University of Groningen

Faculty of Arts

MASTER THESIS

Jianqiang Ma

N-gram Feature Generation for Conditional Random Fields based Chinese Word Segmentation

Supervisor: Dr. Gertjan van Noord

European Masters Program in Language and Communication Technologies (LCT)
2010

University of Malta

Department of Intelligent Computer Systems

MASTER THESIS

Jianqiang Ma

N-gram Feature Generation for Conditional Random Fields based Chinese Word Segmentation

Supervisor: Michael Rosner

European Master of Science in Human Language Science and Technology (HLST)
2010

Abstract

This thesis proposes an approach to generating n-gram features for Conditional Random Fields (CRFs) based Chinese word segmentation (CWS) systems.

Current systems typically adopt only unigrams or bigrams of either Chinese character themselves or certain traits of characters (e.g. tones, whether it is a digit) as features, though longer n-grams may be useful to describe long-distance dependencies and other phenomena. Inspired by works in error mining, a framework of n-gram expansion has been proposed to generate n-gram of arbitrary length as features for CRF based CWS. The expansion is only triggered when certain criteria are satisfied. Under this framework, we have proposed three specific expansion methods, based on mutual information, label-entropy and label distribution bin, respectively.

Expanded n-grams can be further fed to an iterative process that ranks these candidates and selects most promising ones. Those n-gram features are added into the feature set of CRFs algorithm, and the usefulness of selected features is evaluated by measuring overall performance gain of the CWS system. Using the UPUC corpus in SIGHAN Bakeoff 2006, experiments show that error reductions between 5%~13% are achieved by adding these generated features to a CRFs based CWS system that adopts standard features. Besides, the overall system performance is comparable to top results in Bakeoff 2006.

Declaration

I hereby declare that this diploma thesis is my own work and where it draws on the work of others it is properly cited in the text.

I understand that may thesis may be made available to the public.

August 23, 2010 Jianqiang Ma

Acknowledgements

I wish to express my greatest gratitude to my advisors Dr. Gertjan van Noord (Groningen) and Mr. Mike Rosner (Malta) for their guidance, support and advices. I would like to thank Dr. Chunyu Kit of the City University of Hong Kong, who inspired and supported this work. I also want to thank Daniel de Kok, who introduced me the Error Mining project and helped me a lot on C++ programming. I'd like to thank all the faculty members and fellow students in Groningen and Malta, from whom I benefited a lot both academically and personally.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Chinese Word Segmentation: the Problem

Unlike most European languages such as English, Chinese and many other Asian language such as Japanese, Tai do not delimit words by explicit white-space. In particular, a Chinese sentence is made up of a sequence of Chinese characters directly adjacent to one another, as shown in Figure 1.

A sentence in Chinese: 互联网是一个伟大的发明。

Engllish translation: Internet is a great invention.

Figure 1 A sample Chinese sentence with English tanslation

The task of Chinese word segmentation (CWS) is to segment an input sequence of characters into a sequence of words. The sentence in Figure 1 after the segmentation is shown in Figure 2. In that sentence, the first three Chinese characters together form the word meaning *Internet*, thus there is a white space after the third character to indicate the word boundary. Most Chinese words consist of two or more characters, but some characters can stand alone as words themselves, as "是" in the sample sentence.

A segmented sentence in Chinese: 互联网 是 一个 伟大的 发明。

Corresponding English words: Internet be/is a great invention.

Figure 2 The sentence in Figure 1 after segmentation. The segmentation shows word boundaries with while space

1.2 Chinese Word Segmentation: Debate and Usefulness

There has been much debate over what how to define a Chinese word. According to Sproat et al. (1996), there is only 0.76 (out of 1.00) agreement among human judges on what constitutes a Chinese word. If words are ill defined, then why shall we do the CWS anyway? It seems that researchers tend to process Chinese in the same way as

in European languages, whose basic unit is word. The advantage is that once the Chinese characters are segmented into words, most statistical techniques for English can be adapted easily to solve Chinese problems, as long as annotated corpora are built. Thus CWS is considered as a precursor for further natural language processing (NLP) which separates at word level (e.g. part-of-speech tagging, parsing, information retrieval and machine translation).

If solving those tasks is the ultimate goal, then probably it is not necessary to have CWS as a separate step in the language-processing pipeline. In fact, recently research paid much attention to process CWS and POS tagging as a joint process and it showed improvements over a pipelined fashion (Ng and Low, 2004; Jiang et al., 2008). People may even ask why should we need the word level anyway, given the Chinese are naturally represented by characters? Is it possible to process Chinese on the basis of characters directly? There are attempts to solve various NLP tasks based on characters, such as character-level dependency parsing (Zhao, 2009).

However these topics are beyond the coverage of this thesis, which still focuses on Chinese word segmentation itself and describes the proposed methods for extending current approaches to the problem.

1.3 Challenges of CWS

First, ambiguity in the segmentation. Just as an English sentence may have several parsing trees, plausible or implausible, there are often potentially several ways to segment a Chinese sentence. There are at least two types of ambiguities. One is that each segmentation is plausible yet has different meaning, as shown in Figure 3. The other one is that every single word in a segmentation is a valid word in the dictionary, but the segmentation as a whole is not plausible, as shown in Figure 4. Usually people pay more attention to type two ambiguity since the potential segmentations may be semantically incorrect and hurt further processing. It also implies a challenge in CWS: it is not possible to segment sentences by simply looking up dictionaries. Besides these two ambiguity problems, there are also granularity problems in CWS. For example, linguists have different views on whether 北京大学 (Peking University) is a single word or a phrase made up of two words 北京 (Peking) and 大学(University).

This example is similar to the English word “dataset”, which sometimes appears as the phrase “data set”.

A sentence in Chinese: 乒乓球拍卖完了

Interpretation 1: 乒乓球 拍卖 完了 (The ping-pong ball is auctioned)

Interpretation 1: 乒乓球拍 卖 完了 (The ping-pong racket is sold out)

Figure 3 An example sentence that has two plausible segmentation

A sentence in Chinese: 这是一个发展中国家

Correct segmentation: 这 是 一个 发展中 国家(It is a developing country)

Incorrect segmentation: 这 是 一个 发展 中国 家 (It is a develop China family)

Figure 4 An example of type two ambiguity. The character sequence 发展中国家can be segmented as either 发展中／国家(developing/country) or 发展／中国／家 (develop/China/country). Both segmentations are made up of valid words in Chinese, but only the former one makes sense.

Second, Out-of-vocabulary words (OOV) are a major cause of segmentation errors. On one hand, new words and terms appear rapidly every year in the information era and it is not possible to find a dictionary that has a wide coverage of them, as new words are often invented by unpredictable morphological transformations or through transliteration of foreign words. On the other hand, past research (Sproat and Emerson, 2003) reports that about 60% of incorrect segmentation by CWS systems are related with OOV. Thus OOV becomes a bottleneck of most CWS systems.

1.4 Approaches and Contribution of this Work

Conditional random field (CRF) based CWS systems have the state-of-the art performances in CWS evaluations. However, due to the huge computational cost, features used in CRF based systems are usually defined as unigrams or bigrams of Chinese characters. In this thesis, we have proposed a family of n-gram expansion based methods that could select potentially useful n-grams of arbitrary length from all the possible n-grams in given corpora, in the hope of describing longer dependencies and other rich linguistic phenomena while preventing the exponential growth of feature numbers. These methods share a uniform framework of n-gram expansion,

inspired by the method proposed in De Kok et al. (2009) for error mining in parsing results. The expansion processes are governed by some pre-defined criteria, which are based on both widely used concepts such as mutual information and originally proposed indicators such as label distribution bin to define these criteria.

These features can be integrated in CRF based CWS systems and improve their performance. In addition, these methods provide a generic way for automatic feature generation that is potentially useful for CRF based systems for other sequence labeling tasks.

Using corpora of CWS Bakeoff 2006 (Levow, 2006), our experiment shows that a considerable error reduction is achieved by adding these generated features, compared with CRF based methods that adopt standard features, and the overall system performance is comparable to top results in Bakeoff 2006.

Finally, N-expander, the system that implemented the proposed methods, is an open source software public available at http://gitorious.org/n-generator.

1.5 Structure of the Thesis

In Chapter 2, we review past approaches to Chinese word segmentation, both dictionary-based and statistical-based. We also briefly introduce works in error mining, by which our proposed methods are inspired. Chapter 3 describes the general framework for n-gram expansion based feature generation. Chapter 4 introduces three proposed methods, based on mutual information of Chinese characters, label entropy and label distribution bin, respectively. Chapter 5 describes some key algorithms and data structures in the implementation. Chapter 6 presents the experiments on standard data set with evaluation. Chapter 7 concludes the whole thesis.

Chapter 2 Related Work

Various methods have been proposed to address CWS. These methods roughly fall into two categories: heuristic dictionary-based methods and statistical machine learning methods. This chapter will give a general review of both with a focus on statistical machine learning methods, especially conditional random field (CRF). Since the framework for our approaches to generating n-gram based features is inspired by the n-gram expansion methods in De Kok et al.(2009), which were originally proposed for error mining in parsing results, several works of this topic will also be covered in this chapter.

2.1. Dictionary-based Methods for CWS

Most of the earlier work on Chinese word segmentation is lexical knowledge based (Liang, 1986; Kit et al., 1989) which relies on dictionaries to conduct the segmentation. As simple and efficient as it is, a considerable success has been achieved.

2.1.1 Maximum matching (MM)

With maximum matching (Liang, 1986), a character string is compared with the entries of a lexicon so that all the substrings constituting lexicon items are highlighted. The principle of maximum matching is to find the best segmentation with fewest and longest words among all the possible substring chains. It operates by scanning the text from left to right and iteratively matching the input string with the longest word in the dictionary until the end of the sentence is reached. Unfortunately, as the segmentation is determined locally, the resulting sentence segmentation is always suboptimal.

The MM approach always leads to one segmentation pattern resolving the ambiguity problem by ignoring it. As is the nature of the method, MM by itself is unable to deal with the composition of segmentation errors. For example, segmenting sentence in

“这些学生会游泳” MM always gives an incorrect segmentation “这些 学生会 游泳 ” as a result of the fact that it favors longer word 学生会 (“student union”) over short words 学生 (student) and 会 (“can/be able to”).

Despite all this weaknesses, MM has the advantages of simplicity in both algorithm and implementation and it can give a reasonable performance provided the dictionary suits the task well. There are also some variations of MM.

2.1.2 Heuristic rules

There have been a number of linguistic heuristics for resolving ambiguities in MM. Some rules describe constraints on syntactic or semantic features (Yeh and Lee, 1991) and lexical heuristics (Wang et al., 1990). One example is to use word boundary hints. In addition to punctuation marks, there are some other indications that could be helpful to improve segmentation accuracy. Those indications are heuristic linguistic knowledge such as those characters 1) that can be used only to begin a word; 2) that can be used to end a word and 3) that can only be used as a single character word.

