# Technical Report of *Chinese Morphological Analyzer(Chen)*

Vernkin Smith

September 18, 2009

## Abstract

Practical results show that performance of statistic segmentation system outperforms that of hand-crafted rule-based systems. And the evaluation shows than the accuracy drop caused by out-of-vocabulary (OOV) words is at least five times greater than that of segmentation ambiguities. The better performance of OOV recognition the higher accuracy of the segmentation system in whole, and the accuracy of statistic segmentation systems with character-based tagging approach outperforms any other word-based system. This Report is about a supervised machine-learning approach to Character-based Chinese word segmentation. A maximum entropy tagger is trained on manually annotated data to automatically assign to Chinese characters, or *hanzi*, tags that indicate the position of a *hanzi* within a word.

# Changes

| Date | Author | Notes |
|---|---|---|
| 2009-09-10 | Vernkin | Initial version |
| 2009-09-11 | Vernkin | Add Change Log Section and Wisenut QC's Experiment Section for Experiments Chapter. |
| 2009-09-11 | Vernkin | Update the Conclusion Chapter basing QC's experiment and Project Schedule. |

# Contents

# Chapter 1

# Overview

## 1.1 A Decade Review of Chinese Word Segmentation

During the last decade, especially since the First International Chinese Word Segmentation Bakeoff was held in July 2003, the study is automatic Chinese word segmentation has been greatly improved. Those improvements could be summarized as following: (1) on the computation sense Chinese words in read text that have been well-defined by "segmentation guidelines + lexicon + segmented corpus"; (2) practical results show that performance of statistic segmentation system outperforms that of hand-crafted rule-based systems; (3) the evaluation shows than the accuracy drop caused by out-of-vocabulary (OOV) words is at least five times greater than that of segmentation ambiguities; (4) the better performance of OOV recognition the higher accuracy of the segmentation system in whole, and the accuracy of statistic segmentation systems with character-based tagging approach outperforms any other word-based system.

## 1.2 Main Difficulties in Chinese Segmentation

It is generally agreed among researchers that word segmentation is a neccessary first step in Chinese language processing. However, unlike Engish text in which sentences are sequences of words delimited by white spaces, in Chinese text, sentences are represented as strings of Chinese characters or *hanzi* without similar natural delimiters. Therefor, the first step in a Chinese language processing task is to identify the sequence of words in a sentence and mark boundaries in appropriate places. This may sound simple enough but in reality identifying words in Chinese is a non-trivial problem that has drawn a large body of research in the Chinese language processing community.

It is easy to demonstrate that the lack of natural delimiters itself is not the heart of the problem. In a hypothetical language where all words are represented with a finite set of symbols, if one subset of the symbols always start a word and another subset, mutually exclusive from the previous subset, always end a word, identifying words would be a trivial exercise. Nor can the problem be attributed to the lack of inflectional morphology. Although it is true in Indo-European languages inflectional affixes can generally be used to signal word boundaries, it is conceivable that a hypothetical language can use symbols other than inflectional morphemes to serve the same purpose. Therefore the issue is neither the lack of natural word delimiters nor the lack of inflectional morphemes in a language, rather it is whether the language has a way of unambiguously signaling the boundaries of a word.

The real difficulty in automatic Chinese word segmentation is the lack of such unambiguous word boundary indicators. In fact, most hanzi can occur in different positions within different words. The examples in Table 1 show how the Chinese character 产(“produce”) can occur in four different positions. This state of affairs makes it impossible to simply list mutually exclusive subsets of hanzi that have distinct distributions, even though the number of hanzi in the Chinese writing system is in fact finite. As long as a hanzi can occur in different word-internal positions, it cannot be relied upon to determine word boundaries as they could be if their positions were more or less fixed.

Table 1. A hanzi can occur in multiple word-internal positions

| Position | Example |
|---|---|
| Left | 产生 'to come up with' |
| Word by itself | 产小麦 'to grow wheat' |
| Middle | 生产线 'assemby line' |
| Right | 生产 'to produce' |

The fact that a hanzi can occur in multiple word-internal positions leads to ambiguities of various kinds. For example, 文 can occur in both word-initial and word-final positions. It occurs in the word-final position in 日文(“Japanese”) but in the word-initial position in 文章(“article”). In a sentence that has a string “日文章”, as in (1a), an automatic segmenter would face the dilemma whether to insert a word boundary marker between 日 and 文, thus grouping 文章 as a word, or to mark 日文 as a word, to the exclusion of 章. The same scenario also applies to 章, since like 文, it can also occur in both word-initial and word-final positions.

1. (a): Segmentation 1

日文 章魚 怎么 说?

Japanese octopus how say

“How to say octopus in Japanese?”

1. (b): Segmentation 2

日 文章 魚 怎么 说?

Japan article fish how say

Ambiguity also arises because some hanzi should be considered to be just word components in certain contexts and words by themselves in others. For example, 魚 can be considered to be just a word component in 章魚. It can also be a word by itself in other contexts. Presented with the string 章魚 in a Chinese sentence, a human or automatic segmenter would have to decide whether 魚 should be a word by itself or form another word with the previous hanzi. Given that 日, 文章, 章魚, 魚 are all possible words in Chinese, how does one decide that 日文 章魚 is the right segmentation for the sentence in (1) while 日 文章 魚 is not? Obviously it is not enough to know just what words are in the lexicon. In this specific case, a human segmenter can resort to world knowledge to resolve this ambiguity, knowing that 日 文章 魚 would not make any kind of real-world sense.

# Chapter 2

# Introduce to Character-based Chinese Segmentation

Follow the *Overview* and *Abstract*, this chapter would introduce Character-based Chinese Segmentation. The Maximum Entropy Modeling (see chapter-3) used in the segmentation and more detail see the following chapters. In this charper, we first formalize the idea of tagging *hanzi*(Chinese Character) based on their word-internal positions and describe the tag set we used.

