Mohaema Segmenter: Chinese Word Segment System

Based on the Maximum Matching Algorithm

Zhang Yinqing[1], Qin Yu[2], Zhang Chaoran[3]

1 Establishing the algorithm, creating a crawler for the standard corpus on http://g20.org/, debugging

2 Building the User Interface, testing the whole system

3 Searching good segmented corpus, using the program to turn it into lexicon and enabling users add new words into the lexicon

# Algorithm

Zhang Yinqing

## The Maximum Matching Algorithm

The basic form is to resolve the ambiguity of a single word (Yi-Ru Li, personal communication, January 14, 1995). For example, suppose C1, C2,... Cn represent characters in a string. We are at the beginning of the string and want to know where the words are. We first search the lexicon to see if \_C1\_ is a one-character word, then search \_C1C2\_ to see if it is a two-character word, and so on, until the combination is longer the longest words in the lexicon. The most plausible word will be the longest match. We take this word, then continue this process until the last word of the string is identified.

Another variant of maximum matching done by Chen and Liu (1992) is more complex than the basic form. Their maximum matching rule says that the most plausible segmentation is the three-word chunk with maximum length. Again, we are at the beginning of a string and want to know where the words are. If there are ambiguous segmentations (e.g., \_C1\_ is a word, but \_C1C2\_ is also a word, and so on), then we look ahead two more words to find all possible three-word chunks beginning with \_C1\_ or \_C1C2\_. For example, if these are possible three-word chunks:

1. \_C1\_ \_C2\_ \_C3C4\_ 谨 对 各位

2. \_C1C2\_ \_C3C4\_ \_C5\_ 谨对 各位 同

3. \_C1C2\_ \_C3C4\_ \_C5C6 谨对 各位 同事

The chunk with maximum length is the third one. The first word, \_C1C2\_ of the third chunk, will be considered as the correct one. We take this word, proceed to repeat this process from character C3, until the last word of the string is identified. Chen and Liu (1992) reported that this rule achieved 99.69% accuracy and 93.21% of the ambiguities were resolved by this rule.

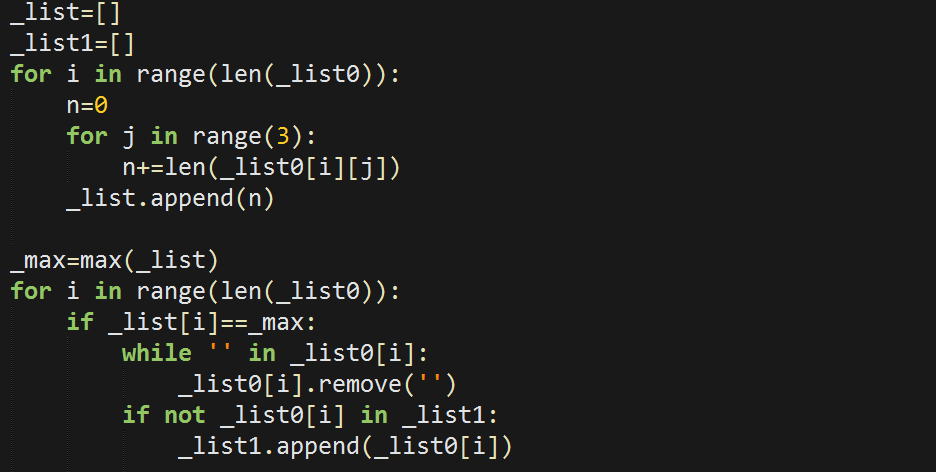
## Ambiguity Resolution Rules

Four ambiguity resolution rules were used. The maximum matching rules applied to ambiguous segmentations from both simple and complex matching algorithms. The rest three rules did not (and could not) apply to ambiguous segmentations from the simple matching algorithm.

**Rule 1: Maximum matching (Chen & Liu 1992).**

(a) Simple maximum matching: Pick the word with the maximum length.

(b) Complex maximum matching: Pick the first word from the chunk with maximum length. If there are more than one chunks with maximum length, apply the next rule.



