{i,y} \leftarrow \frac{\alpha_{i,y} e^{\eta \nabla_{i,y}}}{\sum_y \alpha_{i,y} e^{\eta \nabla_{i,y}}} \qquad (2.12)$$

where

$$\nabla_{i,y} = \begin{cases} 0 & \text{for } y = y_i \\ 1 - M_{i,y} & \text{for } y \neq y_i \end{cases}$$

```
Inputs: Training Data $\langle(x_1, y_1), \ldots, (x_m, y_m)\rangle$; number of iterations T
Initialization: Set $\mathbf{w} = \mathbf{0}$, $\gamma = \mathbf{0}$, $\sigma = \mathbf{0}$, $\tau = \mathbf{0}$
Algorithm:
 for $t$ = 1,..., T do
   for $i$ = 1,..., m do
     Calculate $y_i'$, where $y_i' = \underset{y \in GEN(x)}{\text{argmax}}\ \Phi(x_i, y) \cdot \mathbf{w}$
     if $t \neq$ T or $i \neq$ m then
       if $y_i' \neq y_i$ then
         // Update active features in the current sentence
         for each dimension $s$ in $(\Phi(x_i, y_i) - \Phi(x_i, y_i'))$ do
           if $s$ is a dimension in $\tau$ then
             // Include the total weight during the time
             // this feature remains inactive since last update
             $\sigma_s = \sigma_s + w_s \cdot (t \cdot m + i - t_{\tau_s} \cdot m - i_{\tau_s})$
           end if
           // Also include the weight calculated from comparing $y_i'$ with $y_i$
           $w_s = w_s + \Phi(x_i, y_i) - \Phi(x_i, y_i')$
           $\sigma_s = \sigma_s + \Phi(x_i, y_i) - \Phi(x_i, y_i')$
           // Record the location where the dimension $s$ is updated
           $\tau_s = (i, t)$
         end for
       end if
     else
       // To deal with the last sentence in the last iteration
       for each dimension $s$ in $\tau$ do
         // Include the total weight during the time
         // each feature in $\tau$ remains inactive since last update
         $\sigma_s = \sigma_s + w_s \cdot (T \cdot m + m - t_{\tau_s} \cdot m - i_{\tau_s})$
       end for
       // Update weights for features appearing in this last sentence
       if $y_i' \neq y_i$ then
         $\mathbf{w} = \mathbf{w} + \Phi(x_i, y_i) - \Phi(x_i, y_i')$
         $\sigma = \sigma + \Phi(x_i, y_i) - \Phi(x_i, y_i')$
       end if
     end if
   end for
 end for
Output: The averaged weight parameter vector $\gamma = \sigma/(\text{mT})$
```

Figure 2.13: The averaged perceptron learning algorithm with lazy update procedure

and $\eta$ is the learning rate which is positive and controls the magnitude of the update.

With these general definitions, Globerson et al. [15] proposed the EG algorithm with two schemes: the batch scheme and the online scheme. Suppose we have $m$ training examples. In the batch scheme, at every iteration, the $\alpha_i$s are simultaneously updated for all $i = 1, \dots, m$ before the weight parameter vector $\mathbf{w}$ is updated; while for the online scheme, at each iteration, a single $i$ is chosen and its $\alpha_i$'s are updated before the weight parameter vector $\mathbf{w}$ is updated. The pseudo-code for the batch scheme is given in Figure 2.14, and that for the online scheme is given in Figure 2.15.

```
Inputs: Training Data $\langle(x_1, y_1), \dots, (x_m, y_m)\rangle$; learning rate $\eta > 0$; number of iterations T
Initialization: Set $\alpha_{i,y}$ to initial values; calculate $\mathbf{w} = \sum_{i,y} \alpha_{i,y} [\Phi(x_i, y_i) - \Phi(x_i, y)]$
Algorithm:
  for $t = 1, \dots, T$ do
    for $i = 1, \dots, m$ do
      Calculate Margins: $\forall y$, $M_{i,y} = \Phi(x_i, y_i) \cdot \mathbf{w} - \Phi(x_i, y) \cdot \mathbf{w}$
    end for
    for $i = 1, \dots, m$ do
      Update Dual Variables: $\forall y$, $\alpha'_{i,y} \leftarrow \frac{\alpha_{i,y} e^{\eta \nabla_{i,y}}}{\sum_y \alpha_{i,y} e^{\eta \nabla_{i,y}}}$
    end for
    Update Weight Parameters: $\mathbf{w} = \sum_{i,y} \alpha'_{i,y} [\Phi(x_i, y_i) - \Phi(x_i, y)]$
  end for
Output: The weight parameter vector $\mathbf{w}$
```

Figure 2.14: The batch EG algorithm

Hill and Williamson [16] analyzed the convergence of the EG algorithm, and Collins [7] also pointed out that the algorithm converges to the minimum of

$$\sum_i \max_y (1 - M_{i,y})_+ + \frac{1}{2}\|\mathbf{w}\|^2 \tag{2.13}$$

where

$$(1 - M_{i,y})_+ = \begin{cases} (1 - M_{i,y}) & \text{if } (1 - M_{i,y}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

While in the dual optimization representation, the problem becomes choosing $\alpha_{i,y}$ values to maximize

$$Q(\alpha) = \sum_{i,y \neq y_i} \alpha_{i,y} - \frac{1}{2}\|\mathbf{w}\|^2 \tag{2.14}$$

```
Inputs: Training Data $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$; learning rate $\eta > 0$; number of iterations T
Initialization: Set $\alpha_{i,y}$ to initial values; calculate $\mathbf{w} = \sum_{i,y} \alpha_{i,y} [\Phi(x_i, y_i) - \Phi(x_i, y)]$
Algorithm:
  for $t = 1, \ldots, T$ do
    for $i = 1, \ldots, m$ do
      Calculate Margins: $\forall y$, $M_{i,y} = \Phi(x_i, y_i) \cdot \mathbf{w} - \Phi(x_i, y) \cdot \mathbf{w}$
      Update Dual Variables: $\forall y$, $\alpha'_{i,y} \leftarrow \frac{\alpha_{i,y} e^{\eta \nabla_{i,y}}}{\sum_y \alpha_{i,y} e^{\eta \nabla_{i,y}}}$
      Update Weight Parameters: $\mathbf{w} = \sum_{i,y} \alpha'_{i,y} [\Phi(x_i, y_i) - \Phi(x_i, y)]$
    end for
  end for
Output: The weight parameter vector $\mathbf{w}$
```

Figure 2.15: The online EG algorithm

where

$$\mathbf{w} = \sum_{i,y} \alpha_{i,y} [\Phi(x_i, y_i) - \Phi(x_i, y)]$$

Here is a simple example for updating dual variables and the weight parameter vector in one iteration. Suppose for a particular training example, there are four segmentation candidates, each with its feature vector **f** containing three features $f_1$, $f_2$ and $f_3$. Let's assume that segmentation candidate 1 (i.e. Seg #1) is the truth. Table 2.4 shows the update that occurs in one iteration.

| | Seg #1 (truth) | Seg #2 | Seg #3 | Seg #4 |
|---|---|---|---|---|
| **f** | $\{f_1 = 2, f_2 = 2\}$ | $\{f_1 = 2, f_3 = 1\}$ | $\{f_2 = 1, f_3 = 1\}$ | $\{f_1 = 1, f_2 = 3\}$ |
| Initial $\alpha$ | 0.25 | 0.25 | 0.25 | 0.25 |
| Initial **w** | {0.5, 1.5, -1.25} | | | |
| Margin | 0 | 1.5 | 2 | 6.75 |
| $\nabla$ | 0 | 1 - 1.5 = -0.5 | 1 - 2 = -1 | 1 - 6.75 = -5.75 |
| $e^{\nabla}$, with $\eta = 1$ | $e^0 = 1$ | $e^{-0.5} \approx 0.61$ | $e^{-1} \approx 0.37$ | $e^{-5.75} \approx 0.0032$ |
| Updated $\alpha$ | 0.5042 | 0.3076 | 0.1866 | 0.0016 |
| Updated **w** | {0.1882, 0.3764, -0.1914} | | | |

Table 2.4: Example for updating dual variables and the weight parameter vector in EG

## 2.4 Summary

In this chapter, various general methods that deal with Chinese word segmentation have been explained. The dictionary-based matching methods, used in [12, 25, 39, 52], are simple and efficient. However, the performance is dependent on the size of the dictionary. On the other hand, sequence learning approach does not carry out word matching, but rather it considers segmentation as a character or subword tagging task, attaching probabilities to tagging decisions. This state-of-the-art type of approach is applied extensively in segmentation systems such as [24, 29, 30, 48, 52, 54]. In addition, global linear models compute global scores based on features computed over the whole sentence [19, 26, 53].

# Chapter 3

# Majority Voting Approach

In this chapter, we discuss our Chinese word segmentation system, which is based on majority voting among three models: a greedy longest matching model, a conditional random field (CRF) model with maximum subword-based tagging [52], and a CRF model with minimum subword-based tagging. In addition, our system contains a post-processing component to deal with data inconsistencies. Testing our system in the third SIGHAN bakeoff on the closed track of CityU, MSRA and UPUC corpora, we show that our majority voting method combines the strength from these three models and outperforms the individual methods.

## 3.1 Overall System Description

Our majority voting word segmentation system proceeds in three steps. In the first step, the greedy longest matching method, which is a dictionary-based matching approach, is used to generate a segmentation result. Also at the same time, the CRF model with maximum subword-based tagging and the CRF model with minimum subword-based tagging, both of which will be explained later in this chapter, are used individually to produce segmentation results. In the second step, a character-level majority voting method takes these three segmentation results as input and creates the initial output. In the last step, a post-processing procedure is applied on this initial output to correct certain data inconsistency errors and to get the final output. This post-processing procedure merges adjoining word candidates to match with the dictionary entries and splits word candidates which are inconsistent with entries in the training corpus. The overview of the whole system is shown in Figure 3.1.

This chapter is organized as follows: In Section 3.2, we provide a brief review of the

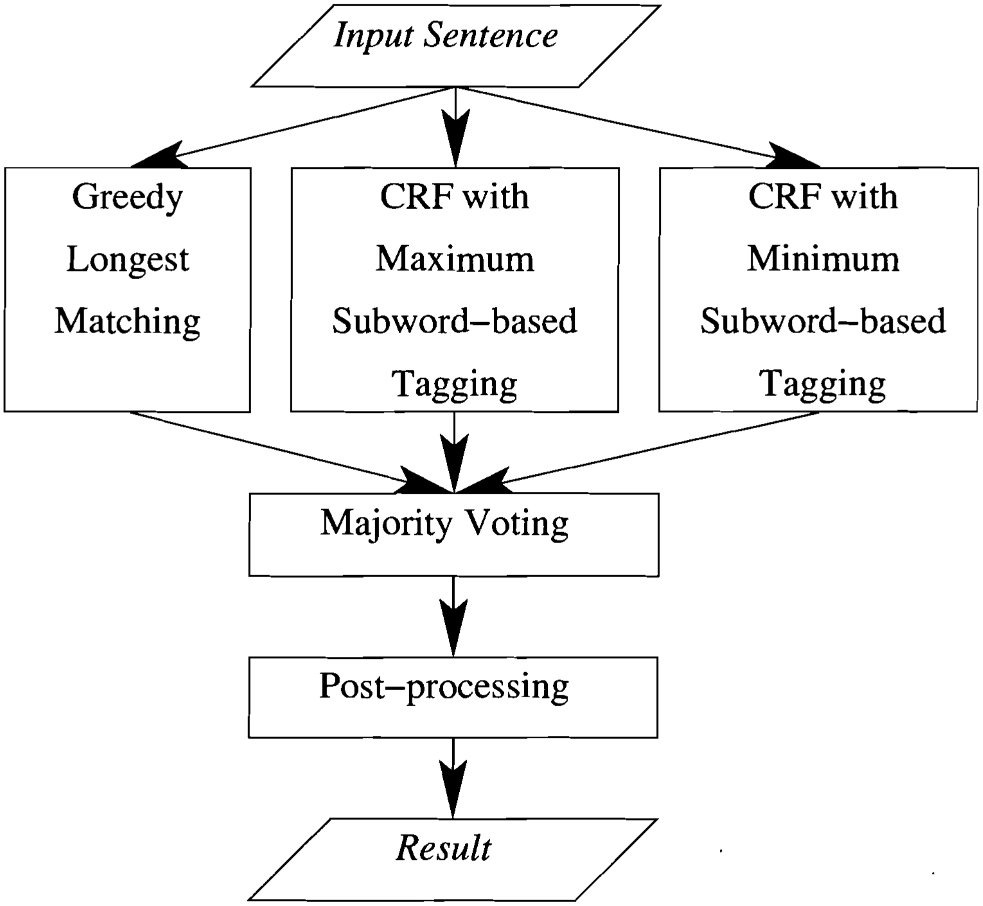
Input Sentence
Greedy Longest Matching
CRF with Maximum Subword–based Tagging
CRF with Minimum Subword–based Tagging
Majority Voting
Post–processing
Result

Figure 3.1: Overview of the majority voting segmentation system

greedy longest matching method. Section 3.3 describes the CRF model using maximum subword-based tagging. Section 3.4 describes the CRF model using minimum subword-based tagging, and in Section 3.5, the majority voting process is discussed. Section 3.6 talks about the post-processing step, attempting to correct the mistakes. Section 3.7 shows the experimental results and analyzes the errors. Section 3.8 briefly discusses a modified majority voting system in which minimum subword-based CRF-tagged candidate is substituted with character-based CRF-tagged candidate, while Section 3.9 summarizes this whole chapter.

## 3.2 Greedy Longest Matching

Recall from Chapter 2 that the greedy longest matching algorithm is a dictionary-based segmentation algorithm. It starts from the beginning of a sentence and proceeds through the whole sentence, attempting to find the longest sub-sequence of characters matching any dictionary word at each point. This algorithm is simple to understand, easy to implement, efficient, and it maximizes the usage of dictionary. However, limited dictionary sizes combined with frequent occurrences of unknown words become a major bottleneck if the greedy longest matching approach is applied by itself.

## 3.3 CRF Model with Maximum Subword-based Tagging

Conditional random field, the discriminative sequence learning method, has been widely used in various tasks [11, 21, 36, 55], including Chinese word segmentation [30, 52, 54]. In this approach, most existing systems apply the character-based tagger. For example, "都(all)/至关重要(extremely important)" is labeled as "都-O 至-B 关-I 重-I 要-I", using the 3-tagset.

In 2006, Zhang et al. [52] proposed a maximum subword-based "IOB" tagger for Chinese word segmentation. This tagger proceeds as follows:

- First, the entire word list is extracted from the training corpus. Meanwhile, the frequency count for each word in the list is recorded, and the words are sorted in decreasing order according to their frequency count.
- Next, all the single-character words and the most frequent multi-character words are extracted from this sorted list to form a lexicon subset.

- Then, this subset is applied on the training data to tag the whole corpus in subword format. For example, suppose we have the single-character words "都"(all), "至"(to) and "关"(close), and the most frequent multi-character word "重要"(important) in the lexical subset, then "都/至关重要" in the previous example is labeled as "都-O 至-B 关-I 重要-I" instead.

- After that, the tagged corpus is fed into *CRF++*, for training the discriminative model. At the same time, the test data is segmented with the greedy longest matching method, using the lexicon subset as the dictionary.

- In the last step, *CRF++* labels these initially segmented test data, according to the learnt model, to produce the final segmentation result.

We implemented this maximum subword-based CRF learning as one of our three systems to produce an initial segmentation for majority voting. Also, in all our experiments, we defined the most frequent words to be the top 5% in the sorted multi-character word list. Zhang et al. [52] observed in their experiments that, with the CRF-based tagging approach, a higher out-of-vocabulary recall rate is achieved, at the cost of getting a very low in-vocabulary recall rate. We claim that the majority voting procedure will take advantage of the dictionary information used in the greedy longest matching method and that of the frequent word information used in this maximum subword-based CRF model to raise the low in-vocabulary recall rate. Also, the voting procedure will benefit from the high out-of-vocabulary recall rate achieved from the CRF-based tagging algorithms.

