论文的关键词有着其特殊的重要使命，首先当然是方便别人浏览，可以一目了然的知道论文论述的主题，从而决定是否要花费时间阅读正文，节约大家的时间；其次也是更重要的一点，能够方便论文的归类和搜索。所以对待任意一段文本，如果我们能快速得到它的关键词，也就能达到和论文一样的效果。本demo用python语言结合jieba分词库+urllib爬虫库+beautifulsoup的html分析工具实现了tf-idf法判断文本关键词的效果，有些地方显的略为复杂是因为加入了自己对文章关键词选取的个人理解，接下来和大家分享。

#coding:utf-8

import jieba

jieba.load\_userdict("myDictionary.txt")

import jieba.posseg as psg

from collections import Counter

from urllib import request

from urllib.parse import quote

from bs4 import BeautifulSoup

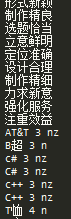
import string

import chardet

import math

import operator;

开头给出程序的编码和一些import的操作。load的txt文件是自建的词汇字典，是在jieba原始字典的基础上加入一些自己将要查看的文章有一定可能要用到的一些词汇，它们有的是专业词汇，有的是两三个词汇合在一起构成的词汇（但是在你将要涉猎的论文主题经常出现），因为代码是为有关博物馆的文章所写，所以myDictionary.txt的内容基本上如下截图：



按照jieba给出的官方说明，一个词一行，可以在词后边加上词频和词性，以空格隔开。接下来初始化了一堆接下来将要用到的变量，这里逐一介绍：

s = u''

f = open('2.txt',encoding='UTF-8')

line = f.readline()

while line:

s=s+line

line = f.readline()

f.close()

S保存了从txt读进的待操作的文本内容，当然2.txt就是你想要操作的文章

importantWord=[]

f = open('moreAttentionWord.txt',encoding='UTF-8')

line = f.readline()

while line:

importantWord.append(line)

line = f.readline()

f.close()

importantWord保存着重要的词汇，和mydictionary新加入的一样就是那些你依仗自己的经验感觉有可能会出现的东西（无论长短），只不过上边的dictionary是给jieba用的，这个importantword我是用来给自己后面处理分出来的词汇用的。

stopWord=[]

f = open('stopwords.txt',encoding='UTF-8')

line = f.readline()

while line:

stopWord.append(line.split('\n')[0])

line = f.readline()

f.close()

stopword保存着停用词，jieba会根据你的stopwords.txt里的词来对分词出的结果进行筛选，一般保存着如下截图中的这种没有实际意义的词汇。



word=[]

cixing=[]

phrasestore=[]

nstore=[]

vstore=[]

sstore=[]

剩下这几个list分别是一会用来保存东西的中间容器，后边再说。

store=[(x.word,x.flag) for x in psg.cut(s)]

for x in store:

if x in stopWord:

continue

word.append(str(x).split(',')[0].split("'")[1])

cixing.append(str(x).split(',')[1].split("'")[1])

首先利用psg的cut将文本s的内容分词，按照jieba官方给出的格式store链表将会以 词，词性 的形式保存下来（没错，就是这么为所欲为就分好了）。然后对分好的词进行处理，如果在停用词表里就不要这个词，否则把对应的词和词性装入word和cixing链表。

for i in range(1,len(word)):

if cixing[i]=='n' and cixing[i-1]=='n' and len(word[i])>1 and len(word[i-1])>1:

phrasestore.append(word[i-1]+word[i])

if cixing[i]=='v' and cixing[i-1]=='n' and len(word[i])>1 and len(word[i-1])>1:

phrasestore.append(word[i-1]+word[i])

if cixing[i]=='n' and cixing[i-1]=='v' and len(word[i])>1 and len(word[i-1])>1:

phrasestore.append(word[i-1]+word[i])

if cixing[i]=='a' and cixing[i-1]=='n' and len(word[i-1])>1:

phrasestore.append(word[i-1]+word[i])

if cixing[i]=='d' and cixing[i-1]=='v' and len(word[i-1])>1:

phrasestore.append(word[i-1]+word[i])

if cixing[i]=='vn' and cixing[i-1]=='n'and len(word[i])>1 and len(word[i-1])>1:

phrasestore.append(word[i-1]+word[i])

if cixing[i]=='n' and cixing[i-1]=='vn' and len(word[i])>1 and len(word[i-1])>1:

phrasestore.append(word[i-1]+word[i])

if cixing[i]=='vn' and cixing[i-1]=='v' and len(word[i])>1 and len(word[i-1])>1:

phrasestore.append(word[i-1]+word[i])

if cixing[i]=='v' and cixing[i-1]=='vn' and len(word[i])>1 and len(word[i-1])>1:

phrasestore.append(word[i-1]+word[i])

if cixing[i]=='a' and cixing[i-1]=='vn' and len(word[i-1])>1:

phrasestore.append(word[i-1]+word[i])

