19011402 南佳霖

摘要

本文以python3.9环境为基础，以AMiner网站为爬取对象建立语料库，在实现正向、逆向最大匹配算法的基础上，以jiaba分词为标准验证准确率，完成中文文本分词目标

正向、逆向最大匹配算法实现中文分词

——Python实现

目录

[一． 引言 1](#_Toc85682548)

[1.1 研究背景 1](#_Toc85682549)

[1.2 技术概况 1](#_Toc85682550)

[1.3 作业目录 1](#_Toc85682551)

[二． 建立语料库 2](#_Toc85682552)

[2.1文本爬虫 2](#_Toc85682553)

[2.2文本预处理 3](#_Toc85682554)

[三． 算法实现 4](#_Toc85682555)

[3.1 算法思想 4](#_Toc85682556)

[3.2 实现细节 4](#_Toc85682557)

[四． 结果分析 5](#_Toc85682558)

[4.1 jieba分词 5](#_Toc85682559)

[4.2 P值 5](#_Toc85682560)

[4.3 R值 5](#_Toc85682561)

[4.4 F值 5](#_Toc85682562)

[4.5结果展示 6](#_Toc85682563)

[4.6 结果分析 6](#_Toc85682564)

[五． 优化策略 6](#_Toc85682565)

[六． 实验总结 6](#_Toc85682566)

[附：源代码A——爬虫 1](#_Toc85682567)

[附：源代码B——正向最大匹配算法 2](#_Toc85682568)

[附：源代码C——逆向最大匹配算法 4](#_Toc85682569)

[附：源代码D——评价程序 5](#_Toc85682570)

# 引言

## 研究背景

随着互联网技术的快速发展，谷歌、必应、百度等通用搜索引擎成为互联网世界的重要组成部分，其核心技术是基于自然语言处理的全文检索技术。在中文的句子中，词与词之间不使用分隔符或空格，这使得计算机对于词的准确识别变得特别困难，想建立关键词的索引必须首先使用分词技术进行句子中词语的切分。

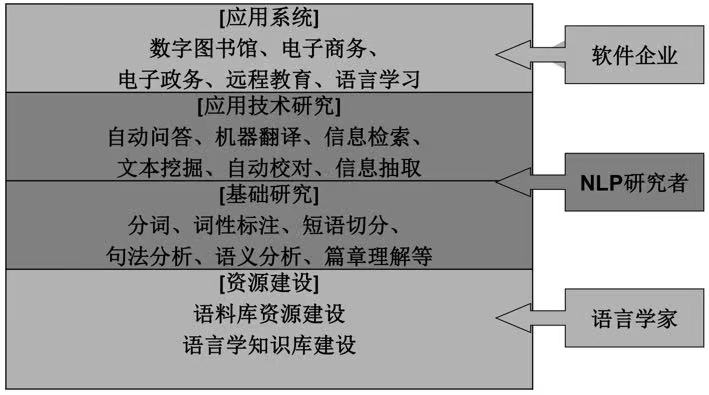


图1-1 NLP的不同层次

所以，中文分词是语言信息处理的基础性工作，是语言机器翻译、智能检索、自然语言理解与处理等智能信息应用的前提。分词系统的好坏将直接影响以此为基础的智能处理系统性能的优劣。

## 技术概况

### 基于规则词典的中文分词法

又叫做机械分词方法，亦称为基于字符串匹配的方法。该方法按照一定的策略将待分析的汉字串与词典中的词条进行配，若在词典中找到对应字串，则匹配成功。

其中，一个简单的词典可以用列表结构来实现，如线性表、哈希表等。关键在于字典数据的查询效率，考虑到汉语由多个语素构成此，按照一定的顺序逐一进行匹配，故可以用树形结构构建中文词典。

