Chinese Sentence Tokenization Using a Word Classifier

Benjamin Bercovitz
Stanford University
CS229
berco@cs.stanford.edu

ABSTRACT

In this paper, we explore a Chinese sentence tokenizer built using a word classifier. In contrast to the state of the art conditional random field approaches, this one is simple to implement and easy to train. The work is broken down into two pieces: the sentence maximizer makes guesses over a large number of sentence tokenization candidates and scores each one. The highest scored sentence tokenization is returned by the algorithm. The word classifier tells the sentence maximizer about whether a particular character sequence is a word or not. The obvious word classifier for words that were seen in the training set is a simple dictionary. We have gone further by building a new word classifier out of character- and word-level features using logistic regression. The performance of the new word classifier is promising, but the sentence maximization objective will have to be constrained more to get meaningful practical results by reducing the impact of false positives.

1. INTRODUCTION

A first step in applying information retrieval techniques to a corpus of text is tokenization, the process of breaking down blocks of text in the form of documents into single words that span the vocabulary of the search terms that users may wish to query for. In many languages, tokenization is trivial or fairly straightforward. For example in English, simply removing punctuation and splitting by whitespace does a fairly good job. Some European languages benefit from a form of processing called stemming that groups related words together into a single index term. However, tokenizing Chinese is hard because it is written without whitespace between the words. Typically, a Chinese word may contain from one to four characters, and sometimes contains eight or more when transliterating foreign words and names. Human Chinese readers, of course, can easily distinguish the word boundaries anyways based on context, but basic tokenization schemes do not comprehend these semantic boundaries.

There are a variety of approaches that have been taken, with everything from part-of-speech trees to conditional random fields and Markov chains involved. In contrast to implementing something complicated with a rich theoretical model, the goal of this work is to build a good-performing tokenizer based on a simple model that uses a binary word classifier.

2. PROBLEM

Given a Chinese sentence, the ideal tokenization algorithm will be able to produce the same tokenization as a human Chinese reader. Formally, we want

$$tokenize(S) = W, \text{ where}$$
$$W = \{w_1, w_2, \cdots, w_n\}, w \in \{\Sigma^*\} \text{ and } S = "w_1\, w_2\, w_3\ \cdots\ w_n"$$

To simplify our problem to dealing with character sequences only, we will tokenize the sentence into phrases by punctuation before starting. This is a trivial tokenization, searching for the set of predefined Unicode symbols that are defined as punctuation for the Chinese code page. Thus evaluation will be done phrase by phrase, even though we will loosely refer to these phrases as sentences. The distinction is not very important in our context, because sentences are just strings of words in our definition.

3. MODELS

3.1 Sentence Likelihood Maximization

The intuition behind this maximization objective is very simple. True sentences are made up of words. Each token has some probability of being a word. The probability of being a sentence is thus proportional to the product of the each token being a word.

We will first introduce a Naive Bayes assumption for sentences. The sentence likelihood will not capture whether a sentence makes sense, nor even if it is grammatically correct (in the linguistic sense). Rather, we will assume that a sentence is any string of valid words, and that each word is independent. This implies,

$$P(S|w_1, w_2, \cdots w_n) = P(S) \prod_{w_i \in W} P(w_i)$$

We already know the classification result: all the sentences we are provided in the testing set are true sentences. We are looking for the sets of words that gives a positive sentence classification.

$$tokenize(S) = \arg\max_W \prod_{w_i \in W} P(w_i)$$

3.2 Word Maximization as an Estimate

The intuition behind word maximization is very similar to sentence likelihood maximization, but it is reformulated as features that are word-based instead of sentence-based. Each word is classified by the algorithm and if the length-normalized sum of classifications is maximized, then so will the likelihood that the sentence is properly tokenized, under the Naive Bayes assumption for sentences. This is proposed because it is not clear how to find many features suitable for linear classifiers at the sentence level. Formally,

$$tokenize(S) = \arg\max_W \sum_{w_i \in W} l_{w_i}^{\alpha}\, h_\theta(w_i)$$

Each of the hypothesis functions is a binary classifier that takes features that can be extracted at the word level. The length normalizer $l_{w_i}^{\alpha}$ is simply the length of the word raised to α. Without this factor, the maximization would prefer short words because each hypothesis function outputs either 0 or 1. A neutral α is 1, whereas $\alpha > 1$ gives preference to longer words, which is desirable for Chinese. [1]

Training the classifier is nonobvious. Since the corpus of training data is all positive examples, we need to generate incorrect tokens in the same fashion that incorrect tokens will be encountered during classification. Training all possible tokenizations of the training set will involve a massive amount of data. We will return to this topic in Section 5.