接下来的这部分操作是我结合自己对关键词的理解增加的一部分操作。主要来说就是，由于关键词经常是一个词组而不是一个词，即使我们可以通过像词典里添加一些可能会出现的词组来缓解这个问题，然而很明显这是杯水车薪，你无法预测到所有有可能出现的词组。所以我们需要自己来构造这些词组。方法很简单，对于刚刚存到word表里的每一个词，把它和它的前一个词连接起来，如果这两个词的词性满足一定构成词组的要求（比如我使用的，如果前一个词和后一个词的词性分别是：名词+名词，动词+名词，名词+动词，形容词+名词等等），我们就把他们连起来存到phrasestore表里。

for temp in [(x.word,x.flag) for x in psg.cut(s) if x.flag.startswith('n')]:

if str(temp).split(',')[0].split("'")[1] in stopWord:

continue

nstore.append(str(temp).split(',')[0].split("'")[1])

for temp in [(x.word,x.flag) for x in psg.cut(s) if x.flag.startswith('v')]:

if str(temp).split(',')[0].split("'")[1] in stopWord:

continue

vstore.append(str(temp).split(',')[0].split("'")[1])

for temp in [(x.word,x.flag) for x in psg.cut(s) if x.flag.startswith('x')]:

if str(temp).split(',')[0].split("'")[1] in stopWord:

continue

if len(str(temp).split(',')[0].split("'")[1])>2:

if '\\' not in str(temp).split(',')[0].split("'")[1]:

sstore.append(str(temp).split(',')[0].split("'")[1])

for temp in [(x.word,x.flag) for x in psg.cut(s) if x.flag.startswith('l')]:

if str(temp).split(',')[0].split("'")[1] in stopWord:

continue

sstore.append(str(temp).split(',')[0].split("'")[1])

这一堆内容则是根据jieba分词的词性把词性为名词、动词、字符串和惯用语（一般碰到我们自己加入到词典里的词组或短句jieba检测出来会归类为这两类）分别存到nstore、vstore、sstore里去。

tempn = Counter(nstore).most\_common(10)

tempv = Counter(vstore).most\_common(3)

temps = Counter(sstore).most\_common(7)

tempphrase = Counter(phrasestore).most\_common(10)

with open('result.txt','w+',encoding='UTF-8') as f:

for x in tempn:

if x not in importantWord:

f.write('{0},{1}\n'.format(x[0],x[1]))

for x in tempv:

if x not in importantWord:

f.write('{0},{1}\n'.format(x[0],x[1]))

for x in temps:

if x not in importantWord:

f.write('{0},{1}\n'.format(x[0],x[1]\*3))

for x in tempphrase:

if x not in importantWord:

f.write('{0},{1}\n'.format(x[0],x[1]\*3))

for x in importantWord:

if x in s:

f.write(x.split('\n')[0])

f.write(",")

f.write(str(s.count(x)\*10))

f.write('\n')

这里使我们分词部分的最后一块内容，用counter库的相应接口统计nstore、vstore、sstore、phrasestore里出现频率最高的一些词，我们把他们存到result.txt为后边TF-IDF做准备。注意的是在这边为了加强importantword里面的词的权重，我们得把前几个store含有的importantword先删去，最后在result.txt加入文本中出现的importantword和出现次数，为了增加权重我们把这个次数\*10。

接下来进入计算IF-IDF的阶段。这一过程中我们主要要利用爬取百度文库的某词汇数量来计算IDF。

dictionary = {}

fr = open("result.txt",encoding='UTF-8')

line = fr.readline()

while line:

# if len(line.split(',')[0])>1:

dictionary[line.split(',')[0]] = int(line.split(',')[1])

line = fr.readline()

fr.close()

print(dictionary)

首先定义一个叫dictionary的字典，key是前边我们jieba分词到result.txt的结果，value是这一词的词频（可能乘了权重）。

for key in dictionary:

url="https://wenku.baidu.com/search?word=%E2%80%9C"+key+"%E2%80%9D&lm=0&od=0&ie=utf-8"

s = quote(url,safe=string.printable)

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

response = request.urlopen(s)

html = response.read()

charset = chardet.detect(html)

html = html.decode(charset.get('encoding'),'ignore')

soup = BeautifulSoup(html,"html.parser")

if soup.find("span",class\_="nums")==None:

url="https://zhidao.baidu.com/search?lm=0&rn=10&pn=0&fr=search&ie=utf-8&word=%E2%80%9C"+key+"%E2%80%9D"

s = quote(url,safe=string.printable)

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

response = request.urlopen(s)

html = response.read()

charset = chardet.detect(html)

html = html.decode(charset.get('encoding'),'ignore')

soup = BeautifulSoup(html,"html.parser")

if soup.find("span",class\_="f-lighter lh-22")==None:

dictionary[key]=dictionary[key]\*math.log(100000000/(1+100000))

print(key)

print(dictionary[key])

else:

dictionary[key]=dictionary[key]\*math.log(100000000/(1+int(str(soup.find("span",class\_="f-lighter lh-22")).split('共')[1].split('条')[0].replace(',', ''))))

print(key)

print(dictionary[key])

else:

dictionary[key]=dictionary[key]\*math.log(100000000/(1+int(str(soup.find("span",class\_="nums")).split('约')[1].split('篇')[0].replace(',', ''))))