The feature template for sequence learning in this method and in all our CRF-related experiments is adapted from [52] and summarized in Table 3.1, defining the symbol $c$ to be character in a character-based CRF method or word in a subword-based CRF method, and defining the symbol $t$ to be the observation. 0 means the current position; -1, -2, the first or second position to the left; 1, 2, the first or second position to the right.

## 3.4 CRF Model with Minimum Subword-based Tagging

In our third model, we apply a similar approach as in the previous model. However, instead of using the maximum subwords, we explore the minimum subword-based tagger. At the beginning, we build the dictionary using the whole training corpus. Without extracting the most frequent words as is done in the maximum subword tagger, the whole dictionary

| | |
|---|---|
| Word features | $c_0$<br>$c_{-1}$<br>$c_1$<br>$c_{-2}$<br>$c_2$<br>$c_{-1}c_0$<br>$c_0c_1$<br>$c_{-1}c_1$<br>$c_{-2}c_{-1}$<br>$c_0c_2$ |
| Context features | $t_{-1}t_0$ |

Table 3.1: CRF feature template

is used to recognize subwords with minimum length in the training corpus. Each training sentence is tagged in the same "IOB" format as before. Suppose "a", "ac", "de" and "acde" are the only entries in the dictionary, and let us assume that "ac" and "de" are the top frequent words appearing in the training corpus. Then, for the word "acde", here is the comparison between the maximum subword tagging and the minimum subword tagging:

- Maximum subword tagging: ac/B de/I
- Minimum subword tagging: a/B c/I de/I

After tagging the training corpus, the *CRF++* package is used to train this type of model, using feature templates identical to the ones in the maximum subword tagging approach.

Meanwhile, the greedy shortest matching algorithm is applied on the unsegmented test data to produce the initial segmentation. When the CRF model training finishes, this initial segmentation is fed into the CRF model for labeling, assigning the "IOB" tags for each entity.

The whole process in this step can be represented as shown in Figure 3.2.

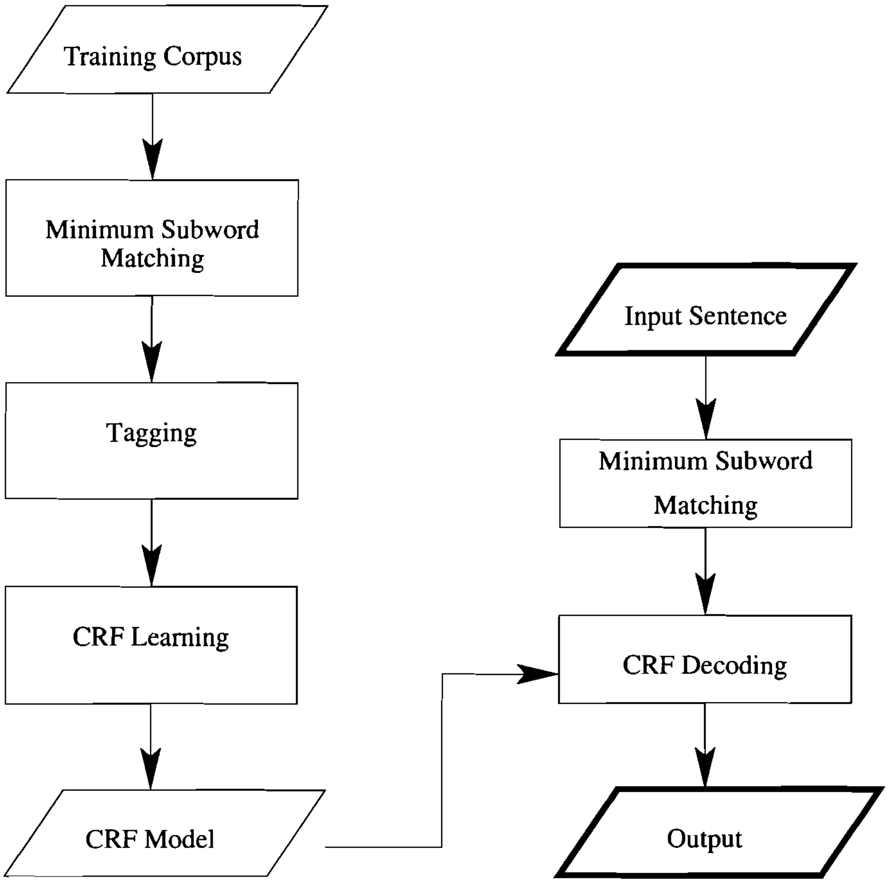

Figure 3.2: Overview of the minimum subword-based tagging approach

## 3.5 Character-level Majority Voting

### 3.5.1 The Voting Procedure

In this next step, with the segmentation results from the greedy longest matching, from the CRF model with maximum subword-based tagging, and from the CRF model with minimum subword-based tagging in hand, we apply the character-level majority voting algorithm. First, for each character in a segmented sentence, we tag it either as "B" if it is the first character of a word or a single-character word, or as "I" otherwise. (In word segmentation, there is no "O" tag required – it can be always replaced with "B" tag. So for example, "OBI" can be written as "BBI" without loss of information.) Then, for each corresponding character from a specific sentence, if at least two of the models provide the same tag, that specific tag will be assigned as the final tag for that character. For instance, considering the character sequence "acde", Table 3.2 illustrates the voting procedure. Suppose "a/c/de" is the segmentation result from the greedy longest matching. "acd/e" is the result from the CRF model with maximum subword-based tagging, and "ac/d/e" is the result from the CRF model with minimum subword-based tagging. Then, for "a", since all segmentation results assign "B' to it, "a" is tagged as "B"; for "c", because two of the segmentation methods tag it as "I", "c" is tagged as "I". Similarly, the tag for each remaining character is determined by this majority voting process, and we get "ac/d/e" as the final output in this example.

| Method | Output | | | |
|---|---|---|---|---|
| Greedy Longest Matching | a/**B** | c/B | d/**B** | e/I |
| CRF model with Maximum Subword Tagging | a/**B** | c/**I** | d/I | e/**B** |
| CRF model with Minimum Subword Tagging | a/**B** | c/**I** | d/**B** | e/**B** |
| VOTING RESULT | a/B | c/I | d/B | e/B |

Table 3.2: An example for the character-level majority voting

## 3.6 Post-processing Step

To test the performance of each of the three models and that of the majority voting, we divide the MSRA corpus into a training set and a held-out set. The training set takes 80%

of the whole MSRA training corpus sentences, and the held-out set takes the remaining 20% of the MSRA training corpus sentences. Table 3.3 shows the performance on the held-out set. Throughout the experiments we conduct, we discover that those two CRF models produced segmentation results with high F-scores, and that the voting process improves the performance further.

| | $P$ | $R$ | $F$ | $R_{OOV}$ | $R_{IV}$ |
|---|---|---|---|---|---|
| Greedy Longest Matching | 91.9 | 95.7 | 93.7 | 2.1 | 98.3 |
| CRF with Maximum Subword Tagging | 95.6 | 95.4 | 95.5 | 68.9 | 96.1 |
| CRF with Minimum Subword Tagging | 95.5 | 95.0 | 95.3 | 68.6 | 95.8 |
| Majority Voting | 96.2 | 96.3 | 96.2 | 65.2 | 97.1 |

Table 3.3: Performance (in percentage) on the MSRA held-out set

While analyzing errors with the segmentation result from the held-out set, we find two types of inconsistency problem: first, the inconsistency between the dictionary and the segmentation result: that is, certain characters that consistently appear together as a single word in the dictionary are separated into consecutive word candidates in the test result. Second, the inconsistency among words in the dictionary. For instance, both "科学研究"(scientific research) and "科学(science)/研究(research)" appear in the training corpora, reflecting different semantic meaning or being merely an error made by the human expert.

To deal with the first phenomena, for the segmented result, we merge adjoining word candidates to match the dictionary entries, if the frequency of the adjoined word in the training corpus is higher than that of the separated word sequence in the same corpus. Suppose "a/b/c/de" is the original voting result, and the word "abc" appears more frequently than the word sequence "a/b/c" in the training corpus. Then, we merge "a", "b" and "c" together to form the output "abc/de".

For the second problem, we introduce the *split* procedure, which examines the word and word sequence frequencies in the training corpus as well. In our system, we only consider two consecutive word candidates. First, all word bigrams are extracted from the training corpus, and their frequencies are counted. After that, for example, if the word bigram "a/b" appears more often than the word candidate "ab", then whenever in the test result we encounter "ab", we split it into "a/b".

The post-processing steps detailed above take word and word sequence frequencies into consideration, attempt to maximize the value of known words in the training corpus as well as deal with the word segmentation inconsistencies in the training corpus. The performance after post-processing on MSRA held-out set is shown in Table 3.4 with bold numbers representing the highest F-score. From the table, we see that, after post-processing, the performance catches up with that produced by the character-based CRF method.

| | $P$ | $R$ | $F$ | $R_{OOV}$ | $R_{IV}$ |
|---|---|---|---|---|---|
| Majority Voting | 96.2 | 96.3 | 96.2 | 65.2 | 97.1 |
| After Post-Processing | 96.5 | 96.4 | **96.5** | 65.2 | 97.3 |
| Character-based CRF | 96.5 | 96.4 | **96.5** | 65.2 | 97.3 |

Table 3.4: Performance (in percentage) after post-processing on the MSRA heldout set

## 3.7 Experiments and Analysis

### 3.7.1 Experiment Corpora Statistics

The majority voting system is evaluated with the CityU corpus, the MSRA corpus, and the UPUC corpus from the third SIGHAN bakeoff. The statistics, including sentence, word, and character information for each training corpus, is summarized in Table 3.5. In addition, Table 3.6 shows their corresponding test set sizes, counting the number of sentences.

| | CityU corpus | MSRA corpus | UPUC corpus |
|---|---:|---:|---:|
| Number of Sentences | 57, 275 | 46, 364 | 18, 804 |
| Number of Words | 70, 290, 106 | 1, 266, 171 | 1, 144, 899 |
| Number of Word Types | 151, 180 | 125, 266 | 74, 764 |
| Number of Characters | 15, 074, 091 | 2, 169, 879 | 1, 235, 673 |
| Number of Character Types | 10, 224 | 9, 534 | 8, 586 |

Table 3.5: Statistics for the CityU, MSRA and UPUC training corpora

### 3.7.2 Results on the Experiment Corpora

To observe the result of majority voting and the contribution of the post-processing step, the experiment is run on each corpus by first producing the result of majority voting and then

| | CityU test set | MSRA test set | UPUC test set |
|---|---|---|---|
| Number of Sentences | 7,511 | 4,365 | 5,117 |

Table 3.6: Number of sentences for the CityU, MSRA and UPUC test sets

producing the result from the post-processing step. In addition, the performance from the standard character-based CRF method is recorded for comparison. In each experiment, the precision ($P$), recall ($R$), evenly-weighted F-score ($F$), out-of-vocabulary recall rate ($R_{OOV}$), and in-vocabulary recall rate ($R_{IV}$) are recorded.

Tables 3.7, 3.8, 3.9 and Figures 3.3, 3.4, 3.5 show and compare the scores for different algorithms applied on the CityU corpus, on the MSRA corpus, and on the UPUC corpus, respectively, with bold numbers representing the highest F-score on each corpus.

| | $P$ | $R$ | $F$ | $R_{OOV}$ | $R_{IV}$ |
|---|---|---|---|---|---|
| Greedy Longest Matching | 88.2 | 93.0 | 90.6 | 0.9 | 96.9 |
| CRF with Maximum Subword Tagging | 95.8 | 96.0 | 95.9 | 75.2 | 96.9 |
| CRF with Minimum Subword Tagging | 95.5 | 95.9 | 95.7 | 73.2 | 96.8 |
| Majority Voting | 95.8 | 96.5 | 96.1 | 69.9 | 97.6 |
| After Post-processing | 96.1 | 96.5 | **96.3** | 69.9 | 97.7 |
| Character-based CRF | 95.7 | 95.7 | 95.7 | 78.3 | 96.5 |

Table 3.7: Performance (in percentage) on the CityU corpus

| | $P$ | $R$ | $F$ | $R_{OOV}$ | $R_{IV}$ |
|---|---|---|---|---|---|
| Greedy Longest Matching | 90.0 | 94.9 | 92.4 | 2.2 | 98.1 |
| CRF with Maximum Subword Tagging | 94.9 | 94.6 | 94.8 | 64.9 | 95.7 |
| CRF with Minimum Subword Tagging | 94.9 | 94.3 | 94.6 | 65.5 | 95.3 |
| Majority Voting | 95.3 | 95.5 | 95.4 | 61.6 | 96.6 |
| After Post-processing | 95.7 | 95.6 | **95.7** | 61.6 | 96.8 |
| Character-based CRF | 95.2 | 94.3 | 94.7 | 66.9 | 95.3 |

Table 3.8: Performance (in percentage) on the MSRA corpus

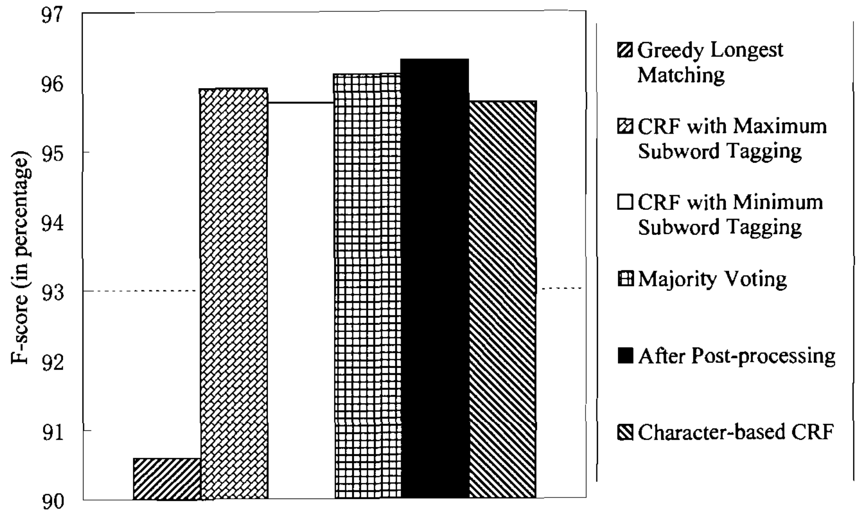

Figure 3.3: Comparison for F-scores on the CityU corpus, with histogram representation

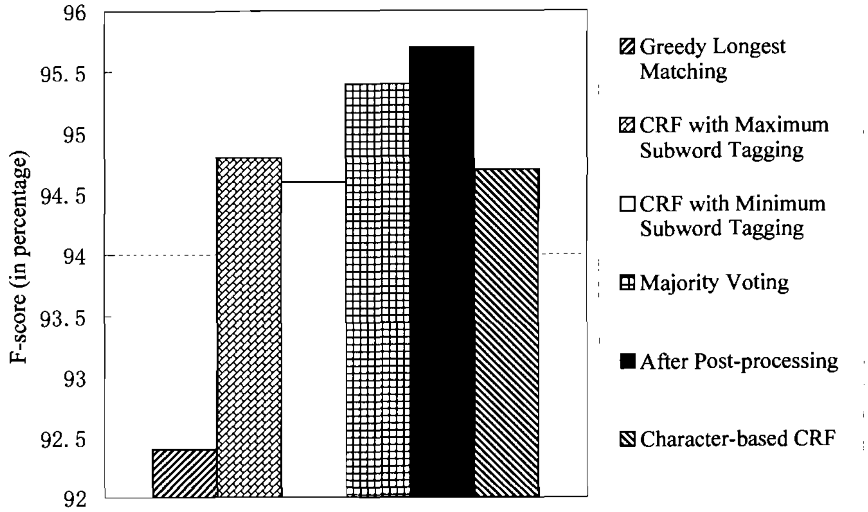

Figure 3.4: Comparison for F-scores on the MSRA corpus, with histogram representation

| | $P$ | $R$ | $F$ | $R_{OOV}$ | $R_{IV}$ |
|---|---|---|---|---|---|
| Greedy Longest Matching | 79.0 | 86.9 | 82.8 | 1.1 | 95.1 |
| CRF with Maximum Subword Tagging | 91.0 | 92.7 | 91.8 | 66.6 | 95.2 |
| CRF with Minimum Subword Tagging | 90.8 | 92.5 | 91.7 | 66.9 | 95.0 |
| Majority Voting | 90.9 | 93.2 | 92.0 | 63.0 | 96.1 |
| After Post-processing | 91.0 | 93.2 | 92.1 | 62.9 | 96.0 |
| Character-based CRF | 92.2 | 93.1 | **92.7** | 71.4 | 95.2 |

