

# 自然语言处理第二次大作业——自动拼写检查程序设计说明报告

## Automatic spell checker design specification report

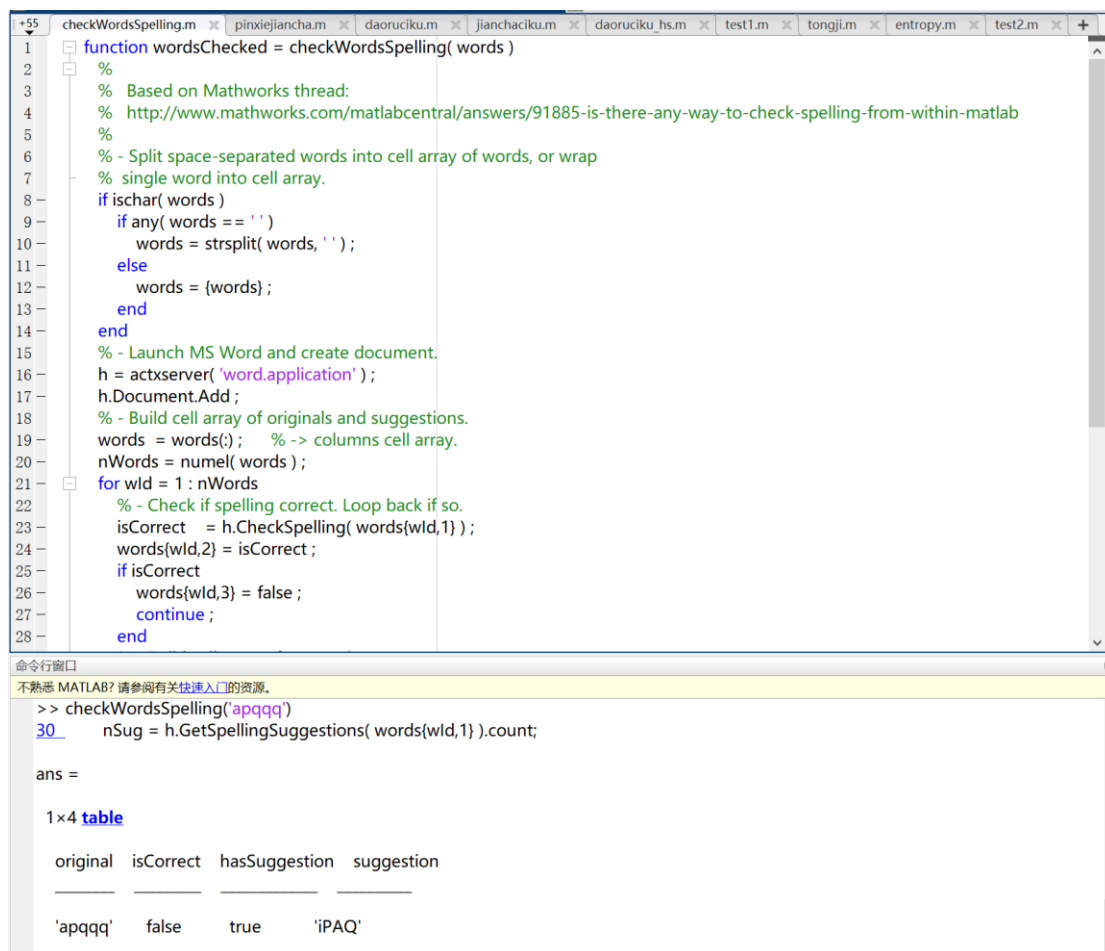
北京航空航天大学(BUAA) 自动化科学与电气工程学院(SASEE)

