Technical Report of *Language Analyzer* Comprehension

Zhongxia Li

December 31, 2010

Abstract

This document is a comprehensive technical report for *Language Analyzer*, including *Chinese Morphological Analyzer*.

Language Analyzer (*LA*) is a essential module for text processing in SF1-R system, it's goal is to convert the input text which was filled with words and characters into more consistent *index terms*. Index terms are the representation of the content of a document that are used for indexing and searching. *LA* provides various analyzing methods for text processing, the first pass of processing is *tokenization* in which the characters are split apart into basic categories, and then other analyzing methods are applied. For Chinese language, the Chinese segment processing for detecting words and other morphological analysis, such as part-of-speech (POS), will be applied.

This report is based on the technical reports ([1], [2] and [3]) and source code review of *LA*, *CMA* and *SF1-R*, also referred to few papers. This report also described another statistical model (CRF) that can be employed to language analyzer.

Changes

Date	Author	Notes
2010-12-17	Zhongxia Li	Initial version.
2010-12-24	Zhongxia Li	Add Chapter 5 (Chinese Morphological Analyzer).
2010-12-31	Zhongxia Li	Update Chapter 5 (refined, add section of CRF); add *References*.

Contents

Chapter 1

Overview

1.1 What Is Language Analyzer?

The *Language Analyzer* is an important component for text processing in SF1-R system, it aims to convert the input text which was filled with words and characters into more consistent *index terms* by applying various analyzing methods including morpheme analyzing. The index terms are the representation of the content of a document that are used for indexing and searching. Text preprocessing is an essential pre-stage for indexing and query for a search engine.

1.2 Terms of Output

The input text of *LA* maybe consist of alphabetic, numeric or specail characters, and more often Chinese characters. Generally, *LA* generates three categories of terms as outputs.

- *Raw Term*: These are the raw terms that completely represent the original raw text.
- *Primary Term*: Within the Raw Terms, the terms that consists either alphabetic or numeric characters fall into this category.
- *Secondary Term*: The terms that are analyzed from the Primary Terms by analyzers fall into this category.

Each term is assigned a word offset corresponding to the term's relative position in the original text. The offset of expanded primary and secondary terms are set based on the raw terms.

1.3 Architecture of Language Analyzer Modules

There are two main kinds of processors for *Language Analyzer* which are Tokenizer and Analyzers, the input text is first passed through the Tokenizer and then processed using other optional analyzers such as StemAnalyzer, NGramAnalyzer, ChineseAnalyzer, etc. The basic architecture of *Language Analyzer* modules for SF1-R is show as figure 1.1. LA Manager exposes interfaces to other modules of SF1-R.

