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:

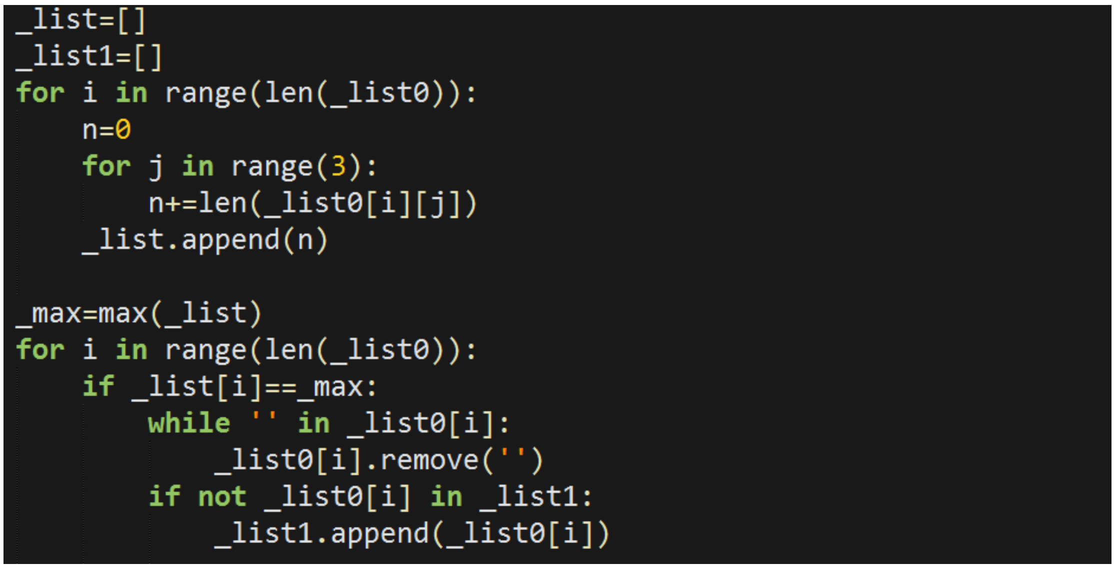
*1. _C1_ _C2_ _C3C4_*	*谨 对 各位*
*2. _C1C2_ _C3C4_ _C5_*	*谨对 各位 同*
3. _C1C2_ _C3C4_ _C5C6	*谨对 各位 同事*

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.



```
_list=[]
_list1=[]
for i in range(len(_list0)):
    n=0
    for j in range(3):
        n+=len(_list0[i][j])
    _list.append(n)

_max=max(_list)
for i in range(len(_list0)):
    if _list[i]==_max:
        while '' in _list0[i]:
            _list0[i].remove('')
        if not _list0[i] in _list1:
            _list1.append(_list0[i])
```