字典树又称为Trie树、单词查找树或前缀树，是一种用于快速检索的多叉树型结构，插入、查找的时间复杂度均为O(N)。

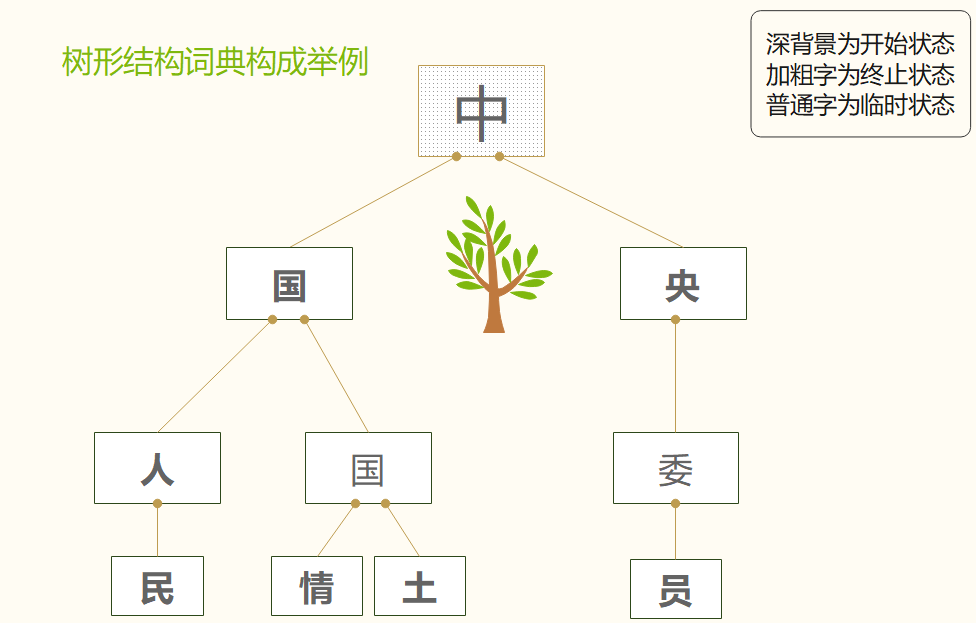


图1-2 中文Trie树

按照扫描方向的不同，串匹配分词方法可以分为正向匹配（从左到右）和逆向匹配（从右到左）；按照不同长度优先匹配的情况，亦可以分为最大匹配和最小匹配。由于汉语的语法特点，一般不使用最小匹配法，故常用的种机械分词方法有以下两种：

1）正向最大匹配法（FMM）；

2）逆向最大匹配法（RMM）；

此次实验基于以上两种方法进行分词处理，在第二部分详细说明。

### 基于统计的分词法

这种方法只需对语料中的字组频度进行统计，不需要切分词典，因而又叫做无词典分词法或统计取词方法。通过给出的大量已经分词的文本，利用机器学习模型学习词语切分的规律（训练），从而实现对未知文本的切分。随着大规模语料库的建立，统计机器学习方法的研究和发展，基于统计的中文分词方法渐渐成为了主流方法。

主要统计模型有：

1. N元文法模型（N-gram）
2. 隐马尔可夫模型（Hidden Markov Model ，HMM）
3. 最大熵模型（ME）
4. 条件随机场模型（Conditional Random Fields，CRF）

## 作业目录

本次实验源代码及数据同步在GitHub；作业文件夹以“NLP\_Lab2\_19011402\_南佳霖”命名，主目录中包括：

1. 本实验报告
2. Pycharm2020环境下的总工程文件包
3. 爬取的.txt语料文件包 all.txt
4. 预处理后的语料all\_without\_punctuation.txt
5. 预处理源代码preProcess.py
6. FMM源代码FMM.py
7. RMM源代码RMM.py
8. 评价函数源代码Evalute.txt
9. Readme.md

# 建立语料库

获取中文语料数据、建立语料库是进行分词的基础和前提。而通过手动录入或复制粘贴的方法效率极低且质量低下。网络爬虫技术易学易用，可以在互联网中自动爬取语料数据，基于此技术可以迅速高效地建立语料库。