2.1.3 Vitibi decoding

Guo (1997) introduced a word n-gram based dynamic programming method to solve the ambiguity problem in MM and decrease the error rate by one order of magnitude. A word lattice is built to keep all the possible segmentation results for each character sequence, each of which consists of sequences of valid words in the dictionary. In this case, every word is associated with a unigram and each word transition is associated with a word or word class. The Viterbi algorithm (Forney, 1973) is then applied to find the best path, which takes the word unigram and bigram into accounts. The resulting path will be the segmentation. Please note that MM itself can be viewed as an extreme case of Viterbi, in which only a single path is kept during the whole process.

2.2 Statistical Learning Methods

From a perspective of statistical learning, the CWS problem is just a special case of sequence labeling, which is the problem to assign a single label to each element in a sequence. For example, part-of-speech (POS) tagging is a well-known sequence labeling problem, in which the words are the elements and the labels to be assigned are POS tags. Chinese word segmentation can also be formulated as a sequence labeling problem in the following way: the task is to assign each character in the sentence a particular tag, which indicates the position of that character within a word. For example, using the simplest B-C tag set, in which B means the starting of a new word and C means the continuation of the current word, the sentence segmentation in Figure 5 can be represented as assigning the tag sequence of BBBCC to the original character sequence. Please note that there are several different tag-set for CWS. A very popular tag-set in practice is the "BMES" tagset, for each multi-character word, its first character is labeled as "B" (Beginning) tag , its last character is labeled the "E" (End) tag, while each remaining character is given the "M" (Middle) tag. Besides, a single-character word will be tagged as "S" (Single).

The whole thing is similar to POS tagging and information extraction in English, thus most statistical techniques originally proposed to tackle these problems can be applied to the CWS problem.

A Chinese sentence before segmentation: 她是荷兰人

The above sentence after segmentation: 她 是 荷兰人。

Corresponding English words: She is Dutch.

labeling with BC tag set: 她-B, 是-B, 荷-B, 兰-C, 人-C

labeling with BMES tag set: 她-S, 是-S, 荷-B, 兰-M, 人-E

Figure 5 A Chinese sentence and its segmentation indicated by white space between characters and labeled by two tag-sets

2.2.1 Hidden Markov Models

In sequence labeling, the baseline method is to predict each label independently. In this way, the problem is transformed into multiclass classification tasks, in which

each label is a separate class. Then we can apply any machine learning method for classification to deal with this. This method in fact achieves reasonable result in many tasks.

However, labels in many cases are not independent from one another. For example, it is not possible to have a tag sequence such as CCCCC for a Chinese character sequence, since there should be at least one B tag (meaning the beginning of a new word) before a sequence of consecutive C tags, which means the continuation of a word. Even in this simplest tag-set, labels are dependent on one another. It is thus understandable that the performance can be improved by including local sequence dependency.

The classic way of doing this is to introduce a hidden Markov model (HMM, Rabiner, 1989). As it is one of the most well known models in the field, we skip the mathematical details here and only summarize the main idea. There are two probability distributions in the model: an emission distribution (how likely the character 她 appears, given the current label is B) and a transition distribution (how likely an E tag follows a M tag). HMM assumes that the transition probabilities satisfy the Markov assumption, whose first-order case is that the probability of label at t+1 only relies on the label at time t and is independent of label at time t-1. Under this assumption, the Viterbi algorithm can be used to decode the best tag sequence.

There are “three classic problems” (Rabiner, 1989) for HMM, shown as following. Much of the popularity of HMM is due to the fact that there are simple solutions for training (Baum-Welch), decoding (Viterbi) and evaluation (backward-forward).

1. The Evaluation Problem: What is the probability that the emissions are generated by the model?

2. The Decoding Problem: What is the most likely state sequence in the model that produced the observation?

3. The Learning Problem: How should we adjust the model parameters in order to maximize the omission probability given the parameters.

2.2.2 Maximum Entropy Markov Models

The potential problem with using HMMs is that the emission probabilities are of the form P(character|label), where our ultimate goal is to model p(label | character). The latter is desired since many tasks would benefit from a richer representation that describes emissions in terms of many overlapping features, such as capitalization, word endings, part-of-speech in English.

Maximum Entropy Markov models (MEMMs, McCallum et al., 2000) address this problem. To allow for non-independent, difficult to enumerate observation features, it proposed a conditional model that represents the probability of reaching a state given an emissions and the previous state. These conditional probabilities are specified by exponential models based on arbitrary observation features. The exponential models follow from a maximum entropy argument, and are trained by generalized iterative scaling (GIS), which is similar in form and computational cost to the expectation-maximization (EM) algorithm. The “three classic problems” of HMMs can all be straightforwardly solved in this new model with new variants of the forward-backward, Viterbi and Baum-Welch algorithms. Compared with HMMs, MEMMs are only slightly more complex to train than HMMs, but have much better performances.

2.2.3 Conditional Random Fields (CRFs)

A problem of MEMMs is that when the models are trained, they are trained against correct previous labels. That is, when creating a classification example corresponding to the label at time t+1, features that depend on the label at time t are included. Even though these will always be correct at training time, they can be wrong at test time, which leads to the famous "label bias" problem. The Conditional Random Fields (CRFs, Lafferty et al., 2001) solved this problem in a principled way. While a MEMM uses per-state exponential models for the conditional probabilities of next states given the current state, a CRF has a single exponential model for the joint probability of the entire sequence of labels given the observation sequence. CRFs are reported to outperform MEMMs, MaxEnt and many other popular learning models in a number of natural language processing (NLP) applications (Rosenfeld et al., 2006). CRFs are first applied to CWS in by Peng et al. (2004), treating CWS as a binary

decision task to determine whether a Chinese character in the input is the beginning of a word. In addition, they use features from hand-prepared lexicons and bootstrapping, where new words are identified using the current model to augment the current lexicon, and the new lexicon is used to train a better model.

More formally, the probability assigned to a label sequence for an unsegmented sequence of characters by a CRF is given by the equation below:

$$P_\lambda(y|s) = \frac{1}{Z}\exp(\sum_{c\in C}\sum_{k}\lambda_k f_k(y_c, y_{c-1}, s, c))$$

In the formula, y is the label sequence for the sentence, s is the unsegmented characters, Z is a normalization term, f_k is a feature function and λ_k is the respective weight, C is the tag set and c indexes into characters in the sequence to be labeled. As mentioned in chapter 1, our work focus on generating useful n-gram based features to improve the performance of CRFs based CWS systems. In the experiments of this thesis, the CRF++ package (http://crfpp.sourceforge.net/) , a popular implementation of CRFs, is used.

2.3 Error Mining

As our proposed methods are inspired by the work by De Kok et al. (2009), which is a extension of previous methods in error mining, we present three important literatures in this topic here. The concept of error mining first appeared in the grammar engineering field. Though wide-coverage grammars can cover a large number of grammatical and lexical phenomena, they often fail to reach the same high accuracy for domain-specific texts as for general domain texts, because of missing lexicon entries, fixed expressions, and grammatical constructions. One type of parsing errors is that the parser can not find an analysis that accounts for the full sentence. This indicates that the grammar or lexicon is incomplete. As it is a tedious task to find incomplete descriptions for a wide coverage grammar by hand, error mining was proposed as a technique to automatically identify a problematic grammar or lexicon.

The problem of error mining for parsing results is summarized as following: After a corpus is parsed, it is split up into two sub-corpora, which consists of parsable and unparsable sentences, respectively. If some n-grams that only exist in the unparsable

corpus satisfy certain pre-defined criteria, they will be filtered out and be considered as the cause of the parsing errors. In other words, these n-grams are chosen as feature to represent the parsing errors. The methods used in error mining to identify useful n-grams, especially the n-gram expansion method inspired our proposed methods for selecting n-gram based features.

2.3.1 Van Noord (2004)

Van Noord (2004) defines an n-gram's suspicion of being the cause of a parsing error as a ratio:

$$S(w_i..w_j) = \frac{C(w_i..w_j|error)}{C(w_i..w_j)}$$

, where $C(w_i \ldots w_j)$ is the total number of occurrences of n-gram $w_i...w_j$ in sentences of both sub-corpora, and $C(w_i...w_j|error)$ is the number of occurrences of the n-gram in the unparsable sub-corpora. The longer n-gram $w_i...w_j$ is only considered if its suspicion is higher than each of its substrings.

The basis case for an n-gram is a word (unigram), but a unigram is not always sufficient to indicate the problem, that's why longer n-grams are considered. An illustrative example is that the Dutch parser Alpino can parse more than 90% of sentences in which unigram "via" occurs but fails to parse almost every sentence in which the bigram "via via" occurs ("via via" is a Dutch expression meaning "through a indirect way").

While this method works well with n-grams that only occur in the unparsable sub-corpus, it also assigns a high suspicion to n-grams that just accidentally appear in unparsable sub-corpora, which is not desirable.

2.3.2 Sagot and de la Clergerie (2006)

The error mining method proposed by Sagot and de la Clergerie (2006) solves the above problem and gradually shift the blame to specific n-grams in unparsable

sentences by taking the following into account:

* A form also occurs in parsable sentences is less likely to be the cause of an error.

* The suspicion rates of forms in the same sentence are interdependent and their sum is constant.

The method can be described mathematically by formulas:

$$S_f = \frac{1}{|O_f|} \sum_{o_{i,j} \in O_f} S_{i,j}$$

$$S_f^{(n+1)} = \frac{1}{|O_f|} \sum_{o_{i,j} \in O_f} S_{i,j}^{(n)}$$

In the formulas above, form *f* represents a surface-level item (unigram or bi-gram); sentence *s*i can be seen as a sequence of observations *o*i ,j of forms; each such observation has an *observation suspicion S*i ,j; and the bag of all observations of a form is denoted as Of . The *observation suspicion* is the suspicion of an n-gram within a given sentence. The suspicion of an n-gram outside the context of a sentence is then defined to be the average of all observation suspicions, which is called *mean suspicion rate*, denoted as Sf. As shown in the formula below, the observation suspicions themselves are dependent on the n-gram suspicions, making the method an iterative process.

$$S_{i,j}^{(n+1)} = error(s_i) \frac{S_{F(o_{i,j})}^{(n+1)}}{\sum_{1 \le j \le |S_i|} S_{F(o_{i,j})}^{(n+1)}}$$

To bootstrap the mining process, the suspicions of observed forms are first initialized by dividing suspicion equally across observed forms in a (unparsable) sentence. For each observation *oi ,j* in a sentence *si* we use:

$$S_{i,j}^{(0)} = \frac{error(s_i)}{|S_i|}$$

, where error (si) is 1 for unparsable sentences and 0 for parsable sentences.

As any iterative process, error mining resumes until convergence on a fix-point takes place. While Sagot and de la Clergerie seem to use 50 cycles, observed mean variation in suspicion rate: 0.01% as the stopping condition, De Kok et al use a different threshold in their iterative process: stop when the maximum suspicion delta is below a certain threshold (default: 0.001).

In a word, the key idea of this method is that the method is based on the recursive dependence between suspicion and observation suspicions, and it works with only unigrams and bigrams, since it encounters data sparseness problems when trying longer n-grams.

2.3.3 De Kok et al. (2009)

De Kok et al (2009) take sparseness into account, producing n-grams that are as long as necessary to identify problematic patterns, but not longer. This method combined the previous two methods and introduced an expansion function. The error mining is realized in a two-pass manner:

1. Candidate selection. Using a ratio-based preprocessor, similar to Van Noord (2004), to compute suspicion of n-grams and to expand unigrams (words) stepwise on each position of an unparsable sentence to n-grams long enough to indicate the problem. These expanded n-grams are called candidates.