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_*

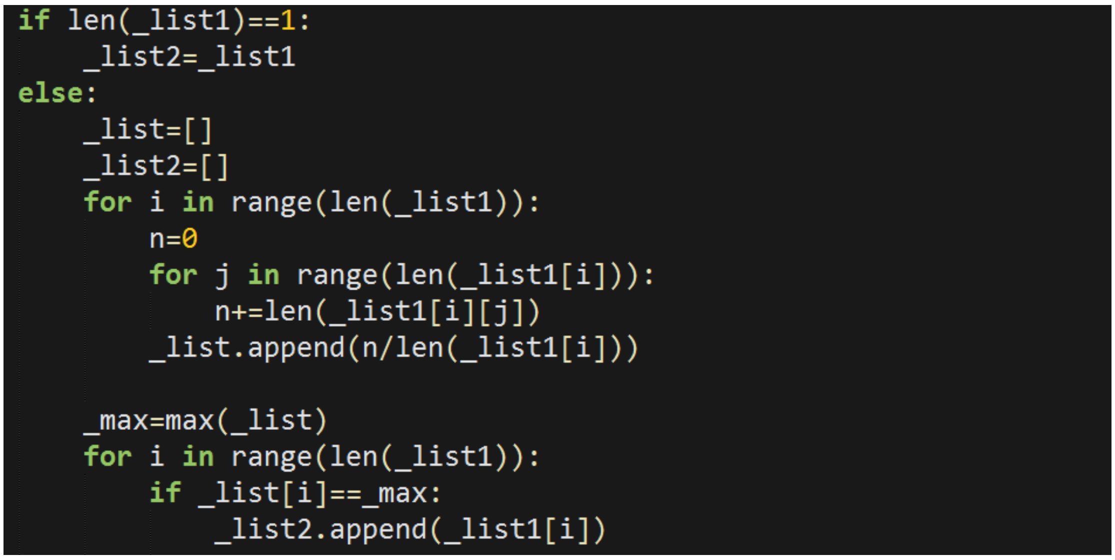
*2. _C1C2C3_*

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.



```
if len(_list1)==1:
    _list2=_list1
else:
    _list=[]
    _list2=[]
    for i in range(len(_list1)):
        n=0
        for j in range(len(_list1[i])):
            n+=len(_list1[i][j])
        _list.append(n/len(_list1[i]))

    _max=max(_list)
    for i in range(len(_list1)):
        if _list[i]==_max:
            _list2.append(_list1[i])
```

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:

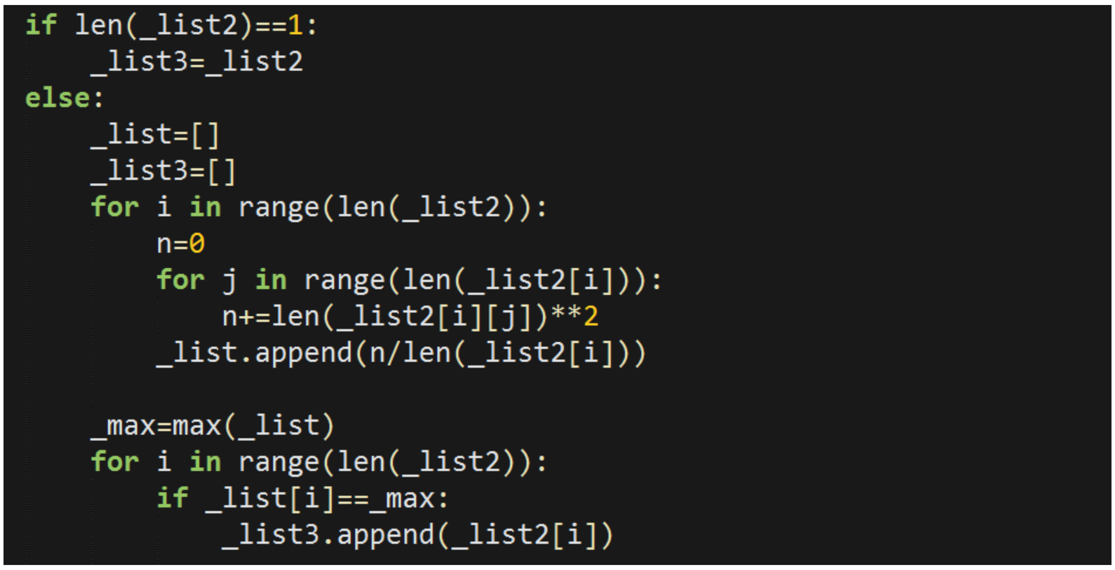
*1. _C1C2_ _C3C4_ _C5C6_*

*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.



```python
if len(_list2)==1:
    _list3=_list2
else:
    _list=[]
    _list3=[]
    for i in range(len(_list2)):
        n=0
        for j in range(len(_list2[i])):
            n+=len(_list2[i][j])**2
        _list.append(n/len(_list2[i]))

    _max=max(_list)
    for i in range(len(_list2)):
        if _list[i]==_max:
            _list3.append(_list2[i])
```

