

enron 公司电子邮件检索系统

2019 年 10 月 5 日 最新更新 @jack-lio

信息检索系统原理

作业一 邮件检索

- 邮件数据：安然公司150位用户50万封电子邮件
 - <http://www.cs.cmu.edu/~enron/>
- 检索模型：向量空间模型
- 编程语言：C/C++/C#/JAVA/Python均可
- 截止时间：10月8日
- 注意事项：
 - 可以按照作者、标题、内容等进行邮件检索
 - 索引构建核心环节、向量空间模型核心环节，必须独立实现，不可以借助工具包等
 - *选做*：附件检索、GUI界面或Web界面呈现

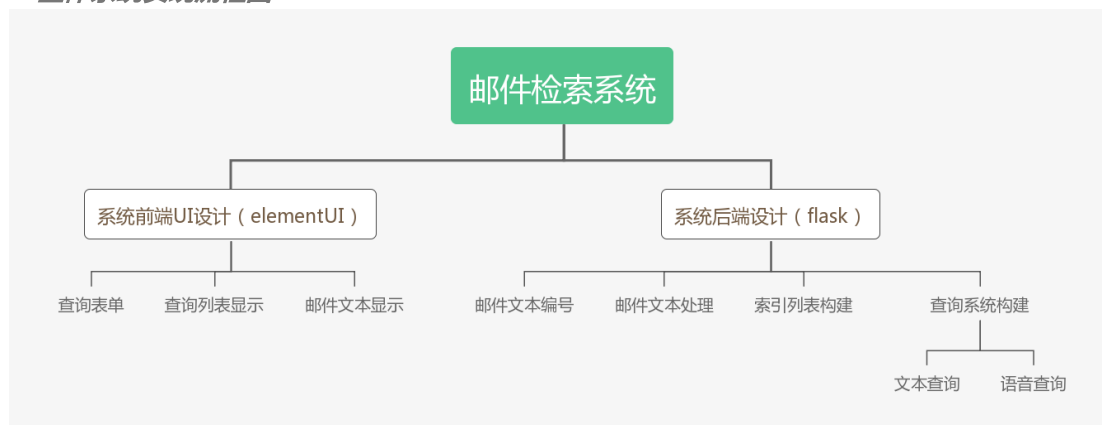
23

• 项目说明

项目内容：本系统基于向量空间模型实现对enron公司150位用户50万封电子邮件进行检索 检索模型：向量空间模型 编程语言：Python

工具包：nltk、flask、elementUI ... 数据来源：[eron 电子邮件数据集](#) 实现功能：基于发件时间、发件人、收件人、主题、正文对邮件进行检索，同时拓展了功能实现语音检索功能，对系统进行整合优化，建立前端UI界面对系统进行友好调用。

=>整体系统实现流程图



=>系统启动步骤

1. 解压缩包，将压缩包解压的文件夹，放在和邮件解压文件夹（邮件解压文件不在作业压缩包中,要自己解压添加，文件的目录结构若未改动则应该运行没有问题）根目录同一路径下，

文件树如下所示：





















```
| - |
|-enron_mail_20150507
|      | --- maildir
|      | --- 邮件用户列表分类
|-homework //作业提交压缩包解压
|      | --- （文件夹解压所之后的程序和中间文件）
```

2. 在控制台运行命令 `python backend.py` 启动flask后端，启动完成后，打开index.html页面即进行查询。（因索引文件较大，所以启动会耗一些时间，同时保证可用运行内存在4G以内）
3. 提示: index.html中使用的js、cs文件来自网络请求，所以在打开前端页面之前保证网络连接OK！