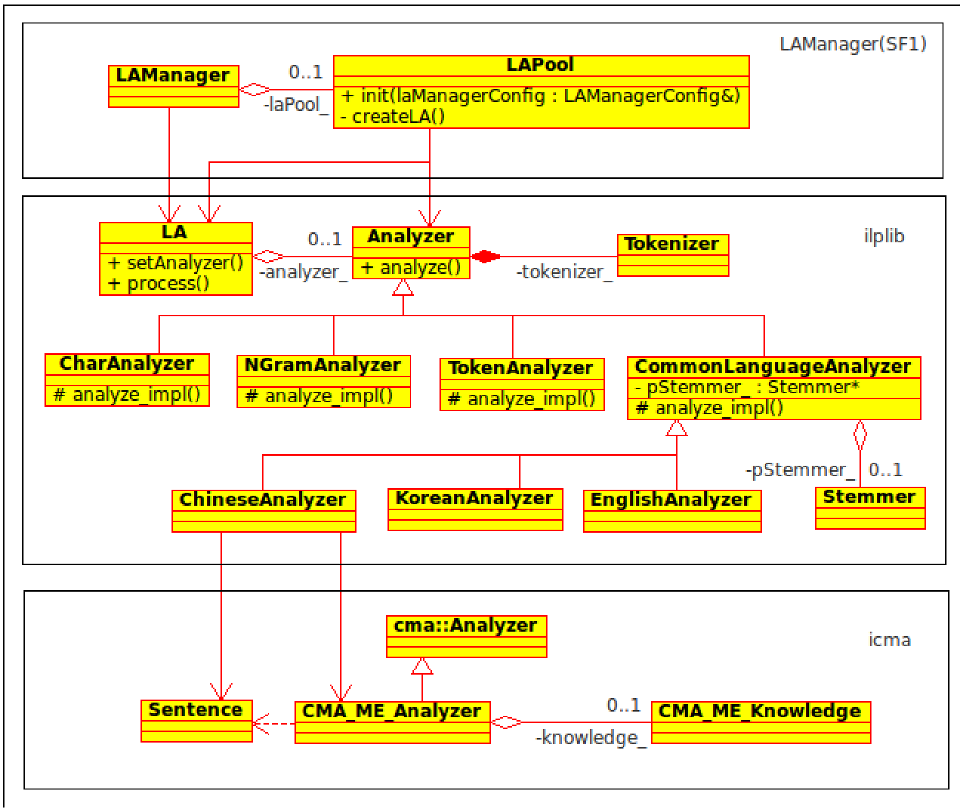

Figure 1.1: Basic Architecture of Language Analyzer Modules

For CJK family of languages, the key problem is word segmentation, where the breaks corresponding to words or terms must be identified in the continuous sequence of characters. The *Chinese Morphological Analyzer* (*CMA*) module implemented several Chinese segmentation algorithms which are optional, including dictionary based algorithms and char-based tagging approach. The *Maximum Entropy Module* is employed for building Chinese segment tagger, POC (Position of Character), and building part-of-speech (POS) tagger.

Chapter 2

Tokenizer

2.1 Introduction

The tokenizing is done by categorizing characters into two basic categories: delimters (white space and special characters by default) and content characters (alphabetic, numeric and other language characters). Continuous characters in the same category are grouped into a single term.

Each term is assigned with a word offsets which shows it's relative position in the original text. Take input text "It's like A@B" for example, the output terms and their offsets are show in table 2.1.

Table 2.1: Raw Term Example

TERM	WORD OFFSET
It	0
'	1
s	2
like	3
A	4
@	5
B	6

Notice that the white space doesn't occupy a position. So the white spaces that are between "s" and "like" and between "like" and "A" are ignored.

As we can see in table 2.1, these terms are Raw Terms. The Raw Terms are used to rebuild the original text, since they exactly represented the original text when put together. The Primary Terms, in this example, are 'It' 's' 'like' 'A' and 'B' whose offsets are based on Raw Terms.

2.2 Options for Tokenizer

Table 2.1 showed an output sequence of terms for our example, however, there are several options for us to decide how the specail characters(such as ['], [@]) will be parsed. The options are: *Allow*, *Divide* and *Unite*, as described in the table below.

Method	**Description**
Allow	The characters set as *allow* will be removed from the list of delimiters. E.g. "A@B" => "A@B"
Divide	The characters will be set to be a delimiter. E.g. "A@B" => "A", "B"
Unite	The character will be used to concatenate the two adjacent tokens on either side of the character. E.g. "A@B" => "AB"

The following shows the primary terms when different options are set for Tokenizer.

- Allow: The character is recognized as an content character.

TERM	**WORD OFFSET**
It's	0
like	1
A@B	2

- Divide: The character is recognized as a delimiter. Acts like the default Tokenizer.

TERM	**WORD OFFSET**
It	0
s	2
like	3
A	4
B	6

- Unite: The character is used to concatenate two terms on both side.

TERM	WORD OFFSET
Its	0
like	1
AB	2

2.3 Setting Options in Configuration File

We can set the options for Tokenizing in the configration file of SF1-R as below.

```
<Tokenizing>
 <Tokenizer id="tok_divide" method="divide" value="@#$" code=""/>
 <Tokenizer id="tok_unite" method="unite" value="/" code=""/>
</Tokenizing>
```

In the *Tokenizing* Element of the XML file of configration, we can edit the attributes of *Tokenizer* elements to set options. The *id* attribute is the name of Tokenizer, the *method* attribute is the option we chose, the *value* attribute indecates what character(s) to apply this method, the character(s) also can be represented as UCS2 *code* in *code* attribute.

Chapter 3

Analyzer

3.1 Introduction

When text are tokenized into a list of Raw Terms and Primary Terms, the Primary Terms can be further processed by several Filters and Analyzers. The terms created during this process are Secondary Terms. Secondary Terms have the same word offsets as the Primary Terms that they expanded from.