此实验中，采用txt文件的形式存储生语料（text\_lib文件夹中）。

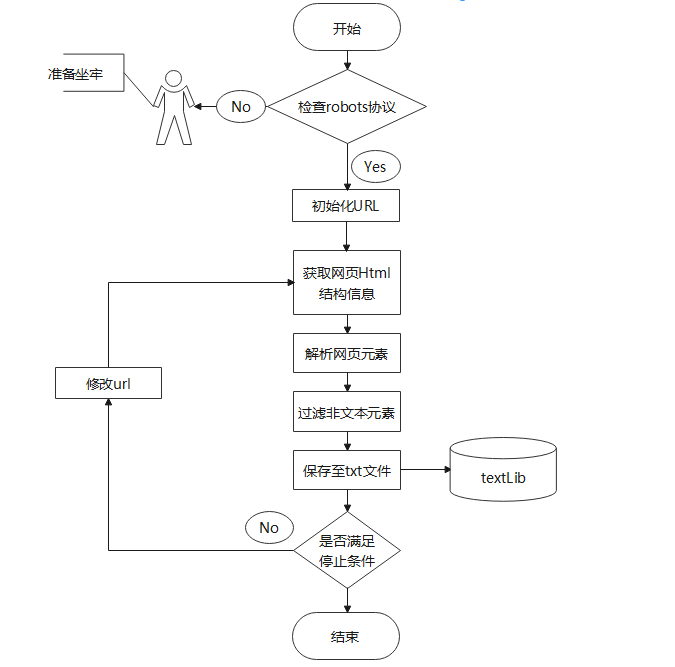
## 2.1文本爬虫

### 2.1.1 工作原理

网络爬虫是一段爬取网页内容的程序。

在联网的状态下，爬虫将发起网络请求访问某一网站，若能够访问则自动下载保存其中的内容。然后通过爬虫的解析模块，解析得到该取页面中的元素，包括文字、图片、链接等。得益于HTML页面的结构化设计，爬虫在分析网页结构时可以完全不依赖于用户手动干预就可以实现自动运行。

### 2.1.2 工作流程



3-2 爬虫工作流程

### 2.1.3 爬取对象

在实验一中，完成了英文电影网站[IMDb](https://www.imdb.com/title/tt0111161/reviews?ref_=tt_urv)的影评爬取。此次实验，则选用中文网站[AMiner](https://www.aminer.cn/)为爬取对象以建立中文语料库。相对于“AI报告”、“会议论文”等板块，“趋势分析”板块中文章内容英文及术语简写较少，更适合建立中文语料库，所以选择爬取“趋势分析”板块内文章。



图3-1 AMiner网站

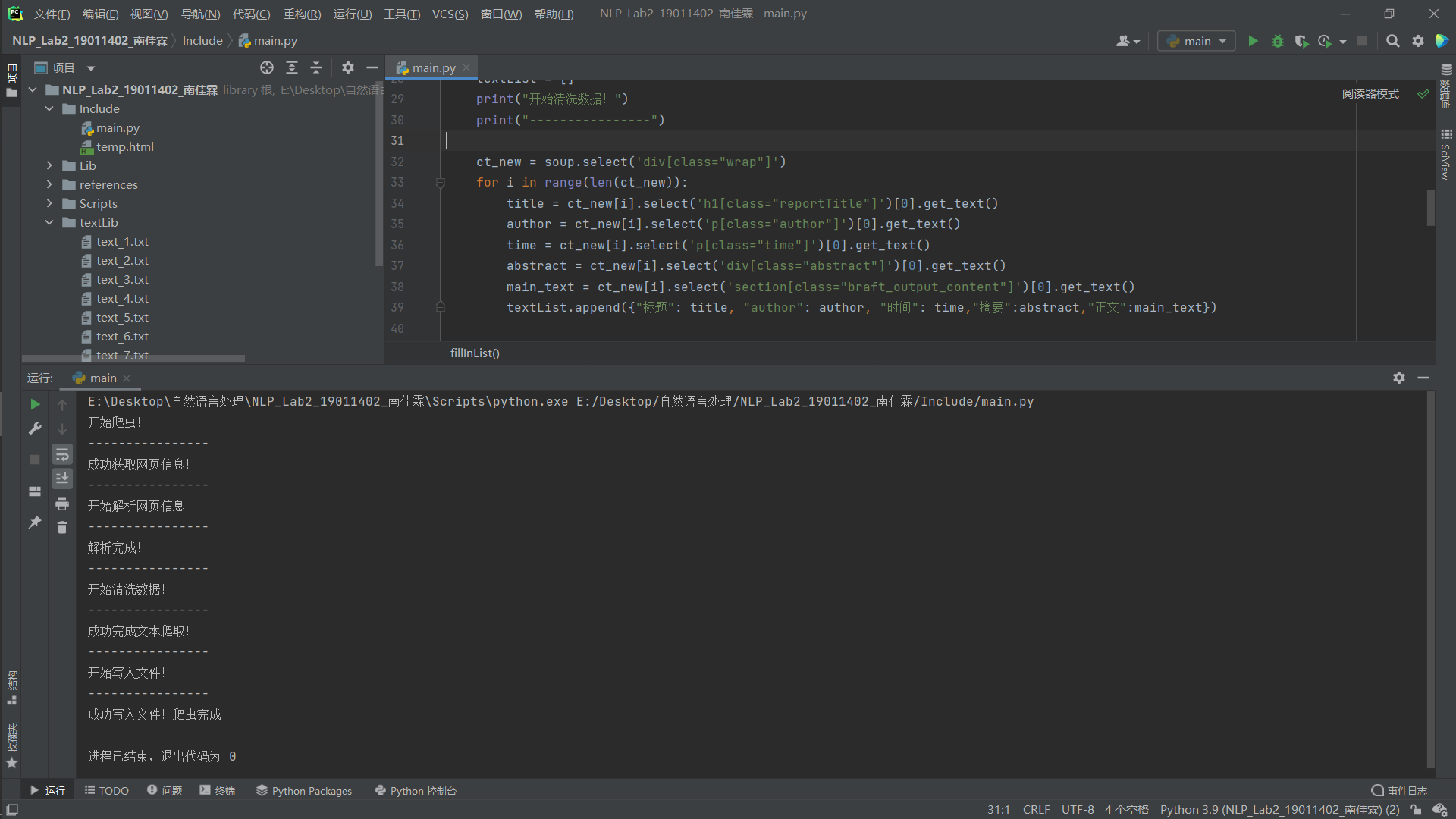
Aminer是一个科技情报大数据挖掘与服务系统平台，由清华大学计算机科学与技术系教授唐杰率领团队建立，平台提供信息工程和AI领域的最新资讯。