First we convert the manually segmented words in the corpus into a tagged sequence of Chinese characters. To do this, we tag each character with one of the four tags, L, R, M and R depending on its position within a word. It is tagged L if it occurs on the left boundary of a word, and forms a word with the character(s) on its right. It is tagged B if it occurs on the beginning position of a word, and forms a word with 0 to n character(s) on its left. It is tagged E if it occurs in the non-beginning position of a word. We call such tags position-of-character (POC) tags to differentiate them from the more familiar part-of-speech (POS) tags. For example, the manually segmented string in (2a) will be tagged as (2b):

Example Senetence 2:

(a) 上海 计划 到 本 世纪 末 实现 人均 国内 生产 总值 五千 美元

(b) 上/B 海/E 计/B 划/E 到/B 本/B 世/B 纪/E 末/B 实/B 现/E 人/B 均/E 国/B 内/E 生/B 产/E 总/B 值/E 五/B 千/E 美/B 元/E

(c) Shanghai plans to reach the goal of 5,000 dollars in per capita GDP by the end of the century.

Given a manually segmented corpus, a POC-tagged corpus can be derived trivially with perfect accuracy. The reason why use such POC-tagged sequences of characters instead of applying $n$-gram rules to segmented corpus directly[Palmer, 1997[5]; Hockenmaier and Brew, 1998[3]; Xue, 2001[6]] is that they are much easier to manipulate in the training process. In addition, the POC tags reflect our observation that the ambiguity problem is due to the fact that a hanzi can occur in different word-internal positions and it can be resolved in context. Naturally, while some characters have only one POC tag, most characters will receive multiple POC tags, in the same way that words can have multiple POS tags. Table 2 shows how all four of the POC tags can be assigned to the character 产 (“produce”):

Table 2. A character can receive as many as two tags

| Position | Tag | Example |
|---|---|---|
| Beginning | B | 产生 'to come up with' |
| Non-Beginning | E | 生产线 'assemby line' |
| Non-Beginning | E | 生产 'to produce' |

If there is ambiguity in segmenting a sentence or any string of hanzi, then there must be some hanzi in the sentence that can receive multiple tags. For example, each of the first four characters of the sentence in (1) would have two tags. The task of the word segmentation is to choose the correct tag for each of the hanzi in the sentence. The eight possible tag sequences for (1) are shown in (3a), and the correct tag sequence is (3b).

Example Senetence 3:

(a) 日/B,E 文/B,E 章/B,E 鱼/B,E 怎/B 么/E 说/B ?

(b) 日/B 文/E 章/B 鱼/E 怎/B 么/E 说/B ?

Also like POS tags, how a character is POC-tagged in naturally occurring text is affected by the context in which it occurs. For example, if the preceding character is tagged R or R, then the next character can only be tagged L or R. How a character is tagged is also affected by the surrounding characters. For example, 关(”close”) should be tagged E if the previous character is 开 (”open”) and neither of them forms a word with other characters, while it should be tagged B if the next character is 心 (”heart”) and neither of them forms a word with other characters. This state of affairs closely mimics the familiar POS tagging problem and lends itself naturally to a solution similar to that of POS tagging. The task is one of ambiguity resolution in which the correct POC tag is determined among several possible POC tags in a specific context. Our next step is to train a maximum entropy model on the perfectly POC-tagged data derived from a manually segmented corpus to automatically POC-tag unseen text.

# Chapter 3

# Maximum Entropy Modeling

This chapter provides a brief introduction to Maximum Entropy Modeling. The Advantage of maximum entropy model includes: Based on features, allows and supports feature induction and feature selection; offers a generic framework for incorporating unlabeled data; only makes weak assumptions; gives flexibility in incorporating side information; natural multi-class classification.

Maximum Entropy (ME or maxent for short) model is a general purpose machine learning framework that has been successfully applied in various fields including spatial physics, computer vision, and Natural Language Processing (NLP). This introduction will focus on the application of maxent model to NLP tasks. However, it is straightforward to extend the technique described here to other domains.

## 3.1 The Modeling Problem

The goal of statistical modeling is to construct a model that best accounts for some training data. More specific, for a given empirical probability distribution $\tilde{p}$, we want to build a model p as close to $\tilde{p}$ as possible.

Of course, given a set of training data, there are numerous ways to choose a model p that accounts for the data. It can be shown that the probability distribution of the form 3.1 is the one that is closest to $\tilde{p}$ in the sense of Kullback-Leibler divergence, when subjected to a set of feature constraints:

$$p(y \mid x) = \frac{1}{Z(x)} \exp\left[\sum_{i=1}^{k} \lambda_i f_i(x, y)\right] \tag{3.1}$$

Here $p(y \mid x)$ denotes the conditional probability of predicting an *outcome* $y$ on seeing the *context* $x$. $f_i(x, y)$'s are feature functions (described in detail later), $\lambda_i$'s are the weighting parameters for $f_i(x, y)$'s. $k$ is the number of features and Z(x) is a normalization factor (often called partition function) to ensure that $\sum_y p(y|x) = 1$.

ME model represents evidence with binary functions1 known as *contextual predicates* in the form:

$$f_{cp,y'}(x, y) = \begin{cases} 1 & \text{if } y = y' \text{ and } cp(x) = true \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Where *cp* is the contextual predicate that maps a pair of *outcome* $y$ and *context* $x$ to $\{true, false\}$.

The modeler can choose arbitrary feature functions in order to reflect the characteristic of the problem domain as faithfully as possible. The ability of freely incorporating various problem-specific knowledge in terms of feature functions gives ME models the obvious advantage over other learn paradigms, which often suffer from strong feature independence assumption (such as naive bayes classifier).

For instance, in part-of-speech tagging, a process that assigns part-of-speech tags to words in a sentence, a useful feature may be (DET is DETERMINER for short):

$$f_{prev\_tag\_DET,NOUN}(x, y) = \begin{cases} 1 & \text{if } y = NOUN \text{ and } prev\_tag\_DET(x) = true \\ 0 & \text{otherwise} \end{cases}$$

which is *activated* when previous tag is DETERMINER and current word's tag is NOUN. In Text Categorization task, a feature may look like (RO is ROMANTIC for short):

$$f_{doc\_has\_RO,love\_story}(x, y) = \begin{cases} 1 & \text{if } y = love_story \text{ and } doc\_has\_RO(x) = true \\ 0 & \text{otherwise} \end{cases}$$

which is *activated* when the term ROMANTIC is found in a document labeled as type:love_story.