Table 3.9: Performance (in percentage) on the UPUC corpus

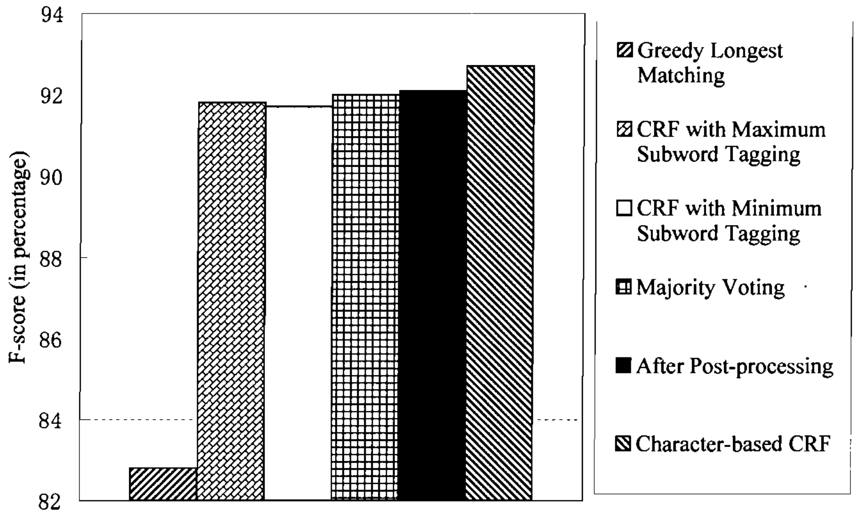

Figure 3.5: Comparison for F-scores on the UPUC corpus, with histogram representation

From those tables, we observe that a simple majority voting algorithm produces an F-score that is higher than each individual system. In addition, the post-processing step indeed helps improve the performance. Also, we see that the system benefits from the CRF-based taggers in that $R_{OOV}$ is significantly higher than the greedy longest matching algorithm, and it also benefits from the dictionary-based algorithms in that $R_{IV}$ is much higher than the CRF-based taggers. Moreover, comparing with the state-of-the-art character-based CRF algorithm, our majority voting system has higher F-scores in the CityU corpus and the MSRA corpus, although it has lower performance in the UPUC corpus. In particular, we observe that for each of these three corpora, $R_{IV}$ from our system is higher than the value achieved from the character-based CRF algorithm, indicating that our system makes better use of the dictionary. On the other hand, $R_{OOV}$ is lower than the character-based CRF algorithm, showing that the character-based CRF algorithm is still a sophisticated algorithm for dealing with out-of-vocabulary words.

Next, with the MSRA corpus, we compare the performance of segmentation algorithm using unigram word frequencies to that of the greedy longest matching algorithm. Also, by substituting the greedy longest matching algorithm with the unigram word frequency-based algorithm, the performance of majority voting is re-examined. Table 3.10 shows the performance.

| | $P$ | $R$ | $F$ | $R_{OOV}$ | $R_{IV}$ |
|---|---|---|---|---|---|
| Segmentation using unigram word frequencies | 88.7 | 94.7 | 91.6 | 2.2 | 97.9 |
| CRF with Maximum Subword Tagging | 94.9 | 94.6 | 94.8 | 64.9 | 95.7 |
| CRF with Minimum Subword Tagging | 94.9 | 94.3 | 94.6 | 65.5 | 95.3 |
| New Majority Voting | 95.3 | 95.5 | 95.4 | 61.6 | 96.7 |
| Original Majority Voting | 95.3 | 95.5 | 95.4 | 61.6 | 96.6 |
| Greedy Longest Matching | 90.0 | 94.9 | 92.4 | 2.2 | 98.1 |
| Character-based CRF | 95.2 | 94.3 | 94.7 | 66.9 | 95.3 |

Table 3.10: Performance (in percentage) on the MSRA corpus, substituting the greedy longest matching method with the unigram word frequency-based method in majority voting

From Table 3.10, we find that the segmentation algorithm with unigram word frequencies has lower performance than the greedy longest matching algorithm, and also that by substituting the greedy longest matching algorithm with the unigram word frequency-based algorithm in majority voting, the F-score does not have any improvement. Thus, in the remaining experiments of this thesis, we will ignore the segmentation algorithm using unigram word frequencies.

### 3.7.3 Error analysis

Examining the segmentation result from these three corpora, we find that the errors that occur in the system are mainly caused by the following factors:

First, there is inconsistency between the gold segmentation and the training corpus, and even within the training corpus or within the gold segmentation itself. Although the inconsistency problem within the training corpus is intended to be tackled in the post-processing step, we cannot conclude that the segmentation for certain words in the gold test set always follows the convention in the training data set. Moreover, errors made by human experts in the gold standard test set contribute to the inconsistency issue as well. For instance, in the UPUC gold set, the person name “赖英照”(Lai, Yingzhao) has two distinct segmentations, “赖/英照”, and “赖英照”. In addition, the inconsistency issue, perhaps as a result of some subtle context-based differences, is hard to model by current methods. For example, in the MSRA training corpus, “中国政府”(Chinese government) and “中国/政府” both appear, but in distinct surrounding context. Either of these two segmentations is acceptable. For instance, in the phrase “中国政府和人民对完成目标充满信心”(Chinese government and people are confident to achieve the goal), it is segmented as “中国(Chinese)/政府(government)/和(and)/人民(people)/对(to)/完成(achieve)/目标(goal)/充满(fulfill)/信心(confidence))” since the adjective “中国”(Chinese) is related to both the nouns “政府”(government) and “人民”(people). On the other hand, for the phrase “中国政府有信心完成这一目标”(Chinese government is confident to achieve this goal), “中国政府” is considered as a single unique entity and thus is not separated. This inconsistency issue lowers the system performance.

Second, we don’t have specific steps to deal with words having common suffixes such as “者”(person). Compared to our system, Zhang et al. [51] proposed a segmentation system that contains a morphologically derived word recognition post-processing component to solve this problem. We only focus on the closed track and thus we cannot include such a step. This prevents us from identifying certain types of words such as “劳动者”(worker).

In addition, unknown words are still troublesome because of the limited size of the training corpora. In the class of unknown words, we encounter person names, numbers, dates, organization names and words translated from other languages. For example, in the CityU test results, the translated person name "米哈伊洛维奇"(Mihajlovic) is incorrectly separated as "米哈伊洛" and "维奇". Moreover, in certain cases, person names can also create ambiguity. Take the person name "秋北方"(Qiu, Beifang) in the UPUC test set for example, without understanding the semantic meaning of the whole sentence, it is difficult even for human to determine whether it is a person name or it represents "秋"(autumn), "北方"(north), with the meaning of "the autumn in the north".

## 3.8 Minimum Subword-based CRF versus Character-based CRF

From the experiments with the MSRA held-out set and with the CityU, MSRA and UPUC test sets, we observe that although the minimum subword-based CRF algorithm produces result whose F-score is relatively high, the character-based CRF algorithm still outperforms it. Moreover, combining the greedy longest matching algorithm with the maximum subword-based CRF algorithm is already capable to improve the in-vocabulary recall rate by including the dictionary information. We propose that by replacing the result from the minimum subword-based CRF algorithm with that from the character-based CRF algorithm, whose $R_{OOV}$ is much higher, during the voting process, the performance should be increased by raising the $R_{OOV}$.

### 3.8.1 Experiments

To examine the above hypothesis, once again, we first use the MSRA held-out set for the majority voting. The comparison result is shown in Table 3.11, with the bold number representing the highest $R_{OOV}$.

| | $P$ | $R$ | $F$ | $R_{OOV}$ | $R_{IV}$ |
|---|---|---|---|---|---|
| New Majority Voting | 96.2 | 96.2 | 96.2 | **66.3** | 97.0 |
| Original Majority Voting | 96.2 | 96.3 | 96.2 | 65.2 | 97.1 |

Table 3.11: Performance (in percentage) for majority voting using character-based CRF algorithm on MSRA held-out set

From the statistics, we see that, comparing with our previous voting strategy, after voting, with sightly changes in $R$ and $R_{IV}$, $R_{OOV}$ increases more than 1%.

Then, we applied this new majority voting system with the CityU, MSRA and UPUC test sets. Tables 3.12, 3.13, 3.14 and Figures 3.6, 3.7, 3.8 show and compare the scores for different algorithms applied on the CityU corpus, on the MSRA corpus, and on the UPUC corpus, respectively, with bold numbers representing the highest F-score on each corpus.

| | $P$ | $R$ | $F$ | $R_{OOV}$ | $R_{IV}$ |
|---|---|---|---|---|---|
| Greedy Longest Matching | 88.2 | 93.0 | 90.6 | 0.9 | 96.9 |
| CRF with Maximum Subword Tagging | 95.8 | 96.0 | 95.9 | 75.2 | 96.9 |
| Character-based CRF | 95.7 | 95.7 | 95.7 | 78.3 | 96.5 |
| New Majority Voting | 96.3 | 96.6 | 96.4 | 74.1 | 97.5 |
| After Post-processing | 96.5 | 96.6 | **96.6** | 74.1 | 97.6 |
| Original Majority Voting | 95.8 | 96.5 | 96.1 | 69.9 | 97.6 |

Table 3.12: Performance (in percentage) on the CityU corpus, voting using the character-based CRF algorithm

| | $P$ | $R$ | $F$ | $R_{OOV}$ | $R_{IV}$ |
|---|---|---|---|---|---|
| Greedy Longest Matching | 90.0 | 94.9 | 92.4 | 2.2 | 98.1 |
| CRF with Maximum Subword Tagging | 94.9 | 94.6 | 94.8 | 64.9 | 95.7 |
| Character-based CRF | 95.2 | 94.3 | 94.7 | 66.9 | 95.3 |
| New Majority Voting | 95.4 | 95.4 | 95.4 | 62.9 | 96.6 |
| After Post-processing | 95.8 | 95.6 | **95.7** | 62.9 | 96.8 |
| Original Majority Voting | 95.3 | 95.5 | 95.4 | 61.6 | 96.6 |

Table 3.13: Performance (in percentage) on the MSRA corpus, voting using the character-based CRF algorithm

We observe that by replacing the minimum subword-based CRF algorithm with the character-based CRF algorithm during the voting process, the $R_{OOV}$ for the MSRA test set increases 1.1%. Moreover, the F-score for the CityU test set increases 0.3%, and its $R_{OOV}$ increases 4.2%. In addition, for the UPUC test set, its F-score even increases 0.7%, and its $R_{OOV}$ increases 2.6%.

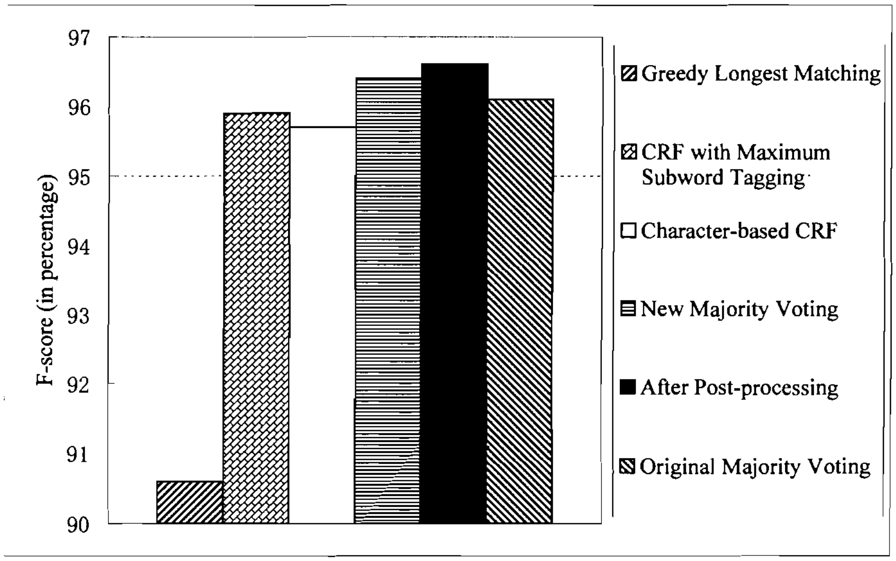

Figure 3.6: Comparison for F-scores on the CityU corpus, voting using the character-based CRF algorithm, with histogram representation

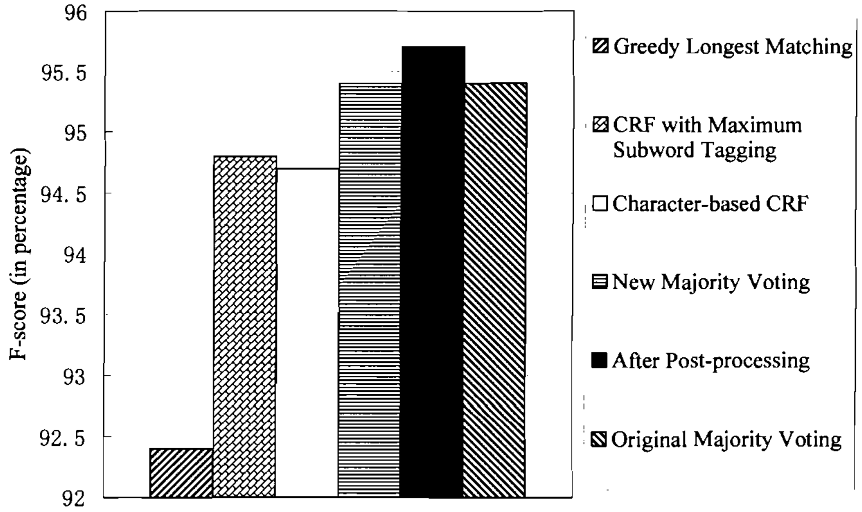

Figure 3.7: Comparison for F-scores on the MSRA corpus, voting using the character-based CRF algorithm, with histogram representation

| | $P$ | $R$ | $F$ | $R_{OOV}$ | $R_{IV}$ |
|---|---|---|---|---|---|
| Greedy Longest Matching | 79.0 | 86.9 | 82.8 | 1.1 | 95.1 |
| CRF with Maximum Subword Tagging | 91.0 | 92.7 | 91.8 | 66.6 | 95.2 |
| Character-based CRF | 92.2 | 93.1 | **92.7** | 71.4 | 95.2 |
| New Majority Voting | 91.4 | 93.3 | 92.4 | 65.6 | 96.0 |
| Post-processing | 91.5 | 93.3 | 92.4 | 65.6 | 96.0 |
| Original Majority Voting | 90.9 | 93.2 | 92.0 | 63.0 | 96.1 |

Table 3.14: Performance (in percentage) on the UPUC corpus, voting using the character-based CRF algorithm

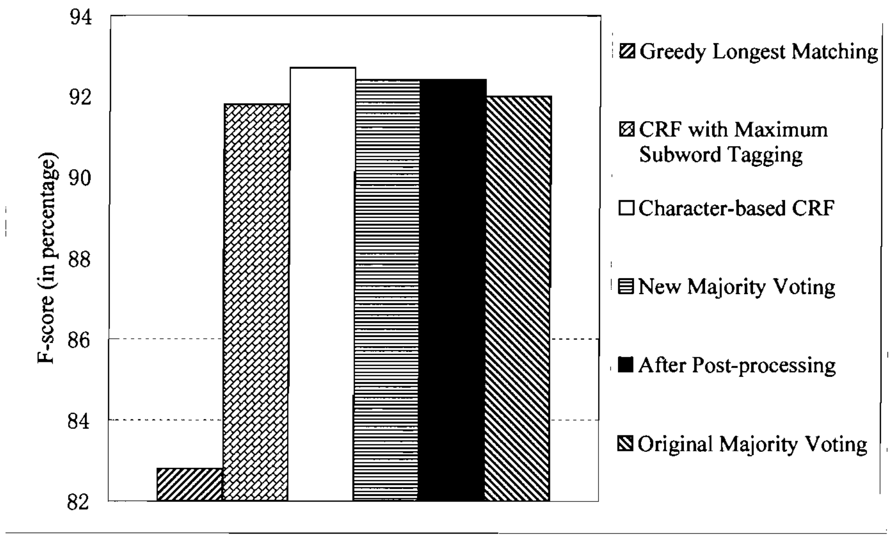

Figure 3.8: Comparison for F-scores on the UPUC corpus, voting using the character-based CRF algorithm, with histogram representation

### 3.8.2 Significance Test

To test whether the improvement is significant, comparing the majority voting method containing the character-based CRF(i.e. Vote_Char_CRF) to the method containing the minimum subword-based CRF(i.e. Vote_Min_CRF), we carry out McNemar's Test[1] [14]. We assume that the errors are independent.