### 2.1.4 实现细节

初始化url后，使用requests库进行网络请求，返回一个response对象。在访问的过程中进行火狐浏览器伪装 *kv = {'user-agent': "Mozilla/5.0"} ，*以及异常处理*r.raise\_for\_status( )* （如果状态不是200.则引发 HTTPError 异常），避免因为网络情况导致的爬取失败问题。

在解析页面后，观察网页源码发现，所需语料信息div的class值为“wrap”，使用 Beautiful Soup库的select方法过滤出该标签内容信息。遍历所有标签，定义一个list型变量存储title、time等信息。

将每一条评论语料存储到 .txt文件中（utf-8） *file = open("reviews.txt", "a", encoding="utf8")*



3-2 程序运行结果演示

## 2.2文本预处理

### 2.2.1文件合并

除了可以个别处理语料，Python的优势之一在于可以自动化批量处理多个文件。本实验中，可以批量合并文档，便于统计文件总字符数，提高样本体量。合并过程如下：

1. 引入glob库，对语料库\内文件进行遍历
2. 新建data存放所有数据
3. 使用readlines( )读取所有行
4. 将data写入文件all.txt

### 2.2.2去除标点

由于爬取的预料并不是纯中文，其中夹杂着大量的标点符号西洋字符，在实现字符匹配算法时要考虑如何处理这些意外符号。

第一种策略是：将符号也作为字进行切分。然而，对于符号的切分并不能体现匹配算法的准确的，在进行评价时，由于大量符号的“正确“切分，会导致算法评价高于实际的情况。

第二种策略是：采用算法去除标点符号，过滤出纯文本或仅带有少量标点的文本。本次实验采用此策略。去除标点过程如下：

* 1. 输入需要处理的字符串
  2. 将想去除的标点建为一个列表
  3. 遍历字符串，若字符不在符号列表中，就将其添加到新列表中
  4. 遍历完成后，将新列表保存至txt文件

# 算法实现

## 3.1 算法思想

### 3.1.1正向最大匹配算法

假设包含在词典中最长词条所含的字的数量为i，然后取当前字符串文本中的前n 个字符作为匹配字段， 在分词词典中进行查找，如果分词词典中有这样的 n 字词，则匹配成功，匹配字段被切分出来作为一个词; 如果在分词词典中没有找到这样的 n 字词，则匹配失败，删除匹配字段中的最后一个字，剩下的字 符作为新的匹配字段，继续匹配，直到匹配成功。