Various analyzing methods are available to be used in SF1-R. Some of these methods are character-based, like token, ngram, and matrix, while some are language based methods. Analyzers will process the tokens returned from the Tokenizer and extract terms from it. These terms are used for indexing and searching.

3.2 Configuration and Options for Analyzer

We can define (set) different Analyzer Methods by setting different options for each Method, the defined Methods can be applied in SF1-R. The following are the core options that are required for each `<Method>` element (except those that marked with "optional"). Other options vary depending on the *analysis*.

Element	Attribute	Description
Method	id	The name of the analyzer instance. The ID is used like variables in `<Property>` configurations in Collections setttings, along with Tokenizer settings.

	analysis	This attribute decides the analyzer type. There are two analysis categories, one in which is language independent, which are *token*, *ngram*, and emphmatrix. The other category is language dependent, it includes *English*, *Chinese*, *Korean* and other languages. The analysis attribute is followed by options that correspond to the option.
	casesensitive (optional)	Default is "yes". This attribute is used to set the case-sensitivity of the document field property. If the setting is turned on, the indexing and searching on the document will be done case sensitivity.
	idxflag (optional)	Default is *all*. Indexing flag to indicates which type of terms should return in the indexing. It has four values: 1) *all* returns both primary and second terms; 2) *prime* returns only primary terms; 3) *second* returns only secondary terms; 4) *none* returns neither primary nor second terms. *Primary Term* is tokenized by tokenizer and *Secondary Term* is analyzed by Specific Language Analyzer basing on the *Primary Term*.
	schflag (optional)	Default is *second*. Searching flag to indicates which type of terms should return in the searching. The detail see *idxflag*.

The following shows an example of Analyzer Methods defined in configuration file of SF1-R.

```
<LanguageAnalyzer dictionarypath="...">
  <Method id="la_token" analysis="token"/>
  <Method id="la_ngram" analysis="ngram" min="2" max="3" maxno↩
      ="2194967296" apart="n" idxflag="second" schflag="second"/>
  <Method id="inner_la_korall_mia" analysis="korean" casesensitive="↩
      yes" >
    <settings mode="label" option="R1H-S-" specialchar="#" ↩
        dictionarypath=""/>
  </Method>
  <Method id="la_mia" analysis="multilang" advoption="default,↩
      inner_la_korall_mia;cn,char" casesensitive="yes" lower="no"/>
  <Method id="inner_la_cnall_sia" analysis="chinese" casesensitive="↩
      yes" >
    <settings mode="label" option="R+H+S+T3" specialchar="#" ↩
        dictionarypath="/home/zhongxia/codebase/icma/db/icwb/utf8"/>
  </Method>
  <Method id="la_sia" analysis="multilang" advoption="default,↩
      inner_la_korall_sia;cn,ma,inner_la_cnall_sia"/>
</LanguageAnalyzer>
```

Chapter 4

Usage Of Language Analyzer

4.1 LA Configuration for Indexing Collection

For each *Property* of *Collection*, we can set *Indexing* Element to indicate which Analyzer or Tokenizer(s) will be applied to parse the *Property* content. As showed below, is an example of cofiguration for data collection *ChnWiki* indexed by SF1-R.

```
<Collection name="ChnWiki" encoding="UTF-8" rankingmodel="ranker1"↩
     wildcardtype="unigram" generalclassifierlanguage="chinese">
  <DocumentSchema>
    <Property name="DOCID" type="string" index="no" />
    <Property name="DATE" type="string" index="no" />
    <Property name="Title" type="string" index="yes" mining="yes">
  <Indexing filter="no" analyzer="la_sia" tokenizer="" rankweight↩
      ="heavy"/>
    </Property>

    <Property name="Content" type="string" index="yes" mining="yes↩
        ">
  <Display length="160" snippet="yes" summary="no" >
    <settings summarynum="2"/>
  </Display>
  <Indexing filter="no" analyzer="la_sia" tokenizer="tok_divide, ↩
      tok_unite" rankweight="normal"/>
    </Property>
  </DocumentSchema>
</Collection>
```

4.2 Application Interface

The LAManager exposes interfaces of *Language Analyzer* to other modules of SF1-R, associated with the configuration settings described above. It's easy to use, initialize LA and parse input text with specified configuration (*AnalysisInfo*), the output term list will be returned.