**Rule 2: Max Average Length (Chen & Liu, 1992).** At the end of each string, it is very likely to have chunks with only one or two words. For example, the following chunks have the same length and the same variance of word lengths.

1. \_C1\_ \_C2\_ \_C3\_

2. \_C1C2C3\_

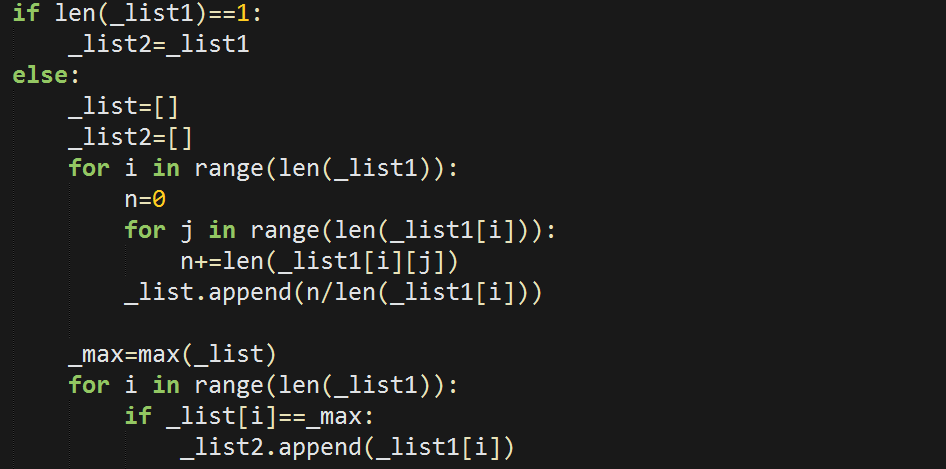
国 际 税收

国 际 税 收

国际 税收

Rule 2 picks the first word from the chunk with largest average word length. In the above example, it picks \_C1C2C3\_ from the second chunk. The assumption of this rule is that it is more likely to encounter multi-character words than one-character words.

This rule is useful only for condition in which one or more word position in the chunks are empty. When the chunks are real three-word chunks, this rule is not useful. Because three-word chunks with the same total length will certainly have the same average length. Therefore we need another solution.



**Rule 3: Smallest variance of word lengths (Chen & Liu, 1992).** There are quite a few ambiguous conditions in which the Rule 1 and Rule 2 cannot resolve. For example, these two chunks have the same length:

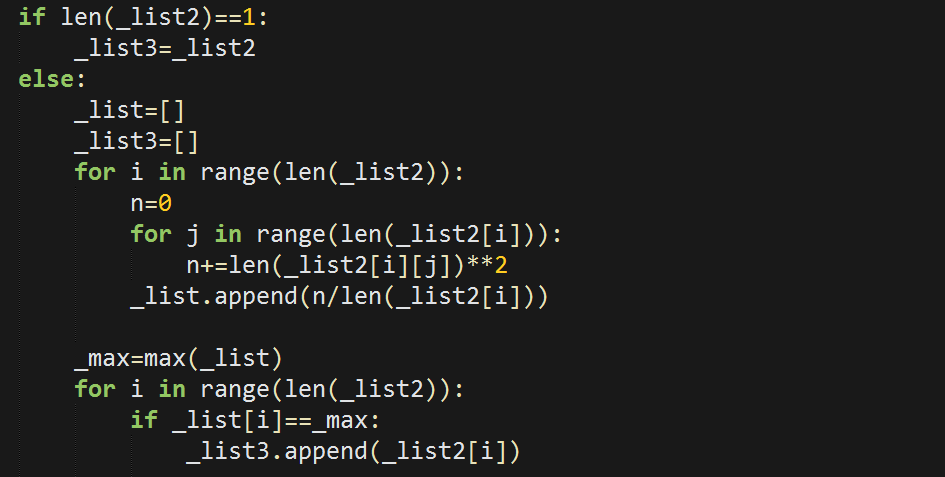
1. \_C1C2\_ \_C3C4\_ \_C5C6\_

2. \_C1C2C3\_ \_C4\_ \_C5C6\_

反腐 败 领域

反腐败 领域

Rule 3 picks the first of the chunk with smallest variance of word lengths. In the above example, it picks \_C1C2\_ from the first chunk. This rule is exactly them as the one proposed by Chen and Liu (1992). (However, they applied this rule immediately after Rule 1.) The assumption of this rule is that word lengths are usually evenly distributed. If there are more than one chunks with the same value of smallest variance of word lengths, apply the next rule.