Once a set of features is chosen by the modeler, we can construct the corresponding maxent model by adding features as constraints to the model and adjust weights of these features. Formally, We require that:

$$E_{\tilde{p}} < f_i >= E_p < f_i >$$

Where $E_{\tilde{p}} < f_i >= \sum_x \tilde{p}(x, y) f_i(x, y)$ is the empirical expectation of feature $f_i(x, y)$ in the training data and E$E_p < f_i >= \sum_x p(x, y) f_i(x, y)$ is the feature expectation with respect to the model distribution $p$. Among all the models subjected to these constraints there is one with the Maximum Entropy, usually called the Maximum Entropy Solution.

## 3.2 Parameter Estimation

Given an exponential model with $n$ features and a set of training data (empirical distribution), we need to find the associated real-value weight for each of the $n$ feature which maximize the model's log-likelihood:

$$L(p) = \sum_{x,y} \tilde{p}(x,y) \log p(y \mid x) \tag{3.3}$$

Selecting an optimal model subjected to given contains from the exponential (log-linear) family is not a trivial task. There are two popular iterative scaling algorithms specially designed to estimate parameters of ME models of the form 3.1: *Generalized Iterative Scaling* [Darroch and Ratcliff, 1972][1] and *Improved Iterative Scaling* [Della Pietra et al.,1997][2].

Recently, another general purpose optimize method *Limited-Memory Variable Metric* (L-BFGS for short) method has been found to be especially effective for maxent parameters estimating problem [Malouf, 2003][4]. L-BFGS is the default parameter estimating method in the implementation.

## 3.3 Usage of the MaxEnt Model

This section covers the basic steps required to build and use a Conditional Maximum Entropy Model.

### 3.3.1 Representing Features

The mathematical representation of a feature used in a Conditional Maximum Entropy Model can be written as:

$$f_{cp,y'}(x,y) = \begin{cases} 1 & \text{if } y = y' \text{ and } cp(x) = true \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

where $cp$ is the $contextual predicate$ which maps a pair of $outcome y$ and $context x$ into $\{true, false\}$.

This kind of math notation must be expressed as features of literal string in order to be used in this toolkit. So a feature in part-of-speech tagger which has the form (DET is DETERMINER for short):

$$f_{prev\_tag\_DET,NOUN}(x,y) = \begin{cases} 1 & \text{if } y = NOUN \text{ and } prev\_tag\_DET(x) = true \\ 0 & \text{otherwise} \end{cases}$$

can be written as a literal string: “tag-1=DETERMINER_NOUN”.

### 3.3.2 Create a Maxent Model Instance

A *maxent* instance can be created by calling its constructor:

```
#include <maxent/maxentmodel.hpp>
using namespace maxent;
MaxentModel m;
```

This will create an instance of MaxentModel class called m. Please note that all classes and functions are in the namespace maxent. For illustration purpose, the include and using statements will be ignored intentionally in the rest of this section.

### 3.3.3 Adding Events to Model

Typically, training data consists of a set of events (samples). Each event has a *context*, an *outcome*, and a *count* indicating how many times this event occurs in training data.

Remember that a *context* is just a group of *contextpredicates*. Thus an event will have the form:

$$[(predicate_1, predicate_2, \ldots, predicate_n), outcome, count]$$

Suppose we want to add the following event to our model:

$$[(predicate_1, predicate_2, predicate_3), outcome1, 1]$$

We need to first create a context:

```
std::vector<std::string> context;
context.append("predicate1");
context.append("predicate2");
context.append("predicate3");
. . .
```

It’s possible to specify feature value (must be non-negative) in creating a context:

```
std::vector<pair<std::string, float> > context;
context.append(make_pair("predicate1",1.0));
context.append(make_pair("predicate2",2.0));
context.append(make_pair("predicate3",3.0));
. . .
```

Before any event can be added, one must call begin_add_event() to inform the model the beginning of training.

```
m.begin_add_event();
```

Now we are ready to add events:

```
m.add_event(context, "outcome1", 1);
```

The third argument of add_event() is the count of the event and can be ignored if the count is 1.

One can repeatedly call add_event() until all events are added to the model.

After adding the last event, end_add_event() must be called to inform the model the ending of adding events.

```
m.end_add_event();
```

### 3.3.4 Training the Model

Train a Maximum Entropy Model is relatively easy. Here are some examples:

```
// train the model with default training method
m.train();
// train the model with 30 iterations of L-BFGS method
m.train(30, "lbfgs");
// train the model with 100 iterations of GIS method and apply
// Gaussian Prior smoothing with a global variance of 2
m.train(100, "gis", 2);
// set terminate tolerance to 1E-03
m.train(30, "lbfgs", 2, 1E-03);
```

The training methods can be either “gis” or “lbfgs” (default). Also, if m.verbose is set to 1 (default is 0), training progress will be printed to stdout.

You can save a trained model to a file and load it back later:

```
m.save("new_model");
m.load("new_model");
```

A file named new_model will be created. The model contains the definition of context predicates, outcomes, mapping between features and feature ids and the optimal parameter weight for each feature.

If the optional parameter binary is true and the library is compiled with zlib support, a compressed binary model file will be saved which is much faster and smaller than plain text model. The format of model file will be detected automatically when loading:

```
m.save("new_model", true); //save a (compressed) binary model
m.load("new_model");       //load it from disk
```

### 3.3.5 Using the Model

The use of the model is straightforward. The *eval*() function will return the probability $p(y|x)$ of an *outcomey* given some *contextx*:

```
m.eval(context, outcome);
```

*eval_all*() is useful if we want to get the whole conditional distribution for a given context:

```
std::vector<pair<std::string, double> > probs;
m.eval_all(context, probs);
```

*eval_all*() will put the probability distribution into the vector probs. The items in probs are the outcome labels paired with their corresponding probabilities. If the third parameter sort_result is true (default) *eval_all*() will automatically sort the output distribution in descendant order: the first item will have the highest probability in the distribution.

# Chapter 4

# POS Tagger

This Chapter discusses the steps involved in building a Part-of-Speech (POS) tagger using Maximum Entropy Model in detail.