The distributions of errors, counted by words, for the CityU test set, for the MSRA test set, and for the UPUC test set are listed in Table 3.15, 3.16 and 3.17, respectively.

| | | Vote_Min_CRF | | |
|---|---|---|---|---|
| | | Correct | Incorrect | Total |
| Vote_Char_CRF | Correct | 211,971 ($n_{00}$) | 639 ($n_{01}$) | 212,610 |
| | Incorrect | 426 ($n_{10}$) | 7,143 ($n_{11}$) | 7,569 |
| | Total | 212,397 | 7,782 | 220,179 |

Table 3.15: Word error distribution on the CityU corpus, from majority voting methods

| | | Vote_Min_CRF | | |
|---|---|---|---|---|
| | | Correct | Incorrect | Total |
| Vote_Char_CRF | Correct | 95,655 ($n_{00}$) | 109 ($n_{01}$) | 95,764 |
| | Incorrect | 149 ($n_{10}$) | 4,447 ($n_{11}$) | 4,596 |
| | Total | 95,804 | 4,556 | 100,360 |

Table 3.16: Word error distribution on the MSRA corpus, from majority voting methods

| | | Vote_Min_CRF | | |
|---|---|---|---|---|
| | | Correct | Incorrect | Total |
| Vote_Char_CRF | Correct | 143,879 ($n_{00}$) | 667 ($n_{01}$) | 144,546 |
| | Incorrect | 392 ($n_{10}$) | 9,926 ($n_{11}$) | 10,318 |
| | Total | 144,271 | 10,593 | 154,864 |

Table 3.17: Word error distribution on the UPUC corpus, from majority voting methods

[1] Script applied from http://www.fon.hum.uva.nl/Service/Statistics/McNemars_test.html

With these statistics, we can perform the McNemar's test, which tests the null hypothesis that the performance improvement between the Vote_Char_CRF method and the Vote_Min_CRF method is simply by chance, and which computes a 2-tailed P-value to test this hypothesis based on the following formula:

$$\textit{P-value} = \begin{cases} 2\sum_{m=0}^{n_{10}} \binom{k}{m}\left(\frac{1}{2}\right)^k & \text{when } n_{10} < k/2 \\ 2\sum_{m=n_{10}}^{k} \binom{k}{m}\left(\frac{1}{2}\right)^k & \text{when } n_{10} > k/2 \\ 1.0 & \text{when } n_{10} = k/2 \end{cases} \tag{3.1}$$

where $k = n_{10} + n_{01}$. Note that $n_{01}$ and $n_{10}$ are defined in Tables 3.15, 3.16 and 3.17.

| Data Set | P-Value |
|---|---|
| CityU | $\leq$ 8.28e-11 |
| MSRA | $\leq$ 0.0152 |
| UPUC | $\leq$ 3.82e-17 |

Table 3.18: P-values computed using the McNemar's test on the CityU, MSRA and UPUC corpora, for comparison of majority voting methods

From the calculated P-values in Table 3.18, we are therefore confident to conclude that, by replacing the minimum subword-based CRF algorithm with the character-based CRF algorithm, the $R_{OOV}$ from the majority voting increases, and the overall performance also increases significantly.

## 3.9 Summary of the Chapter

In this chapter, the Chinese word segmentation system, which is based on majority voting among initial outputs from the greedy longest matching, from the CRF model with maximum subword-based tagging, and from the CRF model with minimum subword-based tagging, is described in detail. Our experimental results show that the majority voting method takes advantage of the dictionary information used in the greedy longest matching algorithm and that of the subword information used in the maximum subword-based CRF model, and thus raises the low in-vocabulary recall rate. Also, the voting procedure benefits from the high out-of-vocabulary recall rate achieved from the two CRF-based tagging algorithms. Our majority voting system improved the performance of each individual algorithm. In addition, we experimented with various steps in post-processing which effectively improved the overall performance. Moreover, we examined that by substituting the minimum subword-based CRF model with character-based CRF model during the majority voting, we can make better usage of its higher out-of-vocabulary recall rate to raise $R_{OOV}$ and the overall performance.

# Chapter 4

# Global Features and Global Linear Models

In this chapter, we propose the use of global features to assist with local features in training an averaged perceptron on N-best candidates for Chinese word segmentation. Our experiments show that by adding global features, performance is significantly improved compared to the character-based CRF tagger. Performance is also improved compared to using only local features. Testing on the closed track of the CityU, MSRA and UPUC corpora from the third SIGHAN bakeoff, our system obtains a significant improvement in F-score from 95.7% to 97.1%, from 95.2% to 95.8%, and from 92.8% to 93.1%, respectively, comparing with the character-based CRF tagger.

The chapter is organized as follows: Section 4.1 provides an overview of the system; Section 4.2 describes the system architecture in depth; Section 4.3 shows and analyzes the experimental results; Section 4.4 explores the weight learning for global features; Section 4.5 compares the N-best list re-ranking method with the beam search decoding approach; while Section 4.6 briefly explores another global linear model – the exponential gradient learning approach; Section 4.7 discusses related works, and Section 4.8 gives the summary for this chapter.

## 4.1 Averaged Perceptron Global Linear Model

Our averaged perceptron word segmentation system implements the re-ranking technique. The overview of the entire system is shown in Figure 4.1. For each of the training corpora, we produce a 10-fold split: in each fold, 90% of the corpus is used for training and 10% is used to produce an N-best list of candidates. The N-best list is produced using a character-based CRF tagger. The true segmentation can now be compared with the N-best list in order to train with an averaged perceptron algorithm. This system is then used to predict the best word segmentation from an N-best list for each sentence in the test data.

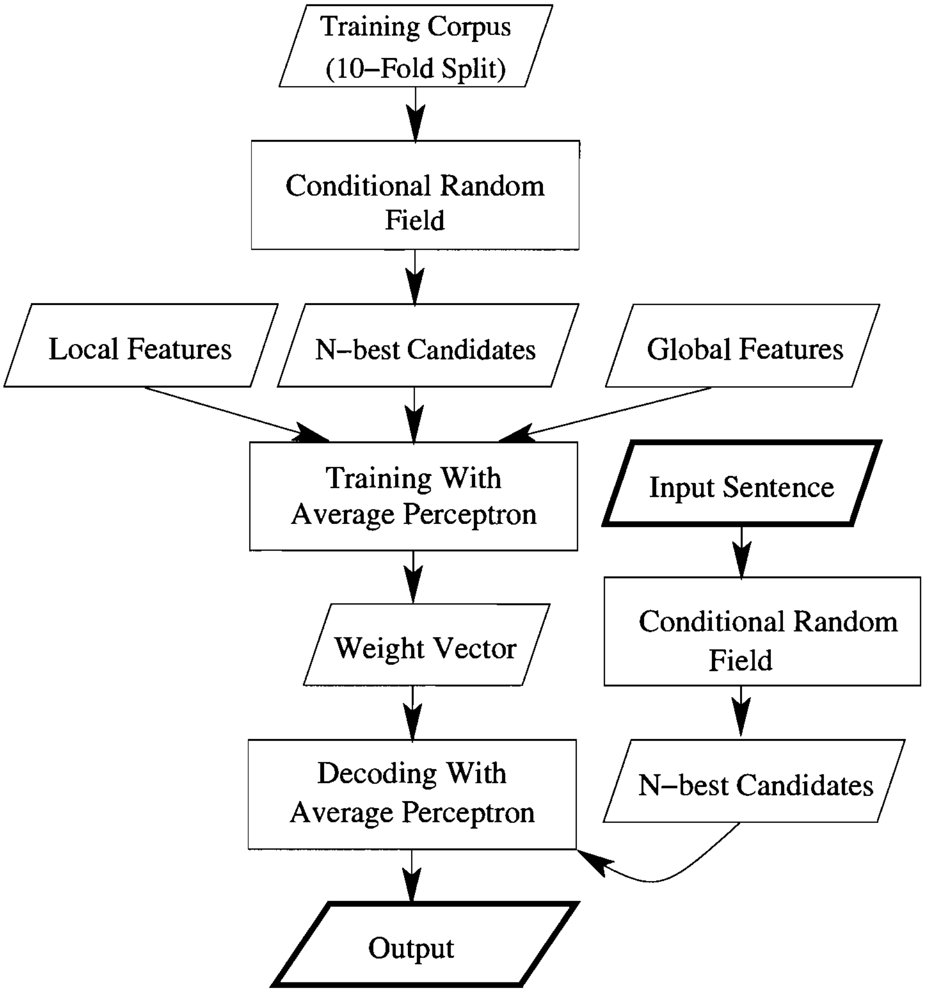

Figure 4.1: Overview of the averaged perceptron system

## 4.2 Detailed System Description

Recall from Chapter 2 that, given an unsegmented sentence $x$, the Chinese word segmentation problem can be defined as finding the most plausible output sentence $F(x)$ from a set of possible segmentations of $x$:

$$F(x) = \underset{y \in GEN(x)}{\text{argmax}} \ \Phi(x, y) \cdot \mathbf{w}$$

where $GEN(x)$ is the set of possible segmentations for the input sentence $x$. The weight parameter vector $\mathbf{w}$, being initially set to $\mathbf{0}$, is maintained and updated while iterating through the training set during the perceptron learning process (See Figure 2.10).

In our system, the averaged perceptron algorithm, which is capable of reducing overfitting on the training data and producing a more stable solution, is implemented. Also, the lazy update process is adapted to reduce the time taken for training.

### 4.2.1 N-best Candidate List

In Figure 2.10, in order to calculate $GEN(x_i)$ in $argmax_{y \in GEN(x_i)}$, the naïve method can be implemented to first generate all possible segmented candidates for the character sequence. For a sentence with $L$ characters, there are $2^{L-1}$ possible segmentations. For example, suppose we have a sentence with 3 characters "abc", then the following 4 candidates are to be generated: "abc", "a/bc", "ab/c", and "a/b/c". When $L$ is large, however, generating those segmentations and picking the one with the highest score is time consuming. For instance, if $L$=20, then over 500,000 candidates are required to be produced and examined.

Our system makes use of the re-ranking approach. Re-ranking has been broadly applied in various natural language tasks such as parsing [6] and machine translation [38]. The general intuition of any re-ranking method is to apply a separate model to re-rank the output of a base system. For each input sentence, this base system produces a set of candidate outputs, and defines an initial ranking for these candidates. The second model attempts to improve upon this initial ranking so that candidates that are closer to the truth get a higher rank. In our system, only a small portion of all possible segmentations, that is, the N-best ranked candidate segmentations, are produced by the *CRF++* package, and the perceptron learning model tends to re-rank these candidates to pick the one closest to the truth.

We use the standard method for producing N-best candidates in order to train our reranker which uses global and local features: 10-folds of training data are used to train the tagger on 90% of the data and then produce N-best lists for the remaining 10%. This process gives us an N-best candidate list for each sentence and the candidate that is most similar to the true segmentation, called $y^b$. Notice that in this modeling process, the characters in the training sentences are assigned "BMES" tags[1], and the same feature templates listed in Table 3.1 are applied in CRF tagging.

Figure 4.2 shows the modified averaged perceptron algorithm on the N-best candidate list.

```
Inputs: Training Data $\langle(x_1, y_1), \ldots, (x_m, y_m)\rangle$; number of iterations T
Initialization: Set $\mathbf{w} = \mathbf{0}$, $\gamma = \mathbf{0}$, $\sigma = \mathbf{0}$
Algorithm:
  for $t = 1, \ldots, T$ do
    for $i = 1, \ldots, m$ do
      Calculate $y_i^{'}$, where $y_i^{'} = \operatorname{argmax}_{y \in \textit{N-best Candidates}} \Phi(y) \cdot \mathbf{w}$
      if $y_i^{'} \neq y^b$ then
        $\mathbf{w} = \mathbf{w} + \Phi(y^b) - \Phi(y_i^{'})$
      end if
      $\sigma = \sigma + \mathbf{w}$
    end for
  end for
Output: The averaged weight parameter vector $\gamma = \sigma/(\mathrm{mT})$
```

Figure 4.2: The averaged perceptron learning algorithm on the N-best list

### 4.2.2 Feature Templates

The feature templates used in our system include both local features and global features. For local features, the 14 feature types from Zhang and Clark's paper [53] in ACL 2007 are adapted, and they are shown in Table 4.1.

While a local feature indicates the occurrence of a certain pattern in a partially segmented sentence, a global feature provides information about the entire sentence. Our

[1] Performance of the CRF tagger might be improved with the use of other tagsets. However, this does not affect our comparative experiments

| | |
|---|---|
| 1 | word $w$ |
| 2 | word bigram $w_1 w_2$ |
| 3 | single character word $w$ |
| 4 | space-separated characters $c_1$ and $c_2$ |
| 5 | character bi-gram $c_1 c_2$ in any word |
| 6 | a word starting with character $c$ and having length $l$ |
| 7 | a word ending with character $c$ and having length $l$ |
| 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 4.1: Local feature templates

global features are different from commonly applied "global" features in the literature, in that they either enforce consistency or examine the use of a feature in the entire training or testing corpus. In our system, two specific global features are included: the sentence confidence score feature and the sentence language model score feature, shown in Table 4.2.

| | |
|---|---|
| 15 | sentence confidence score |
| 16 | sentence language model score |

Table 4.2: Global feature templates

Sentence confidence scores are calculated by *CRF++* during the production of the N-best candidate list, and they measure how confident each candidate is close to the true segmentation. They provide an important initial rank information, which cannot be ignored in the re-ranking phase, for each candidate in the N-best list. The scores are probabilities, which means that for the N-best candidate list $\{c_1, c_2, ..., c_n\}$ of a particular example, $0 \leq P_{c_i} \leq 1$ and $P_{c_1} + P_{c_2} + ... + P_{c_n} = 1$.

Sentence language model scores are produced using the *SRILM* [41] toolkit[2]. They indicate how likely a sentence can be generated given the training data, and they help capture the usefulness of features extracted from the training data. The $n$-gram word statistics, where $n$ is between 1 and 3, for the whole training corpus, is generated by this toolkit and is then applied on each N-best candidate to calculate its language model score. It is normalized using the formula $P^{1/L}$, where $P$ is the probability-based language model score and $L$ is the length of the sentence in words (not in characters). Since *SRILM* returns the score in log-probability form, the value we will use for this feature is $\mid (\log(P))/L \mid$.

In our perceptron learning process, the weights for local features are updated so that whenever a mismatch is found between the best candidate $y^b$ and the current top-scored candidate $C_{top}$, the weight parameter values for features in $y^b$ are incremented, and the weight parameter values for features appearing in $C_{top}$ are decremented. For the global features, however, their weights are not learned using the perceptron algorithm but are determined using a development set.

### 4.2.3 A Perceptron Learning Example with N-best Candidate List

To illustrate the process of perceptron training using N-best candidate list, we provide an example for weight learning, assuming that only local features are applied and that the weights for global features are all zero. For simplicity, only Features 1–5 in Table 4.1 are applied in this example.