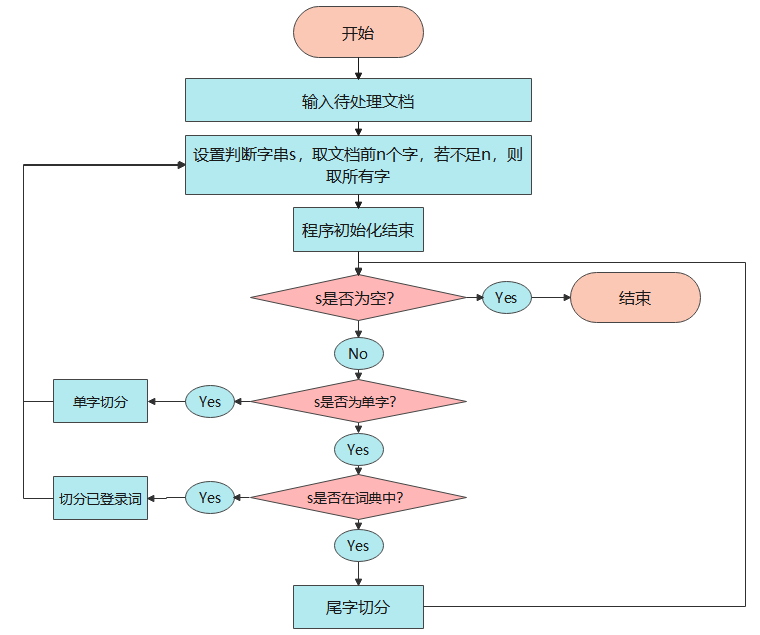


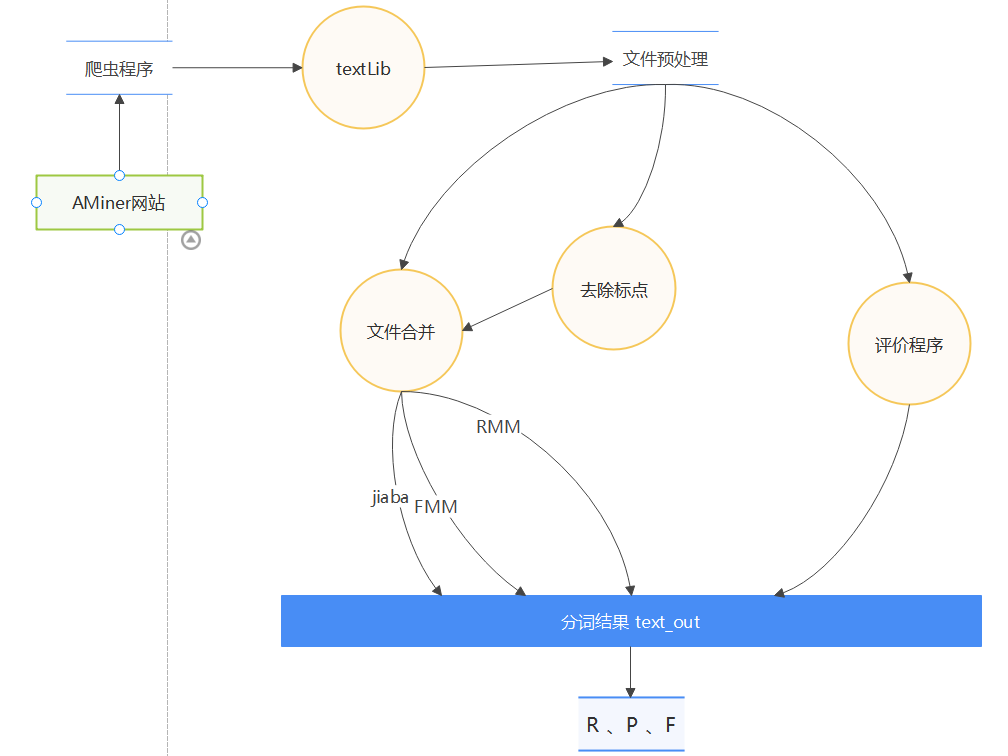
图3-1 FMM流程图

### 3.1.2逆向最大匹配算法

基本过程与正向最大匹配算法相同，取后n个字符作为匹配字段，逆向查找。

## 3.2 实现细节

### 3.2.1流程图



3-2 软件流程图

### 3.2.2词典的使用

本实验第一阶段选用wordlist.Dic作为分词词典。关于Dic文件如何使用，很遗憾我没有找到合适的资料。

于是在本次实验中，将wordlist.Dic转化为txt文件作为词典。这样做即解决了Dic文件无法遍历的问题，也方便扩展词典。

### 3.2.3 评价程序思路

1. 首先对生成的文本和jieba分词文本通过切分形成词汇表，然后对两个词汇表从第一个词开始比较：
2. 如果当前词汇相同，表明结果正确，且之前的词汇拼成的字符串长度相等；
3. 如果当前词汇不同，继续取词汇拼字符串直到两个字符串长度相同；依次进行处理，计算R\R值。