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_*

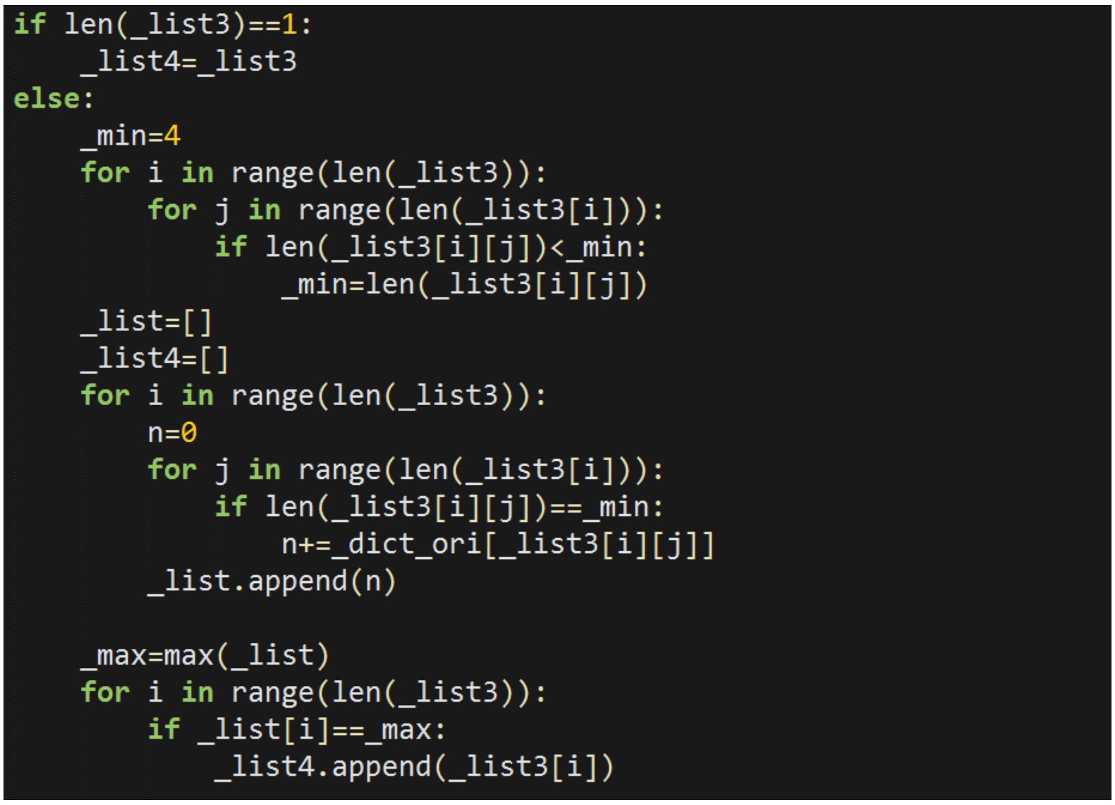
电视 电话 会议 数 百次

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.

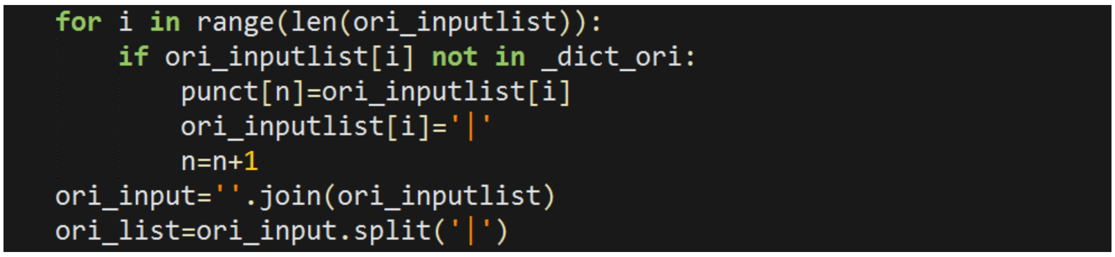


```python
if len(_list3)==1:
    _list4=_list3
else:
    _min=4
    for i in range(len(_list3)):
        for j in range(len(_list3[i])):
            if len(_list3[i][j])<_min:
                _min=len(_list3[i][j])
    _list=[]
    _list4=[]
    for i in range(len(_list3)):
        n=0
        for j in range(len(_list3[i])):
            if len(_list3[i][j])==_min:
                n+=_dict_ori[_list3[i][j]]
        _list.append(n)

    _max=max(_list)
    for i in range(len(_list3)):
        if _list[i]==_max:
            _list4.append(_list3[i])
```