2. Error Mining. Using the iterative process, similar to Sagot and de la Clergerie (2006) to mine the cause of parsing failures. The inputs of error mining are candidates selected in step 1.

The expansion from unigram is necessary since many errors are caused by combinations of words. But an inherent problem of a corpus is data sparseness: longer n-grams are likely to occur by only limited times, which may not be enough to reflect

their real suspicion. Usually, longer n-grams always appear to be more suspicious than it should be. To fix this distortion, the expansion from n-gram (i..j-1) to n-gram (i,…j) is allowed if and only if

$$S(i..j) > S(i..j-1) * extFactor \quad \text{and} \quad S(i..j) > S(i+1..j) * extFactor$$

The extFactor is a function whose value is determined by an empirical formula and always bigger than 1, which requires the expansion only occurs when the suspicious of the longer n-gram is higher enough than the shorter one.

The architecture of the expanders used in our system derives from the candidate selection part of De Kok et al. (2009), and the iterative mining is intergraded in our system as an optional function.

Chapter 3 The Framework of Feature Generation

3.1 Overview

From linguistic intuition, it may be desired to have longer n-grams as features to describe long distance dependencies and other rich linguistic phenomena for the Chinese word segmentation problem, as happened in error mining. A naïve idea is to include all the possible n-grams in a corpus. However, as the number of possible n-grams grows at an exponential rate on the length n-grams, it is simple infeasible or at least unaffordable for the CRF algorithm to compute such a huge feature set.

To illustrate its computational cost, a simple estimation is given as below. Suppose the number of (most frequent) unique Chinese characters (including symbols, digits and alphabets) equals to 5,000. Then the maximum possible number of character bi-grams is 5,000*5000, equals to 25,000,000, and that number of tri-grams is 625,000,000,000. Though the actual numbers for bi-grams and tri-grams may be significantly smaller than the maximum possible, still there are several million unique bi-grams and at least hundreds of million tri-grams in a typical 100 mega-byte corpus. CRF++, a popular C++ implementation of CRF, takes about 10 hours to learn a model for such a corpus with 2 million bigram features, on a modern server of 8 CPUs and 132GB memory. Since the training time for CRF is roughly a linear function of number of features, it will cost more than one month to train a model for the same corpus if all the tri-grams are treated as features (several hundred millions of features).

In fact, even if the computational cost is not a problem, from practical experiences, adding too many irrelevant features may damage the performance. Thus, from both efficiency and effectiveness point of view, we need a method to add useful long n-grams to the feature set without causing exponential growth of number of features.

This thesis proposes n-gram expansion based approaches that could select potentially useful n-grams from all the possible n-grams in a corpus, according to some pre-defined criteria. These approaches consider each position in every sentence of the corpora and try to expand the unigram of the Chinese character at the current position

to a longer n-gram starting from this position. Expansion decisions are based on global information, such as frequency, labeling history of each n-gram being considered. For every position in a sentence, there is one and only one expanded n-gram starting from it, whose length is determined by the expansion process. After these expanded n-grams are collected, there is an optional ranking procedure based on an iterative process that can further choose most promising ones from them. The n-gram expansion framework and iterative ranking algorithm will be described in detail in following sections.

3.2 N-gram Expansion

3.2.1 General expansion framework

As mentioned in the previous section, the aim of n-gram expansion is to select "useful" n-grams as features that convey information about the contexts or dependencies in the hope of improving the performance of the machine learning component of a CWS system. Thus we need both a standard to measure the usefulness of n-grams and a mechanism to select n-grams. From an empirical perspective, the usefulness of a feature (n-gram) can be judged only after comparing the performance difference of the machine learning algorithm with and without that feature. However, as this usually takes too much time, in practice we define criteria to estimate the usefulness of n-grams based on their own characteristics. At this moment, we ignore specific definitions of these criteria, and just assume that each n-gram has a value called information value (i-value), which indicates its usefulness.

Instead of computing the i-value for each possible n-gram and select those with i-value above a certain threshold, we adopt a dynamic expansion strategy (n-gram expansion) as in De Kok et al (2009), which processes the corpora sentence by sentence and iterates through every sentence by the step length of one position, starting from the first position. At each position, it expands unigrams to longer n-grams when there is evidence that it is sensible. Suppose the current sentence is:

c1 c2 c3 …ck

where *ci* is the Chinese character at the *i-th* position of the sentence ($1 \leq ci \leq k$). The n-gram expansion starts from *c1* and first tests whether the bi-gram *c1c2* is more useful

than both of its two corresponding unigrams, *c1* and *c2*, by computing and comparing I(*c1*), I(*c2*) and I*(c1c2*), where I(*x*) is the i-value of the n-gram *x*. If I(*c1c2*)>I(*c1*) and I(*c1c2*)>I(*c2*), it comes to the conclusion that the bigram *c1c2* conveys more useful information than c1 or c2 alone, thus so far the expanded n-gram starting from the first position is *c1 c2*.

In the same way, the expansion from bigram *c1c2* to the trigram *c1c2c3* will be allowed if and only if I (*c1c2c3*) is bigger than both I(*c1c2*) and I(c2c3). The general algorithm is that the expansion to an n-gram *ci..cj* is allowed when I(*ci..cj*) > I(*ci..cj-1*) and I(*ci..cj*) > I(*ci+1..cj*), where $1<i<j\leq k$. Once this condition no longer holds, the expansion for the current position *ci* stops and the expansion for the next position *ci+1* starts. After each position of the sentence is expanded in such a procedure, a sentence is represented by the n-grams *c1..cx*, *c2..cy*, ... , *ck-1...cz*. where *ci..cj* stands for the n-gram starts from the i-th position and ends at the jth position of the sentence ($1<i<j\leq k$).

The major motivation behind this n-gram expansion strategy is to avoid exponential growth of total number of n-grams. In the n-gram expansion, we need explicit evidence (higher i-value) that the longer n-gram is more useful or conveys more information than the current one, before the longer n-gram is reached. Since this is not always the case, many (in fact, the majority of) possible n-grams do not have a chance to appear in the expansion process, thus the size of selected n-grams is controllable. The real challenge left is to define a proper algorithm to compute i-value in such a way that those with higher i-value convey richer context information and are likely to be more useful features.

3.2.2 Data sparseness for expansion

In many methods for i-value computation, the i-value of an n-gram inherently increases with the growth of n-gram length, since longer n-grams occur less frequently and appear to be less ambiguous with respect to its corresponding label sequences. An exaggerated example is an n-gram that equals a whole sentence in the annotated corpus. It is seemingly useful as every character in this n-gram has an unambiguous label, but it is actually not so useful as such as long n-gram is not likely

to occur again in the testing corpus. The expansion conditions do not take this effect into account, allowing for unnecessary expansion of n-grams. To counter this problem, an expansion factor as in De Kok et al (2009) is adopted. This factor depends on the frequency of an n-gram and asymptotically approaches one for higher frequencies. As a result, the longer n-gram either needs to be relatively frequent to disprove data sparseness, or it needs a much more higher i-value than that of its (n-1)-grams. The expansion conditions are changed to:

$I(ci..cj) > I(ci..cj\text{-}1) \cdot extFactor$ and $I(ci..cj) > S(ci+1...cj) \cdot extFactor$

where $extFactor = 1 + exp(-A*|Occurrences\ of\ ci,...cj|)$, in which A = 1.0 proved to be a good setting.

3.3 Iterative Ranking

We can further rank these expanded n-grams via an iterative process and only keep those most promising ones. The iterative process is much the same as in Sagot & De la Clergerie (2006). In short, the idea is that n-grams independently selected in several different sentences are more promising candidate for features and the i-values of n-grams in the same sentence are interdependent and their sum is constant. Specially, for each selected n-gram, there are two i-values, namely *observation i-value* and *mean i-value*. While the suspicion of an n-gram within a given sentence is defined as the *observation i-value*, the suspicion of an n-gram outside the context of a sentence is defined to be the average of all observation i-values, i.e. *mean i-value*. The observation i-values themselves are dependent on mean i-values, making the method an iterative process. In other words, i-values of n-grams are treated in the same way as *suspicion rates* of n-grams in the error mining problem and the iterative process introduced in 2.3.2 are applied to choose most promising n-grams from all the generated n-grams to be features for CWS. We adopted the implementation of De Kok (2009) directly.

Chapter 4 Three N-gram Expansion Methods

The n-gram expansion procedure described in chapter 3 is a quite general framework that can have several implementations to compute the i-value differently, while keeping the same or similar expansion strategy. This section will introduce three specific expanders, based on Chinese character mutual-information, Chinese character label entropy and label distribution bin of Chinese character.

4.1 Mutual Information based Expander

Mutual information (Church and Hanks, 1990) is a widely used concept in information theory to measure the mutual dependence of two variables. Mathematically, the mutual information of two discrete random variables are defined by following formula:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p_1(x)\, p_2(y)} \right),$$

For a bigram of two Chinese characters *xy*, we consider Chinese characters as random variables and define their mutual information as:

$$I(x;y) = \log (p(x,y)/(p(x)*p(y))).$$

For a n-gram of multi Chinese character *c1 c2…ct*, its mutual information can be defined as multi-variable mutual information of these individual characters *c1, c2,…, ct*. However, multi-variable mutual information is quite complex to compute and such a rigorous definition may be unnecessary for our purpose. In fact, we only need a well-defined variable, whose value represents the dependency or correlation of current n-gram of characters and the immediately next character, i.e. whether the immediately next character is part of the current grouping or belongs to the next grouping. These groupings are not necessarily words themselves, but they are expected to provide helpful information for the machine learning algorithms that identify the real boundaries of words.

To make the decision of whether to incorporate the following character into the current n-gram, it is adequate to simply compute the point-wise mutual information of this n-gram and the character immediately next to it. It can be expressed in the following formula:

$$I'(c1,c2,c3,....ct)=\log(p(c1,c2,...ct)/p(c1,c2,...ct-1)*p(ct))$$

This makes sense in our problem scenario, as our goal is to make the expansion decision. Now suppose we have already expanded the n-gram to *c1,...ct-1*, and the next character is ct. Then what we would like to know is how likely the longer n-gram *c1,....ct-1,ct* occurs as a whole in the context. The longer n-gram is likely to be a coherent whole, in the case that the frequency of *c1,...ct-1, ct* is much higher than what it would be if *c1,...ct-1* and *ct* were just combined together by chance. The ratio of these two frequencies is the right hand side of the formula without the log operation.

Please note since it does not make a sense to compare mutual information value of longer n-grams *c1,...ct* with that of shorter ones as suggested by the general expansion framework, we instead compare it with a threshold (denoted as i-parameter) to "guess" whether the occurrence of *c1,...ct-1,ct* is caused by some linguistic constrains or purely by chance. The i-parameter can have a single empirical constant value for all the n-grams, or be computed according to the length of current n-gram or the average frequency of n+1 gram that contains the current n-gram as the affix. Experiments show that a constant threshold is reasonable good for our requirement.