一 准备工作

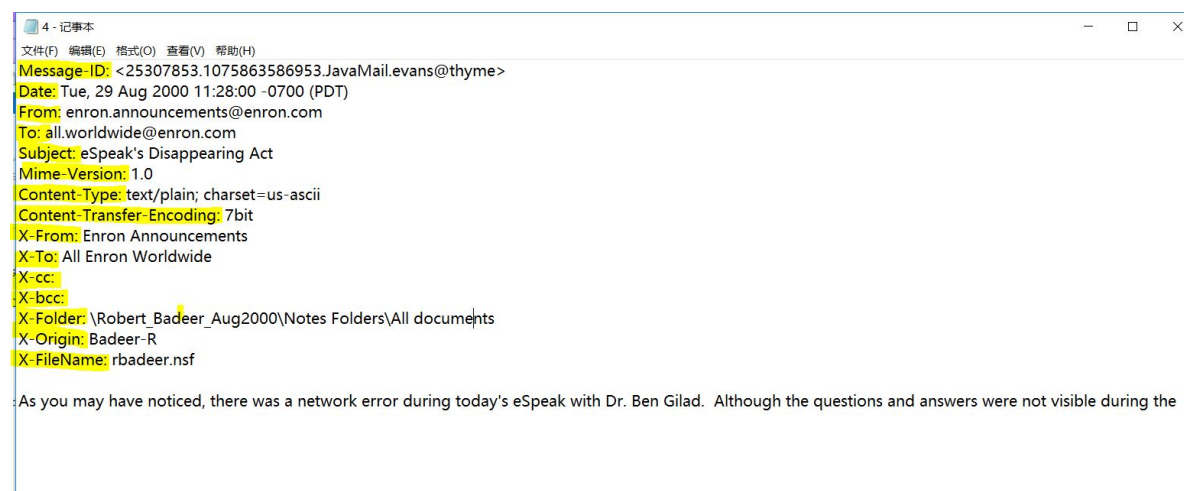
1.1 数据分析

从文件可以看出，enron电子邮件数据已经经过了一部分的处理，所有的邮件按照150个用户进行了分类保存，在每个用户的类别文件夹中，还按照邮件的不同类型、来源、以及用户自行定义的邮件分类系统进行了归类存放。

 allen-p	2019/9/19 12:12	文件夹
 arnold-j	2019/9/19 12:19	文件夹
 arora-h	2019/9/19 12:14	文件夹
 badeer-r	2019/9/19 11:44	文件夹
 bailey-s	2019/9/19 11:51	文件夹
 bass-e	2019/9/19 12:02	文件夹
 baughman-d	2019/9/19 12:16	文件夹
 beck-s	2019/9/19 12:14	文件夹
 benson-r	2019/9/19 11:44	文件夹
 blair-l	2019/9/19 11:44	文件夹
 brawner-s	2019/9/19 12:05	文件夹
 buy-r	2019/9/19 12:10	文件夹
 campbell-l	2019/9/19 12:22	文件夹
 carson-m	2019/9/19 12:16	文件夹
 cash-m	2019/9/19 11:44	文件夹
 causholli-m	2019/9/19 11:46	文件夹
 corman-s	2019/9/19 11:44	文件夹
 crandell-s	2019/9/19 11:54	文件夹
 cuilla-m	2019/9/19 11:44	文件夹
 dasovich-i	2019/9/19 11:51	文件夹

名称	修改日期	类型	大小
_sent_mail	2019/9/19 11:44	文件夹	
all_documents	2019/9/19 11:44	文件夹	
cal_articles	2019/9/19 11:44	文件夹	
california	2019/9/19 11:44	文件夹	
capx	2019/9/19 11:44	文件夹	
contacts	2019/9/19 11:44	文件夹	
deleted_items	2019/9/19 11:44	文件夹	
discussion_threads	2019/9/19 11:44	文件夹	
dj_articles	2019/9/19 11:44	文件夹	
inbox	2019/9/19 11:44	文件夹	
memo_s	2019/9/19 11:44	文件夹	
move	2019/9/19 11:44	文件夹	
notes_inbox	2019/9/19 11:44	文件夹	
press_releases	2019/9/19 11:44	文件夹	
sent_items	2019/9/19 11:44	文件夹	
var	2019/9/19 11:44	文件夹	

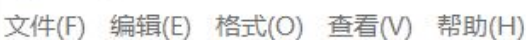
从邮件的内容来看，邮件基本个数格式包括邮件的ID编码、收发时间、发件人、收件人、主题、mime-version（多用途互联网邮件拓展 版本）、信息类型与编码（规定信息类型和采用的编码格式）、编码转换方式声明（content-transfer-encoding）、邮件抄送密送信息、邮件体等等。