## 4.1 The Tagging Model

The task of POS tag assignment is to assign correct POS tags to a word stream (typically a sentence). The following table lists a word sequence and its corresponding tags:

To attack this problem with the Maximum Entropy Model, we can build a conditional model that calculates the probability of a tag $y$, given some contextual information $x$:

$$p(y|x) = \tfrac{1}{Z(x)} \exp\left[\textstyle\sum_{i=1}^{k} \lambda_i f_i(x, y)\right]$$

Thus the possibility of a tag sequence $\{t_1, t_2, \ldots, t_n\}$ over a sentence $\{w_1, w_2, \ldots, w_n\}$ can be represented as the product of each $p(y|x)$ with the assumption that the probability of each tag $y$ depends only on a limited context information $x$:

$$p(t_1, t_2, \ldots, t_n | w_1, w_2, \ldots, w_n) \approx \textstyle\prod_{i=1}^{n} p(y_i|x_i)$$

Given a sentence $\{w_1, w_2, \ldots, w_n\}$, we can generate $K$ highest probability tag sequence candidates up to that point in the sentence and finally select the highest candidate as our tagging result.

## 4.2 Feature Selection

We select features used in the tagging model by applying a set of feature templates to the training data. The features are:

1. Prefix / suffix characters of the word;
2. Whether it is numeric;
3. Whether it is hyphen;
4. The word and the POS of the last/previous word;
5. The word and the POS of the last/previous 2th word;
6. the words and the POSs combination of the last/previous 1st the 2th words.

The following table is the features which are selected from the actual sentence:

Table 4.1 The Features Example

| |
|---|
| curword=years |
| tag-1=CD |
| word-2=, |
| tag-1,2=,CD |
| word+1=old |
| curword=old |
| word-1=years |
| tag-1=NNS |
| tag-1,2=CD,NNS |
| word+2=will |
| word-1=old |
| prefix=E |
| prefix=El |
| suffix=r |
| suffix=er |
| suffix=ier |

Table 4.2 The Contextual Predicates Example

| Condition | Contextual Predicates |
|---|---|
| $w_i$ is not rare | $w_i = X$ |
| $w_i$ is rare | $X$ is prefix of $w_i$, $\|X\| \leq 4$ |
| | $X$ is suffix of $w_i$, $\|X\| \leq 4$ |
| | $X$ contains number |
| | $X$ contains uppercase character |
| | $X$ contains hyphen |
| $\forall w_i$ | $t_{i-1} = X$ |
| | $t_{i-w} t_{i-1} = XY$ |
| | $w_{i-1} = X$ |
| | $w_{i-2} = X$ |
| | $w_{i+1} = X$ |
| | $w_{i+2} = X$ |

Please note that if a word is rare (occurs less than 5 times in the training set) several additional contextual predicates are used to help predict the tag based on the word's form. A useful feature might be:

$$f(x,y) = \begin{cases} 1 & \text{if y=VBG and } current\_suffix\_is\_ing(x) = true \\ 0 & \text{otherwise} \end{cases}$$

and is represented as a literal string: "suffix=ing_VBG".

chapterSegment Tagger

This Chapter discusses the steps involved in building a Segment Tagger tagger using Maximum Entropy Model in detail. Compared with POS, Segment Tagger uses POC (Position of Character)

## 4.3 POC (Position of Character)

There are two POC tags: B (the beginning position of the word) and E (th non-beginning position of the word). See the following example:

Table 5.1: The two POC tags of the Character 产.

| Tag | Example |
|---|---|
| B | 产生 'to come up with' |
| E | 生产线 'assembly line' |

## 4.4 POC Tagger

The POC tagger here uses the same probability model as the POS tagger. The probability model is defined over $H \times T$, where $H$ is the set of possible *contexts* or "*histories*" and $T$ is the set of possible tags. The model's joint probability of a history $h$ and a tag $t$ is defined as

$$p(h,t) = \pi\mu \prod_{j=1}^{k} \alpha_j^{f_j(h,t)} \tag{4.1}$$

where $\pi$ is a normalization constant, $\{\mu, \alpha_1, ..., \alpha_k\}$ are the model parameters and $\{f_1, ..., f_k\}$ are known as features, where $f_j(h,t) \in \{0,1\}$. Each feature $f_j$ has a corresponding parameter $\alpha_j$, hat effectively serves as a "*weight*" of this feature. In the training process, given a sequence of characters $\{c_1, ..., c_k\}$ and their POC tags $\{t_1, ..., t_k\}$ as training data, the purpose is to determine the parameters $\{\mu, \alpha_1, ..., \alpha_k\}$ that maximize the likelihood of the training data using $p$:

$$L(P) = \prod_{i=1}^{n} P(h_i, t_i) = \prod_{i=1}^{n} \pi\mu \prod_{j=1}^{k} \alpha_j^{f_j(h_i,t_i)} \tag{4.2}$$

The success of the model in tagging depends to a large extent on the selection of suitable features. Given $(h,t)$, a feature must encode information that helps to predict $t$ . The features used are instantiations of the feature templates. Feature templates (2) to (4) represent character features while (5) represents tag features. $C_{-3}...C_3$ are characters and $T_{-3}...T_3$ are POC tags. Feature template (1) represents the default feature.

Feature templates:

1. Default feature
2. The current character ($C_0$)
3. The previous (next) two characters ($C_{-2}, C_{-1}, C_1, C_2$)
4. The previous (next) character and the current character ($C_{-1}, C_0, C_1$), the previous two characters ($C_{-2}, C_{-1}$), and the next two characters $C_1, C_2$).
5. The previous and the next character ($C_{-1}, C_1$).
6. The tag of the previous character $T_{-1}$, and the tag of the character two before the current character $T_{-2}$

In general, given $(h, t)$, these features are in the form of co-occurrence relations between $t$ and some type of context $h$ , or between $t$ and some properties of the current character. For example,

$$f_i(h_i, t_i) = \begin{cases} 1 & \text{if } t_{i-1} = L \ \& \ t_i = R \\ 0 & otherwise \end{cases}$$

This feature will map to 1 and contribute towards $p(h_i, t_i)$ if $c_{i-1}$ is tagged $L$ and $c_i$ is tagged $R$.