# 结果分析

## 4.1 jieba分词

### 4.1.1 jieba库

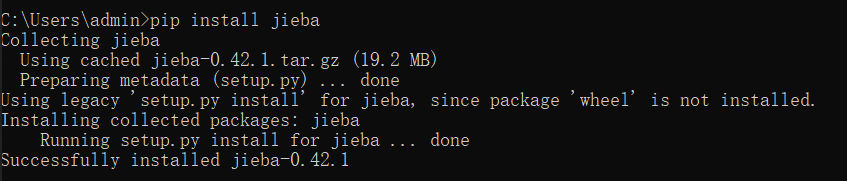
[“结巴”库](https://github.com/fxsjy/jieba)是一款非常优秀的中文分词工具，它支持四种分词模式：

* 精确模式：将句子最精确地切开，适合文本分析；
* 全模式，把句子中所有的可以成词的词语都扫描出来, 速度非常快，但是不能解决歧义；
* 搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。
* paddle模式，利用PaddlePaddle深度学习框架，训练序列标注（双向GRU）网络模型实现分词。同时支持词性标注。

本次实验中，采用jieba分词的结果作为标准。

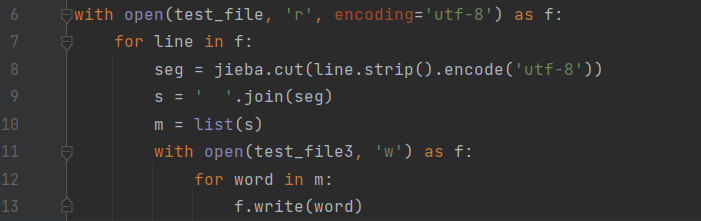
### 4.1.2 安装与使用

在控制台输入pip install jieba ，出现如下界面即安装好jieba分词工具。



4-1 使用pip安装jieba库

jieba.cut 方法可自动将字符串切分，返回的结构是一个可迭代的 generator，可以使用 for 循环来获得分词后得到的每一个词语(unicode)。



4-2 调用jieba.cut进行自动分词

## 4.2 P值

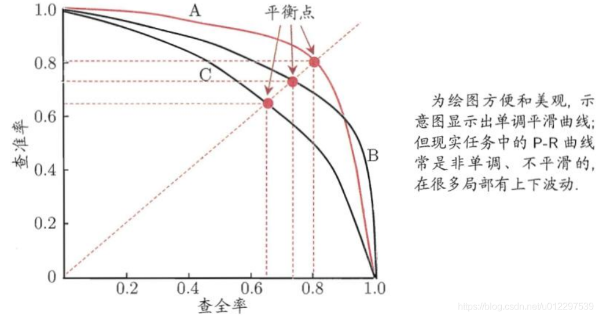
* 正确率（Precision）：测试结果中正确的结果个数占系统所有输出结果的比例。

## 4.3 R值

* 召回率（Recall ratio）：测试结果中正确结果过的个数占标准答案总数的比例。

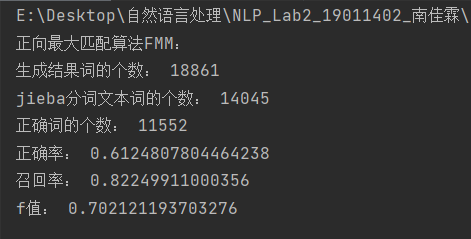
## 4.4 F值

* F值（F-Measure）：在某些情况下，P和R是矛盾的， F是正确率与召回率的综合值。

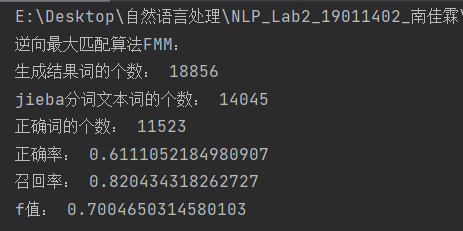


4-4 P-R图

## 4.5结果展示



4-4 FMM算法分析



4-5 RMM算法分析

## 4.6 结果分析

FMM和RMM方法较容易实现，但存在着明显缺点

1. 对歧义字段难以正确切分，无法发挥上下文作用
2. 对于MaxLen的长度很难确定
3. 过分依赖词典规模，且词典主题也与分词效果有关

# 优化策略

基于词典的分词方法通常会遇到切分歧义和未登录词识别的问题。歧义消解和未登录词识别是匹配算法的主要优化方法。其中，未登录词又分为仿词、命名实体和新词。对于这三种不同的类别，有着不同的优化策略。