Application Interfaces:

```
bool LAManager::getTermList(
  const izenelib::util::UString & text,
  const AnalysisInfo& analysisInfo,
  la::TermList& termList );

bool LAPool::init(const sf1v5::LAManagerConfig & laManagerConfig);
```

Usage:

```
LAManagerConfig config;
AnalysisInfo analysisInfo;
String input_text = "content to be parsed";

// setting config

LAPool::getInstance()->init( config );
LAManager laMgr_ = new LAManager();

la::TermList& termList; // output
laMgr_->getTermList(input_text, analysisInfo, termList )
```

Chapter 5

Chinese Morphological Analyzer

5.1 Introduction

For CJK family of languages, a key problem is word segmentation, where the breaks corresponding to words or terms must be identified in the continuous sequence of characters. The *Chinese Morphological Analyzer* (*CMA*) module implemented several Chinese segmentation algorithms which are optional, including dictionary based algorithms and char-based tagging approach. The *Maximum Entropy Module* is employed to build Chinese segment tagging, POC (Position of Character), and part-of-speech (POS) tagger.

There are two main difficulties in Chinese segmentation, segmentation ambiguities and out-of-vocabulary (OOV) words. Practical results show that performance of statistic segmentation system outperforms that of handcrafted 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 applied 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, tags that indicate the position of Chinese character within a word.

5.2 Part-Of-Speech Tagging

This section describea the details of Part-Of-Speech Tagging by using Maximum Entropy Modeling.

5.2.1 Maximum Entropy Model for POS Tagging

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:

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

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:

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

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.

5.2.2 Feature Selection

We select features used in the tagging model by applying a set of feature templates to the training data. The features are (index 0 is current word, negative value is previous index while positive index is latter index):

1. A Single Word: $(C_i),\ i \in [-2, 1]$;
2. Two Adjacent words: $(C_i, C_{i+1}),\ i \in [-2, 0]$;
3. The previous and latter words: (C_{-1}, C_1);
4. The previous, latter and current words: C_{-1}, C_0, C_1;
5. The previous two POS: T_{-2}, T_{-1}.

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

The Sentence for the example:

日文(Japanese)/N 章鱼(Octopus)/N 怎么(How to)/R 说(Say)/V ?/W,

In English: How to say octopus in Japanese?

(the words in the braces are English meanings for Chinese words, doesn't appear in the original sentence).

Bound indicates that value is no available. The following is the example of features:

For word 日文 (Fully):
C_{-2} = Bonud
C_{-1} = Bound
C_0 = 日文
C_1 = 章鱼
$C_{-2,-1}$ = Bound, Bound
$C_{-1,0}$ = Bound, 日文
$C_{0,1}$ = 日文, 章鱼
$C_{-1,1}$ = Bound, 日文
$C_{-1,0,1}$ = Bound, 日文, 章鱼
$T_{-2,-1}$ = Bound, Bound
For word 文章 (Partial):
C_{-0} = 文章
C_{-1} = 日文

5.3 Chinese Word Segmentation

5.3.1 Dictionary Based Approach

Since the problem of automatic Chinese word segmentation being proposed, many researchers have being made efforts to find solutions for it. In the early decades, many dictionary based algorithms has been developed for Chinese Word Segmentation, such as forward (backword) maximum matching, bilateral scanning, etc. The dictionary base algorithms for Chinese Word Segmentation is totally dictionary based matching, the algorithms implemented in *CMA* are Forward Maximum Matching and Forwards Minimum Matching algorithm. The Tire-Tree has been selected as an appropriate data structure to construct word dictionary.

As example, for Forwards Minimum Matching based approach, when scanning the input text, the shortest word matched according the dictionary will be selected. Suppose the input text is ABCDEF (A to H represents a character respectively), and the dictionary contains AB, ABC, DEF. After processing by minimum matching algorithm, the output segments will be AB, C, DEF.

The main goal to apply word segmentation in a searche engine is to generate index terms, as user's keyword should be match the indexed terms as better as possible. All the ambiguities should be contained in the result.

5.3.2 Character Based Tagging

This section will introduce character-based Chinese word segmentation by applying the *Maximum Entropy Model*. 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 two tags, B or E depending on its position within a word. It is tagged B 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 tagged with E in other cases (In the non-beginning position of a word). We call such tags as position-of-character (POC) tags to differentiate them from the more familiar part-of-speech (POS) tags. For example, the manually segmented string in (a) will be tagged as (b):

Example Senetence:

(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.