Suppose the whole training set only contains one training example, and its N-best list includes 6-best possible segmentation sentences, as shown below, and the number of iteration $t$ is set to be 3:

- **The Best Candidate**: 我们(we)/生活(live)/在(in)/信息(information)/时代(age)
- Candidate 1: 我们(we)/生(born)/活(alive)/在(in)/信息(information)/时代(age)
- Candidate 2: 我们(we)/生活在(live in)/信息(information)/时代(age)
- Candidate 3: 我们(we)/生活(live)/在(in)/信息时代(information age)
- Candidate 4: 我们(we)/生活在(live in)/信息时代(information age)

[2] available from http://www.speech.sri.com/projects/srilm/

- Candidate 5: 我们(we)/生(born)/活(alive)/在(in)/信息时代(information age)
- Candidate 6: 我们(we)/生活(live)/在(in)/信息(information)/时代(age)

During the first iteration, since $\mathbf{w}$ is initialized to zero, all those six candidates have equal possibility of being chosen as the top-scored one because the score for each of them is zero. We choose Candidate 1 to be our candidate. Comparing the best candidate and Candidate 1,

- The Best Candidate: 我们(we)/生活(live)/在(in)/信息(information)/时代(age)
- Candidate 1: 我们(we)/生(born)/活(alive)/在(in)/信息(information)/时代(age)

we see that they are not identical. Thus, we reward features appearing in the best candidate and penalize features in this chosen candidate, and achieve the following weight vector $\mathbf{w}_1$:

| | | | | |
|---|---|---|---|---|
| $F_1$ | 生活(live) : | 1 | 生(born) : | -1 |
| | 活(alive) : | -1 | | |
| $F_2$ | (我们(we), 生活(live)) : | 1 | (生活(live), 在(in)) : | 1 |
| | (我们(we), 生(born)) : | -1 | (生(born), 活(alive)) : | -1 |
| | (活(alive), 在(in)) : | -1 | | |
| $F_3$ | 生(born) : | -1 | 活(alive) : | -1 |
| $F_4$ | (生(born), 活(alive)) : | -1 | | |
| $F_5$ | (生(born), 活(alive)) : | 1 | | |

Table 4.3: Updated weight vector $\mathbf{w}_1$ in the perceptron learning example

In the second iteration, each candidate in the 6-best list is re-scored using this updated weight vector. For example, for Candidate 3, its weight corresponding to each feature is summarized to be:

- Weight for $F_1$ in Sentence 3: 生活(live) : 1
- Weight for $F_2$ in Sentence 3: (我们(we), 生活(live)) : 1 ; (生活(live), 在(in)) : 1
- Weight for $F_5$ in Sentence 3: (生(born), 活(alive)) : 1

Therefore, the score for Candidate 3 is 4 (i.e. $Score(s_3) = 4$). Similarly, we calculate scores for the remaining sentences in this 6-best list, and get $Score(s_1) = -8$, $Score(s_2) = 1$, *Score*($s_3$) = 4, *Score*($s_4$) = 1, *Score*($s_5$) = -8, and *Score*($s_6$) = 4. From these scores, we see that, Candidate 3 and Candidate 6 have the highest score. We choose Candidate 3 to be our top candidate.

Comparing the best candidate with Candidate 3:

- The Best Candidate: 我们(we)/生活(live)/在(in)/信息(information)/时代(age)
- Candidate 3: 我们(we)/生活(live)/在(in)/信息时代(information age)

we realize that these two sentences are not identical, either. Therefore, we again reward features appearing in the best candidate and penalize features in Candidate 3, and achieve the following updated weight vector $\mathbf{w}_2$, with the boldfaced part being the updated or newly added features:

| | | | | |
|---|---|---|---|---|
| $F_1$ | 生活(live) : | 1 | 生(born) : | -1 |
| | 活(alive) : | -1 | **信息(information)** : | 1 |
| | **时代(age)** : | 1 | **信息时代(information age)** : | -1 |
| $F_2$ | (我们(we), 生活(live)) : | 1 | (生活(live), 在(in)) : | 1 |
| | (我们(we), 生(born)) : | -1 | (生(born), 活(alive)) : | -1 |
| | (活(alive), 在(in)) : | -1 | **(在(in), 信息(information))** : | 1 |
| | **(信息(information), 时代(age))** : | 1 | **(在(in), 信息时代(information age))** : | -1 |
| $F_3$ | 生(born) : | -1 | 活(alive) : | -1 |
| $F_4$ | (生(born), 活(alive)) : | -1 | **(息(breath), 时(hour))** : | 1 |
| $F_5$ | (生(born), 活(alive)) : | 1 | **(息(breath), 时(hour))** : | -1 |

Table 4.4: Updated weight vector $\mathbf{w}_2$ in the perceptron learning example

In the third iteration, each candidate in the 6-best list is re-scored using this freshly updated weight vector, and get the score for each sentence in the 6-best list: *Score*($s_1$) = -3, *Score*($s_2$) = 5, *Score*($s_3$) = 1, *Score*($s_4$) = -2, *Score*($s_5$) = -11, and *Score*($s_6$) = 9. Due to the highest score Candidate 6 gets, it is selected as the current top candidate. By comparing the best candidate with Candidate 6, we find that they are identical, and therefore, the weight vector does not change: $\mathbf{w}_3 = \mathbf{w}_2$. The updating for the weight vector is finished, and in the averaged perceptron learning, the final target vector is calculated by

$$\frac{1}{\mathrm{mT}}(\mathbf{w}_1 + \mathbf{w}_2 + \mathbf{w}_3) = \frac{1}{1 \times 3}(\mathbf{w}_1 + \mathbf{w}_2 + \mathbf{w}_3)$$

to become the one in Table 4.5.

| | | | | |
|---|---|---|---|---|
| $F_1$ | 生活(live) : | 1 | 生(born) : | -1 |
| | 活(alive) : | -1 | 信息(information) : | 2/3 |
| | 时代(age) : | 2/3 | 信息时代(information age) : | -2/3 |
| $F_2$ | (我们(we), 生活(live)) : | 1 | (生活(live), 在(in)) : | 1 |
| | (我们(we), 生(born)) : | -1 | (生(born), 活(alive)) : | -1 |
| | (活(alive), 在(in)) : | -1 | (在(in), 信息(information)) : | 2/3 |
| | (信息(information), 时代(age)) : | 2/3 | (在(in), 信息时代(information age)) : | -2/3 |
| $F_3$ | 生(born) : | -1 | 活(alive) : | -1 |
| $F_4$ | (生(born), 活(alive)) : | -1 | (息(breath), 时(hour)) : | 2/3 |
| $F_5$ | (生(born), 活(alive)) : | 1 | (息(breath), 时(hour)) : | -2/3 |

Table 4.5: Final weight vector in the perceptron learning example

## 4.3 Experiments

To test the performance of our system, the CityU, MSRA and UPUC corpora from the third SIGHAN bakeoff are used once again, following the closed track.

### 4.3.1 Parameter Pruning

First, we experimented on the development set of the UPUC corpus to find a suitable value for the parameter $n$, the maximum number of N-best candidates. This oracle procedure proceeds as follows: 80% of the training corpus is used to train the CRF model, which is used to produce the N-best outputs for each sentence on the remaining 20% of the corpus. Then, these N candidates are compared with the true segmentation, and for each training sentence, the candidate closest to the truth is chosen as the final output. As we increase the value of $n$, for some sentences, its $n$-best candidate list is more likely to contain a segmentation that will improve the overall F-score (Figure 4.3). However, the cost of choosing a larger value of $n$ leads to increased time complexity in the perceptron learning algorithm. To balance accuracy and speed, we choose $n$ to be 20 in all remaining experiments.

Next, the weight for sentence confidence score $S_{crf}$ and that for language model score $S_{lm}$ are determined. In this step, each training corpus is separated into a training set, which contains 80% of the training corpus, and a development set containing the remaining 20% of the training corpus. Then, the perceptron algorithm is applied on the training set with different $S_{crf}$ and $S_{lm}$ weight values, and for various number of iterations. The weight values

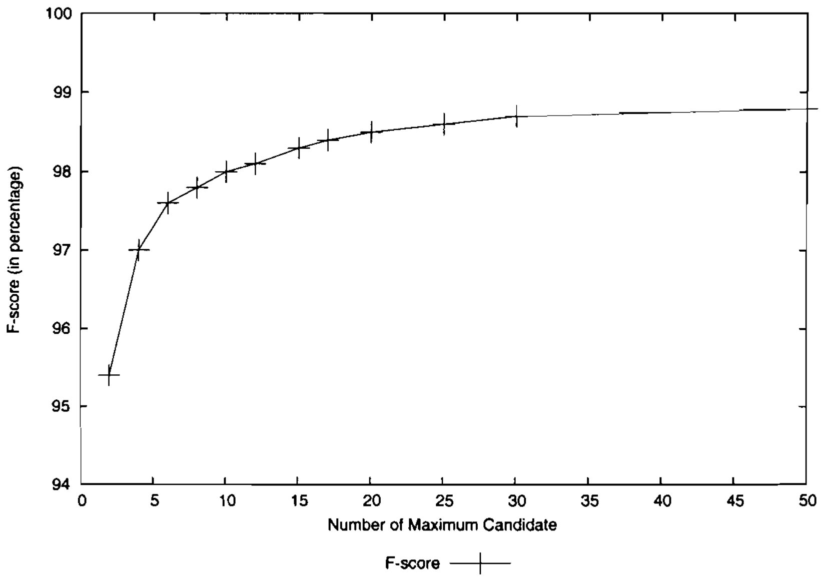

Figure 4.3: F-score on the UPUC development set with different $n$

we test include 2, 4, 6, 8, 10, 15, 20, 30, 40, 50, 100 and 200, across a wide range of scales. As we can see, there will be a significant number of testing scenarios (i.e. 12×12 = 144 testing scenarios) in order to pick the most suitable weight values for each corpus. To simplify the process, we assume that the weights for both $S_{crf}$ and $S_{lm}$ are equal. Figures 4.4, 4.5, 4.6 show the F-scores on each of the three corpora using different $S_{crf}$ and $S_{lm}$ weight values with different number of iterations $t$. From the tables, we observe that when the weight for $S_{crf}$ and $S_{lm}$ increases, F-score improves; however, if the weight for $S_{crf}$ and $S_{lm}$ becomes too large to overrule the effect of weight learning on local features, F-score drops down. For the remaining experiments, the weight for $S_{crf}$ and $S_{lm}$ is chosen to be 15 for the CityU corpus, to be 15 for the MSRA corpus, and to be 20 for the UPUC corpus.

While determining the weights for global features, the number of training iterations can be determined as well. The trends from Figures 4.4, 4.5, 4.6 show that, as the number of iterations increases, F-score stabilizes for most cases, reflecting convergence of the learning algorithm. From the learning curves, we can fix the number of training iterations to be 7 for the CityU and MSRA corpora, and to be 9 for the UPUC corpus.

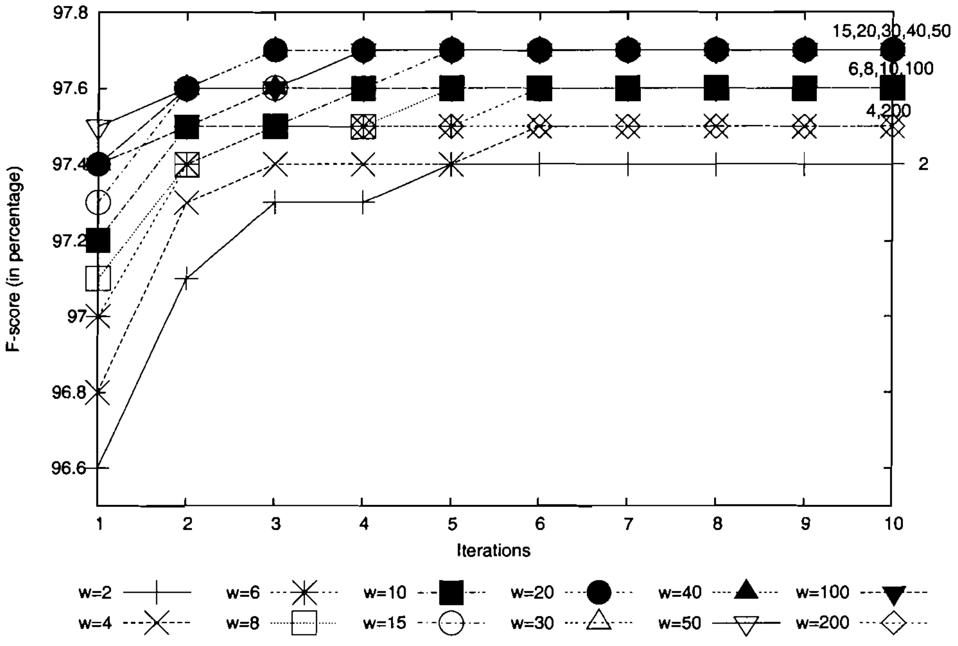

Figure 4.4: F-scores on the CityU development set

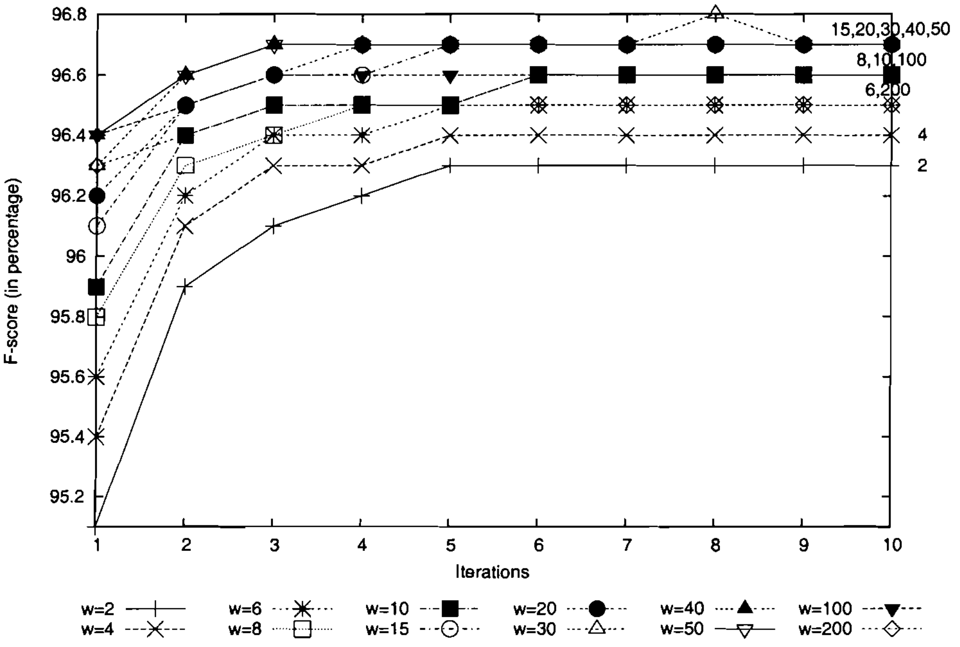

Figure 4.5: F-scores on the MSRA development set

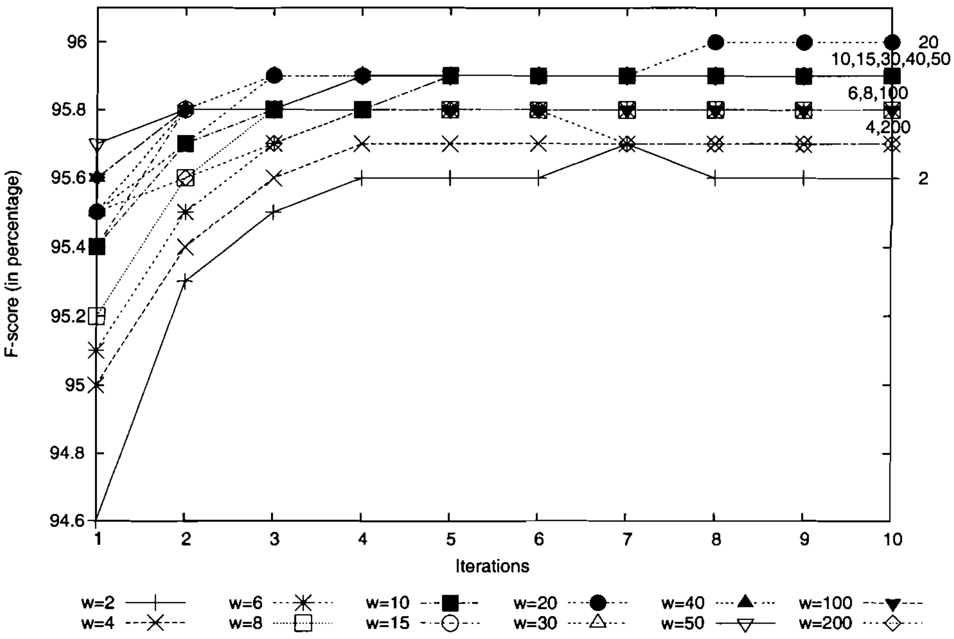

Figure 4.6: F-scores on the UPUC development set

### 4.3.2 Experiment Results

The performance is measured using F-score($F$), precision($P$), recall($R$), in-vocabulary recall rate($R_{IV}$) and out-of-vocabulary recall rate($R_{OOV}$). We compare our approach with the averaged perceptron using only local features, the character-based CRF method, and the base-line performance using the greedy longest matching method, and the results are listed in the Table 4.6. For each corpus, the bold number shows the highest F-score. Figure 4.7 intuitively compares the F-score results among these various methods.