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.



```python
    for i in range(len(ori_inputlist)):
        if ori_inputlist[i] not in _dict_ori:
            punct[n]=ori_inputlist[i]
            ori_inputlist[i]='|'
            n=n+1
    ori_input=''.join(ori_inputlist)
    ori_list=ori_input.split('|')
```

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

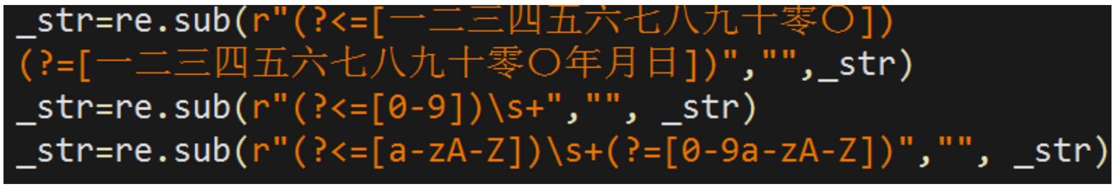
Quantifiers, Numbers, English Words Seeking Method

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.



```
_str=re.sub(r"(?<=[一二三四五六七八九十零〇])
(?=[一二三四五六七八九十零〇年月日])","",_str)
_str=re.sub(r"(?<=[0-9])\s+","", _str)
_str=re.sub(r"(?<=[a-zA-Z])\s+(?=[0-9a-zA-Z])","", _str)
```

(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...)