Apparently, since a constant threshold does not take specific characters of individual n-grams into account, it causes both type I error (false positive, expand those are not supposed to be expanded) and type II error (false negative, fail to expand those should be expanded) of the expander, where there is always a tradeoff between the two. As the CRFs can cope with noisy features to some extent and the aim of feature generation is to provide more potential useful feature, we are more tolerant of type I errors than the type II error, thus we set a relatively low value for the i-parameter.

4.2 Label Entropy based Expander

Entropy was originally a thermodynamic concept to measure how disorganized a system is. Shannon introduced it to information theory as a measure of the uncertainty associated with a random variable. We adopt this term to describe the certainty of an n-gram's corresponding label sequences in the annotated data.

As the CWS problem is treated as a sequence-labeling problem in this study, the central task is to predict the correct label sequences of Chinese character sequences in unseen corpora (testing data). The greatest challenge is that the same Chinese character may have different labels in different contexts. This ambiguity is comparable to the situation in POS tagging, where one word can have multiple part-of-speech tags, which could only be disambiguated when the context is given.

Just like individual characters, n-grams of characters also have ambiguity, which we measured by label entropy. The label entropy of a given n-gram of characters is computed by visiting all the occurrences of this n-gram in an annotated corpus. During the visit, we keep a record of all the unique label sequences (n-grams of labels) that correspond to the current n-gram being considered, with times of occurrence of each unique label sequence throughout the annotated corpus. The label entropy is computed by dividing the maximum value of all the numbers of occurrence of unique label sequences by the their sum. More formally, the label entropy can be defined as follows:

An n-gram of characters, w, occurs n times in the corpus. It is labeled as $t1, t2, \ldots tn$, respectively in the annotated corpus, where ti is the label sequence (n-gram of labels) corresponding to the i-th occurrence of w in the annotated corpus ($1 \leq i \leq n$). Suppose there are m unique label sequence (where $m \leq n$) among $t1, t2, \ldots tn$, which are denoted as $u1, u2, \ldots um$, respectively, and their frequency are denoted as $d1, d2, \ldots dm$, respectively, where $d1+d2+\ldots dm=n$. The maximum number of $d1, \ldots dm$ is denoted as d_max. Then the label entropy LE of the n-gram of characters w is:

$$LE(w)=d\_max/n$$

In the implementation, a hash table H is built incrementally to record unique n-gram of labels (key) and their frequency (value).

For each n-gram of characters, the following procedure (Figure 6) is conducted:

```
for (i=0;i<n;i++)
{    visit i-th occurrence of current n-gram of characters;
     k=its corresponding label sequence in the annotated corpus;
     if k is found (searched) in H
         H[k]++;   // H[k] stands for the value for the key k.
     else
         Insert a new key-value pair (k, 1) in H.
}
```

Figure 6 Algorithm for label entropy computation

The expansion procedure is much the same as in the general expansion framework; the only difference is that the decision of whether to expand to longer n-gram is based on the comparison of label entropy values of the longer n-gram we would like to expand to with that of its two shorter n-grams.

Suppose the current sentence being processed is:

$$c1\ c2\ c3 \ldots cn$$

and we have already expand the n-gram of characters starting from i-th position to

$ci, ci+1, \ldots cj$, where $i<j<n$. We need to decide whether we should further expand this n-gram to

$$ci, ci+1, \ldots, cj, cj+1$$

To do so, we compute LE($ci, \ldots cj+1$), LE($ci, \ldots cj$) and LE ($ci+1$, $cj+1$) and test whether the following condition holds:

LE($ci..cj+1$) > LE($ci..j$) · *extFactor* and LE($ci..cj+1$) > LE($ci+1..cj+1$) · *extFactor*

,where *extFactor* is the factor to deal with data sparseness problem, which is defined as $1+exp(-A*|Occurrences\ of\ ci, \ldots cj+1|)$. We can set further constrain such as the LE value of the finally expanded n-grams should be greater than 0.9 (or equals to 1) to keep only those most promising ones.

As all these expanded n-grams in most cases have only one unique label sequence, it is quite confident to assign a specific label (the one as in that unique label sequence)

to a Chinese character occurs in such a context (n-gram). This provides a very useful feature for the machine learning method to build its model.

4.3 Label distribution bin based expander

4.3.1 Data sparseness in CWS

In this part, we further examine the data sparseness problem in the context of CWS. In general, data sparseness refers to the problem of insufficient quantities of data to estimate parameters that grow exponentially. Let's take the n-gram model as an example. For a language whose vocabulary size is v, if we would like to estimate the frequency distribution of all the n-grams of length m, we have to estimate v^m parameters, since there are v^m possible combinations for basic words to form an n-gram of length m. Since language distributions obey Zifp's law, the corpus size needed to estimate these parameters is typically many times of v^m. In fact, even a corpus as big as English Wikipedia seems to be inadequate to estimate the distribution of frequencies of five-grams.

In CWS, the size of annotated corpora usually ranges from several millions bytes to tens of million bytes, which is too small to estimate longer n-grams of words other than unigram and bi-grams. After having estimated parameters for tri-grams and four-grams from the training data, it is likely that many of these long n-grams never occur in the testing data and many long n-grams occur in testing data are unseen in the training data, i.e. the overlap of long n-grams between training data and testing data is little.

In other words, many longer n-grams generated from the training data do not appear in the testing data, thus are not useful at all. This is a quite frustrating conclusion. As it is not easy to build a significantly bigger corpus, the only way to cope with data sparseness is to decrease the vocabulary size. It is not realistic to decrease the actually size of Chinese characters, so that we define a mapping from Chinese characters to some abstract symbols in such a way that each character is mapped to one and only one abstract symbol and each abstract symbol can have multiple corresponding

characters. Apparently, it makes the vocabulary size of abstract symbol smaller than the original vocabulary size of characters.

To simplify the reasoning, we suppose the ratio of vocabulary size of n-grams and the corpus size required to estimate these n-grams is a constant, 1/k (k is a integer bigger than 1). Assuming the vocabulary size of Chinese character and abstract symbols are v and u, respectively, the required size of the corpus to estimate n-grams of abstract symbols of length m is then $k*u^m/k*v^m = u^m/v^m$ times as that of the corpus to estimate original n-grams of Chinese characters of length m. A more concrete example is as following: If the vocabulary size of abstract symbols is 10 times smaller than that of concrete Chinese characters, the data sparseness problem of estimating trigrams of abstract symbols is 1000 times less severe than that of estimating trigram of characters.

It is apparent that it would make the data less sparse if we estimate parameters for n-grams of abstract symbols instead of n-grams of characters. The challenge is how to define a proper character-symbol mapping, which satisfies the condition that for all the characters mapped to the same abstract symbol they are very similar to each other and are very dissimilar to characters that are mapped to other abstract symbols, according to a certain standard of similarity, i.e. the mapping is equivalent to a good clustering of the characters.

4.3.2 Type of Chinese characters

While the natural cluster for English words are based on their part-of-speech tags, a suitable partition of Chinese characters is according to their types of characters, which simply map characters to their types. Just like the disambiguation problem in English POS tagging, there is also a similar problem of assigning the most suitable type to each character that has multiple types in a given context. In this study, we don't take this into consideration, and assign each character a symbol that stands for the set of all its possible character types. The Chinese character type information is obtained from the Peking University Chinese Treasury (http://icl.pku.edu.cn/).

A potential problem of using character type is that there are around 20 types in total, which lead to only hundreds of bigrams and thousands of trigrams. Thus n-grams of character types may not provide adequate information of the context for CWS. To solve this problem, we in practice adopt the hybrid n-grams of characters and types. For example, an trigram feature may look like *c1,t2, c3*. It represents a character sequence in which the first and last character is *c1* and *c3*, respectively and the character in the middle is any character whose type is *t2*. Please refer to the experiment part to explore how these hybrid n-grams are used in actual feature generation.

To further illustrate the benefit of introducing character types into n-gram features, the following example is given. Suppose a trigram feature *c1, c2, c3* only appear in the testing corpus but not in the training corpus, in which there are only trigram such as *c1, c2', c3*. Then there is no matching between these two tri-grams. However, if we also consider character types, we may find that in fact *c2* and *c2'* are of the same Chinese character type ***t2***, and a hybrid tri-gram feature of *c1, t2, c3* can describe both. Since only features occur in both training and testing corpus are meaningful for a machine learning algorithm, those hybrid n-grams contribute more potentially useful features to the CWS, as they link features pairs in training and testing corpus which would never be matched otherwise.

4.3.3 Label distribution bin

Besides Chinese character types, we also propose a new set of abstract symbols, called label distribution bins. As mentioned earlier, every mapping scheme should guarantee that those characters mapped to the same symbol act similarly in the context. No doubt, character types, which are based on characters' syntactical roles, satisfy this condition. Considering CWS as a sequence-labeling problem, the most important characteristic of a character is how it is labeled in annotated corpora, i.e. the distribution of all its possible labels throughout the annotated corpora. The newly proposed mapping actually maps characters to label distribution bins, each of which represents the distributions of the corresponding labels of a certain character in the annotated corpora.

In practice, there are various label sets to define the positions of characters in words differently. We adopt the simplest label set, the binary label set in this demonstration. In the binary label set, a character is either tagged as "B", meaning the beginning of a new word or "C", meaning the continuation (middle part or the end) of current word. The distribution of labels for a certain character is represented as a value of *p_B*, which is the possibility of this character being labeled as B (as the possibility of this character being labeled as "C" equals to 1-*p_B* in any case, *p_B* alone is adequate to represent the label distribution). For example, character *x1* which only appears as the head in a word will have the *p_B* value of 1, and character *x2* which appears as the head of the word 60% of the time, and as the continuation of a word 40% of the time will have the *p_B* value of 0.6. We may assign abstract symbols to characters according to their *p_B* values, but since many characters have their unique *p_B* values, the vocabulary size of abstract symbol would be too big in this case. Thus, we map *p_B* values in each interval of the value range into different bins–a discretization process. As shown in Figure 7, we may map 0, (0, 0.1], (0.1,0.2], … (0.9,1.0), 1.0 into bin1, bin2, …bin12, respectively, where (x,y] and (x,y) are semi-closed and open range of *p_B* values. Please note that those characters have *p_B* value of 0 or 1 have unambiguous labeling in the corpora, so they themselves are assigned to two separate bins, respectively.