1.2 文档预处理

此阶段主要是对文档中的邮件文件进行统一的处理，提取文件目录下的所有文件，同时对其进行编号，将文件路径和文件的编号存入mail_list_file.txt文件。

```
#对数据集中的所有文件进行编号，将其文件目录输出到一个文件中
#path: 邮件的存储文件夹
#number_file: 为存储邮件编号列表的txt文件路径
#控制台输出的count变量数据显示处理的文件数量
def list_files(path,number_file):
    f = open(number_file,'w',encoding='utf-8')
    # 遍历根目录
    count = 1
    for root,dirs,files in os.walk(path,topdown=True):
        for file in files :
            path_t = os.path.join(root,file)
            f.write(str(count)+'-'+path_t+'\n')
            count=count+1
        print(count)
    f.close()
```

之后按照对于邮件的编号对邮件内容进行分域、分词处理，对不同的文本域有不同的分词处理，所用的分词器利用NITK的分词函数和正则表达式进行，分词器实现的功能有：单词小写正规化，分词，词源恢复等，同时基于Python的counter实现对单词的词频统计。再利用Python具有的dict数据结构，将每个文件的词频统计数据记录到词汇字典当中，字典之后跟着一个数据统计字典，内容为所有该词汇出现的文件编号（key）和出现次数（value）。

```
# 分词&词形还原函数
# sentence 为输入的单词串
# 此函数采用yield 实现迭代器，返回一个单词的迭代器
# 返回值为经过词形还原之后的一个一个的单词或符号
def lemmatize_all(sentence):
    wn1 = wordNetLemmatizer()      # 声明一个词形还原器
    for word, tag in nltk.pos_tag(re.findall(r'[0-9]+[:][0-9]+[:][0-9]+|[0-9]+[:][0-9]+|[0-9]+[:.][0-9]+|[0-9]+|[A-Za-z]+[-\']+[A-Za-z]+|[A-Za-z]+[[@$]+' ,sentence))): # 先正则分词，再给不同的词贴上标签
        if tag.startswith('NN'):   # 根据标注的不同词性对单词进行词形还原
```

```

        yield wn1.lemmatize(word, pos='n')
    elif tag.startswith('VB'):
        yield wn1.lemmatize(word, pos='v')
    elif tag.startswith('JJ'):
        yield wn1.lemmatize(word, pos='a')
    elif tag.startswith('R'):
        yield wn1.lemmatize(word, pos='r')
    else:
        yield word

#输出句子分词之后的词项和词频，并统计句子的单词总数
#调用lemmatize_all(分词) 函数，并将返回的词条迭代器转换为一个单词列表
#通过使用counter函数，将单词列表进行词项统计工作，返回一个字典
#此函数另一个返回值为该输入字符串分词后的单词总数
def split_lemmatize(sentence,type):
    # 全部转换为小写字母
    sentence_proc = sentence.lower()
    # nltk分词以及词形还原
    words = []
    if type == '-C' or type == '-S':
        words = [i for i in lemmatize_all(sentence_proc) ]      # 基于nltk分词，将
        # 结果以列表形式存储
    elif type == '-D':
        words = re.findall(r'[0-9]+[:][0-9]+[:][0-9]+|[0-9]+|[A-Za-
z]+' ,sentence_proc)
    elif type == '-F' or type == '-T':
        words = re.findall(r'[0-9]+[.][0-9]+|[0-9]+|[A-Za-z]+[
@]+' ,sentence_proc)
    # print(sentence)
    # print(words)
    count = Counter(words)      # 使用counter进行计数，counter继承了Python的字典数据
    # 结构能够很好的解决索引问题
    return len(words),count      #返回句子分词和词形还原之后统计的词条总数以及词项频数

# 对文件中不同的文本域进行划分的主要代码，核心思想是利用标识的词项区分 （来自函数
split_words)
content = open(file[1], 'r', encoding='utf-8') # 打开一个文件进行读取
former_word = ''
for line in content:
    top_word = line.split(':')[0]
    if top_word == 'X-FileName':
        word_dict['Content'] = ''
    if top_word in title_list and former_word != 'X-FileName':
        word_dict[top_word] = line.strip(top_word +
':').strip('\n').strip('\t')
        former_word = top_word
        continue
    elif former_word == 'X-FileName':
        word_dict['Content'] = word_dict['Content'] + ' ' +
line.strip('\n').strip('\t')
        continue
    else:
        word_dict[former_word] = word_dict[former_word] + ' ' +
line.strip('\n').strip('\t')
content.close() # 关闭读取文件流

```

二 文件索引构建

利用文件预处理实现的分词词频字典，通过计算tf-idf的值，构建文件索引表，同时记录所有的文件的向量空间长度，将构建的索引表存入index.txt文件，将记录的文件不同域的词项向量空间长度存入file_info.txt文件中。