4. RELATED WORK

Recent work done in this highly specialized area is found in the ACL SIGHAN workshops and the HLT-NAACL conference. SIGHAN has sponsored a number of "bake-offs", or short competitions in recent years to advance the state of the art for Asian language problems. The data for the 2005 SIGHAN bake-off, which was a Chinese tokenization competiion, was used for this work.[1]

In particular, one theoretical model has taken over the field, finding its way into all of the top performing contestants. The work is based on Peng, Feng, and McCallum [2], in which they describe how to use conditional random fields for Chinese tokenization. In contrast to this work where sentences are considered as a whole chunk, the work with conditional random fields is stream-based, working with just a few characters in a buffer at a time. The test performance of some contestants exceeded 90%, which is more than we can possibly hope for with this basic approach.

The theoretical underpinnings of using conditional random fields are completely opposite of the stated strategy we have taken here. Instead of assuming independence between words, the conditional random fields approach indeed relies on that dependence. In truth, some of the features used in the word classifier actually depend on character ordering, so it uses a similar idea.

5. SENTENCE MAXIMIZER

Before worrying about how to classify candidate character sequences as words or not words, it is necessary to verify that doing so will be useful. To test the usefulness of the sentence maximization technique, several experiments were run where the word classifier had perfect forward knowledge of the words. The classifier consisted of a dictionary of all the words in the training set, and it returned 1 if the candidate sequence was in the dictionary and 0 otherwise. Thus it can be thought of as an ideal classifier, so long as the dictionary is pre-populated with the words from the test set. This allowed the isolation of the sentence maximization step and allowed comparison of the various methods. We built three different sentence maximizers based on the model described in Section 3.2.

5.1 Brute Force

Initially, we thought that it would be possible to use a brute force search to find the tokenization of words that has the highest score. Although a brute force list of permutations of word breaks would be guaranteed to include the optimal solution (and indeed, every other solution, providing a perfect test case for our scoring function), it was impractical to use.

[1] It turns out that using α=1 will result in many ties where two shorter words must be combined to make the correct word.

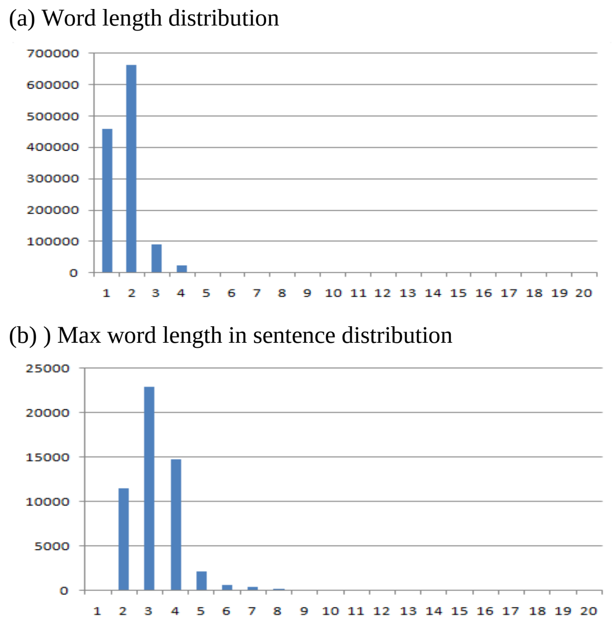


Figure 1. Word length distributions

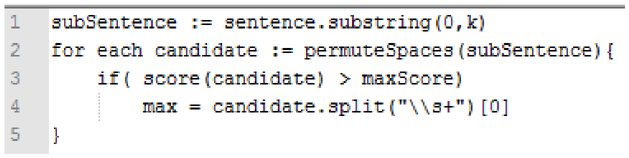
Initially it was expected that phrases would be at most around 10 characters, but it was found that this was an unreasonable assumption, so the exponential number of permutations to test was wholly impossible on available hardware. It was necessary to innovate a method for generating possibilities that would include the optimal solution without exponentially many non-optimal solutions.