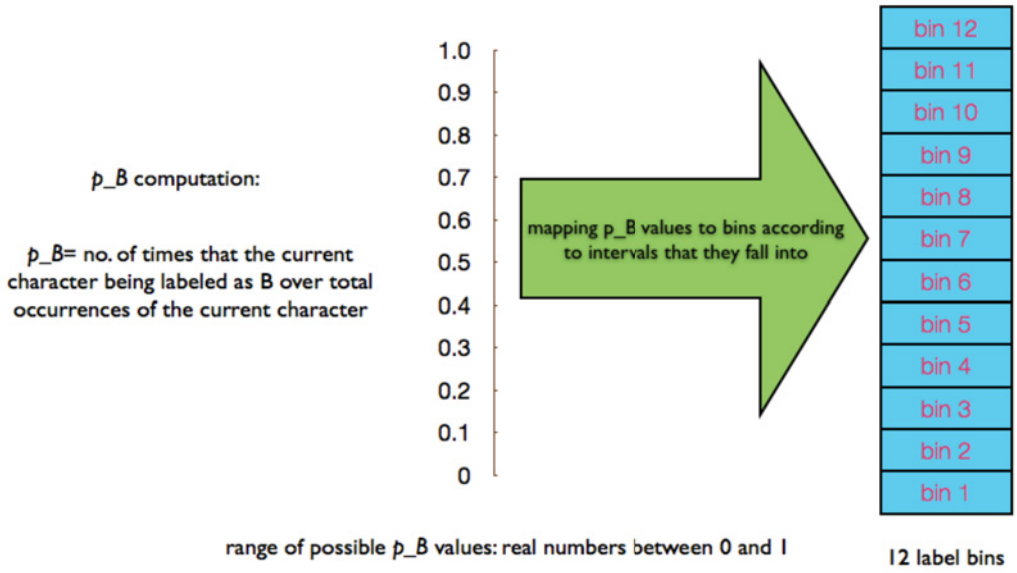

Figure 7 Discretization: mapping *p_B* values to label bins

The character-*p_B* mapping and *p_B*-bin mapping imply that each character has a unique corresponding bin, which is called label distribution bin. The nice thing of this composed mapping is that we can modify the length of the intervals, i.e. the scaling of discretization to control the vocabulary size of abstract symbols. Besides fixed intervals of value range as in this example, we can also make it dynamic to make sure there are roughly same numbers of characters in each bin. Although this is only an example based on binary label set, the same logic applies to cases with more complicated label set of six or more tags.

It is quite straightforward to compute the distribution of labels for each character. Suppose there are six possible labels, each of each corresponds to an integer between 1 and 6. Then for each unique character, the following procedure is conducted (Figure 8):

```
// n is the total number of occurrence of current character

for (i=0;i<n;i++)
{
     visit ith occurrence of current character;
     k=its corresponding label in the annotated corpus;
     if (k==1)
       L[1]++;
     else if (k==2)
       L[2]++;
     else if (k==3)
       L[3]++;
     else if (k==4)
       L[4]++;
     else if (k==5)
       L[5]++;
     else
       null; // we don't consider the case k=6, as it is dependent on
the rest five;
}

for (t=1;t<n; t++)
    P[t]=L[t]/n;   //compute the raw label probability for current
character

interval_num=10;  //10,the interval for each label probability value

//discretization
for (t=1;t<6; t++)
{
    if (P[t]=0);
      D[t]=0;
     else if (P[t]=1);
      D[t]=11;
```

```
      else
          D[t]=floor (P[t]*interval_num) +1
         //floor (x) above return the largest integer smaller than x

      }

bin=0;
for (t=1;t<6; t++)
     bin=bin*(interval_num+1)+D[t];
     return  bin;  //bin is a integer that represents the label
distribution bin of current character
```

Figure 8 Algorithm for label distribution bin calculation

4.3.3 Expander V.S. feature template

We can utilize label distribution bin in an expander. N-grams of characters are expanded in the same way as in mutual information based expansion and the only difference is that the mutual information used is that of n-grams of label distribution bins, rather than of n-grams of characters.

Another way of using label distribution bins is to incorporate all the hybrid bi-grams of label distribution bins and characters for each bi-gram of characters in a template way. In other words, if the original bi-gram *c1c2* is considered as a feature, *c1b2*, *b1c2* and *b1b2* are also considered as features, where *b1* and *b2* are label distribution bins of *c1* and *c1*, respectively.

As there are only tens of Chinese character types, in a template way of using character types, we also add hybrid tri-grams with one position as character type, as well as hybrid bi-grams. If bi-gram *c1c2* is a feature,tri-grams *c1c2t3*, *t1c2c3*, *c1t2c3*, as long as *c1t2* and *t1c2* are considered as features, where *t1*,*t2* and *t3* are the character types of *c1*, *c2* and the character after *c2* (i.e. *t3*), respectively.

Chapter 5 Implementation

In this chapter, we will discuss some important details of the system implementation. Based on the methods proposed in the previous chapter, our system, N-generator is implemented in C++. We will start with software or libraries used in our system; then focus on the overall architecture and algorithm/data structure details of important components.

5.1 Used Tools

Error Mining in Parsing Results Project

The error mining in parsing results project (http://github.com/danieldk/errormining/) implements the error mining system as described in De Kok et al. (2009). We utilize some of its basic data structures such as form, sentence to keep the features, and modified the expander to serve as the framework for the feature generation. Besides, the suffix array implementation from this software is integrated in our system to compute n-gram frequencies.

UTF8-CPP

UTF8-CPP (http://utfcpp.sourceforge.net/) is a library for handling UTF-8 encoded Unicode strings. It is written in STL style, thus is easy to use. The utf8.h headfile is called in our system to decode UTF-8 bytes.

CRF ++

CRF++ (http://ctfpp.sourceforge.net/) is a simple, customizable, and open source implementation of Conditional Random Fields (CRFs) for segmenting/labeling sequential data. CRF++ is designed for generic purpose and will be applied to a

variety of NLP tasks, such as Named Entity Recognition, Information Extraction and Text Chunking.

Both the training file and the test file need to be in a particular format for CRF++ to work properly. Generally speaking, training and test file must consist of multiple tokens. In addition, a token consists of multiple (but fixed-numbers) columns. The definition of tokens depends on tasks and each token must be represented in one line, with the columns separated by white space (spaces or tab characters). A sequence of token becomes a sentence. To identify the boundary between sentences, an empty line is put. The last column represents a true answer tag, which is going to be trained by CRF. CRF++ is used as the experiment platform to test the usefulness of the generated features.

5.2 System Description

The N-generator implements the feature generation approaches as described in previous chapters, which take training and testing corpora for Chinese word segmentation task as inputs and automatically generates features for CRF learning by processing the corpora. The system can be viewed as a pipeline of three major parts, namely pre-processor, feature selector and feature writer, as shown in Figure 9.

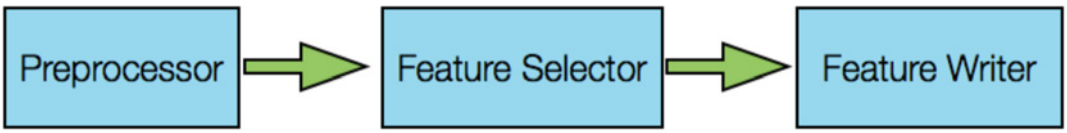

Figure 9 The architecture of N-generator system

The preprocessor reads the corpora and builds up useful classes/data structures for the feature selector. The feature selector, which is the core part of the system, utilizes these data structures to select features and finally the feature writer generates features for each character in the corpora according to feature definition and input format of CRF learning algorithm.

In a unified framework, this system implements three major feature generation (n-gram expansion) methods proposed in previous chapters, which are based on mutual

information, label entropy and label distribution bin, respectively. All of these methods share the above system architecture, all the codes in preprocessor and feature writer, and some of codes in feature selector. The component, which we call "expander" in the feature selector part, is the only place where their differences are reflected. In actual implementation, there is a unique expander for each method.

The feature selector selects n-grams of characters as the potential features for CRF learning. Those n-grams are selected by considering the global information of each character occurs in each sentence of the corpora, such as frequency, label entropy, and label distribution. Since the selection is realized by iteratively process characters in sentences and expand the n-gram starting from each character to longest possible, the core module is called "expander".

While the feature selector is the central part of the system, the preprocessor and the feature generator solves practical problems. One important objective of the preprocessor is to process Chinese characters in utf8 encoding, in which the number of bytes required to store one character is not fixed but variant from 2 to 3. Another function of the preprocessor is to convert original corpora into vector of integers in order to improve computational efficiency. Moreover, the preprocessor builds up important data structures such as suffix array and index table for the fast computation of frequency of character based n-grams in the feature selector part.

The features selected by feature selector are actually records of n-grams of characters and the places of their occurrences in the corpora. The feature writer defines features for each occurrence of character in the form of "whether this character occurs as the *i*-th position of *x*-th n-gram in the corpus" and write features and other formatting information into the resulting corpus as the input for CRFs learning algorithm.

5.2.1 Character encoding and integer vector building

1) Chinese characters encoded in UTF-8

As mentioned in previous section, one function of the preprocessor is to process Chinese characters encoded in UTF-8, which is the most popular encoding of Chinese characters in the Internet era. As there are thousands of unique characters in Chinese, it is not possible to represent them with an encoding system whose code length is just

1 byte, i.e. 2^8 possibilities, just like ASCII. UTF-8 (8-bit Unicode Transformation Format) is a variable-length character encoding for Unicode, which is also backwards-compatible with ASCII, i.e. it uses just one byte to represent ASCII characters. For this reason, it is the dominant character encoding for files, emails and web pages. UTF-8 encodes each Unicode character as a variable number of 1 to 4 bytes, where the number of bytes depends on the integer value assigned to the Unicode character.

While most functions in high-level programming languages such as C++ operates only on char/byte level, CWS algorithms process characters, which consist of variable number of bytes as in UTF-8. Thus, it is necessary to decode UTF-8 codes back to Chinese characters, i.e. to group byte sequences into sequences of units, each of which consists of 2-3 bytes and represents a Chinese character. This is realized by testing byte sequences with UTF-8 encoding rules. In our implementation, we utilize UTF8-CPP (refer to 5.1.3) to translate bytes in UTF-8 format to Unicode points, each of which is an integer represents one Chinese character.

2) Compression to improve space efficiency

In NLP systems for European languages, a common practice for dealing with large corpora is to represent words in a more compact way to decrease space cost and improve compute efficiency. One way of doing this is to build a perfect hash table to map each word to an integer, since a word usually takes several bytes in storage while an integer can take only two to four bytes.

Similarly, we also build a perfect hash to map Chinese characters to short integers and represent the corpora as vectors of integers. And this compression is processed together with UTF-8 decoding. The raw corpora in the form of byte sequences are fed to the pre-processer, which finds out boundaries of characters in the byte sequences and maps each unique character to a unique integer. A perfect hash table is built and kept during this process to store this mapping relationship. A more important reason for this is that the implementation of suffix array, a data structure and a sorting algorithm used in feature selector, requires that its inputs to be vectors of integers in the range of 1 to k, and each number in the range occurs at least once. After pre-processing, the raw corpora file is converted into a vector of integers for further processing.

3) Labeling corpora according to a tag set.

An extra task of pre-processing is to label the training corpora in accordance to the meaning of tags in the chosen tag-set. While word boundaries in training corpora are represented as a white space between two characters, machine learning methods such as CRF requires that their input are represented as symbols with tags. In CWS, the tags for characters represent the position of one character in a word. To illustrate the difference, the following example is given. A sentence in an un-annotated corpus may look like this the on in Figure 10:

马耳他是一个美丽的国家。

Figure 10 A Chinese sentence without word segmentation

After (manual) annotation, this sentence becomes the one shown in Figure 11:

Chinese: 马耳他 是 一个 美丽的 国家 。

English Translation: *Malta is a beautiful country* .

Figure 11 A Chinese sentence after word segmentation

Since the first three characters in the sentence, “马”, “耳” and “他” forms one word meaning “Malta” and the fourth character “是 “ alone forms one word meaning “is”, the annotation process is just to insert one while space between “他” and “是 “to indicate word boundary. The rest characters are annotated in the same manner.