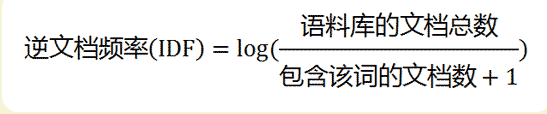
print(key)

print(dictionary[key])

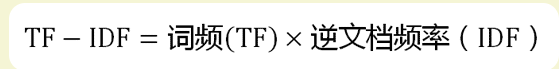
这一堆是这一部分的重点，主要思路是利用百度文库中搜索双引号+我们要查询的词来精确搜索我们的词在文库中出现的条目数（如下图）



然后利用这个数值和我们的词频按照给定的公式计算IDF



最后利用tf（就是词频）和idf计算tf-idf



当然，上边这部分代码关于爬虫部分的一些操作（我选用百度文库而不是中国知网的原因就是这里用的是get而不是post比较好操作）这里先不赘述。

new\_dictionary=sorted(dictionary.items(),key=operator.itemgetter(1),reverse=True)

flag=0

for i in range(0,len(new\_dictionary)):

for j in range(0,i):

if (new\_dictionary[i][0] in new\_dictionary[j][0]) or (new\_dictionary[j][0] in new\_dictionary[i][0]):

flag=1

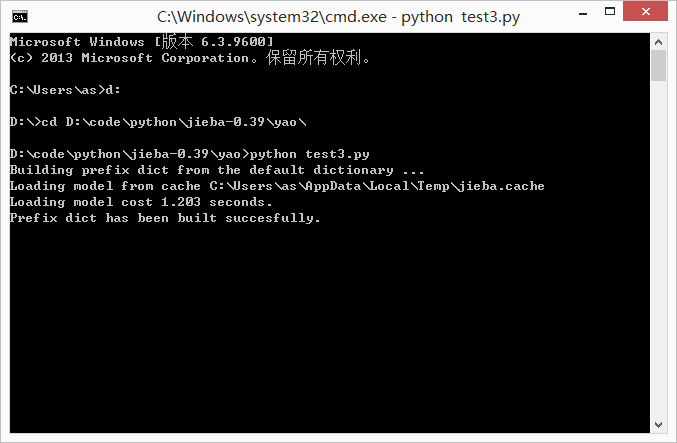
if flag==0:

print(new\_dictionary[i][0])

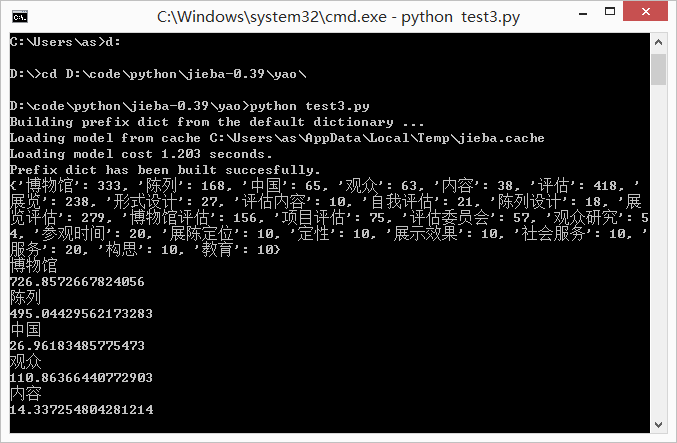
flag=0

最后我们用sorted函数为我们的dictionary排个序，然后对排名前几的词当做关键词进行输出。输出的时候还有一些小想法，首先如果某个词发现是前面已输出词的子串，那么就放弃这个词；还有将最后输出的结果的这些词再分析一下，如果某个词重复出现（比如博物馆评估和展览评估和观众评估，这其中的评估这个词），那么我们把它单独拿出来也作为一个结果。

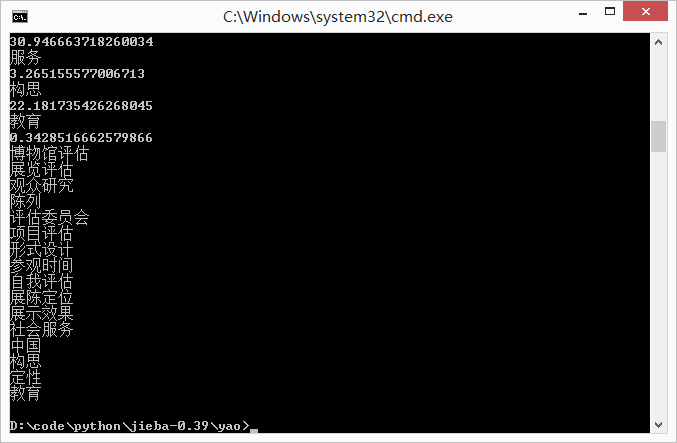
看一下输出的结果（python3）：



首先是调用jieba的一些常规输出



这里对应的是第二部分开始构造字典那句print（dictionary），接下来计算每个词的td-idf并输出结果



最后根据排序的结果输出程序给出的结果作为预测的关键字