In the next section, the details of applying *Maximum Entropy Model* to POC tagging will be introduced.

5.3.3 Maximum Entropy Approach for POC Tagging

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

$$p(h,t) = \pi\mu \prod_{j=1}^{k} \alpha_j^{f_j(h,t)} \tag{5.1}$$

Where π 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 α_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:

$$L(P) = \prod_{i=1}^{n} P(h_i, t_i) = \prod_{i=1}^{n} \pi\mu \prod_{j=1}^{k} \alpha_j^{f_j(h_i,t_i)} \tag{5.2}$$

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 (1) to (3) for common characters and (1) for special characters represent character features while (2) for special characters represents tag features. $C_{-2}...C_1$ are characters and $T_{-1}...T_1$ are POC tags.

There are two groups of feature templates basing on the context. The C represents character array, and $C[0]$ is the current character, $C[1]$ is the previous character and so on. The T represents the characters' types array (type see section-**??**), and $T[0]$ is the type of the current character.

The default feature set(eight features) is:

1. The single character group: $(C_i),\ i \in [-2, 1]$;
2. The adjacent two characters group: $(C_i, C_{i+1}),\ i \in [-2, 0]$;
3. The previous and next characters (C_{-1}, C_1).

The feature set for special characters (like digits, letters) (four features) is:

1. The next character C_1;
2. The Single Type group: $(T_i)\ i \in [-1, 1]$.

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,

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

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

The feature templates encode three types of contexts. First, features based on the current and surrounding characters are extracted. Given a character in a sentence, this model will look at the current character, the previous two and next characters. For example, if the current character is 们 (plural marker), it is very likely that it will occur in the non-begin position of a word, thus receiving the tag E. On the other hand, for other characters, they might be equally likely to appear on the beginning of a word. 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 B if the next character is 护 ("protect"). However, if the previous character is 热 ("warm"), then it should perhaps be tagged E. Second, for special characters, features based on the previous tags (2) 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. When the training is completed, 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

$$P(t_1, ..., t_n \mid C_1, ..., C_n) = \prod_{i=1}^{n} P(t_i \mid h_i) \tag{5.3}$$

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

$$P(t \mid h) = \frac{p(h,t)}{\sum_{t' \in T}^{P(h,t')}} \tag{5.4}$$

5.4 Conditional Random Field

Many statistical machine learning methods has been employed to natural language processing, statistical probabilistic models such as HMM, ME(applied in 5.2 and 5.3.3), HEMM has being used for language processing tasks. However, MEMMs and other non-generative finite-state models based on next-state classifiers, such as discriminative Markov models (Bottou, 1991), share a weakness we call here the *label bias problem*: the transitions leaving a given state compete only against each other, rather than against all other transitions in the model.

(Lafferty et al., 2001 [4]) introduces conditional random fields (CRFs), a sequence modeling framework that has all the advantages of MEMMs but also solves the label bias problem in a principled way.

Conditional random field (CRF) is a statistical sequence modeling framework first introduced into language processing in (Lafferty et al., 2001 [4]). In (Peng et al., 2004 [5]), this framework is used for Chinese word segmentation by treating it as a binary decision task, such that each Chinese character is labeled either as the beginning of a word or not.

Bibliography

[1] Technical report of la manager. *iZENEsoft*, April 2010.

[2] Technical report of chinese morphological analyzer. *iZENEsoft*, August 2010.

[3] Search formula-1 administrator's manual. *iZENEsoft*, pages 40–47, April 2010.

[4] John D. Lafferty, Andrew McCallum, and Fernando C.N.Pereira. Conditional random fields: Proba- bilistic models for segmenting and labeling sequence data. *Proceedings of the Eighteenth In- ternational Conference on Machine Learning*, 2001.

[5] Fuchun Peng, Fangfang Feng, and Andrew McCallum. Chinese segmentation and new word detection using conditional random fields. *COLING 2004*, 2004.

[6] Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 1996.

[7] Hai Zhao, Chang-Ning Huang, and Mu Li. An improved chinese word segmentation system with conditional random field. *The Third SIGHAN Chinese Language Processing Bakeoff*, 2006.