The word length distributions were analyzed, as seen in Figure 1. This information proved very useful, as it confirmed that words longer than 5 characters were exceedingly rare (and 95% of the sentences did not contain any word longer than 4). The first pass at using this information was to improve the brute force search by adding filtering on any partition that would leave a subsequence of more than 5 characters. Unfortunately, filtering alone this did not make the problem tractable (2^40 sequences take a very long time to enumerate) and it seems that even with the word length distribution the permutations list will be exponential, but the insight of the distribution helped drive the next attempt.

5.2 Random Draws from PDF

Armed with the discrete probability distribution function for the word length, the next try at enumerating possible sentences was to repeatedly draw word lengths from the distribution and use them to form the word breaks. With 10,000 draws, it was found that the real sentence was included in the draws over 90% of the time (averaged over the whole training set). Shorter sentences needed fewer draws, so a reasonable improvement was to use 100 times length as the number of draws. However, 10% candidate error on top of significant baseline scoring error obtained from limited experiments with the brute force method (of about 7%) was still unsatisfactory.

(a) Pseudocode for dynamic algorithm



```
subSentence := sentence.substring(0,k)
for each candidate := permuteSpaces(subSentence){
    if( score(candidate) > maxScore)
        max = candidate.split("\\s+")[0]
}
```

(b) Training error versus k for dynamic algorithm

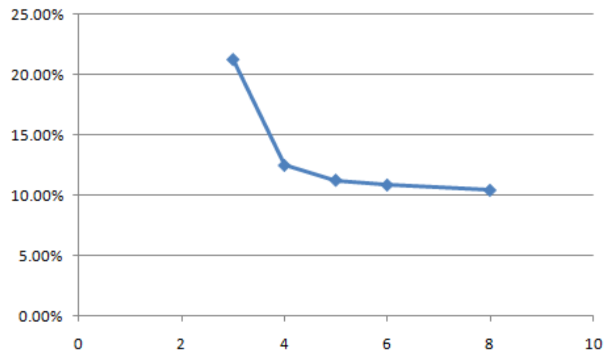


Figure 2. Dynamic optimizer

5.3 Dynamic Programming Algorithm

The short maximum word length created an opportunity for using a dynamic programming algorithm. Since the max word length is limited, intuitively finding the best first word among the first 5 characters would be just as good as looking for it among the first 25. All permutations of the spaces within the 5 characters are made. Once the first word is found, it can be "dropped" from the remaining characters array and progress has been made. Repeating this process gives a solution in O(n * 2^k) time, where k is a small constant (the max number of characters to consider). The algorithm is presented as Figure 2(a).

The solution is not necessarily the same as the correct one, but in practice it frequently is, as is seen in Figure 2(b). This algorithm, using the ideal classifier and a length-biased scoring function, approached only 10% scoring error + candidate error (they cannot be separated because they are intertwined in the dynamic programming approach), which was comparable to the brute force method. This implies that most of the remaining problem is scoring error, for which we need to evolve the maximization objective in Section 3.2 to further improve performance.

5.4 Scoring Error

Reducing the scoring error toward zero presented additional challenges. Since the focus of the project was intended to be on machine learning techniques and the most promising improvements here did not involve those techniques, they were not explored to a great extent. It seemed to be a battle of diminishing returns because of two facts. First, the dynamic programming method did very well. Besides from getting nearly 90% of the sentences perfect, it got overall about 97% of the actual word breaks correct. An analysis of the 3% which were mistakes showed that in a number of these cases, there was an ambiguity in the dataset (it can be argued that the computed answer was also valid under the tokenization rules). Indeed, for the intended applications (IR-like systems that required tokenization) the mistakes were often harmless (an equivalent in English might be "can not" versus "cannot"). Instead of spending much time working out how to squeeze away the last few bad mistakes, a baseline of performance was established and efforts moved on to the evaluation of different word classifiers beyond the simple dictionary.