邵金鑫(Jinxin SHAO)

neushaojinxin@163.com

### 0.参考资料

查看相关例程，比如很常用的一个 matlab 的拼写检查程序,主要是是调用微软 word 里的拼写检查代码做错误判断，而且给拼写建议也用的是 word 里的拼写建议的接口，自己写了一个小循环，可能是取最有可能的给建议，看不到具体 dll 是怎么工作的，但是很有效。



```
1 function wordsChecked = checkWordsSpelling( words )
2 %
3 % Based on Mathworks thread:
4 % http://www.mathworks.com/matlabcentral/answers/91885-is-there-any-way-to-check-spelling-from-within-matlab
5 %
6 % - Split space-separated words into cell array of words, or wrap
7 % single word into cell array.
8 if ischar( words )
9     if any( words == ' ' )
10         words = strsplit( words, ' ' );
11     else
12         words = {words};
13     end
14 end
15 % - Launch MS Word and create document.
16 h = actxserver( 'word.application' );
17 h.Document.Add ;
18 % - Build cell array of originals and suggestions.
19 words = words(:); % -> columns cell array.
20 nWords = numel( words );
21 for wld = 1 : nWords
22     % - Check if spelling correct. Loop back if so.
23     isCorrect = h.CheckSpelling( words{wld,1} );
24     words{wld,2} = isCorrect ;
25     if isCorrect
26         words{wld,3} = false ;
27         continue ;
28     end
29 end
```

命令窗口

```
>> checkWordsSpelling('apqqq')
30 nSug = h.GetSpellingSuggestions( words{wld,1} ).count;

ans =

1x4 table

    original    isCorrect    hasSuggestion    suggestion
    _____    _____    _____    _____
    'apqqq'      false         true             'iPAQ'
```

