Implementation of Chinese Segmentation Tool using Hidden Markov Model

Yang Zhang (11529139)

I. Abstract

This report summarized the implementation of a Chinese segmentation tool by applying HMM (Hidden Markov Model). The reasons that HMM is suitable to solve Chinese segmentation problem will be covered. In the experiment part, empirical data and results are provided to support the theoretical ideas.

II. Introduction

Text segmentation is an important part of natural language processing. The purpose of word segmentation is to divide written text into meaningful units, such as words, sentences, or topics. Some languages have explicit word boundary markers, for instance, written English uses spaces to mark single word. However, Chinese segmentation is non-trivial. Written Chinese does not have a significantly clear marker for each word which makes the segmentation sometimes ambiguous. Here is an interesting example,

新书架

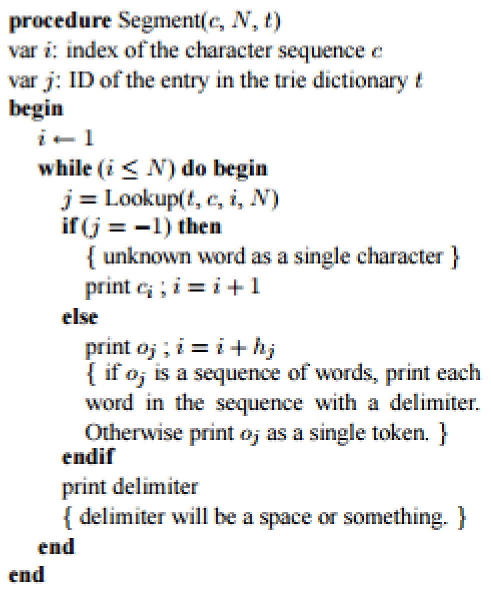
The above Chinese phase have two possible segmentation:

1. **新 / 书架**
2. **新书 / 架**

The first segmentation means a new bookshelf, while the second segmentation indicates the bookshelf with new books. We can see from the example that segment Chinese is not obvious even if it is a short phase. As a Chinese myself, it is quite interesting to implement a Chinese segmentation tool with machine learning techniques and see how it works.

III. Problem Setup and Related work

The main task of Chinese segmentation is to tokenize a Chinese sentence into batch of semantic meaning chunks. Chinese segmentation is important in machine translation to other language. In the past, without the magic of machine learning, an algorithm named Max-Matching is widely used to solve the algorithm.



```
procedure Segment($c$, $N$, $t$)
var $i$: index of the character sequence $c$
var $j$: ID of the entry in the trie dictionary $t$
begin
  $i \leftarrow 1$
  while ($i \leq N$) do begin
    $j$ = Lookup($t$, $c$, $i$, $N$)
    if ($j = -1$) then
      { unknown word as a single character }
      print $c_i$ ; $i = i + 1$
    else
      print $o_j$ ; $i = i + h_j$
      { if $o_j$ is a sequence of words, print each
      word in the sequence with a delimiter.
      Otherwise print $o_j$ as a single token. }
    endif
    print delimiter
    { delimiter will be a space or something. }
  end
end
```

The idea behind max-matching is very simple. First, we define a max-matching number N, which means we check a sequence of character no more than this number. Then, the algorithm will divide the input sequence into subsequences with size N with sliding window method. For each of the subsequences, the algorithm looks up each token (from length 1 to length of the subsequences) in a dictionary. If the token is in the dictionary, the token is marked as a segment. There are two variances of Max-Matching algorithm, which are forward max-matching and backward max-matching. For forward max-matching, the algorithm lock up from the front, and vice versa. However, this algorithm shows significant limitations. One is that max-matching completely ignores the dependency between adjacent characters. This will damage the performance of segmentation significantly.

IV. Solution Approach and Experiments

Using a static model could overcome the lack of dependency in Max-Matching algorithm. In this report, I will mainly focus on applying Hidden Markov Model. HMMs can be regarded as Bayesian networks and HMMs represent probability distributions over sequences of observations. This feature of HMMs fits the domain of Chinese segmentation problem well. Specifically, two probability distributions are built while training a HMM model. (Note: assume that current character only has dependency with previous character)

The State Transformation Matrix $\mathbf{T_{ij}}$ (L_i indicates the state label for the i^{th} character)

$$T_{ij} = P(L_j \mid L_i) = P(L_j , L_i) / P(L_i) = \text{count}(L_j , L_i) / \text{count}(L_i)$$

The Mixture Matrix $\mathbf{M_{ij}}$ (C_i indicates the ith character)

$$M_{ij} = P(C_j \mid L_i) = P(C_j, L_i) / P(L_i) = count(C_j, L_i) / count(L_i)$$

Four state labels are defined:

1. B --- Beginning character of a word
2. M --- Middle character of a word
3. E --- Ending Character of a word
4. S --- Single character of a word

The unicode range of Chinese character is from U+4E00 to U+9FFF, therefore it is very likely that the training sets do not contain every possible character. Laplace smoothing should be applied in building mixture matrix.

$$M_{ij} = P(C_j \mid L_i) = P(C_j, L_i) / P(L_i) = (count(C_j, L_i) \mathbf{+ 1}) / count(L_i)$$

1. **Code Preparation**

   The Chinese segmentation tool is implemented in Java. The HMM model implementation is imported from open source library JaHmm .

2. **Data Preparation and Experiment Procedure**

   I used the msr_training.utf8 as training dataset, which contains 86924 training sentences (around 200,0000 segmented Chinese word).
   To test the model, I used the file bundle pku_test.utf8 and pku_test_gold.utf8 that contains 1944 sentences (pku_test_gold.utf8 is the file that has correctly segmented sentences from pku_test.utf8).
   Notes: all files are from package icwb2-data of SIGHAN Bakeoff 2005

3. **Experiment Results**

Measurement	**Value**
Total Insertions	1432
Total deletions	4463
Total correct word	63127
True word count	88696
Test word count	85665
Recall	0.712
Precision	0.737

Both precision and recall are around 70%, which is not a very satisfied result. One possible reason is that HMM model has relatively loose rule in segmentation. From some specific results, I found a lot of words do not exist in Chinese literature. Interestingly, the loose control is not always bad. I also observed that some rare words are segmented correctly such as people's names.

V. Conclusions

In this project, the Chinese segmentation tool was implemented successfully. However, the performance of the tool needs improvement. In my opinion, applying Laplace smoothing might negatively affect the performance. Laplace smoothing is effective to deal with sparse cases, but it also under estimates or over estimates the true probability. Good-Turing is a better smoothing method to estimate probabilities.

Last but not the least, this project provides me a hand-on experience on structure prediction area. It also motivated me to learn new concepts such as graphic model in this area and inspired me to explore and learn in the future.

VI. Acknowledgments and References

Team member: Yang Zhang

1. Manabu Sassano, *Deterministic Word Segmentation Using Maximum Matching with Fully Lexicalized Rules.*
2. Zoubin Ghahrammani, *An Introduction to Hidden Markov Models and Bayesian Networks.*
3. Bill MacCartney, *NLP Lunch Tutorial: Smoothing*
4. Data package from Linguistic Data Consortium of U-Peen
5. JaHmm is open source libaray *https://github.com/KommuSoft/jahmm*