From these experimental results, we see that our system outperforms the dictionary-based greedy longest matching method. Also, our system improves the performance of the character-based CRF. In addition, using global features, our model achieved better overall performance than the perceptron algorithm using only local features.

### 4.3.3 Significance Test

To test whether the improvement is significant, compared to the character-based one-best CRF result, we carry out McNemar's test and assume that the errors are independent.

| Corpus | Setting | F | P | R | $R_{IV}$ | $R_{OOV}$ |
|---|---|---|---|---|---|---|
| CityU | Averaged Perceptron with global and local features | **97.1** | 97.1 | 97.1 | 97.9 | 78.3 |
|  | Averaged Perceptron with only local features | 96.7 | 96.7 | 96.6 | 97.5 | 77.4 |
|  | Character-based conditional random field | 95.7 | 95.7 | 95.8 | 96.6 | 77.7 |
|  | Greedy longest matching | 90.6 | 88.2 | 93.0 | 96.9 | 0.9 |
| MSRA | Averaged Perceptron with global and local features | **95.8** | 95.9 | 95.7 | 96.9 | 62.0 |
|  | Averaged Perceptron with only local features | 95.5 | 95.6 | 95.3 | 96.3 | 65.4 |
|  | Character-based conditional random field | 95.2 | 95.6 | 94.8 | 95.8 | 67.1 |
|  | Greedy longest matching | 92.4 | 90.0 | 94.9 | 98.1 | 2.2 |
| UPUC | Averaged Perceptron with global and local features | **93.1** | 92.5 | 93.8 | 96.1 | 69.4 |
|  | Averaged Perceptron with only local features | 92.5 | 91.8 | 93.1 | 95.5 | 68.8 |
|  | Character-based conditional random field | 92.8 | 92.2 | 93.3 | 95.5 | 70.9 |
|  | Greedy longest matching | 82.8 | 79.0 | 86.9 | 95.1 | 1.1 |

Table 4.6: Performance (in percentage) on CityU, MSRA, and UPUC Corpora

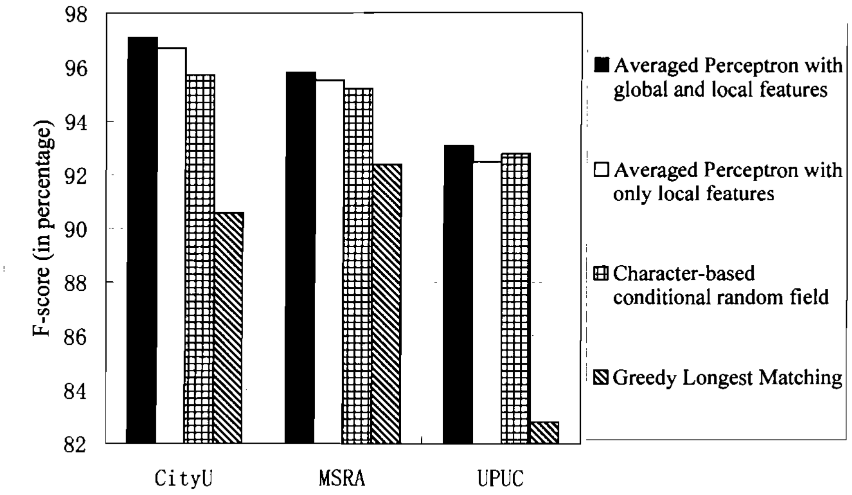

Figure 4.7: Comparison for F-scores on the CityU, MSRA, and UPUC corpora, with histogram representation

The distributions of the errors, counted by words, for the CityU test set, for the MSRA test set, and for the UPUC test set are listed in Table 4.7, 4.8 and 4.9, respectively.

| | | one-best CRF result | | |
|---|---|---|---|---|
| | | Correct | Incorrect | Total |
| perceptron re-ranking | Correct | 209,583 ($n_{00}$) | 4,175 ($n_{01}$) | 213,758 |
| | Incorrect | 1,337 ($n_{10}$) | 5,084 ($n_{11}$) | 6,421 |
| | Total | 210,920 | 9,259 | 220,179 |

Table 4.7: Word error distribution on the CityU corpus

With these statistics, we can perform the McNemar's test, which tests the null hypothesis that the performance improvement between the global-feature-based averaged perceptron and the character-based CRF is simply by chance, and which computes a 2-tailed P-value to test this hypothesis.

From the calculated P-value in Table 4.10, we are therefore confident to conclude that, for all CityU, MSRA and UPUC test sets, the difference in performance between the averaged perceptron using global features and the character-based CRF is significant.

| | | one-best CRF result | | |
|---|---|---|---|---|
| | | Correct | Incorrect | Total |
| perceptron re-ranking | Correct | 94,308 ($n_{00}$) | 1,763 ($n_{01}$) | 96,071 |
| | Incorrect | 835 ($n_{10}$) | 3,454 ($n_{11}$) | 4,289 |
| | Total | 95,143 | 5,217 | 100,360 |

Table 4.8: Word error distribution on the MSRA corpus

| | | one-best CRF result | | |
|---|---|---|---|---|
| | | Correct | Incorrect | Total |
| perceptron re-ranking | Correct | 142,581 ($n_{00}$) | 2,636 ($n_{01}$) | 145,217 |
| | Incorrect | 1,932 ($n_{10}$) | 7,715 ($n_{11}$) | 9,647 |
| | Total | 144,513 | 10,351 | 154,864 |

Table 4.9: Word error distribution on the UPUC corpus

| Data Set | P-Value |
|---|---|
| CityU | $\leq$ 2.04e-319 |
| MSRA | $\leq$ 7e-74 |
| UPUC | $\leq$ 2.5e-25 |

Table 4.10: P-values computed using the McNemar's test on the CityU, MSRA and UPUC corpora, for comparison between the averaged perceptron using global features and the character-based CRF

### 4.3.4 Error Analysis

By analyzing segmentation results, we discover that, for the CityU output, errors repeatedly involve punctuation as well as numbers. For instance, the front double quote in [“/六一/”] is mistakenly combined with its following word to become [“六一/”]. As another example, the number “18” as in “18/日”(the 18th day in a month) in certain sentences is broken into two words “1” and “8”. However, we notice that these errors also happen in the output produced by character-based conditional random field, from which our N-best list is generated.

In addition to punctuation and number errors, personal names are another major source of errors, not only in the CityU output, but also in the MSRA as well as the UPUC segmentation results.

While for the UPUC output, the suffix character “号”(date) as in “5月/15号”(May 15th) also tends to cause errors to happen. In the UPUC's training set, the character “号” with the context of “date” seldom occurs; therefore, there are few patterns that contain “号” and thus, not only for producing N-best candidates using *CRF++* but also in weight update during perceptron learning, its related patterns cannot be emphasized. However, this suffix frequently appears in the test set, and it is therefore segmented incorrectly (e.g. to become “15/号”).

## 4.4 Global Feature Weight Learning

The word segmentation system, designed by Liang in [26], incorporated and learned the weights for mutual information (MI) features, whose values are continuous. In the weight learning process, in order to deal with the mismatch between continuous and binary features, Liang transformed the MI values into either of the following forms:

- Scale the MI values into some fixed range [a, b], where the smallest MI value maps to a, and the largest MI value maps to b.

- Apply z-scores instead of their original MI values. The *z-score* of an MI value $x$ is defined as $\frac{x-\mu}{\sigma}$ where $\mu$ and $\sigma$ represent the mean and standard deviation of the MI distribution, respectively.

- Map any MI value $x$ to a if $x < \mu$, the mean MI value, or to b if $x \geq \mu$.

Testing with those various transformations, Liang shows that the weight for the MI feature can be learned in the same way as the weight for the binary features and that the word segmentation performance increases by normalizing the MI values. Especially, the highest increase is obtained by normalizing them with z-scores.

For our global features, both sentence confidence score and sentence language model score have the property that their values are also continuous rather than discrete. Therefore, we try to see whether Liang's method to incorporate MI features could be applied to automatically learn weights for our two global features during perceptron training, instead of manually fixing their weight using the development set.

We experiment with the transformations on those two global features with the UPUC and CityU corpora. Table 4.11 provides the performance on their development sets as well as test sets.

| Method | F-score (UPUC corpus) | | F-score (CityU corpus) | |
|---|---|---|---|---|
| | held-out set | test set | held-out set | test set |
| Without global features | 95.5 | 92.5 | 97.3 | 96.7 |
| Fixed global feature weights | **96.0** | **93.1** | **97.7** | **97.1** |
| Threshold at mean to 0,1 | 95.0 | 92.0 | 96.7 | 96.0 |
| Threshold at mean to -1,1 | 95.0 | 92.0 | 96.6 | 95.9 |
| Normalize to [0,1] | 95.2 | 92.1 | 96.8 | 96.0 |
| Normalize to [-1,1] | 95.1 | 92.0 | 96.8 | 95.9 |
| Normalize to [-3,3] | 95.1 | 92.1 | 96.8 | 96.0 |
| Z-score | 95.4 | 92.5 | 97.1 | 96.3 |

Table 4.11: F-scores (in percentage) obtained by using various ways to transform global feature weights and by updating their weights in averaged perceptron learning. The experiments are done on the UPUC and CityU corpora.

Different transformations give different performance. Among the normalization methods, the one with z-scores has the highest F-score. However, all of those accuracies are worse than our previous method for fixing global feature weights using the development set. As a result, better methods to perform weight updates for global features need to be explored further in the future.

## 4.5 Comparison to Beam Search Decoding

### 4.5.1 Beam search decoding

In Zhang and Clark's paper [53], instead of applying the N-best re-ranking method, their word segmentation system adapts beam search decoding [10, 33], using only local features.

In beam search, the decoder generates segmentation candidates incrementally. It reads one character at a time from the input sentence, and combines it with each existing candidate in two ways, either appending this new character to the last word, or considering it as the beginning of a new word. This combination process generates segmentations exhaustively; that is, for a sentence with $k$ characters, all $2^{k-1}$ possible segmentations are generated.

We implemented the decoding algorithm following the pseudo-code described by Zhang and Clark [53]. According to their pseudo-code (see Figure 4.8), one source agenda and one target agenda, both being initially empty, are used. In each step, the decoder combines the character read from the input sentence with each candidate in the source agenda, and puts the results into the target agenda. After processing the current character, the source agenda is cleared, each item in the target agenda is copied back into the source agenda to form the new candidates which will be used to process the next character, and then, the target agenda is cleared. After the last character in the current input sentence is processed, the candidate with the best score in the source agenda is returned by the decoder for training the averaged perceptron.

To guarantee reasonable running speed, the beam size is limited to be $B$, a value that is usually much less than $2^{k-1}$, which means that after processing each character, only the $B$ best candidates are preserved.

### 4.5.2 Experiments

During training with the Peking University corpus (PU), which contains 19,056 sentences, from the first SIGHAN bakeoff, we observe that the running speed for this beam search decoding based segmentation system is low. To examine whether we can use only a partial training corpus in the averaged perceptron learning without downgrading the accuracy, we divide the PU corpus into a training set (80% of the corpus) and a development set (20% of the corpus). Initially, only 1,000 sentences from the training set are used in the perceptron learning. Then, 1,000 more sentences are incrementally added in each following experiment

```
Inputs: raw sentence sent - a list of characters
Initialization: Set agendas $src = [[]]$, $tgt = []$
Variables: candidate sentence item - a list of words
Algorithm:
  for $index = 0 \ldots sent$.length-1 do
    var $char = sent[index]$
    for each item in src do
      //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 then
        var $item_2 = item$
        $item_2[item_2$.length-1].append(char)
        tgt.insert($item_2$)
      end if
      $src = tgt$
      $tgt = []$
    end for
  end for
Outputs: src.best_item
```

Figure 4.8: Beam search decoding algorithm, from Figure 2 in Zhang and Clark's paper [53]

to observe the changes in the F-score. For all experiments, the number of iterations are set to be 6, and the beam size is fixed to be 16, matching the parameters used in [53]. Table 4.12 and Figure 4.9 show the performance.

| Number of Training Sentences | 1,000 | 2,000 | 3,000 | 4,000 | 5,000 |
|---|---|---|---|---|---|
| F-score on the held-out set | 86.6 | 89.6 | 91.0 | 92.0 | 92.6 |
| Number of Training Sentences | 6,000 | 7,000 | 8,000 | 9,000 | 10,000 |
| F-score on the held-out set | 93.1 | 93.4 | 93.8 | 94.1 | 94.2 |
| Number of Training Sentences | 11,000 | 12,000 | 13,000 | 14,000 | 15,244 |
| F-score on the held-out set | 94.5 | 94.7 | 94.8 | 95.0 | 95.2 |

Table 4.12: F-scores (in percentage) with different training set sizes for the averaged perceptron learning with beam search decoding

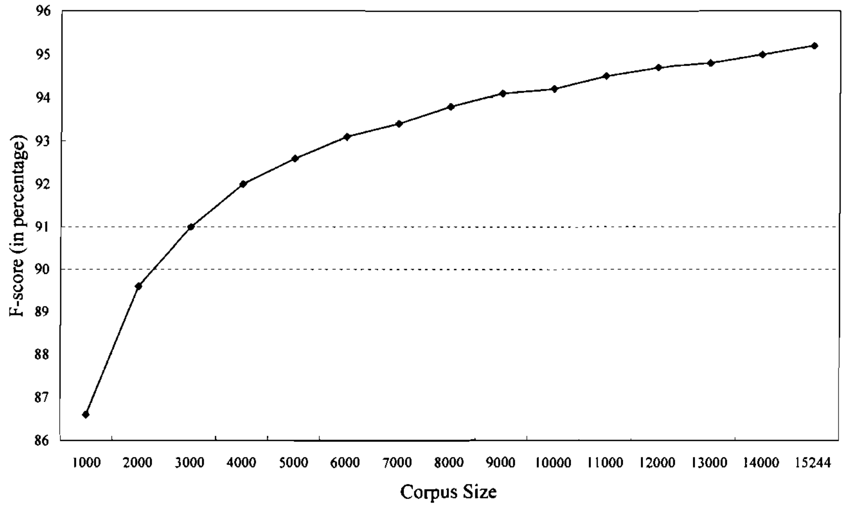

Figure 4.9: F-scores (in percentage) on the PU development set with increasing training corpus size

From Figure 4.9, we see that, by increasing the number of training examples, the F-score increases instead of converging to a certain value. As a result, despite low running speed, the whole training corpus has to be involved in order to produce the highest accuracy.

Then, we applied this system to the PU corpus to confirm the correctness of our implementation and to replicate the experimental result produced in [53] on the PU corpus.

Finally, the performance of this system is compared with that of the N-best re-ranking system on the PU corpus from the first SIGHAN bakeoff, and on the CityU, MSRA, UPUC corpora from the third SIGHAN bakeoff. For simplicity, the beam size was set to be 16 for all corpora, and the number of iterations was set to be 7, 7 and 9 for the CityU, MSRA and UPUC corpora, respectively, corresponding to the iteration values we applied on each corpus in the re-ranking system.