The feature templates encode three types of contexts. First, features based on the current and surrounding characters $(2, 3, 4, 5)$ are extracted. Given a character in a sentence, this model will look at the current character, the previous two and next two characters. For example, if the current character is 们 (plural marker), it is very likely that it will occur as a suffix in a word, thus receiving the tag $R$. On the other hand, for other characters, they might be equally likely to appear on the left, on the right or in the middle. In those cases where it occurs within a word depends on its surrounding characters. For example, if the current character is 爱 (“love”), it should perhaps be tagged $L$ if the next character is 护 (“protect”). However, if the previous character is 热 (“warm”), then it should perhaps be tagged $R$. Second, features based on the previous tags (5) are extracted. Information like this is useful in predicting the POC tag for the current character just as the POS tags are useful in predicting the POS tag of the current word in a similar context. For example, if the previous character is tagged $I$ or $R$, this means that the current character must start a word, and should be tagged either $L$ or $I$. Finally, a default feature (1) is used to capture cases where no other features are available. When the training is complete, the features and their corresponding parameters will be used to calculate the probability of the tag sequence of a sentence when the tagger tags unseen data. Given a sequence of characters $c_1, ..., c_n$, the tagger searches for the tag sequence $t_1, ..., t_n$ with the highest probability

$$P(t_1, ..., t_n \mid C_1, ..., C_n) = \prod_{i=1}^{n} P(t_i \mid h_i) \tag{4.3}$$

and the conditional probability of for each POC tag $t$ given its history $h$ is calculated as

$$P(t \mid h) = \frac{p(h,t)}{\sum_{t' \in T}^{P(h,t')}} \tag{4.4}$$

## 4.5 Types of Characters

All the types of characters are list below:

Table 5.2: The types of Characters

| | |
|---|---|
| INIT | the initial type, used for the algorithm |
| DIGIT | the digit character, like ”0” and ”1” |
| PUNC | the punctuation character, like ”.” and ”,” |
| LETTER | the letter character, like ”a” and ”D” |
| OTHER | other character, like Chinese character ”汉” |

The rules to combine special strings:

1. digits are digits.

2. punctuations are punctuations.

3. other characters are other characters.

4. any combination of digits, hyphens and letters or simple letters and letters.

The class CMA_CType is the class to identify the types of the charcters, and each encoding (gb2312 and big5 have its own CMA_CType). To identify the type of a character, besides that character, the type of the previous character and next character are all required. It is because some characters' type varies under different context. For example, "分"(cent) in the "五分之一"( = 1/5) is a number, and the type of the previous character is number("五"(five)) and next character must be "之"(of somebody).

And the class CMA_WType uses rules to combine special strings to identify the type of the word.

## 4.6 Post-Processing

This processing is invoked before segmenting using MaxEnt Model.

The post processing use the forward direction maximum matching to combine the successive words if necessary. The basic unit for the post-processing is the words from the segmenting using MaxEnt Model. And those word wouldn't be divided into small parts in the post-processing.

For example, one segmented result is A/B/C/D/E/F/G (A to F represents a word), and words AB, ABCG, CDE exist in the dictionary. The steps of Post-Processing for this example are:

1. Search via VTrie, start with A, moreLong is true (dictionary exists words like A*) and exists is false (word A not exists). If the moreLong is true, search via VTrie will continue based on the previous result (that is, search nodes just under A in the VTrie).

2. For B, moreLong is true and exists is true (AB is in the dictionary). Record longest existent word found, and this step is recording AB. As mention above, continue to search based on the result of the B.

3. For C, moreLong is true and exists is false. Thus continue to search but no record ABC hear.

4. For D, moreLong is false, thus it is time to find the best word find from the previous several steps. The best word here is the longest found word record so far and AB is for this case. Thus, the new search via VTrie will begin with C (just following AB).

5. For C and D, moreLong is true. For E, moreLong is false but exists in the dictionary, thus CDE will combined as one word and search begins with F.

6. For F, moreLong is false, no matter whether exists in the dictionary( as it is first search via VTrie), F would be regarded as one word and next search begin with G.

7. And Continues with all the rules mentioned above.

# Chapter 5

# Experiments

## 5.1 Testing environment

Table 5.1: Computer Environment

| | |
|---|---|
| Platform | Fedora 8.04 Kernel Linux 2.6.24-21-generic |
| Memory | 3.7GB |
| CPU | Double Intel(R) Core(TM) 2 Duo CPU E6550 @ 2.33GHz |

## 5.2 Dataset

The Test Datasets include two dataset: PFR (From *PeopleDaily* Newspapers) is for the testing while development, and CTB is for the final test. For each dataset, the training set and test set extracted from the dataset randomly.

Table 5.2: Dataset Statistics

| Dataset | #Training Set | #Test Set | OOV rate |
|---|---|---|---|
| PFR | 5.5m | 502k | 0.027 |
| CTB | 1.4m | 173k | — |

The size of the Training Set and Test Set is the size of the raw text, that is, the size doesn't include the pos. For the Training Set with POS tagged, the size is 9.7m for PFR and 3.1m for the CTB.

## 5.3 Experiment Result

The segmentation and pos tagging results are shown in table 6.3.

Table 5.3: Segmentation And POS Tagging Result

| Corpus | R | P | F | POS Accuracy | Execute Time |
|---|---|---|---|---|---|
| PFR | 0.947 | 0.960 | 0.953 | 0.965 | 3.9s |
| CTB(1) | 0.900 | 0.899 | 0.899 | 0.901 | 1.31s |
| CTB(2) | 0.925 | 0.954 | 0.939 | 0.934 | 1.32s |

In the Table 6.3, the meanings of columns are:

- $R$(Recall) is defined as the number of correctly segmented words divided by the total number of words in the gold standard.
- $P$(Precision) is defined as the number of correctly segmented words divided by the total number of words in our segmentation result.
- $F$(F-score) is defined as $F = \frac{2*R*P}{R+P}$
- $POSAccuracy$ is defined as the $POS_{accuracy} = \frac{Correct\ Tagged\ POS}{Total\ Correct\ POS}$
- $ExecuteTime$ is the time used to execute the segmentation and POS tagging.

From the experiment result, the PFR gains highest scores is because some adjustments are used while developing with the PFR. Hence, CTB test result represents more general purpose.