### 1.词库下载并导入

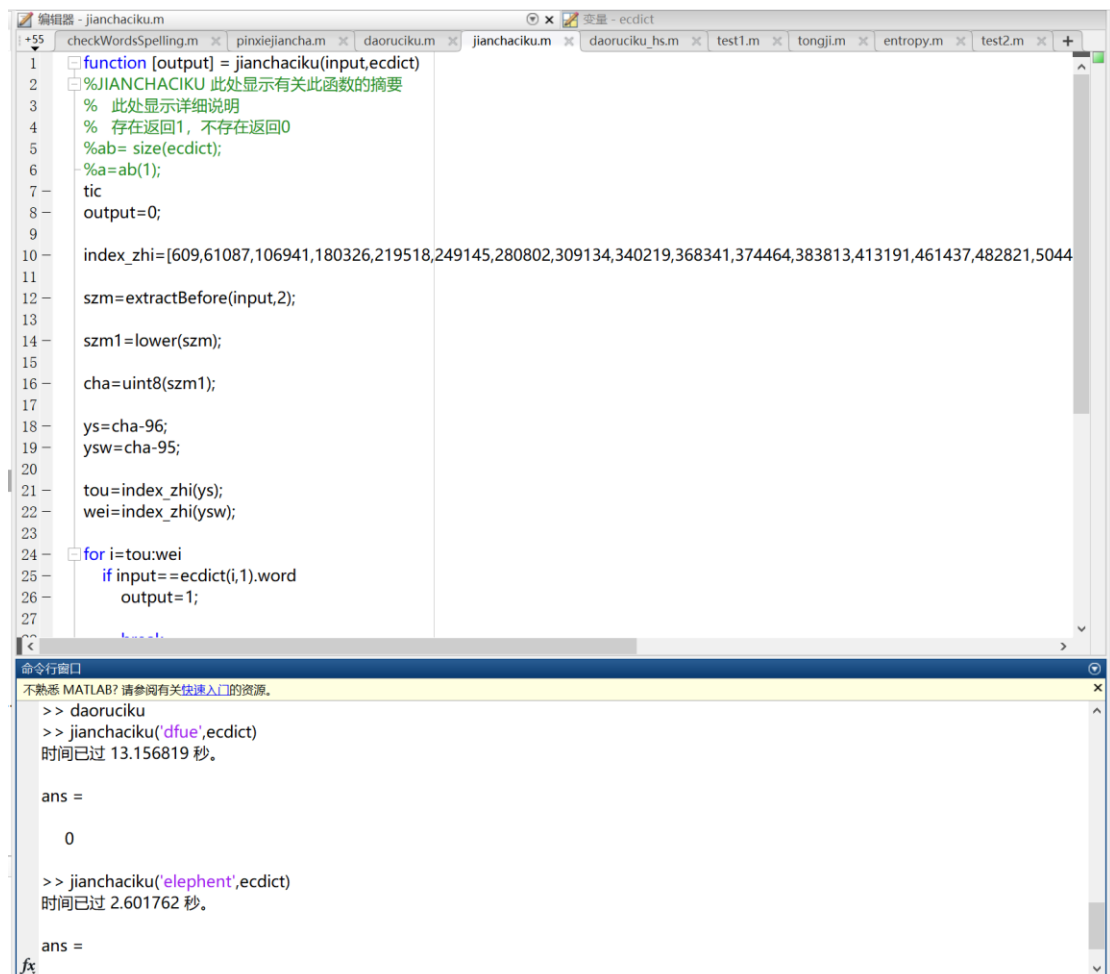
<https://github.com/skywind3000/ECDICT>

skywind3000 expand tab			Latest commit a0c44b8 13 days ago
LICENSE	Initial commit	2 years ago	
README.md	Fix typos.	28 days ago	
dictutils.py	initial commit	10 months ago	
ecdect.csv	initial commit	10 months ago	
ecdect.mini.csv	添加字典的mini版, 方便用户预览	8 months ago	
lemma.en.txt	initial commit	10 months ago	
linguist.py	initial commit	10 months ago	
resemble.txt	initial commit	10 months ago	
stardict.7z	initial commit	10 months ago	
stardict.py	expand tab	13 days ago	
wordroot.txt	initial commit	10 months ago	

770612x1 table															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1
1	"word"														
2	"hood"														
3	"'s Gra...														
4	"tween"														
5	"twee...														
6	"-ability"														
7	"-able"														
8	"-ably"														
9	"-acy"														
10	"-ad"														
11	"-ade"														
12	"-adelp...														
13	"-aemia"														
14	"-age"														
15	"-agog...														
16	"-aholic"														
17	"-al"														
18	"-algia"														
19	"-alia"														
20	"-ally"														
21	"-an"														
22	"-ance"														

词库共有 770612 个词，按字母表顺序排序，完全可以满足词库查找到的需求，然而没有词频数据，只有释义词根等信息，对我们做拼写检查没有帮助，所以不进行导入，则本工作无法实现基于先验贝叶斯概率的错误词频推荐。

## 2.暴力搜索检查程序



```
function [output] = jianchaciku(input,ecdect)
%JIANCHACIKU 此处显示有关此函数的摘要
% 此处显示详细说明
% 存在返回1, 不存在返回0
%ab= size(ecdect);
%a=ab(1);
tic
output=0;

index_zhi=[609,61087,106941,180326,219518,249145,280802,309134,340219,368341,374464,383813,413191,461437,482821,5044

szm=extractBefore(input,2);

szm1=lower(szm);

cha=uint8(szm1);

ys=cha-96;
ysw=cha-95;

tou=index_zhi(ys);
wei=index_zhi(ysw);

for i=tou:wei
    if input==ecdect(i,1).word
        output=1;
    end
end
```

```
>> daoruciku
>> jianchaciku('dfue',ecdect)
时间已过 13.156819 秒。

ans =

    0

>> jianchaciku('elephant',ecdect)
时间已过 2.601762 秒。

ans =
```

自己写了一个暴力搜索匹配的小程序在字库里查询，虽然已经做了一些前期工作，把 26 个字母的索引位置保存并根据查询字符的首字母进行指针位置开始查找，但速度还是比较慢，一个词不在词库里查找要 10 多秒时间，如果是 26 个字母都试一遍一共试 5 个字母就是 1300 多秒，这个显然是不能接受的，查询了 matlab 现有的库函数，发现 2 个库函数特别适合工作要求，而且工作速度非常快，大概 1,2 秒就能查完 26 个字母 7 万多个字库，**所以本次工作的全部查找都是基于这 2 个库函数进行。**