For N-best re-ranking method on the PU corpus, we applied the same parameter pruning process as before to select the weight value for global features and the iteration number. Figure 4.10 compares the F-scores using different $S_{crf}$ and $S_{lm}$ weight values with different number of iterations $t$ on the PU development set. We chose the global feature weight values to be 40 and the number of iterations to be 6.

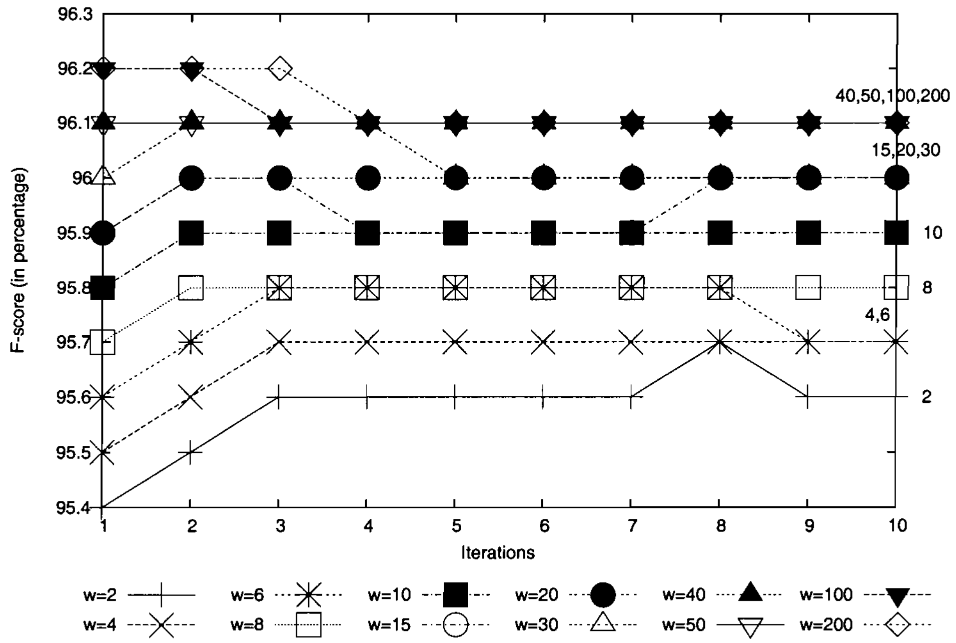

Figure 4.10: F-scores on the PU development set

Figure 4.11 shows the comparison results between the averaged perceptron training using the beam search decoding method and that using the re-ranking method. For each corpus, the bold number represents the highest F-score. From the result, we see that on

| Corpus | Setting | F | P | R | $R_{IV}$ | $R_{OOV}$ |
|---|---|---|---|---|---|---|
| PU | Averaged perceptron with beam search decoding | **94.1** | 94.5 | 93.6 | 69.3 | 95.1 |
| | Averaged perceptron with re-ranking, containing global and local features | 93.1 | 93.9 | 92.3 | 94.2 | 61.8 |
| | Averaged perceptron with re-ranking, containing only local features | 92.2 | 92.8 | 91.7 | 93.4 | 62.3 |
| CityU | Averaged perceptron with beam search decoding | 96.8 | 96.8 | 96.8 | 97.6 | 77.8 |
| | Averaged perceptron with re-ranking, containing global and local features | **97.1** | 97.1 | 97.1 | 97.9 | 78.3 |
| | Averaged perceptron with re-ranking, containing only local features | 96.7 | 96.7 | 96.6 | 97.5 | 77.4 |
| MSRA | Averaged perceptron with beam search decoding | 95.8 | 96.0 | 95.6 | 96.6 | 66.2 |
| | Averaged perceptron with re-ranking, containing global and local features | 95.8 | 95.9 | 95.7 | 96.9 | 62.0 |
| | Averaged perceptron with re-ranking, containing only local features | 95.5 | 95.6 | 95.3 | 96.3 | 65.4 |
| UPUC | Averaged perceptron with beam search decoding | 92.6 | 92.0 | 93.3 | 95.8 | 67.3 |
| | Averaged perceptron with re-ranking, containing global and local features | **93.1** | 92.5 | 93.8 | 96.1 | 69.4 |
| | Averaged perceptron with re-ranking, containing only local features | 92.5 | 91.8 | 93.1 | 95.5 | 68.8 |

Table 4.13: Performance (in percentage) Comparison between the averaged perceptron training using beam search decoding method and that using re-ranking method

the CityU, MSRA and UPUC corpora, although the beam search decoding based system still outperforms the re-ranking based system using only local features, the re-ranking based system containing global features performs as good as or even better than the beam search decoding based system. Therefore, it confirms again that global features have great influence on performance in most cases.

On the other hand, for the PU corpus from the first SIGHAN bakeoff, the re-ranking based method has worse performance than the beam search decoding based one. To explore the rationale behind this phenomena, for each of the CityU, MSRA, UPUC, and PU test sets, we examine how many sentences in the gold standard also appear within the 20-best candidate list. Table 4.14 shows the corresponding ratios. From this table, we find that

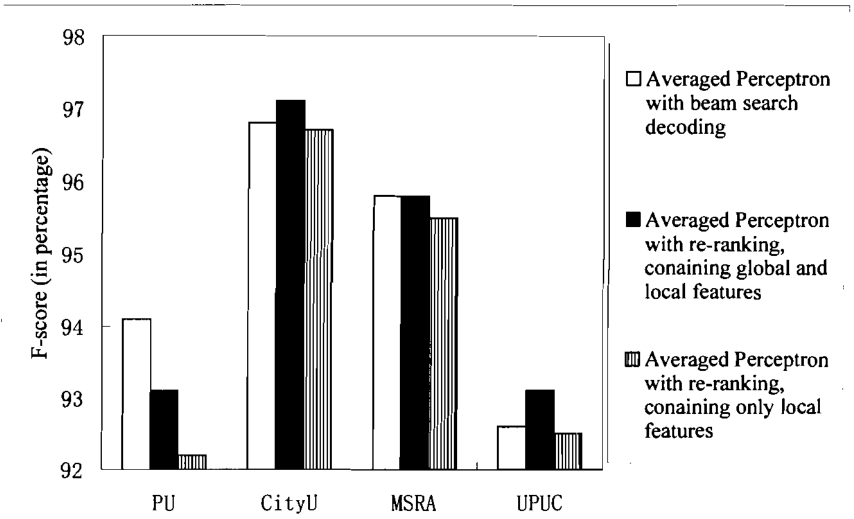

Figure 4.11: Comparison for F-scores between the averaged perceptron training using beam search decoding method and that using re-ranking method, with histogram representation

| | CityU | MSRA | UPUC | PU |
|---|---|---|---|---|
| Ratio | 88.2% | 88.3% | 68.4% | 54.8% |

Table 4.14: The ratio from examining how many sentences in the gold standard also appear within the 20-best candidate list, for the CityU, MSRA, UPUC, and PU test sets

for the PU test set, almost half of the true segmentations are not seen in the 20-best list, which seriously affects the re-ranking process to pick up the correct candidate. While for the CityU and MSRA corpora, nearly 90% of the gold standard appear in the 20-best candidate lists, which provide better chances for the correct candidates to be picked up. Thus, in order for the re-ranking method to have high performance, the quality of its candidate list is extremely important. In comparison, for the beam search decoding based method, the process of incremental character concatenation can produce more candidates to be examined, without limiting itself to certain pre-defined 20-best candidates.

## 4.6 Exponentiated Gradient Algorithm

In the next experiment, we implement the batch exponentiated gradient (EG) algorithm (see Figure 2.14) containing those two global features, explore the convergence for primal (Equation 2.7) and dual (Equation 2.8) objective functions, and compare the performance with the perceptron learning method, on the UPUC corpus.

In implementing the batch EG algorithm, during the initialization phase, the initial values of $\alpha_{i,y}$ are set to be 1/(number of N-best candidates for $x_i$). Also, in the dual variable update stage, considering Equation 2.6

$$\alpha'_{i,y} \leftarrow \frac{\alpha_{i,y}e^{\eta\nabla_{i,y}}}{\sum_y \alpha_{i,y}e^{\eta\nabla_{i,y}}}$$

where

$$\nabla_{i,y} = \begin{cases} 0 & \text{for } y = y_i \\ 1 - M_{i,y} & \text{for } y \neq y_i \end{cases}$$

In order to get $\alpha'_{i,y}$, we need to calculate $e^{\eta\nabla_{i,y}}$. When each $\nabla$ in the N-best list is positively or negatively too large, numerical underflow occurs. To avoid this problem, $\nabla$ is normalized, and the above equation is modified to become

$$\alpha'_{i,y} \leftarrow \frac{\alpha_{i,y}e^{\eta\nabla_{i,y}/\sum_y|\nabla_{i,y}|}}{\sum_y \alpha_{i,y}e^{\eta\nabla_{i,y}/\sum_y|\nabla_{i,y}|}} \tag{4.1}$$

in our implementation.

As before, the weight for global features is pre-determined using the development set and is fixed during the learning process. Taking the difference on learning efficiency between online update for perceptron learning and batch update for EG method into consideration, the maximum number of iterations is set to be larger (T = 25) in the latter case during parameter pruning. The weight for the global features are tested with 2, 5, 10, 30, 50, 70, and 90. Figure 4.12 shows the performance on the UPUC held-out set with various parameters.

The experimental result tells us that, initially, as the number of iteration increases, the F-score produced by EG method increases as well. However, larger numbers of iterations could introduce over-fitting, causing the F-score to drop. In addition, the figure shows that a larger weight for the global features produces better segmentation result. Therefore, we select the number of iterations to be 22 and the weight for global features to be 90, and

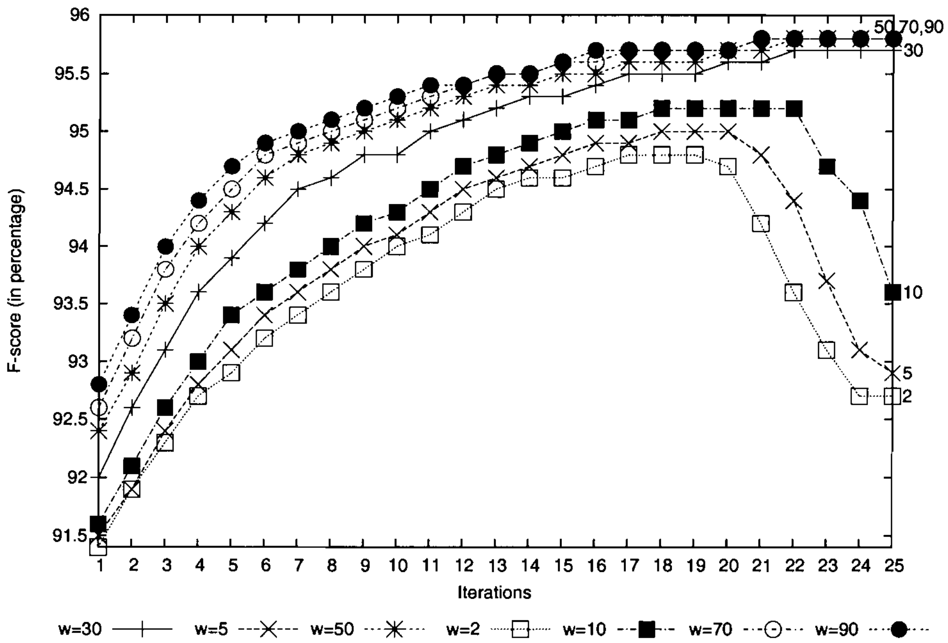

Figure 4.12: F-scores on the UPUC development set for EG algorithm

apply these parameters on the UPUC test set. Table 4.15 lists the resulting performance. In this table, not only do we show the performance produced by the EG method using global features, but also we list the performance from the EG method with only local features and that from perceptron learning methods. Moreover, performance of the EG method with the same number of iterations (t = 9) as the averaged perceptron method and that from the character-based CRF method are listed as well. The bold number represents the highest F-score.

From Table 4.15 and Figure 4.13, we see that the averaged perceptron with global features still gives the highest F-score. Although the EG algorithm with global features has better performance than the character-based CRF method, it takes more iterations in achieving that result, and also it still performs worse than the averaged perceptron with global features.

Continuing to run the EG algorithm for more iterations (T = 120) with the weight of global features being fixed at 90, Figure 4.14 gives us the changes in primal and dual objective functions. From the figure, we can see that the algorithm does in fact converge to the maximum margin solution on this data set. However at iteration 120, the F-score

| Setting | F | P | R | $R_{IV}$ | $R_{OOV}$ |
|---|---|---|---|---|---|
| EG algorithm with global and local features | 93.0 | 92.3 | 93.7 | 96.1 | 68.2 |
| EG algorithm with only local features | 90.4 | 90.6 | 90.2 | 92.2 | 69.7 |
| EG algorithm with global and local features, with number of iterations being 9 | 92.4 | 91.7 | 93.1 | 95.5 | 67.6 |
| Averaged Perceptron with global and local features | **93.1** | 92.5 | 93.8 | 96.1 | 69.4 |
| Averaged Perceptron with only local features | 92.5 | 91.8 | 93.1 | 95.5 | 68.8 |
| Character-based CRF method | 92.8 | 92.2 | 93.3 | 95.5 | 70.9 |

Table 4.15: Performance (in percentage) from the EG algorithms, comparing with those from the perceptron learning methods and the character-based CRF method

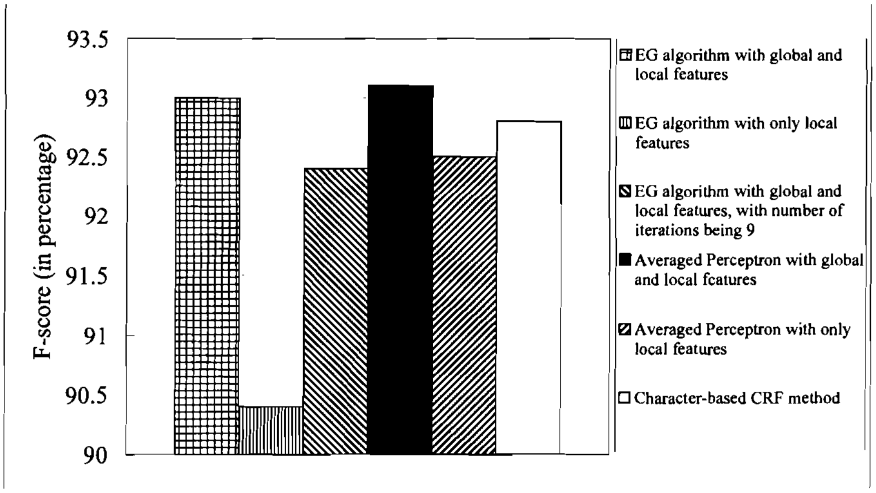

Figure 4.13: Comparison for F-scores from the EG algorithms, with those from the perceptron learning methods and the character-based CRF method, using histogram representation

remains 0.930, which is the same as the F-score produced in the 22nd iteration.