此次实验中，主要以加入未登录词扩大词典规模为主要优化手段。

# 实验总结

代码运行的速度慢得离谱，这是我没想到的，甚至一度以为自己交不上这次作业了。现在是凌晨三点四十一分，终于写到了实验总结，激动得差点哭出来。

一个人做lab的工作量确实有些大，却的确是学到了些东西的。借这个机会看了些舆情分析的资料，没准真的可以做毕设hh

实验的重点放在了实现算法进行分词，对于一些优化策略如“双向匹配”并没有深入思考，希望下几次实验能再完整些叭。

# 附：源代码A——爬虫

**import** **requests**

**from** **bs4** **import** BeautifulSoup

**def** getHtmlText(url):

**try**:

kv = {'user-agent': "Mozilla/5.0"} *# 修改头部信息，伪装成火狐访问*

r = requests.get(url, headers=kv, timeout=30)

r.raise\_for\_status() *# 如果状态不是200.则引发HTTPError异常*

r.encoding = r.apparent\_encoding

print("成功获取网页信息！")

print("----------------")

**return** r.text

**except**:

print("爬取失败!")

print("----------------")

**return** ""

**def** fillInList(html):

print("开始解析网页信息")

print("----------------")

soup = BeautifulSoup(html, "html.parser") *# 得到一个BeautifulSoup的对象(标准的缩进格式)*

print("解析完成！")

print("----------------")

textList = []

print("开始清洗数据！")

print("----------------")

ct\_new = soup.select('div[class="wrap"]')

**for** i **in** range(len(ct\_new)):

title = ct\_new[i].select('h1[class="reportTitle"]')[0].get\_text()

author = ct\_new[i].select('p[class="author"]')[0].get\_text()

time = ct\_new[i].select('p[class="time"]')[0].get\_text()

abstract = ct\_new[i].select('div[class="abstract"]')[0].get\_text()

main\_text = ct\_new[i].select('section[class="braft\_output\_content"]')[0].get\_text()

textList.append({"标题": title, "author": author, "时间": time,"摘要":abstract,"正文":main\_text})

print("成功完成文本爬取！")

print("----------------")

**return** textList

**def** storeInFiles(textList):

print("开始写入文件！")

print("----------------")

num = 9

file = open("..**\\**textLib**\\**text\_" + str(num) + ".txt", "w", encoding="utf8") *# 文件命名*

**for** each **in** textList:

file.write(str(each)) *# 存入.txt文件*

file.close()

print("成功写入文件！爬虫完成！")

textList.clear()

**if** \_\_name\_\_ == "\_\_main\_\_":

print("开始爬虫！")

print("----------------")

url1= "https://www.aminer.cn/research\_report "

html = getHtmlText(url9) *# 使用request库函数获取网页信息*

textList = fillInList(html) *# 使用BS4库函数进行页面解析*

storeInFiles(textList)

# 附：源代码B——正向最大匹配算法

1. word\_list = '../wordlist.txt' # 词典
2. test\_file2 = '../all\_without\_punctuation.txt' # 测试语料
3. test\_file3 = '../text\_out/text\_out\_FMM.txt' # 生成结果

6. def get\_dic(test\_file): # 读取文本返回列表
7. with open(test\_file, 'r', encoding='utf-8', ) as f:
8. **try**:
9. file\_content = f.read().split()
10. **finally**:
11. f.close()
12. chars = list(set(file\_content))
13. **return** chars

16. # 读取词典
17. dic = get\_dic(word\_list)

20. def readfile(test\_file2):
21. max\_length = 5
23. h = open(test\_file3, 'w', encoding='utf-8', )
24. with open(test\_file2, 'r', encoding='utf-8', ) as f:
25. lines = f.readlines()
27. **for** line in lines: # 对每行进行正向最大匹配处理
28. max\_length = 5
29. my\_list = []
30. len\_hang = len(line)
31. **while** len\_hang > 0:
32. tryWord = line[0:max\_length]
33. **while** tryWord not in dic:
34. **if** len(tryWord) == 1:
35. **break**
36. tryWord = tryWord[0:len(tryWord) - 1]
37. my\_list.append(tryWord)
38. line = line[len(tryWord):]
39. len\_hang = len(line)
41. **for** t in my\_list: # 将分词结果写入生成文件
42. **if** t == '**\n**':
43. h.write('**\n**')
44. **else**:
45. h.write(t + " ")
47. h.close()
49. **if** \_\_name\_\_ == "\_\_main\_\_":
50. readfile(test\_file2)