①`tf = strcmpi(wordip,ecdect.word);`

第一个是 `strcmpi` 不分大小写字符匹配函数，用这个来进行匹配，1-2 秒就能找出单词是否在词库里，是返回 1 不是返回 0。

②`expression =convertStringsToChars(shuchu1(i,1));`

`match = regexpi(ecdect.word,expression,'match');`

第二个是 `regexpi` 不分大小写特征匹配函数, 只要规定好 `expression` 里的特征, 就可以返回和 `expression` 里特征相符合的字典库里的所有字符, 作者通过 `mathwork` 说明文档发现了几个与本次工作特别匹配的特征用法。

如 `expression="^app.e$";`

本身一定要是 `char` 格式才方便操作和函数的接口匹配, 如果最开始是生成 `string` 格式, 则一定要转化成 `char` 格式再进行操作。`^` 字符的含义是匹配字符串的开始, `$` 字符串的含义是匹配字符串的结束, `.` 字符串的含义是只要是 26 个字母加数字都可以代表这个字符串, 所以上述字符串代表的特征是 `app?e` 以任意字母代表 `?` 的一个单词。通过这种方式, 就可以非常快速的进行匹配字符串在词库里的查找, 作者经过测试, 这种方法非常高效。

### 3.expression 特征表达小程序

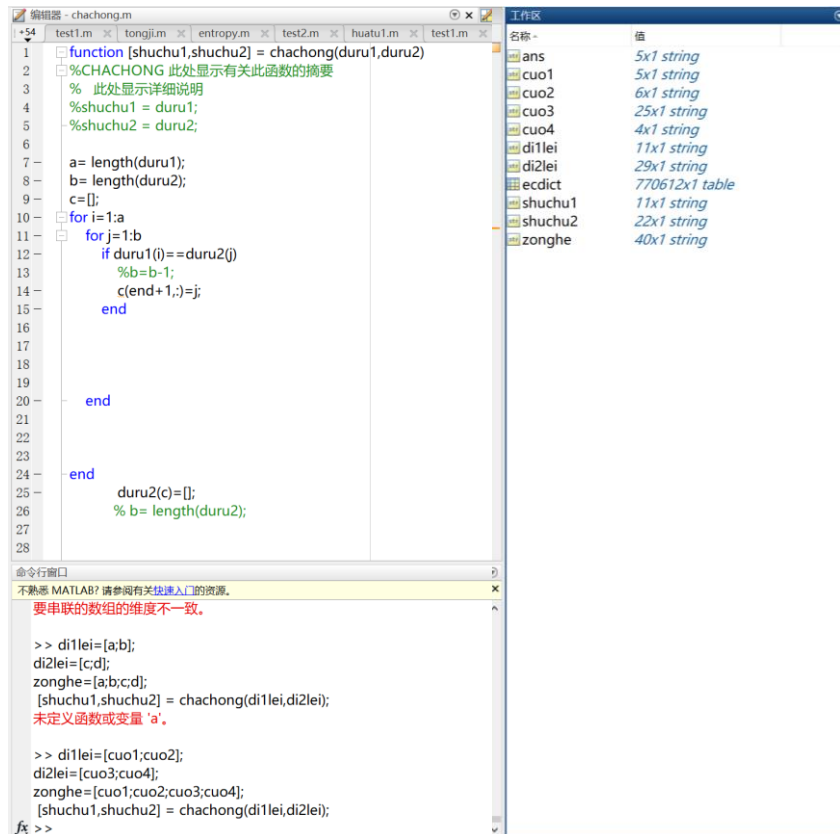
基于上述的思想, 作者写了一个 `function` 专门来生成不同错误所对应的 `expression` 表达, 实际上也是非常简单的操作, 就是往原有的字符串里, 加点, 去字符, 加头的 `^` 和尾部的 `$`。作者简单想了一下, 把错误分成了 4 种不同的类型, 第一种是输入的时候少打了一个字母, 就在原有字符的任意空档位置加上一个点; 第二种是输入的时候把一个字母打错了, 就去掉一个字母, 在原有位置加上一个点; 第三种是输入的时候即打错了一个字母, 又少打了一个字母, 就要即去掉一个字母, 在任意位置加上一个字母。很明显这种错误是包含错误 2 的, 但没有关系, 后期的处理中会将与错误二重复的部分去掉, 只留下不同的; 第四种错误是连续两个字母都打错了, 则要去掉 2 个字母, 加上 2 个点。