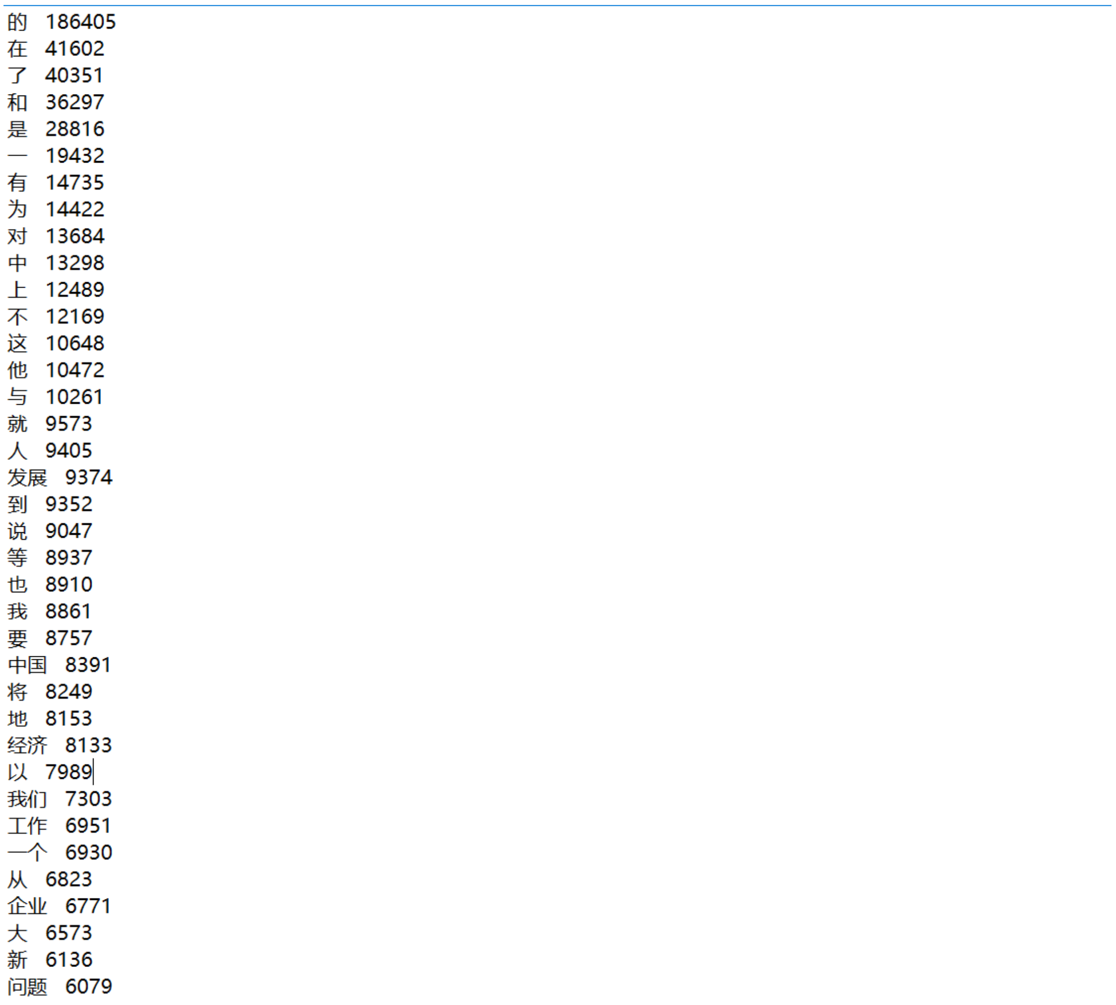
Lexicon

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.



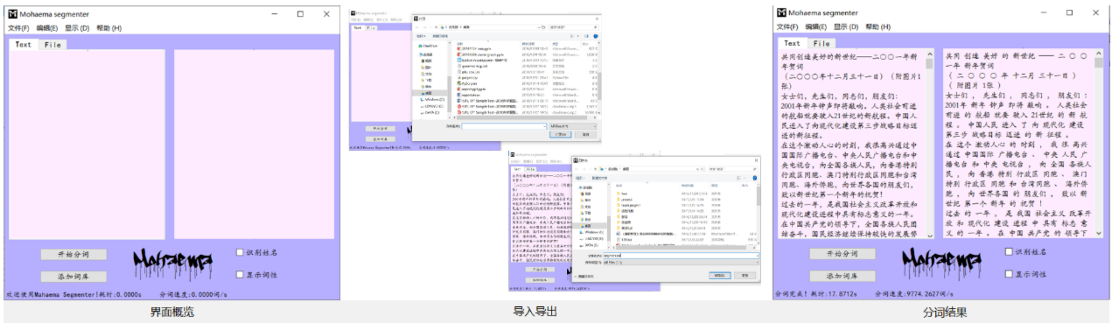
```
的 186405
在 41602
了 40351
和 36297
是 28816
一 19432
有 14735
为 14422
对 13684
中 13298
上 12489
不 12169
这 10648
他 10472
与 10261
就 9573
人 9405
发展 9374
到 9352
说 9047
等 8937
也 8910
我 8861
要 8757
中国 8391
将 8249
地 8153
经济 8133
以 7989
我们 7303
工作 6951
一个 6930
从 6823
企业 6771
大 6573
新 6136
问题 6079
```