A CRF system, however, requires an input in the form as following (Figure 12):

Figure 12 CRF input format for CWS problem

Each row of the input is made up of one character and its tag. The boundaries of words are translated into the sequence of tags. The pre-processor builds a vector of tags while processing the annotated corpus. The fragment of the tag vector corresponding to above example is therefore <B, B_2, E, S, B, E, B, B_2, E, B, E,S>.

4) Key class of the preprocessor: UTF8-Reader

HanCharTable Class

HanCharTable keeps 2 perfect hash tables. One is called Charater2hash, which hashs Chinese characters to integers from 1 to k, where k is the number of unique characters in corpora. The other is called hash2Unicode, which maps hash codes in the previous hash table to the Unicode integers of their corresponding characters. The class support ***NewIterm*** functions to record character-Unicode pairs.

UTF8-Reader Class

UTF8-Reader processes Chinese characters encoded in utf-8 and also tags the corpora according the labeling rule of the chosen tag set. It operates two data objects: one is a HanCharTable, which keep the mapping between Chinese characters and hash codes; the other one is CodedCorpus, which is a vector of integers to store corresponding hash codes of Chinese characters in the order that the characters appear in the corpora.

Initially, the HanCharTable is empty and then the corpora are fed to the UTF8-Reader, which process the corpora line by line. When a line is being processed, the UTF8-Reader calls methods in external library UTF-CPP to extract one character per time from the byte sequence of the current line. The NewItem function of HanCharTable deals with the character and its Unicode integer. If that character is already in the record of HanCharTable, the NewIterm function does nothing. Otherwise, it keeps the record for that character Unicode pair. In either case, the HanCharTable returns the hash code of that character and it is pushed into CodedCorpus, the integer vector that represents the corpora. The simplified view of the UTF8-Reader class is as in Figure 13:

```
class utf8reader
{
public:
    utf8reader(string Corpus, HanCharTable);
    ...

private:
     HanCharTable;
     vector <int>CodedCorpus;

};
```

Figure 13 The simplified view of UTF8-Reader Class

Here is the pseudo code for the constructor of UTF8-Reader (Figure 14), which is also the procedure it processes the corpora as described in previous paragraph:

```
utf8reader::utf8reader(string raw_corpus, HanCharTable newtable)
{

     string code2utf8(uint32_t c);
    //the function to convert Unicode points to utf8 encoded
     characters

while (the current line is not empty)
     {
        string line=byte sequences of current line;
        check whether the current line is in valid utf-8 encoding;
        string::iterator iter=line.begin();
          while (iter!=the last byte of the current line)
        {
           current_unicode=utf8::next(iter,end_it);
          //Using utf8-CPP class to process the byte sequences,
           extract bytes that represent next character convert it to
           45nicode

             current_character=code2utf8(current_unicode);
           //convert the 45nicode code point back to utf-8 string;
           //Process the (utf8-string, unicodepoint) pair as below
           If (the current character is not yet in the HanCharTable)
                Build a record in to keep that character with its
               Unicode integer
           else
                    do nothing;

           CodedCorpus.push_back(HanCharTable->operator
                                        ()(current_character));
           // push the hashcode of current character into the vector
           that represents the corpus

             }

       }
}
```

Figure 14 Algorithm of UTF8-Reader Constructor

The actual constructor of UTF8-Reader class is much more complicated than the above algorithm. Besides processing the corpora to build up its integer vector representation and maintain mapping records in HanCharTable, it also labels the annotated corpora using the scheme of the chosen tag set. The tag sequence is stored in a vector of string called CorpusTags, which has the same length as CodedCorpus. The tag stored in the i-th position of CorpusTags is the corresponding tag of the Chinese character in the i-th position of CodedCorpus, where 0<I <size of the corpus.

5.2.2 Suffix Array and index table building

1) Suffix Array

A suffix array is an array that contains indices pointing to sequences in the data array, that are ordered by suffix. For example, if the corpus is the string acba, the suffix array is {a, acba, ba, cba}. To simplify the notation, people use integers I to denote the suffix starting at position I in the corpus. In this way, the suffix array is {3, 0, 2, 1}.

The suffix array of a sequence can be used as an index to quickly locate every occurrence of sub-sequence within the sequence. Finding every occurrence of the sub-sequence is equivalent to finding every suffix that begins with the sub-sequence. This becomes easy if the lexicographical ordering is conducted so that these suffixes will be group together in the suffix array and can be found efficient with a binary search.

The preprocessor also builds up suffix arrays that will be used in the feature selector to compute the frequencies of n-grams of characters, which provides important information for the feature selection process. The frequency of any n-gram can be computed by looking up its upper and lower bounds in the suffix array, where the difference is the frequency.

The suffix array implementation in the Error Mining in parsing result project (http://github.com/danieldk/errormining/) has been integrated into our system.

2) Index table

As the feature selection procedure is dynamically processing while scanning the corpora and the number of intermediate candidates is very large, it is inefficient to keep the location information of all these candidates, which are in the form of n-grams. On the contrary, we look up each chosen n-gram's occurrences in the corpora only in the feature writer phase. While suffix arrays provide a compact and relatively fast data structure for looking up n-gram frequencies, they are suitable in cases where hybrid n-grams exist, which is the case in our problem. Hybrid n-grams are those n-grams that consist of different types of information (Figure 15). For example in English, we can define hybrid n-grams of words and POS tags, such as the bigram "ADJ_country", meaning an adjective appears right ahead of the word *country*. This

bigram generalize many bi-grams, such as *good country*, *European country, democratic country*, etc.

Word layer: *Malta is a beautiful country*.

POS tag layer: Noun Be ART ADJ NOUN

Figure 15 Two layers of information for an English sentence

In the CWS problem, the features are not only n-grams of characters, but also be hybrid n-grams of character and character types (Figure 16), similar to hybrid n-grams of words and POS tags in English.

Chinese character layer: 一个美丽的国家

Character type layer: <数><量><形><形><的><名><名>

Figure 16 Two layers of information for a Chinese phrase

In the above example, <数>,<量>,<形> and <名> are character types. A typical hybrid n-gram can be <*形*><*形*>*的*, which is a tri-gram formed by the character "的" after a double occurrence of the character type <形>. It is a generalization of many concrete tri-gram of characters such as *善良的*, *肮脏的*, etc, in which the character types for the characters 善, 良, 肮 and 脏 are <形>. The reason for introducing hybrid n-grams is that we would like to have features as general as possible so that we can use less features to cover more character combinations.

Since we need to look up frequencies of every possible combination of representations that are used, we would have to create d^l suffix arrays to be able to look up hybrid n-gram occurrences with the same time complexity, where d is the number of dimensions and l is the corpus length.

To solve this problem, we developed a data structure called **indextable**, which is for fast looking up the occurrences of hybrid n-grams in the corpus. First, we build a hash table, which we call index table, for each type of information that can be used in n-grams. A hash table contains an instance of such information as a key (e.g. a specific character or character type) and a set of corpus indices where the instance occurred in the corpus as the value associated with that key. Now we can look up the occurrence

of a sub-sequence i..j by calculating the set intersection of the indices of j and the indices found for the sequence i..j -1, after incrementing the indices of i..j -1 by one.

The complexity of calculating frequencies following this method is linear, since the set of indices for a given instance can be retrieved with a O(1) time complexity, while both incrementing the set indices and set intersection can be performed in O(n) time.

The following example illustrates how index table works. The sample corpus and its index are shown in Figure 17 and its indextable is show in Figure 18. Suppose we have known the occurrences of the an n-gram *i…j* (here we take "*your boss*" as the example) in this corpus, then these occurrences can be represented as the a set of indices of last position *j* ({1,8} for "*your boss*") , denoted as *S.* Given *S* ({1,8}), which represents all occurrence of n-gram *i..j* ("*your boss*"), all occurrence of n+1-gram *i..j+1*("*your boss needs*") can by calculated by the set intersection of the indices of *j+1* ({2,6}) and the set *S*, after incrementing the indices of *S* by one ({2,9}). Finally we have {2} to represent all occurrence of "your boss needs". In this way, we can iteratively find out all the occurrences of any n-gram. The implementation of this algorithm and its data structure is straightforward.

Corpus	Your	boss	needs	you	and	you	need	your	boss	.
Corpus Index	0	1	2	3	4	5	6	7	8	9

Figure 17 A sample corpus with index

Index Table

key	value
and	{4}
boss	{1,8}
you	{3,5}
your	{0,7}
need	{2,6}
.	{9}

Figure 18 The index table for the above corpus

5.2.3 Feature selector

The core of the feature selector is a n-gram expander and a miner. As explained in Chapter 3, the expander iterates through a sentence of unigrams, and expands unigrams to longer n-grams when there is evidence that it is useful. The judgment condition is that the expansion to an n-gram *i..j* is allowed when

ngramRatio(*i..j*) > ngramRatio(*i..j*−*1*) and ngramRatio(*i..j*) > ngramRatio(*I* + *1..j*).

This gives us a sentence that is represented by the n-grams *n0..nx*, *n1..ny*, ... *n|si|−1..n|si|−1*, which is stored in a data structure also called **sentence**. **ngramRatio** is a pre-defined function, whose value is determined by the n-gram *i...j*. In the implementation, the i-value is computed by the function called **ngramRatio**. The pseudo code of the original expander is as following (Figure 19).

```
Sentence Expander(vector<int> HashedToekens)
{

   //the ngramRatio is the function that computes the ratio, or i-value of
   //an n-gram

   for (iter is the iterator of HashedTokens;it has not reached the end of
         the sentence; iter++)
     {
      // Initially, the best n-gram is the shortest one we start with.
      // 'Best n-gram' is defined as the n-gram with the highest i-value.

       bestNgram = bigram iter, iter+1;

      //compute the ratio or the i-value of bestNgram
      double bestNgramRatio = ngramRatio(bestNgram);



          for (endIter = iter + 1 + 1;
                  endIter <= HashedToekens.end(); ++endIter)
           {


           // Get the n+1-gram, which we will call an m-gram.
           mgram(iter, endIter);

           // Compute the ratio/i-value of mgram
           double mgramRatio = ngramRatio(mgram);

           // Get the second n-gram within the current m-gram.
           ngram(iter + 1, endIter);

            // Calculate the expansion factor, if it is sensible
          factor = expansionFactor(mgram.first, mgram.second);
```

```
// If the m-gram has higher ratio (i-value) we'll extend the n-gram.
        // Otherwise, stop the expansion process
              if (mgramRatio > factor * bestNgramRatio
                    && mgramRatio > factor * ngramRatio(ngram)
                          {
                                bestNgram = mgram;
                                bestNgramRatio = mgramRatio;
                          }
              else
                                break;
                    }

              Insert bestNgram into the Sentence vector

      }

      return sentence;

}
```

Figure 19 The architecture of the expander

Our implementation of the expander shares the architecture with the expander in the error mining project. The differences are: 1) In our implementation, there are three different ngramRatio functions that implement three of the proposed methods for computing information value (i-value), respectively. 2) We modified the codes to better process Chinese characters.

As the other parts of the implementation are quite straightforward and more engineering based, we skip the in-detailed introduction.