5.5 Basic Word Classifier Performace

Using a dictionary alone did quite well. On 10-fold cross validation, using just the dictionary yielded 68% accuracy (out of an ideal of 90%) at the whole sentence level. Looking at individual word breaks, it got 90% correct, compared to a 97% ideal. Thus, the focus of the machine learned word classifiers studied here is to bring that closer to the ideal.

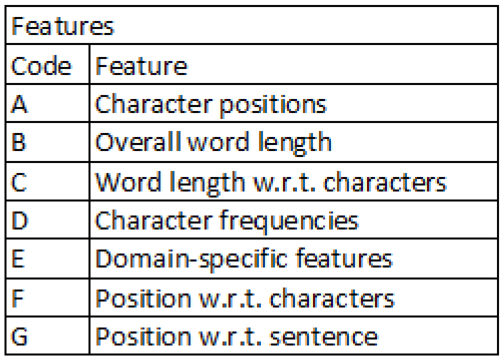
6. NEW WORD CLASSIFIER

Once the sentence maximizer strategy was verified, we focused efforts on building a binary classifier for words using logistic regression. The parameters were optimized using parallel batch gradient descent. We tried a stochastic gradient descent approach, but it found it hard to parallelize and so it ran slower even though it came close to convergence in fewer iterations. We noticed that it was more sensitive to local optima as well.

The success of the basic dictionary classifier meant that the main utility of a learned word classifier would be for identifying new words, since known words could simply be read from the dictionary.

We used a test set of 10,000 sentences, a statistics training set of 43,000 sentences, and a classifier training set of 3,000 sentences.

Instead of incorporating the dictionary into the learned classifier, it was found that performance was improved dramatically by keeping it as a separate, perfect-precision classifier that would defer to the learned new word classifier when it did not contain the candidate word.



Features	
Code	Feature
A	Character positions
B	Overall word length
C	Word length w.r.t. characters
D	Character frequencies
E	Domain-specific features
F	Position w.r.t. characters
G	Position w.r.t. sentence

Table 1. List of features in new word classifier

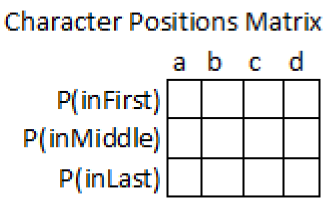
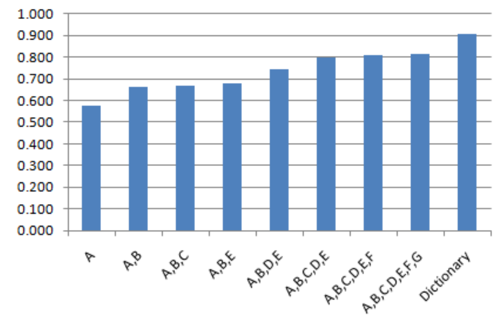


Figure 3. Character positions matrix for "abcd"

(a) Avg. Jaccard similarity with actual sentence



(b) Training/Testing error (word candidates)

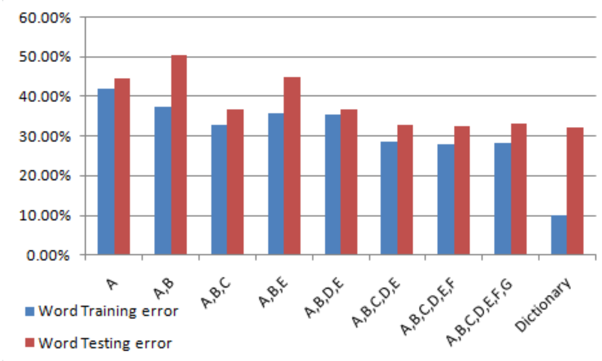


Figure 4. Feature selection

6.1 Features

Several classes of features were chosen to aid in the identification of new words. Because the gap between training and test error remained reasonable, we realized we were underfitting the data and spent a lot of time trying to develop new features. The list of features that were actually closely evaluated in the end are listed in Table 1.

One important feature is the character positions matrix, which is also show in Figure 3. In the statistical training step, we collect character statistics about how often characters start a word, appear in the middle of a word, and end a word. It proves indeed unlikely that a two-character combination that starts with an "ender" and ends with a "starter" would be a word.