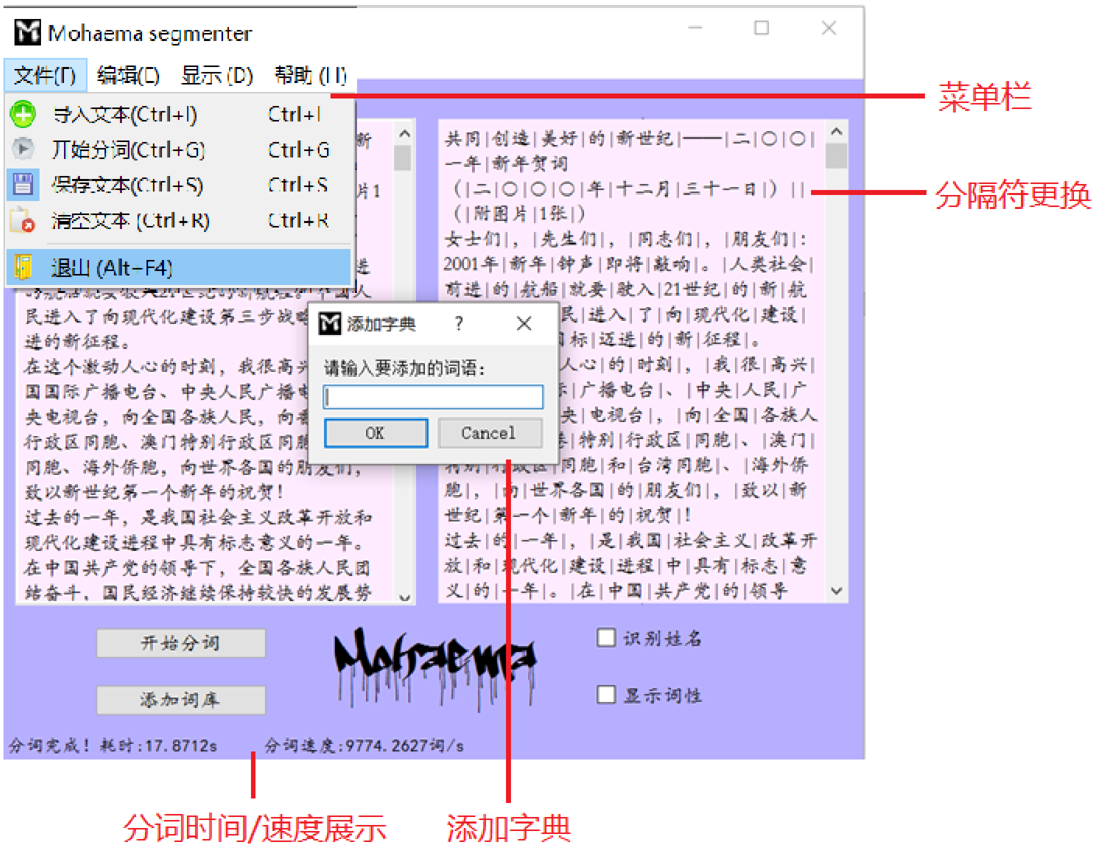
A part of the lexicon.

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)





民进入了向现代化建设第三步战
进的新征程。
在这个激动人心的时刻，我很高兴
国国际广播电台、中央人民广播
央电视台，向全国各族人民，向
行政区同胞、澳门特别行政区同
同胞、海外侨胞，向世界各国的朋友们，
致以新世纪第一个新年的祝贺！
过去的一年，是我国社会主义改革开放和
现代化建设进程中具有标志意义的一年。
在中国共产党的领导下，全国各族人民团
结奋斗，国民经济继续保持较快的发展势

共同|创造|美好|的|新世纪|——|二|〇|〇|
一年|新年贺词
(|二|〇|〇|〇|年|十二月|三十一日|) ||
(|附图片|1张|)
女士们|，|先生们|，|同志们|，|朋友们|：
2001年|新年|钟声|即将|敲响|。|人类社会|
前进|的|航船|就要|驶入|21世纪|的|新|航
民|进入|了|向|现代化|建设|
目标|迈进|的|新|征程|。
人心|的|时刻|，|我|很|高兴|
台|广播电台|、|中央|人民|广
央|电视台|，|向|全国|各族人
|特别|行政区|同胞|、|澳门|
同胞|和|台湾同胞|、|海外侨
胞|，|向|世界各国|的|朋友们|，|致以|新
世纪|第一个|新年|的|祝贺|！
过去|的|一年|，|是|我国|社会主义|改革开
放|和|现代化|建设|进程|中|具有|标志|意
义|的|一年|。|在|中国|共产党|的|领导