Chapter 6 Experiments and Results

6.1 Evaluation Method

Since the function the proposed approaches is to generate features for CRF based CWS systems, the simplest way to evaluate the usefulness of generated features for CRF learning is through comparing the performance of a CRF based CWS system with and without the generated features. To measure CWS system performances, we adopt both data set and evaluation criteria from SIGHAN Bakeoff 2006 (the third Bakeoff).

6.2 The SIGHAN Bakeoff Data and Criteria

To encourage and to promote 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. Since the first Bakeoff in 2003, there have been five Bakeoffs in total. The third Bakeoff was held in 2006, whose results were presented at the fifth SIGHAN Workshop at ACL 2006 in Sydney, Australia (Levow, 2006). We adopt the data set from the third Bakeoff because it is one of the widely used standard data set and it is available to us.

There were four data sets to be evaluated in the third bakeoff: CKIP (from Academia Sinica), CityU (from the City University of Hong Kong), MSRA (from Microsoft Research Asia) and UPUC (from University of Pennsylvania/University of Colorado). All the data set have an Unicode encoding version. 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. The test data is in the same format, except that, of course, 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 I, evenly-weighted F-score(F), out-of-vocabulary recall rate (ROOV), and in-vocabulary recall rate (RIV). 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 that are in the gold standard test set and also in the dictionary.

6.3 Data Set Statistics

All the three proposed methods are evaluated on the UPUC corpus from the third SIGHAN bakeoff. The statistics, including sentence, word, and character information for each training corpus, is summarized in Table 1. And the number of sentences of the testing corpus is shown in Table 2.

	UPUC Corpus
Number of Sentences	18,804
Number of Words	1,144,899
Number of Word Types	74, 764
Number of Characters	1,235,673
Number of Character Types	8,586

Table 1 Statistics of the training data

	UPUC Corpus
Number of Sentences	5,117

Table 2 Number of sentences in testing data

6.4 The Baseline System

As mentioned in 6.1, in order to measure the performance gain contributed by extra features that are generated by our methods, we conduct experiments to compare the overall performance of a CRF based CWS system with and without these extra features. We describe that CRF based CWS, or the baseline system in this section.

As CRF in its nature is a sequence-labeling algorithm, it is straightforward to use it to process CWS, which is a sequence-labeling task. If we treat the CRF implementation as black box, the only thing we need to do is to transform the CWS training data into the required input of CRF and specify parameters. Usually, there are two factors that we can play with once the CRF implementation is given: 1) the standard feature templates; and 2) the tag-set.

The standards feature templates usually include unigrams and bigrams of characters and their traits. It may sound intenerating to also include tri-gram features in the baseline system as our methods generate long n-grams of arbitrary length. But as discussed in section 3.1, the huge number of possible tri-grams makes the computation unaffordable. Thus the baseline system sticks to standard unigram and bigram feature template.

While the 2-tag-set and the 4-tag-set are more popular, the 6-tag-set leads to better performance, according to (Zhao et al, 2006). Since we are interested in both how much performance gain the proposed methods can contribute and the highest possible overall system performance with generated features, we chose the 6-tag-set.

We built a CRF based Chinese word segmenter similar to the one described in (Zhao et al, 2006). We adopt their so-called six-tag-set and unigram/bigram of character feature templates and use CRF++, their chosen implementation package of CRFs to act as the baseline system.

6.4.1 Tag sets

As introduced in Chapter two, there are various tag-sets. Two most popular tag sets, the 2-tag-set and the 4-tag-set ("BMES") are shown in Table 3. Generally speaking, activated feature functions in practice are determined by both feature template and tag set. Zhao et al. (2006) claims that tagging longer words will be more effective and overall performance will be improved if the 4-tag-set is extended into the 6-tag-set, by adding new tags, '*B*2' and '*B*3' to the 4-tag-set. We adopted the 6-tag-set (shown in Table 4 in our baseline system.

4-tag –set		2-tag-set	
Tag	Function	Tag	Function
B	begin	Start	start
M	middle		
E	end	NoStart	continuation
S	single		

Table 3 4-tag-set and 2-tag-set for CWS

Word Length	1	2	3	4	5	6	7 or longer
Tag Sequence	S	BE	BB_2E	BB_2B_3E	BB_2B_3ME	BB_2B_3MME	$BB_2B_3M…ME$

Table 4 Definition of 6-tag-set for CWS

6.4.2 Feature templates

The probability model and corresponding feature function is defined over the set $H \times T$, where H is the set of possible contexts (or any predefined condition) and T is the set of possible tags. Generally, a feature function can be defined as in Figure 20, where *hi* and *ti* are elements of H and T, respectively.

$$f(h,t) = \begin{cases} 1, \text{ if } h = h_i \text{ is satisfied and } t = t_j \\ 0, \text{ otherwise,} \end{cases}$$

Figure 20 The feature function of CRF

For convenience, features are generally organized into some groups, which used to be called feature templates. For example, a bigram feature template C_1 stands for the next character occurring in the corpus after each character. The feature templates we used in the baseline system is shown in Table 5

Code	Type	Feature	Function
a	Unigram	C_{-1}, C_0, C_1	The previous, current, and next characters
b	Bigram	$C_{-1}C_0$, C_0C_1	The previous(next) and current characters
c	Jump	$C_{-1}C_1$	The previous and next characters
d	Digital	D_0	Current character is a digital or not

Table 5 Feature templates

6.5 Features Generated by our Methods

As introduced in Chapter 3 and Chapter 4, the proposed feature generation methods are mostly based on n-gram expanders, each of which has its own expansion criteria. In both character mutual information based expander and label entropy based

expander, generated feature candidates are n-grams of characters of arbitrary length. Since the features in CRF are defined with regard to each observation (in CWS, each occurrence of character in the corpus), we transform those selected n-gram of characters as "whether current character appears at the *i-th* position of the *x-th* generated n-grams of characters". For example, "灾区的" ("of the disaster area") is a generated n-grams of character, thus for each occurrence of the character 灾 within the context of this very sequence of characters in the corpus, it will be denoted as "the first character in the n-gram 灾区的". In the same way, each occurrence of 区 and 的 in the same sequence will be denoted as "the 2nd character in the n-gram 灾区的" and "the 3rd character in the n-gram 灾区的", respectively. In real implementation, features of all the characters are represented as a large matrix, each row of which looks like Figure 21.

C	*p1*	*p2*	*p3*	*p4*	*p5*	*p6*	*p7*	*p8*	*p9*	*p10*	*p11*	*p12*

Figure 21 Format of feature matrix

In each row of the matrix, the 1st position, denoted as C is an occurrence of a character in the corpus. The 2nd through the 13th position, denoted as *p1*...*p12*, are slots for integer numbers. Each integer value corresponds to an n-gram generated by the proposed methods and the place where they appear in the current row (ranging from *p1* to *p12*) stands for the position where the current character occurs in that particular n-gram. Since most Chinese words are no longer than 6 characters and they are rarely longer than 12 characters, we set the maximum number of positions to be 12. Let's come back to the example of generated n-gram 灾区的. Suppose the integer number that represents that n-gram is 7100, then a fragment of the feature matrix will look like Figure 22. The format of this matrix complies with the input format of CRF++.

Please note that it is possible that one character appears at different positions of two different (often overlapping) n-grams at the same time. For example, if another generated n-gram is 的人民 ("people of some place/of some property"), whose corresponding integer is 1364, then the character 的 may occur in both n-grams. And if these two n-grams happen to be overlapped as in the character sequence 灾区的人

民, the character 的 has both "the 3rd position of the n-gram 灾区的" feature and "the 1st position of the n-gram 的人民" feature, whose feature matrix representation is shown in Figure 23.

C	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12
…												
灾	7100	0	0	0	0	0	0	0	0	0	0	0
区	0	7100	0	0	0	0	0	0	0	0	0	0
的	0	0	7100	0	0	0	0	0	0	0	0	0
...												

Figure 22 A fragment of the feature matrix (a)

C	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12
…												
灾	7100	0	0	0	0	0	0	0	0	0	0	0
区	0	7100	0	0	0	0	0	0	0	0	0	0
的	**1364**	**0**	**7100**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
人	0	1364	0	0	0	0	0	0	0	0	0	0
民	0	0	1364	0	0	0	0	0	0	0	0	0
...												

Figure 23 A fragment of the feature matrix (b)

Once we have these feature matrixes, they can be used directly as the input for CRF++. Besides all those features as described in the feature template for the baseline system (Table 5), we can define 12 extra unigram feature templates, which are values of *p1* to *p12*, respectively, for the current character.

6.6 Experiments

We applied the mutual information based expander and label entropy based expander to generate n-gram of characters and write the corresponding feature matrix, upon which we define 12 extra unigrams feature templates to represent the feature of "current character appears at the *i-th* position of the *x-th* generated n-grams of characters". Thus we had mutual information based features (MI) and label entropy based features (LE), which can be used as extra feature sets for the baseline system.

As for label distribution bin and character types, we did not actually generate hybrid n-grams of characters and label distribution bins/character types as features. Rather, we assigned the label distribution bin (B) and character type (T) for each character, and write them as the 14th and 15th column of the feature matrix. And then we use templates to define hybrid bi-grams and tri-grams. Some of the templates are shown in Table 6.

Code	Type	Feature	Function
e	Unigram	T_{-1}, T_0, T_1	The character type of previous, current, and next characters
f	Unigram	B_{-1}, B_0, B_1	The label distribution bin of previous, current, and next characters
g	Bigram	$C_{-1}T_0$, T_0C_1, $T_{-1}C_0$, C_0T_1,	The hybrid bigrams of characters and character types of previous (next) and current positions
h	Bigram	$C_{-1}B_0$, B_0C_1, $B_{-1}C_0$,	The hybrid bigram of characters and label distribution bins of previous (next) and current positions
i	Jump	$T_{-1}C_1$, $C_{-1}T_1$ $C_{-1}B_1$, $B_{-1}C_1$	The hybrid bigram of characters, label distribution and characters of previous and next positions

Table 6 Feature templates for character type and label distribution bin

6.7 Results

The results with UPUC corpus are shown Table 7, measured in precision, recall and f-scores. The first three rows are the best result on closed track, the best result on open track and the official baseline (based on maximum matching) on the UPUC corpus, respectively (Levow, 2006). The fourth row is the result from a based line CRF based CWS system, as described in section 6.3. The rest results are also achieved by this system, but with extra feature sets, whose names are indicated in the first column. The notations are: MI stands for mutual information based feature set, LE corresponds to the label entropy based feature set, D represents the "Is digit or alphabet or special symbol" feature. Besides, *e, g, f, h* are the codes for the feature sets used, as defined in Table 6. The symbol "+" indicates a combination of two feature sets. Note that *e*+*g* are actually unigram and hybrid bigram features of word types (T), and *f*+*h* are actually unigram and hybrid bigram feature features of label distribution bins (B).