- tf-idf构建和输出函数代码：

```
indexf = open(index_path, 'w', encoding='utf-8') # 打开索引输出文件流
infof = open(file_info_path, 'w', encoding='utf-8') # 打开文件输出流，输出文件
中不同域文本的向量空间长度
for item in index_dict.keys():
    df = len(index_dict[item])
    indexf.write(item)
    for temp in index_dict[item].keys():
        index_dict[item][temp] = math.log(file_count_num / df) * (1 +
math.log(index_dict[item][temp]))
        words_count_of_file[temp][order[item[-1]]] =
words_count_of_file[temp][order[item[-1]]] + math.pow(index_dict[item][temp], 2)
        indexf.write( format('#'+temp+':%.2f'%index_dict[item][temp]))
    indexf.write('\n')
for item in words_count_of_file:
    infof.write(item)
    for temp in range(len(words_count_of_file[item])):
        if temp != 0:
            infof.write( format(' ,%.4f'%math.sqrt(words_count_of_file[item]
[temp])))
        else :
            infof.write(' ,'+str(words_count_of_file[item][temp]))
    infof.write('\n')
infof.close()
indexf.close()
```

- 文件向量空间长度信息：

```
2, 2617, 64. 47205390635284, 115. 86442943148481, 0, 103. 7765834816689, 37863. 37214833677
3, 959, 60. 76392866303375, 0. 6720136058208708, 13. 415125785221402, 145. 0302367192331, 9932. 36873963022
4, 959, 46. 76248193943759, 0. 6720136058208708, 13. 415125785221402, 145. 0302367192331, 9921. 819552729909
5, 19, 61. 91323996807742, 0. 6720136058208708, 49. 65426321713228, 0, 163. 16002228607138
6, 6, 71. 00896914540832, 0. 6720136058208708, 52. 892318852875945, 82. 93949420516306, 66. 08177994264067
7, 152, 57. 186685112444465, 0. 6720136058208708, 20. 60585763002792, 125. 94150616694014, 1450. 466705337982
8, 306, 55. 62979259310881, 0. 6720136058208708, 16. 405424947536968, 30. 486530667480196, 2769. 743363799667
9, 112, 72. 86034491993419, 0. 6720136058208708, 20. 60585763002792, 118. 84343956146009, 1213. 29041951643
10, 26, 72. 86034491993419, 0. 6720136058208708, 73. 36732928316898, 0, 145. 9667781980086
11, 13, 55. 62979259310881, 0. 6720136058208708, 49. 246409259859284, 103. 06384057203954, 128. 24505010893702
12, 29, 55. 88871712797366, 0. 6720136058208708, 73. 36732928316898, 157. 9548406880032, 104. 4483679621532
13, 51, 55. 376324729021945, 49. 24018216731784, 195. 53533542699728, 138. 9266441159981, 629. 0457679660266
14, 389, 75. 77237000663291, 0. 6720136058208708, 95. 44076106171337, 96. 80429763071382, 3460. 737859881091
15, 369, 75. 77237000663291, 0. 6720136058208708, 52. 892318852875945, 96. 80429763071382, 3457. 4453426385458
16, 1500, 69. 4833747019263, 0. 6720136058208708, 38. 82159865980758, 59. 91955540733593, 13908. 036407582465
17, 1500, 69. 4833747019263, 0. 6720136058208708, 16. 405424947536968, 59. 91955540733593, 13917. 132136759796
18, 837, 46. 38619880999246, 0. 6720136058208708, 20. 60585763002792, 0, 8922. 015652922819
19, 9, 69. 4833747019263, 0. 6720136058208708, 74. 0907409625598, 91. 8949631553173, 111. 45490674414292
20, 104, 55. 51243398953531, 0. 6720136058208708, 20. 60585763002792, 0, 1050. 8134118564162
21, 101, 52. 252822375100905, 0. 6720136058208708, 20. 60585763002792, 112. 83179965647332, 745. 5902060934313
22, 377, 69. 4833747019263, 0. 6720136058208708, 22. 874256042809698, 38. 62135381697468, 2889. 514852360473
23, 24, 61. 35324739281329, 0. 6720136058208708, 64. 99501178522891, 0, 118. 15855489576606
24, 166, 47. 351800669217134, 0. 6720136058208708, 20. 60585763002792, 0, 2047. 5635704073466
25, 96, 47. 351800669217134, 0. 6720136058208708, 20. 60585763002792, 0, 934. 0002308553711
26, 14, 70. 44897657014418, 0. 6720136058208708, 58. 56529265724693, 0, 66. 31286621670804
27, 44, 63. 72497192835706, 0. 6720136058208708, 64. 4329177922277, 0, 291. 1801784167999
```