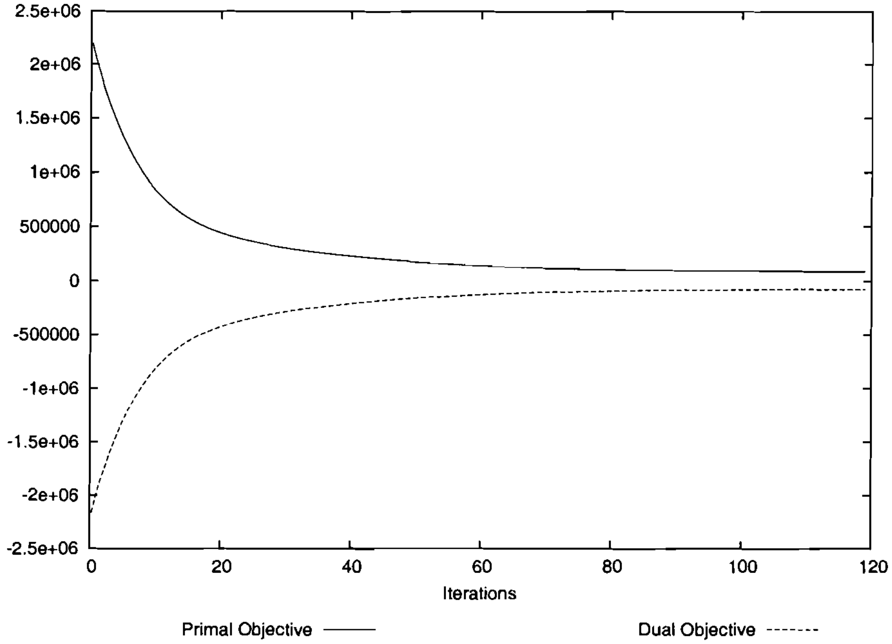

Figure 4.14: EG algorithm convergence on the UPUC corpus

## 4.7 Related Work

Re-ranking over N-best lists has been applied to so many tasks in natural language that it is not possible to list them all here. Closest to our approach is the work done by Kazama and Torisawa [19], for named entity recognition (NER). They proposed a max-margin perceptron algorithm that exploited non-local features on an N-best list. Instead of using averaged perceptron, their method for NER tries to maximize the margin between the best scoring candidate and the second best scoring candidate, applying the original perceptron algorithm with local features and non-local features that are defined on partial sentences. In contrast, the averaged perceptron algorithm is used in our system, and global features are used to examine the entire sentence instead of partial phrases. For word segmentation, Wang and Shi [46] implemented a re-ranking method with POS tagging features. In their approach, character-based CRF model produces the N-best list for each test sentence. The Penn

Chinese TreeBank is used to train a POS tagger, which is used in re-ranking. However, the POS tags are used as local and not global features. Note that our experiments focus on the closed track, so we cannot use POS tags. In machine translation, Shen et al. [38] investigated the use of perceptron-based re-ranking algorithm, looking for parallel hyper-planes splitting the top $r$ translations and the bottom $k$ translations of the N-best translations for each sentence, where $r + k \leq n$, instead of separating the one-best candidate with the rest. Also in the tagging task, Huang et al. [18] modified Collins' re-ranking algorithm [6], utilizing n-gram features, morphological features and dependency features, for the Mandarin POS tagging task.

## 4.8 Summary of the Chapter

In this chapter, we described our system for training a perceptron with global and local features for Chinese word segmentation. We have shown that by combining global features with local features, the averaged perceptron learning algorithm based on re-ranking produces significantly improved results, compared with the algorithm only using local features and the character-based one-best conditional random field method. Also, we attempted to automatically learn weights for global features. In addition, by comparing our system with the beam search decoding based perceptron learning, we show again that global features are useful. Moreover, the performance of the perceptron training is compared with that of the exponentiated gradient method. We show that the averaged perceptron with global features gives higher performance.

# Chapter 5

# Conclusion

In this thesis, we looked at the Chinese word segmentation problem, and reviewed various common approaches applied in the literature. Also, we provided and evaluated two specific Chinese word segmentation systems. This final chapter summarizes the contents of the thesis in Section 5.1, re-emphasizes the contributions of our proposed approaches in Section 5.2, and points out certain possibilities for future work in Section 5.3.

## 5.1 Thesis Summary

In Chapter 2, various approaches dealing with Chinese word segmentation were described. We classified these approaches into three main categories: the *dictionary-based matching* approach, the *character-based or subword-based sequence learning* approach, and the *global linear model* approach. Each of these categories, together with certain specific algorithms, were explained in detail.

Chapter 3 described a character-level majority voting Chinese word segmentation system, which voted among the initial outputs from the greedy longest matching, from the CRF model with maximum subword-based tagging, and from the CRF model with minimum subword-based tagging. In addition, we experimented with various steps in post-processing which effectively improved the overall performance. Moreover, a related voting system, replacing the minimum subword-based CRF model with the character-based CRF model during the majority voting, was described and compared.

Chapter 4 proposed the use of two global features, the sentence confidence score global feature and the sentence language model score global feature, to assist with local features in training an averaged perceptron on N-best candidates for Chinese word segmentation. We performed extensive experiments to compare the performance achieved from our system with that achieved from the character-based CRF tagger, and also with that achieved from perceptron learning using only local features. In addition, the beam search decoding algorithm and the exponentiated gradient algorithm were implemented and compared with our averaged perceptron learning system.

## 5.2 Contribution

The main contributions of this thesis in general are as follows:

- We show that the majority voting approach helps improve the segmentation performance over its individual algorithms. The voting procedure successfully combines the dictionary information used in the greedy longest matching algorithm and in the maximum subword-based CRF model, and thus raises the low in-vocabulary recall rate produced by the two CRF-based discriminative learning algorithms. Also, the voting procedure benefits from the high out-of-vocabulary recall rate achieved from these two CRF-based algorithms. Thus, our majority voting system raises the performance of each individual algorithms.
- We discover that by combining global features with local features, the averaged perceptron learning algorithm based on re-ranking produces significantly improved results, compared to the character-based one-best CRF algorithm and the averaged perceptron algorithm with only local features.

## 5.3 Future Research

There are many aspects of Chinese word segmentation that need to be further explored. For instance, how should we effectively solve the unknown word problem, or at least what new strategies could help to minimize its negative consequence and thus produces a significantly higher out-of-vocabulary recall rate?

Also, in our perceptron training system, only two global features, the sentence confidence score and the sentence language model score, are considered. Are there any other meaningful global features that can also be included? In the future, we would like to explore more global features that are useful for perceptron learning. Also, the weight for the global features are pre-determined and fixed in our system. Although we tried to update the weight for these features during the learning process, the experimental accuracies were not encouraging. Finding better methods for updating weights for the global features so that the perceptron learning process can be done without a development set is an important topic for future work.

In addition, during beam search based perceptron learning, only local features are included. The language model score, as one of the global features, is able to provide additional information about partial sentence candidates, and can be computed through the use of a language model (e.g. use SRILM to score the partial sentence). In our experiments, we attempted to involve the language model score feature in the beam search decoder: First, SRILM is set up on the server side; then, the word segmentation system with beam search decoder is run on the client side. For each sentence, after combining the next character with all candidates up until the current character, the updated candidate list is sent to the server, producing the language model score for each candidate, and then these language model scores are sent back to the client system; After combining the language model feature score with other local feature scores, the top $B$ best candidates in the list, where $B$ is the beam size, are retained and carried on to the next step. During the whole process, the weight for the language model score feature is selected using the development set and is fixed afterward. However, due to the delay caused by client-server communication and the time spent on the calculation of language model score for each candidate, the running speed is extremely slow. Also, the exhaustive weight determination for the global feature on development set takes long time to complete as well. Thus, we have to regrettably abandon this experiment and try to find a faster way to proceed. Effectively using not only the language model score feature but also other global features inside beam search decoding and then comparing its performance with other methods is an interesting experiment that we leave for future work.

# Bibliography

[1] Indrajit Bhattacharya, Lise Getoor, and Yoshua Bengio. Unsupervised sense disambiguation using bilingual probabilistic models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 287–294, Barcelona, Spain, July 2004.

[2] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms using different performance metrics. Technical Report TR2005-1973, Cornell University, 2005.

[3] Keh-Jiann Chen and Shing-Huan Liu. Word identification for mandarin chinese sentences. In *Proceedings of the 14th conference on Computational linguistics*, pages 101–107, Morristown, NJ, USA, 1992. Association for Computational Linguistics.

[4] Wenliang Chen, Yujie Zhang, and Hitoshi Isahara. Chinese named entity recognition with conditional random fields. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*, pages 118–121, Sydney, Australia, July 2006. Association for Computational Linguistics.

[5] Shiren Ye Tat-Seng Chua and Liu Jimin. An agent-based approach to chinese named entity recognition. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[6] Michael Collins. Discriminative reranking for natural language parsing. In *Proc. 17th International Conf. on Machine Learning*, pages 175–182. Morgan Kaufmann, San Francisco, CA, 2000.

[7] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP). MACL*, pages 1–8, Philadelphia, PA, USA, July 2002. Association for Computational Linguistics.

[8] Michael Collins. Ranking algorithms for named entity extraction: Boosting and the voted perceptron. In *Proceedings of ACL 2002*, pages 489–496, Philadelphia, PA, USA, July 2002. Association for Computational Linguistics.

[9] Michael Collins. Parameter estimation for statistical parsing models: theory and practice of distribution-free methods. pages 19–55, 2004.

[10] Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain, July 2004.

[11] Aron Culotta, Andrew McCallum, and Jonathan Betz. Integrating probabilistic extraction models and data mining to discover relations and patterns in text. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 296–303, New York City, USA, June 2006. Association for Computational Linguistics.

[12] Thomas Emerson. The second international chinese word segmentation bakeoff. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, pages 123–133, Jeju Island, Korea, October 2005. Asian Federation of Natural Language Processing.

[13] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Mach. Learn.*, 37(3):277–296, 1999.

[14] L. Gillick and Stephen Cox. Some statistical issues in the comparison of speech recognition algorithms. In *Proc. of IEEE Conf. on Acoustics, Speech and Sig. Proc.*, pages 532–535, Glasgow, Scotland, May 1989.

[15] Amir Globerson, Terry Y. Koo, Xavier Carreras, and Michael Collins. Exponentiated gradient algorithms for log-linear structured prediction. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 305–312, New York, NY, USA, 2007. ACM.

[16] Simon I. Hill and Robert C. Williamson. Convergence of exponentiated gradient algorithms. In *IEEE Trans. Signal Processing*, volume 49, pages 1208–1215, June 2001.

[17] Julia Hockenmaier and Mark Steedman. Generative models for statistical parsing with combinatory categorial grammar. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 335–342, Philadelphia, USA, July 2002. Association for Computational Linguistics.

[18] Zhongqiang Huang, Mary Harper, and Wen Wang. Mandarin part-of-speech tagging and discriminative reranking. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 1093–1102.

[19] Jun'ichi Kazama and Kentaro Torisawa. A new perceptron algorithm for sequence labeling with non-local features. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 315–324.

[20] Balázs Kégl and Guy Lapalme, editors. *Advances in Artificial Intelligence, 18th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2005, Victoria, Canada, May 9-11, 2005, Proceedings*, volume 3501 of *Lecture Notes in Computer Science*. Springer, 2005.

[21] Kyungduk Kim, Yu Song, and Gary Geunbae Lee. POSBIOTM/W: A development workbench for machine learning oriented biomedical text mining system. In *Proceedings of HLT/EMNLP 2005 Interactive Demonstrations*, pages 36–37, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics.

[22] Jyrki Kivinen and Manfred Warmuth. Exponentiated gradient versus gradient descent for linear predictors. Technical Report UCSC-CRL-94-16, UC Santa Cruz, 1994.

[23] Taku Kudo, Yamamoto Kaoru, and Matsumoto Yuji. Applying conditional random fields to japanese morphological analysis. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 230–237, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[24] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[25] Gina-Anne Levow. The third international chinese language processing bakeoff: Word segmentation and named entity recognition. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*, pages 108–117, Sydney, Australia, July 2006. Association for Computational Linguistics.

[26] P. Liang. Semi-supervised learning for natural language. Master's thesis, Massachusetts Institute of Technology, 2005.

[27] Yuan Liu, Qiang Tan, and Xukun Shen. Chinese national standard gb13715: Contemporary chinese word segmentation standard used for information processing. In *Information Processing with Contemporary Mandarin Chinese Word Segmentation Standard and Automatic Segmentation Method*. Tsinghua University Press, 1994.

[28] Albert Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on Mathematical Theory of Automata*, pages 615–622. Polytechnic Institute of Brooklyn, 1962.

[29] Constantine P. Papageorgiou. Japanese word segmentation by hidden markov model. In *HLT '94: Proceedings of the workshop on Human Language Technology*, pages 283–288, Morristown, NJ, USA, 1994. Association for Computational Linguistics.

[30] Fuchun Peng, Fangfang Feng, and Andrew McCallum. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of Coling 2004*, pages 562–568, Geneva, Switzerland, August 2004.

[31] Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

[32] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications inspeech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286. Institute of Electrical and Electronics Engineers, Feburary 1989.

[33] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In Eric Brill and Kenneth Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 133–142, Somerset, New Jersey, 1996. Association for Computational Linguistics.

[34] Frank Rosenblatt. The perception: a probabilistic model for information storage and organization in the brain. 65(6):386–408, 1958.

[35] Erik F. Tjong Kim Sang and Jorn Veenstra. Representing text chunks. In *Proceedings of EACL 1999*, pages 173–179, Bergen, Norway, June 1999. Association for Computational Linguistics.

[36] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 134–141, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[37] Hong Shen and Anoop Sarkar. Voting between multiple data representations for text chunking. In Kégl and Lapalme [20], pages 389–400.

[38] Libin Shen, Anoop Sarkar, and Franz Josef Och. Discriminative reranking for machine translation. In *HLT-NAACL 2004: Main Proceedings*, pages 177–184, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

[39] Richard Sproat and Thomas Emerson. The first international chinese word segmentation bakeoff. In *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing*, Sapporo, Japan, July 2003. Association for Computational Linguistics.

[40] Richard Sproat, William Gale, Chilin Shih, and Nancy Chang. A stochastic finite-state word-segmentation algorithm for chinese. *Comput. Linguist.*, 22(3):377–404, 1996.

[41] Andreas Stolcke. SRILM – an extensible language modeling toolkit. In *Proceedings of the ICSLP*, Denver, Colorado, 2002.

[42] Maosong Sun and Benjamin K T'sou. Ambiguity resolution in chinese word segmentation. In *Proceedings of the 10th Pacific Asia Conference on Language, Information and Computation*, pages 121–126, Hong Kong, China, December 1995. City University of Hong Kong.

[43] Charles Sutton and Andrew Mccallum. An introduction to conditional random fields for relational learning. In *Introduction to Statistical Relational Learning.* MIT Press, 2006.

[44] Cynthia Thompson, Siddharth Patwardhan, and Carolin Arnold. Generative models for semantic role labeling. In *Senseval-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 235–238, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[45] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In *IEEE Transactions on Information Theory*, volume 13, pages 260–269, April 1967.

[46] Mengqiu Wang and Yanxin Shi. Using part-of-speech reranking to improve chinese word segmentation. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*, pages 205–208, Sydney, Australia, July 2006. Association for Computational Linguistics.

[47] Andi Wu. Customizable segmentation of morphologically derived words in chinese. *Computational Linguistics and Chinese Language Processing*, 8(1):1–27, 2003.

[48] Nianwen Xue. Chinese word segmentation as character tagging. *Computational Linguistics and Chinese Language Processing*, 8(1):29–48, 2003.

[49] Shiwen Yu, Huiming Duan, Bin Swen, and Bao-Bao Chang. Specification for corpus processing at peking university: Word segmentation, pos tagging and phonetic notation. *Journal of Chinese Language and Computing*, 13, 2003.

[50] Xiaofeng Yu, Marine Carpuat, and Dekai Wu. Boosting for chinese named entity recognition. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*, pages 150–153, Sydney, Australia, July 2006. Association for Computational Linguistics.

[51] Huipeng Zhang, Ting Liu, Jinshan Ma, and Xiantao Liu. Chinese word segmentation with multiple postprocessors in HIT-IRLab. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, pages 172–175, Jeju Island, Korea, Octber 2005. Asian Federation of Natural Language Processing.

[52] Ruiqiang Zhang, Genichiro Kikui, and Eiichiro Sumita. 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 2006. Association for Computational Linguistics.

[53] Yue Zhang and Stephen Clark. Chinese segmentation with a word-based perceptron algorithm. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 840–847, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[54] Hai Zhao, Chang-Ning Huang, and Mu Li. An improved chinese word segmentation system with conditional random field. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*, pages 162–165, Sydney, Australia, July 2006. Association for Computational Linguistics.

[55] Junsheng Zhou, Liang He, Xinyu Dai, and Jiajun Chen. Chinese named entity recognition with a multi-phase model. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*, pages 213–216, Sydney, Australia, July 2006. Association for Computational Linguistics.