分词完成！ 耗时:17.8712s 分词速度:9774.2627词/s

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

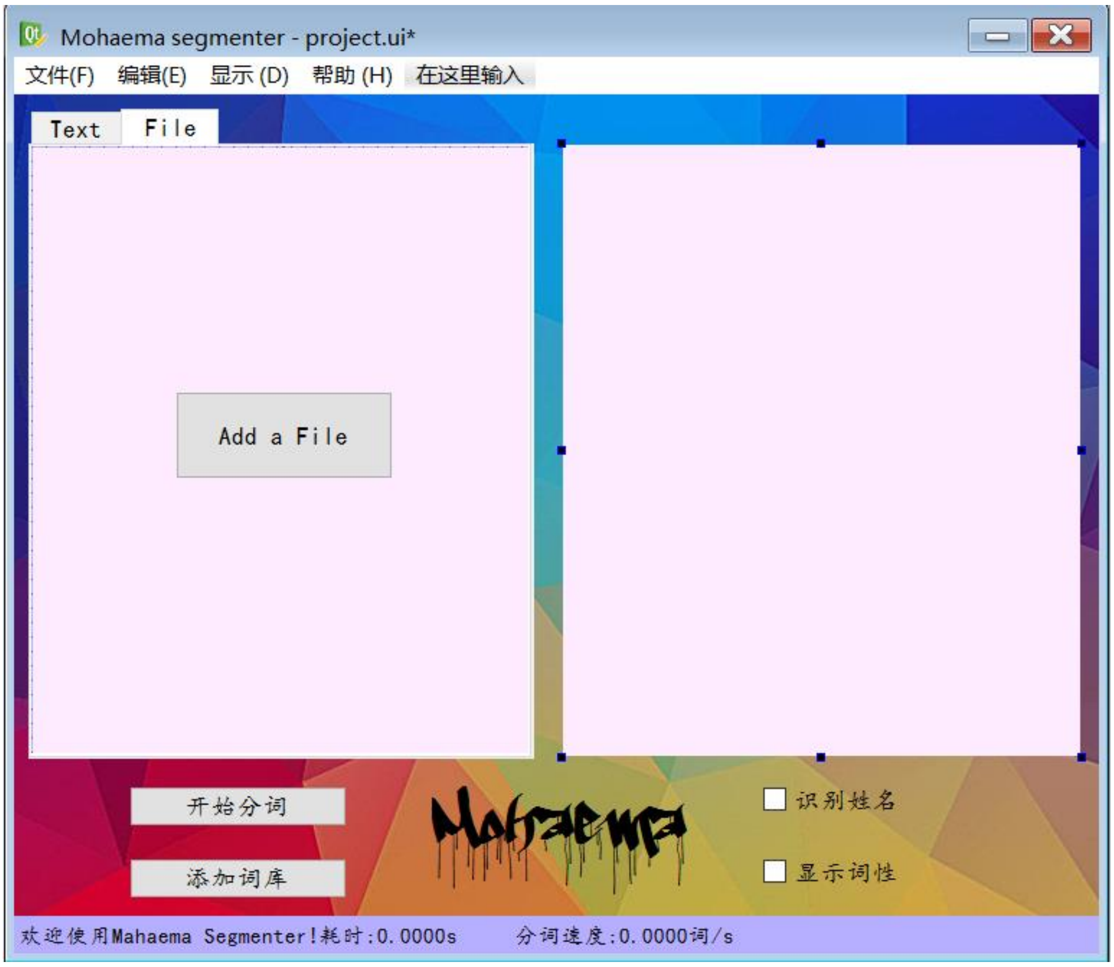
Code: runtime1= time.clock()

```
(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.


Mohaema segmenter - project.ui*
文件(F)
编辑(E)
显示 (D)
帮助 (H)
在这里输入
Text
File
Add a File
开始分词
添加词库
Mahaema
识别姓名
显示词性
欢迎使用Mahaema Segmenter!耗时:0.0000s
分词速度:0.0000词/s