The testing of the CTB include two parts (labeled as CTB(1) and CTB(2)). The only difference between them is that the system dictionary (include the word and POS taggers) in the CTB(2) includes all the words in the Training Set while CTB(1) does not. The only effect of the system dictionary is in the post-processing of the segmentation (combine the successive words if necessary). From the result, the quality of the dictionary obviously affects the segmentation detail. In the CTB(2), the precision achieves the goal. As the dictionary for CTB training dataset is limited (analysed by the error segmentation statistics, lots of frequent-used words are not included). So the proper estimation for the precision of the CTB is 94%.

If the segmentation procedure is without Post-Processing (see section 4.6) of the word segmentation, the precision for the PFR and CTB (CTB(1) and CTB(2) are the same in this case) are 84.2% and 81.2% respectively, and it proves that the Post-Processing is efficient for the precision of the word segmentation.

More detail about error segmentation would be included in the "Detail of Error Segmentation" section.

## 5.4 Wisenut QC's Experiment

The Test Corpus used by the Wisenut QC is from the SIGHAN[1].

Table 5.4: Test Corpus Information

| Name | Sentences | Words |
|---|---|---|
| AS (Academia SINICA) | 14,432 | 122,610 |
| CITYU (Hong Kong City University) | 1,492 | 40,936 |
| MSR (Microsoft Research) | 3,985 | 106,873 |
| PKU (Beijing University) | 1,944 | 104,372 |

According to above table, the CITYU test corpora is much smaller than others, and AS test corpora has many short sentences.

The test results are:

| Model | Evaluation Unit | Test Corpus | | | |
|---|---|---|---|---|---|
| | | AS | CITYU | MSR | PKU |
| AS | P | 0.940 | 0.926 | 0.913 | 0.929 |
| | R | 0.962 | 0.944 | 0.945 | 0.937 |
| | F | 0.951 | 0.935 | 0.929 | 0.933 |
| | SA | 0.767 | 0.394 | 0.372 | 0.322 |
| CITYU | P | 0.931 | 0.938 | 0.912 | 0.919 |
| | R | 0.954 | 0.960 | 0.927 | 0.934 |
| | F | 0.942 | 0.950 | 0.927 | 0.926 |
| | SA | 0.737 | 0.471 | 0.355 | 0.290 |
| MSR | P | 0.919 | 0.912 | 0.933 | 0.935 |
| | R | 0.938 | 0.927 | 0.959 | 0.934 |
| | F | 0.928 | 0.919 | 0.946 | 0.935 |
| | SA | 0.690 | 0.328 | 0.452 | 0.275 |
| PKU | P | 0.918 | 0.914 | 0.923 | 0.953 |
| | R | 0.940 | 0.931 | 0.953 | 0.962 |
| | F | 0.929 | 0.922 | 0.938 | 0.958 |
| | SA | 0.690 | 0.349 | 0.409 | 0.428 |
| ICWB | P | 0.939 | 0.928 | 0.929 | 0.940 |
| | R | 0.961 | 0.945 | 0.957 | 0.944 |
| | F | 0.950 | 0.936 | 0.943 | 0.942 |
| | SA | 0.764 | 0.395 | 0.441 | 0.333 |

[1] http://www.sighan.org/bakeoff2005/

In the 'Evaluation Unit' column, $SA$ indicates the Sentences Accuracy, and $SA = \frac{True\_Sentences}{Total\_Sentences}$

There are five models, AS, CITYU, MSR and PKU models are trained from their associated training corpus separately, and them only contains Simplified or Traditional Chinese.

ICWB models is so-called general-purpose model, its training corpus is from the combination of all the four training corpus, therefor ICWB supports both Simplified and Traditional Chinese. As the limitation of memory, training ICWB model has to set feature cutoff value larger, thus the expression occurs less than 10 times would not be recorded in the model.

The dictionaries are mainly from the dictionary of SIGHAN directly, and all the five models share the same dictionaries.

According to the Wisenut QC's result, The model for associated test copora (like AS model for AS test corpora) gain the best score. For such associated pairs, compare the F-Score, expect the MSR model (0.946), others have passed 0.950, and highest one is PKU (0.958).

Then compare the general-purpose corpus with associated pairs, almost gain the same score with the MS and MSR model, but lower about 1.5% with CITYU and PKU. It can be concluded that expressions in CITYU and PKU are little complicated (as some would be ignored in the training).

For $SA$, it is mainly affected by the average length of sentences, and it is just from reference here.

## 5.5 Detail of Error Segmentation

The example for this section is segmentation result the CTB(1).

The name of the sub-section is the error division cases, such as "ABC to A/BC", where the former case (ABC) is the correct case and the latter one is the wrong case (A/BC).

### 5.5.1 ABC to A/BC

This situations mainly because the dictionary didn't contains the word ABC. In the column $DictionaryExists$, the $N$ in the braces next to a word indicates that word doesn't exist in the dictionary while $Y$ exists.

| Correct Division | Error Division | Dictionary Exists |
|---|---|---|
| 法规性 | 法/规性 | 法规性(N) 法(N) 规性(N) |
| 资质证 | 资质/证 | 资质证(N) 资质(Y) 证(N) |

### 5.5.2 A/BC to ABC

This situation occurs for two reasons, one is error segmentation when using MaxEnt model and the other is error combination in the post-processing.

| No. | Correct Division | Error Division | Dictionary Exists |
|---|---|---|---|
| 1 | 当/到 | 当到 | 当(Y) 到(Y) 当到(N) |
| 2 | 马/上 | 马上 | 马(Y) 上(Y) 马上(Y) |
| 3 | 全/国 | 全国 | 全(Y) 国(Y) 全国(Y) |

Case 1th is the error segmentation from MaxEnt Model.

Case 2nd is the error combination. The whole sentence is "他(He) 从(From) 马(Horse) 上(Up) 掉(Drop) 下来(Down)", the complete English sentence is "He drop down from the horse". When combined as 马上(immediately), it represents totally different meanings. Hence case 2th is the error segmentation. Case 2nd is rarely in the reality. And the longer a word's length, the less possibility error segmentation.

But for case 3, it is correct division for both two cases (that is, both two situations could be found in the corpus). This case is because some inconsistent segmentation guidelines. This case depends on the quality of dataset. The frequent-used words are mostly likely seen in this case.