# 附：源代码C——逆向最大匹配算法

1. word\_list = '../wordlist.txt' # 词典
2. test\_file2 = '../all\_without\_punctuation.txt' # 测试语料
3. test\_file3 = '../text\_out/text\_out\_FMM.txt' # 生成结果

6. def get\_dic(test\_file): # 读取文本返回列表
7. with open(test\_file, 'r', encoding='utf-8', ) as f:
8. **try**:
9. file\_content = f.read().split()
10. **finally**:
11. f.close()
12. chars = list(set(file\_content))
13. **return** chars

16. # 读取词典
17. dic = get\_dic(word\_list)

20. def readfile(test\_file2):
21. max\_length = 5
23. h = open(test\_file3, 'w', encoding='utf-8', )
24. with open(test\_file2, 'r', encoding='utf-8', ) as f:
25. lines = f.readlines()
27. **for** line in lines: # 对每行进行正向最大匹配处理
28. max\_length = 5
29. my\_list = []
30. len\_hang = len(line)
31. **while** len\_hang > 0:
32. tryWord = line[0:max\_length]
33. **while** tryWord not in dic:
34. **if** len(tryWord) == 1:
35. **break**
36. tryWord = tryWord[0:len(tryWord) - 1]
37. my\_list.append(tryWord)
38. line = line[len(tryWord):]
39. len\_hang = len(line)
41. **for** t in my\_list: # 将分词结果写入生成文件
42. **if** t == '**\n**':
43. h.write('**\n**')
44. **else**:
45. h.write(t + " ")
47. h.close()
49. **if** \_\_name\_\_ == "\_\_main\_\_":
50. readfile(test\_file2)

# 附：源代码D——评价程序

1. test\_file = '../text\_out/text\_out\_FMM.txt'
2. test\_file2 = '../text\_out\_jieba.txt'
3. def get\_word(fname):
5. f = open(fname,'r',encoding='utf-8')
6. lines = f.readlines()
8. **return** lines

11. def calc():
12. lines\_list\_sc = get\_word(test\_file)
13. lines\_list\_gold = get\_word(test\_file2)
15. lines\_list\_num = len(lines\_list\_gold)
17. right\_num = 0
18. m = 0#m存逆向结果文本词的总数
19. n = 0#n存gold文本词的总数
21. **for** i in range(lines\_list\_num):
23. line\_list\_sc = list(lines\_list\_sc[i].split())#line\_list\_sc为生成结果每行通过空格切分后的词汇表
24. line\_list\_gold = list(lines\_list\_gold[i].split())#line\_list\_gold为正确结果每行通过空格切分后的词汇表
26. m += len(line\_list\_sc)
27. n += len(line\_list\_gold)
29. str\_sc = ''#存结果文本每行无空格连接起来的字符串
30. str\_gold = ''#存gold文本每行无空格连接起来的字符串
32. s = 0#表示结果文本每行列表的下标
33. g = 0#表示gold文本每行列表的下标
35. **while** s < len(line\_list\_sc) and g < len(line\_list\_gold):
36. str\_word\_sc = line\_list\_sc[s]
37. str\_word\_gold = line\_list\_gold[g]
39. str\_sc += str\_word\_sc
40. str\_gold += str\_word\_gold
42. **if** str\_word\_sc == str\_word\_gold:#如果当前词汇相同，表明结果正确，且之前的词汇拼成的字符串长度相等
43. s += 1
44. g += 1
45. right\_num += 1
47. **else**:#如果当前词汇不同，结果错误，不断取词汇拼字符串直到两个字符串长度相同
49. **while** len(str\_sc) > len(str\_gold):
50. g += 1
51. str\_gold += line\_list\_gold[g]

54. **while** len(str\_sc) < len(str\_gold):
55. s += 1
56. str\_sc += line\_list\_sc[s]
58. g += 1
59. s += 1
60. print("逆向最大匹配算法FMM：")
61. print("生成结果词的个数：", m)
62. print("jieba分词文本词的个数：", n)
63. print("正确词的个数：", right\_num)
64. p = right\_num/m
65. r = right\_num/n
66. f = 2\*p\*r/(p+r) # 当beta=1时
67. print("正确率：", p)
68. print("召回率：", r)
69. print("f值：", f)