在后期的处理中, 12 类错误被归为 1 类, 属于比较可能出现的明显错误, 34 类错误被归为 1 类, 属于比较不容易出现的模糊错误, 关于什么时候开启模糊错误查找, 将在下一部分程序整体设计中进行说明。

进过测试, 这个函数花费时间 大概在 `0.00n` 秒的量级, 基本上没有时间损失。

程序运行结果如下, 如生成一个错误单词 `applq` 的 4 种错误特征类型, 作者为显示方便将 4 个字符串横向放在了一个 `string` 里, 实际生成的结果是 4 个字符串在 4 个字符串里, 这种字符串长度不同的横向拼接实际上是一种非法操作:





可以看出原来的 11,29 输入变为了 11,22 输出，7 种重复情况被删去了，效果很好。

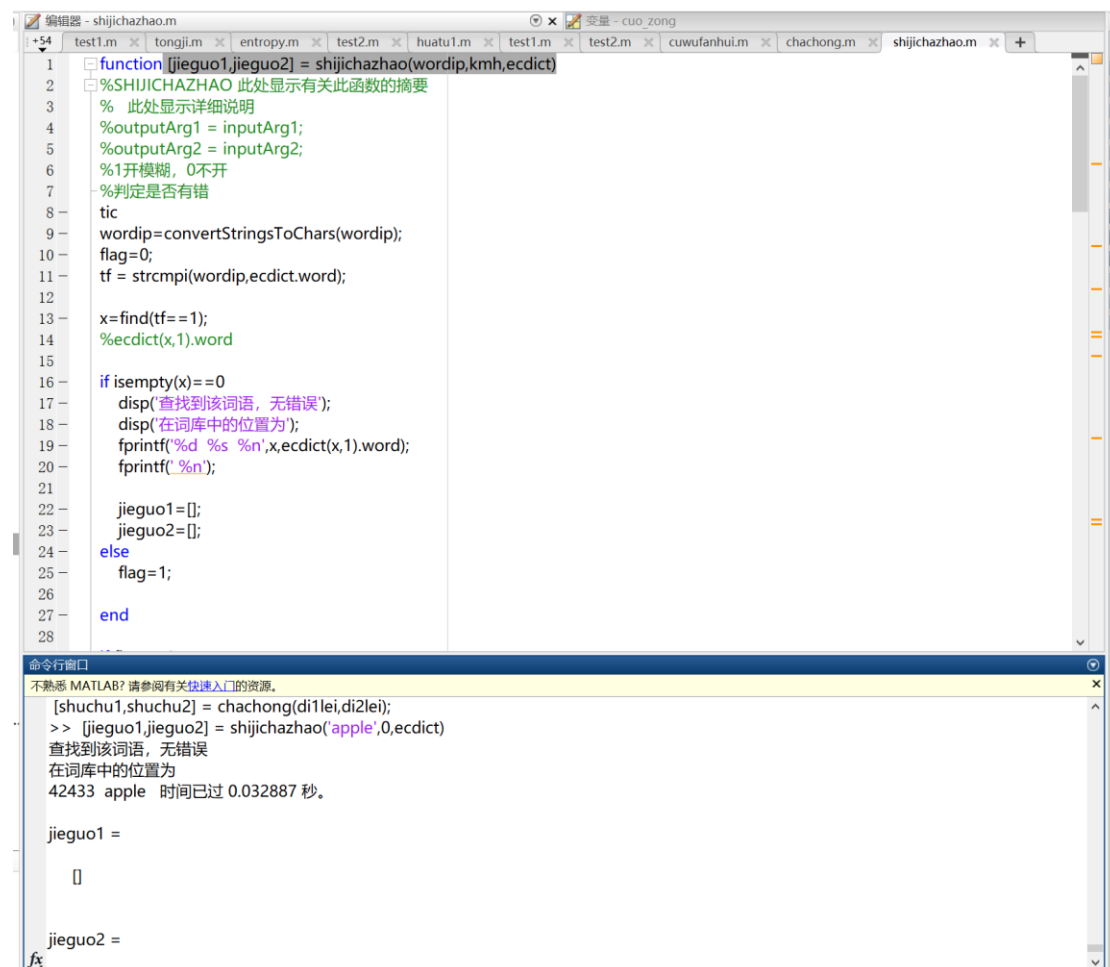
## 5.综合使用上述错误类型进行查找

这里就要规定查找的逻辑了，就是编程者的主要思想，由于 matlab 的运算速度是比较慢的，作者加了一个可以减少计算量的先验规定：

首先人要人工输入一个判断位 kmh（开模糊首字母），只有 kmh 为 1 才开启第二类错误查找，否则只进行第一类错位查找，并且除了开模糊判断位要为 1 以外，输入单词的长度至少要为 5 位及以上，才能进行模糊查找，否则试想一个单词只有 3 个字母或者 4 个字母，如果换掉 2 个字母，会得到非常多的结果。但有一个例外，一旦第一类错误没有找到解，不管是否开了模糊查找，也不管是否够 5 个字母，都要开启模糊查找找所有可能的解。

整体的实验测试如下，首先是正确字符，没有查找的测试，注明 jieguo1 和 jieguo2 虽然没有值但一定要给一个空输出，否则虽然程序可以运行但是会给 1

个 warning，可以看出，如果能直接匹配结果，这个计算时间是可以忽略的，以 apple 的查找为例，直接输出了在词库的 42433 行出现了单次 apple：



```
1 function [jiegua1,jiegua2] = shijichazhao(wordip,kmh,ecdect)
2 %SHIJICHAZHAO 此处显示有关此函数的摘要
3 % 此处显示详细说明
4 %outputArg1 = inputArg1;
5 %outputArg2 = inputArg2;
6 %1开模糊，0不开
7 %判定是否有错
8 tic
9 wordip=convertStringsToChars(wordip);