### 5.5.3 AB/C to A/BC

These situations are crossing ambiguities and are most complicated. And mainly because the error segmentation using MaxEnt model.

| No. | Correct Division | Error Division | Dictionary Exists |
|---|---|---|---|
| 1 | 甲肝/流行 | 甲/肝流/行 | 甲肝(N) 流行(Y) 甲(N) 肝流(N) 行(N) |
| 2 | 其/销售 | 其销/售 | 其(Y) 销售(N) 其销(N) 售(N) |

As shown, most of words in these situation doesn't in the dictionary. Morever, these are not words in the reality. The MaxEnt model focuses on the location of the character in a word, and do not care whether the words exists.

## 5.6 Entity Detection Error

The entities include numbers (in Chinese form, like ”二十二 (twenty two)”), people name, organization names (Like ”中华人民共和国 (People’s Republic of China)”), place names (Like ”黄埔江 (Huangpu River)”) and so on.

The CMA don’t have special knowledge to dealt with most cases of the entities. The limited improvement can be done when detecting numbers.

# Chapter 6

# Conclusion

## 6.1 Quality and Performance

The quality of CMAC is evaluated by authors on some corpus. In case of corpus CTB, it achieves F-Score of word segmentation as 0.899 to 0.939 (see section 5.3), and in case of corpus PFR, it achieves F-score as 0.953.

And from the Wisenut QC's Experiment (corpus are from SIGHAN), for the associated training corpora and test copora, the F-Score varies from 0.946 to 0.958. And for the general-purpose, it is 0.945. This result is similar with the estimation result of CTB (0.940).

The execution time for the word segmentation is about 7.30 seconds for 1 Megabyte text.

The Post-Processing (see section 4.6) of the word segmentation is proved efficient for the precision of CMAC's word segmentation.

The evaluation result shows that CMA is capable in realistic usage of Chinese morphological analysis.

## 6.2 What Affects the Quality

From the experiment result, we could see that the quality is influenced by several factors below.

### 6.2.1 Model Training

The first factor is the training set as the CMAC is statistical model based segmentation. It is impossible to include all the situations in the training set, but it is suggested that dataset is about 10 to 20 Megabyte and is gained from the real data set randomly. And for the training process, some parameters can be higher to fetch the meaningful features. Also, it is better if the training set and testing set are segmented under some principles.

### 6.2.2 Feature Set

The second factor is the feature set that MaxEnt model used. From the experiments, it performs better when feature set varies under different context. For example, if the current character is number (in Chinese and English form), it only cares about whether the previous character is number or letter. If the previous character is number, the current character of course should be possible (possible here indicates the next character maybe number too) ending of the previous character. Suppose the feature set is the same, and features likes $C_{-2}$ and $C_{-2,-1}$ are used to estimate the possibility (Beginning or non-beginning of a word) of the current character, and it may result in current character is the beginning of a word, which is a error segmentation.

### 6.2.3 Dictionary

Improper words in the dictionaries may result in segmentation errors, like the number and its unit are a word in the dictionary. Thus the quality of Dictionary is also importance.

## 6.3 Future work to do

The future work to do is basically two factors which affect the quality of the CMAC.

Research for the proper feature set under the different context. And it should contains more than two dataset.

Update some entity detection rules. Like some characters can be a part of number when behind the number (”余” itself is not a number and in the ”两千余” it is).

Regarding the realistic usage, and comparing with some commercial CMAs, we should select the datasets basing on the applying purpose of the CMA

(For example, for Science articles purpose and Newspapers articles purpose are quite different in the expression forms). Of course it includes the general purpose (like Google).

# Chapter 7

# Guidance to use the library

## 7.1 Interface description

The C++ API of the library is described below:

Class CMA_Factory creates instances for CMA, its methods below create instances of CMA_Factory, Analyzer and Knowledge, which are used in the morphological analysis.

```
static CMA_Factory* instance();
virtual Analyzer* createAnalyzer() = 0;
virtual Knowledge* createKnowledge() = 0;
```

Class Knowledge manages the linguistic information, its methods below load files of system dictionary, user dictionary, stop-word dictionary, and statistical model.

```
virtual int loadSystemDict(const char* binFileName) = 0;
virtual int loadUserDict(const char* fileName) = 0;
virtual int loadStopWordDict(const char* fileName) = 0;
virtual int loadStatModel(const char* binFileName) = 0;
```

Class Analyzer executes the morphological analysis, its methods below execute the morphological analysis based on a sentence, a paragraph and a file separately.

```
virtual void setKnowledge(Knowledge* pKnowledge) = 0;
void setOption(OptionType nOption, double nValue);

virtual int runWithSentence(Sentence& sentence) = 0;
virtual const char* runWithString(const char* inStr) = 0;
virtual int runWithStream(const char* inFileName, const char* outFileName) = 0;
```

Class Sentence saves the analysis results, so that the n-best or one-best results could be accessed.

```
void setString(const char* pString);
const char* getString(void) const;
int getListSize(void) const;
int getCount(int nPos) const;
const char* getLexicon(int nPos, int nIdx) const;
int getPOS(int nPos, int nIdx) const;
const char* getStrPOS(int nPos, int nIdx) const;
double getScore(int nPos) const;
int getOneBestIndex(void) const;
```

## 7.2 How to use the interface

To use the library, follow the following steps:

1. Include the header files.

```
#include "cma_factory.h"
#include "analyzer.h"
#include "knowledge.h"
#include "sentence.h"

using namespace cma;
```