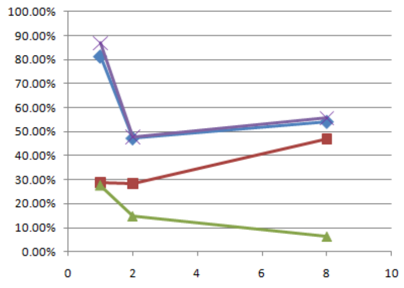
Other features we tried included individual character frequency, word length, and absolute and relative positioning within the sentence. In addition, two domain-specific features for handling written numbers and interspersed ASCII words were evaluated.

6.2 Feature Selection

We performed an evaluation on the different feature sets using various metrics. The two metrics we found most helpful were test/training error over word candidates and a Jaccard similarity of the chosen words in the sentence between the sentence guessed by the sentence maximizer and the real sentence. This similarity can be thought of as a percentage of the word gaps that were guessed correctly at the sentence level using the information from the word classifiers.

The results of the feature selection are shown in Figure 4.

(a) All features



(b) Without G features

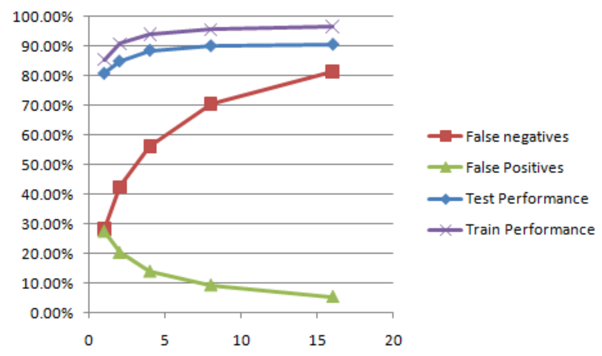


Figure 5. Performance optimization using training bias

Generally, adding features helped performance. The feature sets that seemed to boost performance the most were A, B, and D.

6.3 Optimizing Performance

We found that this particular sentence scoring is very sensitive to false positives, as we will discuss in the next section. Even though the classifier did fairly well when it came to distinguishing words from candidate character combinations, disappointingly, it did not translate into improved performance at the sentence level because every additional word that it got right was counterbalanced by a non-word that it got wrong and caused the sentence optimizer to choose it.

In hopes of obtaining at least some benefit from the new word classifier, we biased the training set by including many times more negative examples. Ideally, the new learned parameters would have a lower false positive rate, at the expense of a higher false negative rate.

The results of this attempt are shown in Figure 5. Figure 5(a) shows the bizarre results when all the features were used. We attributed this to an overfitting caused by some of our sentence position features (G). Once the G features were removed, the expected trade off was observed, as seen in Figure 5(b).

Ultimately, we could not get the learned new word classifier to improve performance on sentences over just using the dictionary as a classifier.

7. DISCUSSION

7.1 Issues with False Positives

The word classifier actually worked well, as we can see from the relatively low test and training error. However, the sentence optimizer was underconstrained and thus very sensitive to false positives. While the dynamic programming approach clearly is obviously very sensitive because it assumes an optimal substructure, the problem actually lies with the maximization equation. It is underconstrained because nearly any false positive will result in an incorrect sentence. This was not detected in the early stages because the basic dictionary classifier had no false positives. Thus it can be thought of as having good training error and poor generalization error.

Conceivably two approaches could improve this. One would be a learned sentence model that follows common word length patterns. The other is adding constraints that exclude more character combinations. For instance, some simple grammar rules might substantially reduce the number of false positives.

7.2 Conclusion

It is possible to build a fairly accurate word classifier using logistic regression. In combination with a dictionary, it performs even better.

The sentence optimizer also did well at turning word classifications into accurate sentence scores, but it had a dismal tolerance for false positives, which made using the learned word classifier difficult. Adding constraints or changing the objective will likely allow the classifier to actually help without hurting too much.

8. ACKNOWLEDGMENTS

Thanks to all the course staff and Prof. Ng for your help and encouragement.

9. REFERENCES

[1] R. Sproat and T. Emerson. The First International Chiense Word Segmentation Bakeoff. *Proc. Of Second SIGHAN workshop on Chinese language processing*.

[2] F. Peng, F. Feng and A. McCallum. Chinese Segmentation and New Word Detection using Conditional Random Fields. *International Conference on Computational Linguistics, 2004*.