Except the label distribution bin based feature set B (i.e. *f*+*h* group), all the other feature sets or feature set combinations contribute to performance improvements of the baseline CRF system, measured by precision, recall and F-scores. Since the F-

score of the CRF baseline system (0.924) is much higher than that of the official baseline (0.828) of Bakeoff-2006 and would rank the 5th in the closed track, some 0.5~1% performance gain over it, as achieved by these feature sets, is reasonably good. These performance improvements equals reduction of errors by 5%~13%, as shown in Table 8. If compared with results submitted to Bakeoff-3, our results would rank 1st to 4th out of 15 total submissions in the closed track, or 3rd to 5th out of 8 total submissions in the open track. Finally, the number of features generated by each method is given in Table 9, together with the corresponding CPU time consumed in training the CRF model.

	Precision	Recall	F-score
Best 2006 closed	*0.926*	*0.940*	*0.933*
Best 2006 open	*0.939*	*0.949*	*0.944*
Baseline 2006	*0.790*	*0.869*	*0.828*
Baseline CRF	0.918	0.930	0.924
MI	0.921	0.932	0.927
MI+D	0.919	0.932	0.926
MI+g	**0.927**	**0.940**	**0.934**
T (e+g)	0.926	0.940	0.932
B(f+h)	0.895	0.910	0.902
LE+g	**0.925**	**0.939**	**0.933**
LE	0.923	0.933	0.928

Table 7 Performances on UPUC Corpus

	Error reduction
MI+T	13.0%
LE+T	11.8%
T	10.5%
LE	7.2%
MI	5.3%
B	-28.9%

Table 8 Error reductions contributed by various feature sets compared with the baseline CFR system

	Num of features	CPU time
Baseline CRF	4,994,184	21559.12
MI	5,227,485	30531.67
MI+D	5,232,271	32598.12
MI+g	12,549,990	29369.29
T (e+g)	12,024,690	11651.35
B(f+h)	15,833,092	4537.48
LE+g	12,186,570	33009.20
LE	5,155,980	31793.84

Table 9 Number of features generated and CPU time consumed by each method

6.8 Discussion

Generalization of surface symbols should have a solid basis in order to be beneficial. Both label distribution bins and character types are means of generalizing concrete Chinese characters into more abstract categories, but their effects on the system performance are completely different. While introducing hybrid bigrams of characters and character types as features has lead to more than 10% of error reduction, features of the hybrid bigrams of label distribution bins and characters has caused 30% more errors than the baseline. Generalization can deal with data sparseness problem, but it should base on some solid principles, either linguistic or computational.

The types of Chinese characters is comparable with part-of-speech tags, or word categories of English words, which is an abstraction on what roles words can play within a sentence. Just like words of same POS tags may appear in similar positions of some phrases, i.e. sequence of words, Chinese characters of same types may also appear in similar positions of some words, i.e. sequence of characters. Thus it is imaginable that types of characters are good indicators for characters' positions in words, i.e. CWS tags of characters. Label distribution bins are expected to act in the similar manner as character types, but proved to be unsuccessful. The reason why it should work is more or less a guess without rigid proving, and some details such as improper scaling and grouping in the discretization process probably should be responsible for the failure.

As discussed in Section 4.3.1, the features generated from the training data have a relatively small overlap with the features generated from the testing data, since the majority of subsequences of characters (both words and non-words) occur only one or a few times in a corpus according to the Zipf's law. This explains why character type features contribute more performance gain than any other feature sets, even though the n-gram expansion based methods (mutual information based and label entropy based in particular) can generate some seeming promising longer n-grams as features, such as 毛泽东 ("Mao Tsedong", which is a word itself and name entity). Of course if those n-grams that are words themselves occur in both training and testing data, it can contribute to correct segmentations. A good example is 进一步 ("further"), whose substrings 进 ("to move forward")and 一步 ("one step") are also valid words. Considering only unigrams and bigrams, the baseline system made the incorrect segmentation, while the system with this feature segmented it correctly.

In more cases, n-grams generated by our methods are not necessarily valid words. Most n-grams covers s span longer than one words, such as 运用他, which is made up of two words, 运用 (to use) and 他 ("he"/prefix of "his"). In this case, since there is no link or dependency between these two parts and the baseline system with unigram and bi gram features can also segment the sentence correctly, long n-grams are not useful. Of course, n-grams are not necessarily to be words themselves to be useful. For example, a generated n-gram, 固定资 is shorter than the phrase 固定资产 ("fixed asset") the and longer than the word 资产 ("asset"). But as all the occurrences of this n-gram are labeled as *B-E-B*, it provides a useful context for correct tagging. However, in general, because of the data sparseness problem, the improvement brought by n-gram of concrete characters is limited, and proposing more general patterns or features seems to be a more promising direction.

Features are overlapping and sometimes contradictory. It is clear that adding bad features may hurt the overall performance, just like the case in label distribution bin based features. But even combining two "beneficial" feature sets may cause a drop in the performance. This is what has happened to the combination of mutual information based features and "Is digit/alphabet/symbol" features (MI+D group). This raises a question of how to do the feature engineering symmetrically to maximize the performance gain.

Chapter 7 Conclusion

7.1 Conclusion

The aim of this work is to improve the performance of the Conditional Random Fields based Chinese word segmentation systems. We try to achieve this goal by automatically generating extra features for the learning framework. As a conditional model, CRFs can adopt arbitrary overlapping features to help with the sequence learning and is considered as the state-of-the-art. But common features only include all the unigrams and bigrams of characters in the sequence, though long n-grams can be useful in describing longer dependencies or other rich linguistic phenomena. Since it is computationally costly and ineffective to include all longer n-grams as features, we have proposed methods to select only a few long n-grams to represent some characteristics of the sequence.

Inspired by n-gram expansion methods in error mining, our proposed methods adopt a unified framework of iteratively processing Chinese characters in the corpus to expand each character to the longest possible n-gram, given there is evidence that it is worthwhile. As the usefulness of the feature can be known only after the learning-testing circle is over, which usually takes too much time, we proposes heuristics based criteria to estimate the usefulness of n-grams beforehand. One criterion is based on mutual information (MI) of characters, as we believe words are more or less sequence of characters that are more frequently combined. Beside this global frequency information, we also take effort to utilize the annotation information in the corpora. The difficulty of labeling is that one character usually has several different tags across the corpora. We treat n-grams as context to decrease ambiguity of labeling a character, and for each ambiguous character we process, we choose those n-grams long enough to disambiguate its labeling as features, which we call label-entropy (LE) based method. Experiment on the SIGHAN Bakeoff-3 UPUC data set indicates that these two methods can contribute 5%~7% performance improvement over a baseline CRF CWS system, which reflects the usefulness of these features.

As a meta-problem in NLP, the data sparseness is also a challenge in CWS. If all the features are based on characters or character sequences, it is likely that many

sequences in testing data are unseen, which is usually caused by out-of-vocabulary or rare combinations of words. If we adopt more abstract symbols as generalization of characters, the possible combinations of abstract symbols are much less than that of characters, thus the chance of occurring unseen patterns are lower. The only requirement is that we find the correct generalization. We have tried both types of characters and label distribution bin (B) of characters as abstract symbols and have conducted experiments with type-character hybrid n-grams and bin-character hybrid n-grams as features. Experiments show that type-character hybrid n-grams contribute as much as 10.5% error reduction while label distribution bin based features have adversely effect on performance.

Combining character type based features with mutual information or label entropy based features can further improve the error reduction to 13% and 11.8%, respectively. The former has the F-score of 0.934 on UPUC corpus and is higher than the result reported by the official 1^{st} /3^{rd} in the closed/open track in Bakeoff-3. Both relative and absolute performance showed that the n-gram expanded based feature generation work is effective.

7.2 Future Work

As we only assign each character a symbol that represents all its possible type of characters as its character type in our experiments, it is possible that the performance can be further improved if the accurate type of character is given. In this case, the CWS task can be treated as a joint learning of characters and character types. And it is even more interesting to do the jointly learning together with POS tagging.

As the purpose of introducing long n-grams is to provide more information on longer dependencies and other linguistic phenomena, it would be nice to add some easily access semantic or grammatical features/models, which may give a more effect description of the characteristics of character sequences. It is also interesting to port the proposed methods to generate features for other sequence labeling problems.

Reference

Kenneth Church and Patrick Hanks. 1990. Word association norms, mutual information and lexicography. In *Computational Linguistics,*16(1).

Thomas Emerson. 2003. The second international chinese word segmentation bakeoff. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*

David Forney. 1973. The Viterbi algorithm. In *Proceedings of the IEEE,* 61(3).

Jin Guo. 1997. Critical Tokenization and its Properties. *Computational Linguistics*, 23(4).

Wenbin Jiang, Liang Huang, Yajuan Lu, and Qun Liu. 2008. A cascaded linear model for joint Chinese word segmentation and part-of-speech tagging. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*.

Chunyu Kit, Yuan Liu and Nanyuan Liang. 1989. On methods of automatic Chinese word Segmentation. *Journal of Chinese Information Processing,* 3(1).

Daniel de Kok, Jianqiang Ma and Gertjan van Noord. 2009. A generalized method for iterative error mining in parsing results.In *Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks, ACL-IJCNLP 2009*

John Lafferty, Andrew McCallum, Fernando Pereira. 2001. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*

Gina-Anne Levow. 2006. The third international Chinese language processing bakeoff: Word segmentation and named entity recognition. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*

Nanyuan Liang. 1986. Shumian hanyu zidong fenci xitong-CDWS [A written Chinese automatic segmentation system-CDWS. *Chinese Information Processing,* 1(1).

Andrew McCallum, Dayne Freitag and Fernando Pereira. 2000. Maximum Entropy Markov Models for Information Extraction and Segmentation. In *Proc. International Conference of Machine Learning*

Hwee Tou Ng and Jin Kiat Low. 2004. Chinese partof-speech tagging: One-at-a-time or all-at-once? word-based or character-based? In *Proceedings of the Empirical Methods in Natural Language Processing Conference*.

Gertjan van Noord. (2004). Error Mining for Wide-Coverage Grammar Engineering. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics: ACL 2004,* 446-453.

Fuchun Peng, Fangfang Feng and Andrew McCallum. 2004. Chinese segmentation and new word detection using conditional random fields. In: *Proceedings of the 20th International Conference on Computational Linguistics*

Lawrence Rabiner. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *In Proceedings of the IEEE*, 77(1)

Binyamin Rosenfeld, Rosen Feldman and Moshe Fresko. 2006. A systematic cross-comparison of sequence classifier. *In SDM 2006*

Benoit Sagot and Eric De la Clergerie. 2006. Error mining in parsing results. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*

Richard Sproat and Thomas Emerson. 2003. First International Chines Word Segmentation Bakeoff. In *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing*.

Richard Sproat, Chilin Shih, William Gale, and Nancy Chang. 1996. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3).

Yongheng Wang, Haiju Su and Yan Mo. 1990. Automatic processing of Chinese words. *Journal of Chinese information Processing* 4(4)

Ching-long Yeh and His-Jian Lee. 1991. Rule based word identification for Mandarin Chinese sentences – a unification approach. *Computer Processing of Chinese and Oriental* Languages, 5(2)

Hai Zhao. 2009. Character-level dependencies in Chinese: usefulness and learning. In *Proceedings of the 12th Conference of the European Chapter of the ACL.*

Hai Zhao, Chang-Ning Huang, Mu Li, and Bao-Liang Lu. 2006. Effective tag set selection in Chinese word segmentation via conditional random field modeling. In *Proceedings of PACLIC-20*