**Rule 4: Largest single word frequency.** This example shows two chunks with the same length, variance, and average word length:

1. \_C1\_ \_C2\_ \_C3C4\_

2. \_C1\_ \_C2C3\_ \_C4\_

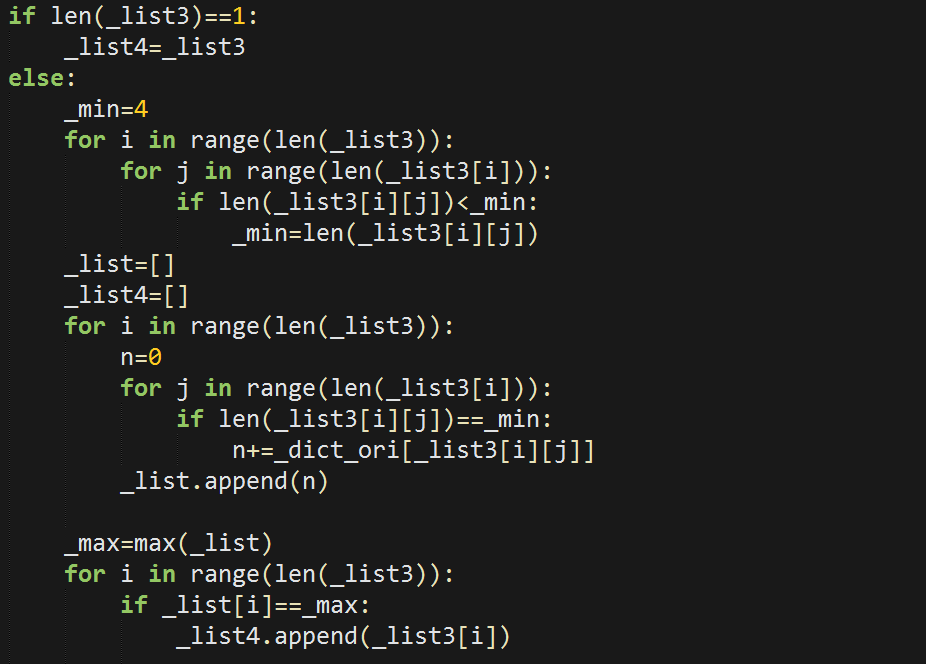
电视 电话 会议 数 百次

电视 电话 会议 数百 次

Both chunks have two one-character words and one two-character word. Which one is more likely to be the correct one? Here we will focus on one-character words. Chinese characters differ in their degree of morphemic freedom. Some characters are rarely used as free morphemes, but others have larger degree of freedom. The frequency of occurrence of a character can serve as an index of its degree of morphemic freedom. A high frequency character is more likely to be a one-character word, and vice versa.

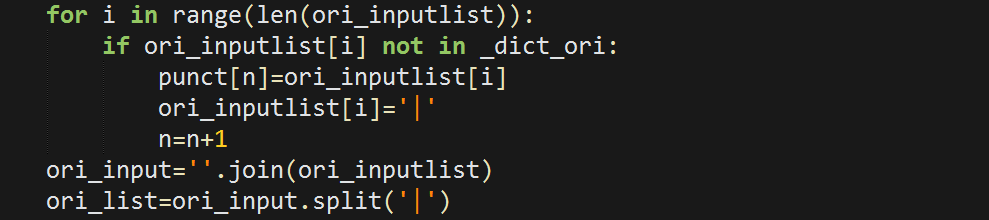
The formula used to calculate the sum of degree of morphemic freedom is to sum log(frequency) of all one-character word(s) in a chunk. The rationale for logarithm transformation is that the same amount of frequency difference does not have a consistent effect across all frequency ranges.