10 flag=0;
11 tf = strcmpi(wordip,ecdect.word);
12
13 x=find(tf==1);
14 %ecdect(x,1).word
15
16 if isempty(x)==0
17     disp('查找该词语，无错误');
18     disp('在词库中的位置为');
19     fprintf('%d %s %n',x,ecdect(x,1).word);
20     fprintf('%n');
21
22     jiegua1=[];
23     jiegua2=[];
24 else
25     flag=1;
26
27 end
28
```

```
命令窗口
不熟悉 MATLAB? 请参阅有关快速入门的资源。
[shuchu1,shuchu2] = chachong(di1lei,di2lei);
>> [jiegua1,jiegua2] = shijichazhao('apple',0,ecdect)
查找该词语，无错误
在词库中的位置为
42433 apple 时间已过 0.032887 秒。

jiegua1 =

    []

jiegua2 =
```

第二个测试，不开启模糊查找的第一类错误输出测试，使用了一个比较简单的例子 applr，找到了 4 个可能的第一类错误 appar，apple，appli 和 apply，花费了时间 38.027988 秒，这个时间还是比较长的，主要花费在比如第一类错误有 15 种，那么要逐次的对这 15 种模式进行查找匹配，可能是 15\*2 之类的一个计算量，作者尝试使用 parfor 操作，但是一般这种查找的循环是不支持并行的，因为 matlab 本质上还是一个科研工具，没有想到什么合适的办法进行优化计算：

```
>> [jiegua1,jiegua2] = shijichazhao('applr',0,ecdect)
输出第一类可能错误

jiegua1 =

4×1 cell 数组

{"appar"}
{"apple"}
{"appli"}
{"apply"}

未开启模糊查找（功能关闭或者字符不足5）
时间已过 38.027988 秒。

jiegua1 =

4×1 cell 数组

{"appar"}
{"apple"}
{"appli"}
{"apply"}

jiegua2 =

[]

fx >> |
```