2. Use the name space of the library.

```
using namespace cma;
```

3. Call the interfaces and handle the result.

```
// create instances
CMA_Factory* factory = CMA_Factory::instance();
Analyzer* analyzer = factory->createAnalyzer();
Knowledge* knowledge = factory->createKnowledge();

//It is suggested to set encoding after crate the Knowledge. Another supported encode type is big5.
knowledge->setEncodeType(Knowledge::ENCODE_TYPE_GB2312);

// If use /dir1/dir2/cate as the cateFile in the trainer (see section "Run the Trainer"
// in this chapter), the parameter for loadStatModel is /dir1/dir2/cate-poc and for the
// loadPOSModel is /dir1/dir2/cate.

// load POC statistical model
knowledge->loadStatModel("...");
// loadPOSModel has to be invoked before loading XXX Dictionaries
knowledge->loadPOSModel("...");

// (optional) load dictionaries
knowledge->loadSystemDict("...");
```

```
knowledge->loadUserDict("...");
knowledge->loadStopWordDict("...");

// (optional) if POS tagging is not needed, call the function below to turn off the analysis and
// output for POS tagging, so that large execution time could be saved when execute
// Analyzer::runWithSentence(), Analyzer::runWithString(), Analyzer::runWithStream().
analyzer->setOption(Analyzer::OPTION_TYPE_POS_TAGGING, 0);

// (optional) set the number of N-best results,
// if this function is not called, one-best analysis is performed defaultly on
// Analyzer::runWithSentence().
analyzer->setOption(Analyzer::OPTION_TYPE_NBEST, 5);

// set knowledge
analyzer->setKnowledge(knowledge);

// 1. analyze a paragraph
const char* result = analyzer->runWithString("...");
...

// 2. analyze a file
analyzer->runWithStream("...", "...");

// 3. split paragraphs into sentences
string line;
vector<Sentence> sentVec;
while(getline(cin, line)) // get paragraph string from standard input
{
    sentVec.clear(); // remove previous sentences
    analyzer->splitSentence(line.c_str(), sentVec);
    for(size_t i=0; i<sentVec.size(); ++i)
    {
        analyzer->runWithSentence(sentVec[i]); // analyze each sentence
        ...
    }
}

// destroy instances
delete knowledge;
delete analyzer;
```

## 7.3 Compile the Source

1. Using shell, go to the project root directory.
2. Type "mkdir build".
3. Type "cd build".
4. Under linux, type "cmake ../source"; Under windows, run in the msys, type "cmake -G 'Unix Makefiles' ../source ".
5. Finally Type "make" to compile all the source

If the external program uses the library, simply add all the header files in the *include* directory under the project root directory, and add the lib/libcmac.a into the library path.

## 7.4 Run the Trainer

The dataset have to be trained by the Trainer. The Trainer is a executable file with name camctrainer under directory bin.

The SYNOPSIS for the trainer is:

```
./cmactrainer mateFile cateFile [encoding] [posDelimiter]
```

The Description for the parameters:

- mateFile is the material file, it should be in the form word1/pos1 word2/pos2 word3/pos3 ...
- cateFile is the category file, there are several files are created after the training, and with cateFile as the prefix, prefix should contains both path and name, such /dir1/dir2/n1.
- encoding is the encoding of the mateFile, and gb2312 is the default encoding. Only support gb2312 and big5 now.
- posDelimiter is the delimiter between the word and the pos tag, like '/' and '_' and default is '/'.

Take "/dir1/dir2/cate" as the cateFile, after the training. The following files are created (All under directory /dir1/dir2):

1. cate.model is the POS statistical model file.
2. cate.pos is the all the POS gained from the training dataset.
3. cate.dic is the dictionary (include words and POS tags) gained from the training dataset. This file is plain text and should be loaded as user dictionary. To convert it to the system dictionary, use Knowledge::encodeSystemDict(const char* txtFileName, const char* binFileName), then the binFileName can be loaded as the system dictioanry.
4. cate-poc.model is the POC statistical model file.

All the files are required to run the program.

## 7.5 Run the Demo

After the training, you can run the demo to segment the file, The Demo is a executable file with name camcsegger under directory bin.

The SYNOPSIS for the demo is:

```
./cmacsegger cateFile inFile outFile [encoding] [posDelimiter]
```

The Description for the parameters:

- cateFile is the category file, there are several files are created after the training, and with cateFile as the prefix, prefix should contains both path and name, such /dir1/dir2/n1. This parameter is the same as the cateFile in the training process.
- inFile the input file.
- outFile the output file.
- encoding is the encoding of the mateFile, and gb2312 is the default encoding. Only support gb2312 and big5 now.
- posDelimiter is the delimiter between the word and the pos tag, like '/' and '_' and default is '/'.

The result with pos tagging can be found in the outFile.

## Appendix A

# Appendix - Project schedules and milestones

| | Milestone | Start | Finish | In Charge | Description | Status |
|---|---|---|---|---|---|---|
| 1 | Survey on the project | 2009-02-01 | 2009-02-13 | Vernkin | 1. Have a general overview of current Chinese Segmentation techniques and their differences. 2. Select a suitable one (Character-based) and do more research on it. | Finished |
| 2 | Design the Architecture of CMA | 2009-02-16 | 2009-02-20 | Vernkin | 1. Basen on WISE KMA Orange, design the Architecture for the MA Systems (CJK). 2. Moreover, design the common Architecture for the CMA. 3. Finally focus on the Architecture for the approach I selected. | Finished |
| 3 | Implement CMA and Unit Test | 2009-02-23 | 2009-03-25 | Vernkin | 1. Implement the Maximum Entropy Model. 2. Implement character-based Segmentation. 3. Integrate all the components into CMA. | Finished |
| 4 | Optimization the Segmentation Logic | 2009-03-26 | 2009-04-25 | Vernkin | 1. Try other features sets; 2. Optimize the code logic to reduce the execute time; 3. List out the error segmentations, classify the reasons and do extra post-segmentation optimization. | Finished |
| 5 | Wrap up the Project | 2009-04-26 | 2009-04-30 | Vernkin | 1. Finished the TR; 2. Review the code and comments. | Finished |
| 6 | QC Testing | 2009-08-31 | 2009-09-11 | Wisenut QC Team & Vernkin | 1. Corporate with QC Team to finish the testing. | Finished |
| 7 | Make the Project as a Product | 2009-09-14 | 2009-09-30 | Vernkin | 1. Organize the System Dictionary; 2. Add the context for some segmentation errors; 3. Optimize some source code; 4. Support Unicode and some project's invoke. | 35% Finished |

# Bibliography

[1] J.N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, Vol. 43:pp 1470–1480, 1972.

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

[3] Julia Hockenmaier and Chris Brew. Error-driven segmentation of chinese. *Communications of COLIPS*, 1(1):69–84, 1998.

[4] Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation, 2003.

[5] David Palmer. A trainable rule-based algorithm to word segmentation. *Proceedings of the 35th Annual Meeting of the Association of Computational Linguistics*, 1997.

[6] Nianwen Xue. Defining and automatically identifying words in chinese. *Ph.D. thesis, University of Delaware*, 2001.