Rule 4 than picks the first word of the chunk with largest sum of log(frequency). Since it is very unlikely that two characters will have exactly the same frequency value, there should be no ambiguity after this rule has been applied.



## Notation Seeking Method

Though using a huge lexicon, there always occurs some unknown words or notation. For the complement of the algorithm, instead of the lexicon of notations, I put all the words not in the lexicon as notation.



Then, the remaining work is only focusing on the ambiguity resolution.

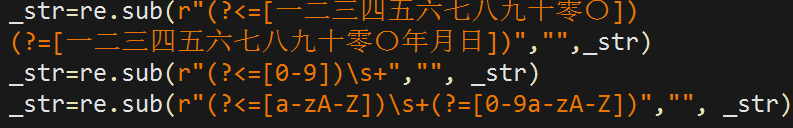
## Quantifiers, Numbers, English Words Seeking Method

In practical, the method above leads a vital problem: the numbers and English words will be cut into pieces. So this method was drew initially aimed at this problem, this part, only needs put them together. As for quantifier, this part is focused on the following problem:

去年 11 月 (Original Method)

去年 11月 (Standard)

Then another methods is using regular expression to find the pattern.



(Only a demo, our team has built a lexicon for quantifiers.)

## Results

Using the paragraphs from g20.org \*segtest.txt

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Test from different paragraphs

1 2 3 4

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Identifications 678 254 473 316

Errors 5 4 2 4

Accuracy 99.26% 98.81% 99.58% 99.37%

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Examples of errors:

阿美 首脑会议 将 讨论 巴以 和 平等 问题

(阿美 首脑会议 将 讨论 巴以 和平 等 问题)

二十国 集团 成员 和 嘉宾 国 领导人

(二十国 集团 成员 和 嘉宾国 领导人)

二十国 集团 要 与时俱进 、 知 行 合一

(二十国 集团 要 与时俱进 、 知行合一)

我们 还 将 通过 支持 非洲 和 最 不 发达国家 工业化

(我们 还 将 通过 支持 非洲 和 最不发达 国家 工业化)

要 充分 倾听 世界 各国 特别 是 发展 中 国家 声音

(要 充分 倾听 世界 各国 特别 是 发展中国家 声音)

这 是 习近 平和 彭丽媛…

(这 是 习近平 和 彭丽媛…)