同样的测试，还是刚才的例子，开启模糊查找之后，观察查找结果和查找时间，如下所示，可以看出，除了刚才 4 种的第一类错误，还有 19 种有可能的第二类错误，当然和第一类错误是不重复的，但是整个查询时间翻了 3 倍多，变成了 112.68 秒，时间可以说是很长了。

```
>> [jiegua1,jiegua2] = shijichazhao('applr',1,ecdect)
输出第一类可能错误

jiegua1 =

4×1 cell 数组

{"appar"}
{"apple"}
{"appli"}
{"apply"}

输出第二类可能错误（模糊）

jiegua2 =

19×1 cell 数组

{"tappr"}
{"appro"}
{"appal"}
{"appel"}
{"aggar"}
{"appar"}
{"apter"}
{"appal"}
{"appar"}
{"appel"}
{"appen"}
{"appet"}
{"appia"}
{"apple"}
{"appli"}
{"apply"}
{"appro"}
{"apptd"}
{"appui"}

fx 时间已过 112.681487 秒。
```



最后加大难度，测试作者编写一种保险情况，当第一类错误的查询结果为空时，强行开启第二类错误的查找，作者在刚才的例子上加了一个改进，将 `applr` 改为 `applrr`。很明显，这种 3 个辅音字母在一起的错误是不可能通过修改 1 个字母得到改进的。

```
>> [jieguo1,jieguo2] = shijichazhao('applrr',0,ecdict)
输出第一类可能错误

jieguo1 =

空的 0×1 cell 数组

第一类错误未找到解，强行开启第二类错误模糊查找
输出第二类可能错误（模糊）

jieguo2 =

5×1 cell 数组

{"appear"}
{"apples"}
{"applet"}
{"applin"}
{"applud"}

时间已过 150.852544 秒。

jieguo1 =

空的 0×1 cell 数组

jieguo2 =

5×1 cell 数组

{"appear"}
{"apples"}
{"applet"}
{"applin"}
{"applud"}

fx >> |
```

可以看出结果是奏效的，得到了可能出现的第二类错误单词表为 `appear`，`apples`，`applet`，`applin` 和 `applud`。但是由于比上一个测试还要多一个字母，查询时间变为了 150 秒，这是一个基本上我们不能再做任何算法改进的时间，因为设计不同的查询规则乃至非常复杂的查询规则实际上是很容易的，但是 `matlab` 机制导致如果在现行基础上把规则变得更为复杂，可能查询需要的时间要进行几倍或者 10 几倍的增加，而这种增加显然是不能被接受的，除非能不使用 `for` 循环或者使用 `parfor` 并行 `for` 循环，但是在查询工作中这种思路很明显是不现实的，作者在多次的 `matlab` 编程中确实出现过一个程序要运行半个小时乃至 1 个小时的情况，但在这样的一个问题中显然是不可接受的，所以说现有的查询方式应该是针对目前可行问题的一种较优方法。

## 6.总结

这次工作由于没有找到带先验词库概率的数据库，所以没法计算贝叶斯概率，如果能和贝叶斯概率结合则预测的结果绝对会更有说服力。此外，从作者本身的编程经验乃至上次的分词信息熵计算编程结果来看，在涉及一些比较新型的领域，比如 nlp，matlab 本身不具备非常好的支持库，并且计算速度已经不能满足新型领域的要求，对于同样一个问题，python1 分钟可能就能解决的问题，matlab 可能要花上十几分钟乃至半小时一小时。缺乏必要的支持库，快速计算能力，现在从 matlab 转向 python 已经是大势所趋，希望能通过这次小程序编写的锻炼能尽快促进转向 python 的编程。