- 文件索引列表信息

```
18:41:00-D#1:0. 907755278982137
6 0800-D#1:0. 6033064765601559#2:0. 6033064765601559#13:0. 6033064765601559#44:0. 6033064765601559#55:0. 6033064765601559#63:0. 603306476560
7 pst-D#1:0. 6033064765601559#2:0. 6033064765601559#13:0. 6033064765601559#44:0. 6033064765601559#55:0. 6033064765601559#63:0. 603306476560
8 1-F#1:6. 214608098422191#923:6. 214608098422191
9 11913372-F#1:6. 907755278982137
10 2-F#1:6. 907755278982137
11 multexinvestornetwork-F#1:6. 907755278982137
12 com-F#1:0. 019182819416773904#2:0. 019182819416773904#3:0. 019182819416773904#4:0. 019182819416773904#5:0. 019182819416773904#6:0. 019182
13 pallen-T#1:1. 8773173575897015#77:1. 8773173575897015#99:1. 8773173575897015#110:1. 8773173575897015#122:1. 8773173575897015#209:1. 87731
14 error-T#1:0. 44005655287778345#3:0. 44005655287778345#4:0. 44005655287778345#5:0. 44005655287778345#6:0. 44005655287778345#7:0. 440056552
15 com-T#1:0. 08121005542554327#3:0. 08121005542554327#4:0. 08121005542554327#5:0. 08121005542554327#6:0. 08121005542554327#7:0. 08121005542
16 december-S#1:4. 605170185988092#110:4. 605170185988092#219:4. 605170185988092#485:4. 605170185988092#807:4. 605170185988092#875:4. 605170
17 14-S#1:5. 521460917862246#219:5. 521460917862246#287:5. 521460917862246#534:5. 521460917862246
18 2000-S#1:5. 809142990314028#111:5. 809142990314028#219:5. 809142990314028
19 bear-S#1:6. 907755278982137
20 stearns-S#1:6. 907755278982137
21 prediction-S#1:6. 907755278982137
22 for-S#1:2. 385966701933097#37:2. 385966701933097#76:2. 385966701933097#102:2. 385966701933097#103:2. 385966701933097#118:2. 3859667019330
23 telecom-S#1:6. 214608098422191#707:6. 214608098422191
24 in-S#1:3. 8632328412587142#9:3. 8632328412587142#271:3. 8632328412587142#277:3. 8632328412587142#451:3. 8632328412587142#623:3. 863232841
```

三 检索系统构建

从文件中读取索引列表信息，通过对查询内容进行相同的分词和tf-idf计算，实现对邮件文本的查询。同时进行拓展，利用百度语音的API实现对于输入语音的文本化，利用文本化的内容实现对于相关内容的查询。将查询结果按照匹配度从高到低进行返回输出。

- 语音文本化代码

```
# 获取语音输入
def get_audio(filepath):
    aa = 'yes' #str(input("=>start recording?      (yes/no) :"))
    if aa == str("yes") :
        CHUNK = 256
        FORMAT = pyaudio.paInt16
        CHANNELS = 1                # 声道数
        RATE = 16000                # 采样率
        RECORD_SECONDS = 10         # 采样时间
        WAVE_OUTPUT_FILENAME = filepath #文件存储路径
        p = pyaudio.PyAudio()

        stream = p.open(format=FORMAT,
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        frames_per_buffer=CHUNK)

        print("***10, "recording begins: please input audio in 20 seconds! ")
        frames = []
        for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
            data = stream.read(CHUNK)
            frames.append(data)
        print("***10, "recording end\n")

        stream.stop_stream()
        stream.close()
        p.terminate()

        wf = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
        wf.setnchannels(CHANNELS)
        wf.setsampwidth(p.get_sample_size(FORMAT))
        wf.setframerate(RATE)
        wf.writeframes(b''.join(frames))
```

```

        wf.close()
        return "succeed recording!"
    elif aa == str("no"):
        exit()
    else:
        return ("incorrect input, please choose again!")
# 获取语音文本化信息
def recognize(sig, rate, token):
    url = "http://vop.baidu.com/server_api"
    speech_length = len(sig)
    speech = base64.b64encode(sig).decode("utf-8")
    mac_address = uuid.UUID(int=uuid.getnode()).hex[-12:]
    rate = rate
    data = {
        "format": "wav",
        "lan": "en",
        "token": token,
        "len": speech_length,
        "rate": rate,
        "speech": speech,
        "cuid": mac_address,
        "channel": 1,
    }
    data_length = len(json.dumps(data).encode("utf-8"))
    headers = {"Content-Type": "application/json",
               "Content-Length": str(data_length)}
    r = requests.post(url, data=json.dumps(data), headers=headers)
    # print(r.json()['result'])
    return r.json()