# Lexicon

Qin Yu

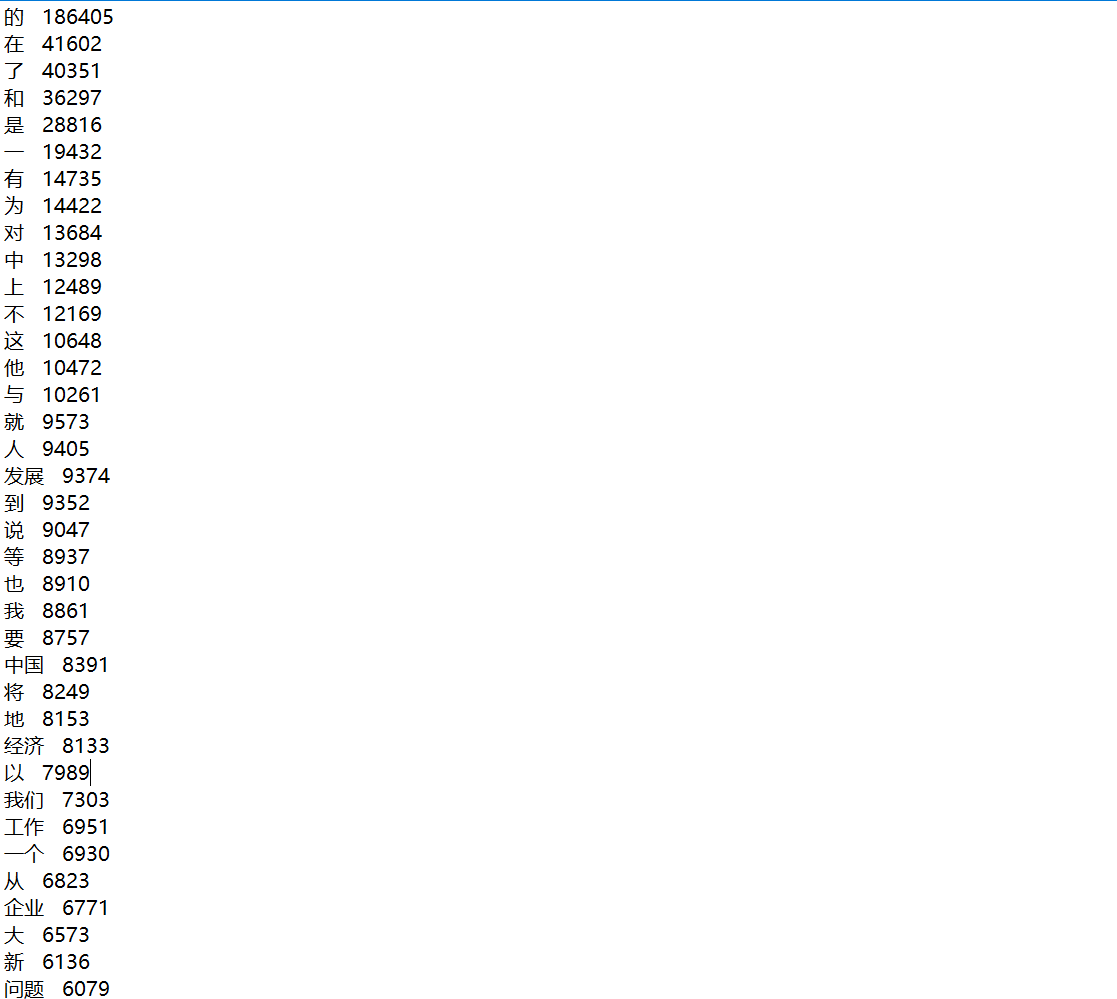
Developing Environment:

JeBrains PyCharm Community Edition 2016.3.1

About the lexicon, we search for some corpus which already had been segmented, such as the corpus from Microsoft and PKU.

And then we code a program to turn it into a lexicon. We are in trouble when open the corpus file, because there are always decoding error. Then we add “encoding = utf-8” to solve the problem successfully. We use some built-in function, like “Counter”, to count the words and their frequency and sort them by the frequency. Finally write the result into a txt type file. And there is a function that enable user to add new words.

At first, I intended to make a dictionary of name and toponym. But in consideration of the place names of China are very complex and the names are very complex, I directly copy a file of toponym from Internet.



A part of the lexicon.

# User Interface

Zhang Chaoran

1: Goal: To design a both delicate and practical UI.

2: Tools:

Python 3.4(32bit), PyQt v4.11.4(for python 3.4, x32),

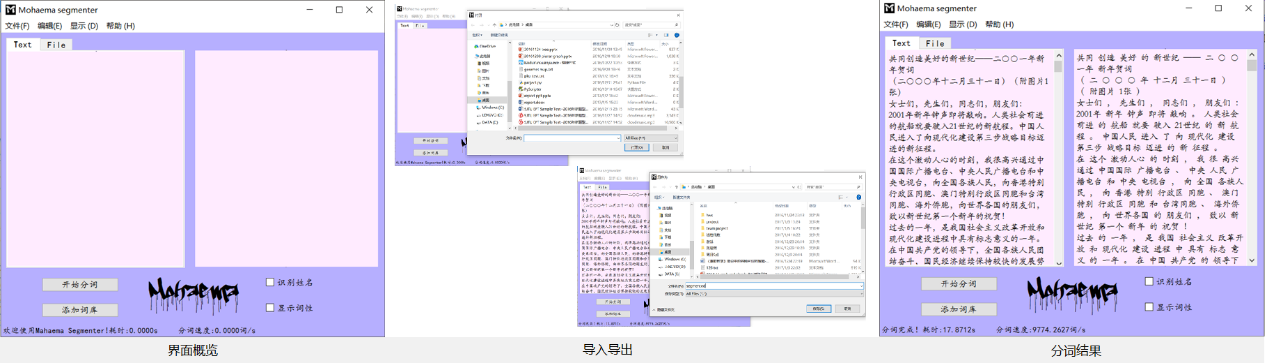
PyScripter for Python 3.4

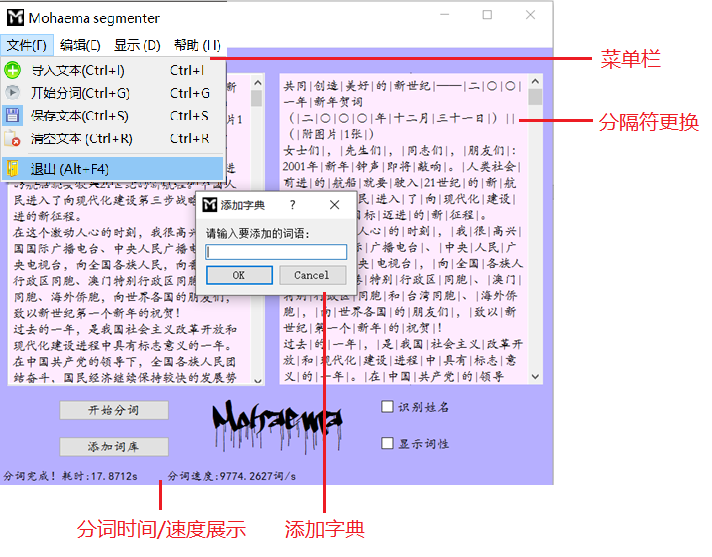
3: Imported Libraries:

Built-in: sys, time

From PyQt4: QtCore, QtGui, uic

4: User Interface: (Screenshot)





Pictures above show the general appearance and functions of this interface. Next, I’ll introduce the construction of each widget.

4.1 Input Frame

Widget name: Text Edit

Function: It allows the user to write text in it, and records the text as the parameter of the segment algorithm.

Code: ori\_text=self.Input.toPlainText()#Input is the name of the object.

Created by PyQt Designer.

4.2 Output Frame

Widget name: Text Browser

Function: It can present the text that was processed by the segment algorithm.

Code: self.Output.setText(new\_text) #Output is the name of the object.

4.3 Buttons

4.3.1 Start button

Code:

self.start.clicked.connect(self.segment0)

self.Go.triggered.connect(self.segment0)#In the menu bar

P.s.: There are 2 ways to start the segment algorithm. It’s said that a good UI design allows the users to activate a function in more than one way. Same to other functions.

4.3.2 Add file

Code: self.import\_1.triggered.connect(self.FileOpen) #In the menu bar

self.AddFile.clicked.connect(self.FileOpen)

These buttons can use the text in a “.txt” file as the input.

4.4.3 Add word

self.addword.clicked.connect(self.addup1)

Though carefully constructed, our dictionary may fail to deal with all kinds of texts. But you can customize the dictionary by adding words into it. By the way, the frequency of added word is always 1.

4.4.4 Edit Buttons

Undo, Redo, cut, copy, paste… These are the basic operations to text. You can get access to them either use hotkeys or by the menu bar.

4.4.5 (Not done yet) Recognizing names& Show nominals

4.5 Hotkeys

Ctrl+R: Clear Inputs

Ctrl+G: Start segmenting

Ctrl+Z,Y,X,C,V: Same to Word operations

Ctrl+I: Open files

Ctrl+S: Save

4.6 Timer

Code: runtime1= time.clock()

(segment………)

runtime2= time.clock()

runtime = runtime2-runtime1-0.1

self.time\_speed.setText('分词完成！'+"耗时:"+str('%.4f'%runtime)+'s '+'分词速度:'+str('%.4f'%(len(ori\_text)/runtime))+"词/s")

An easy time recording method.

4.7 Color

Use Designer’s “Palette” to change the color of background, buttons and frames. P.s.: Background picture can be added by using a “label”, but I haven’t found a proper picture for our UI, so abandoned using a image as background.