```

- 查询代码

```

# 获取键盘输入，进行查询匹配

def query(q_sentence, index_dict, words_count_of_file, type):
    length, count = split_lemmatize(q_sentence, type)
    score = {}
    file_count_num = len(words_count_of_file.keys()) # 文件数

    for file in words_count_of_file.keys():
        score[file] = 0.0
        for item in count.keys(): # 如果没有语音分析结果，说明录取的音频没有
            # 有效的信息，提前退出并返回错误提示
            if item+type not in index_dict.keys():
                continue
            if file in index_dict[item+type].keys():
                w = index_dict[item+type][file]
            else :
                w = 0
            temp =
            (1+math.log(count[item]))*math.log(file_count_num/len(index_dict[item+type].keys
            ())) * w
            score[file] = score[file] + temp
        a_len = words_count_of_file[file][order[type[-1]]]
        if a_len == 0 :
            score[file] = 0.0
            continue

```

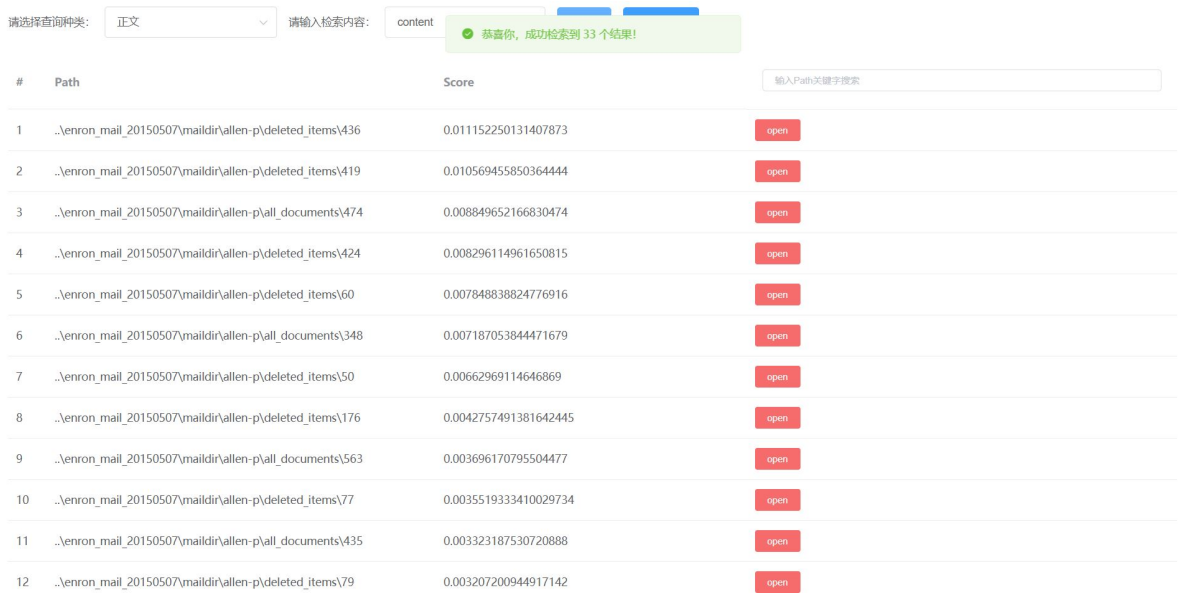


```
score[file] = score[file]/a_len
# 对计算的结果进行排序处理
rank_list = sorted(score.items(),key=lambda x:x[1],reverse=True)
return rank_list
```

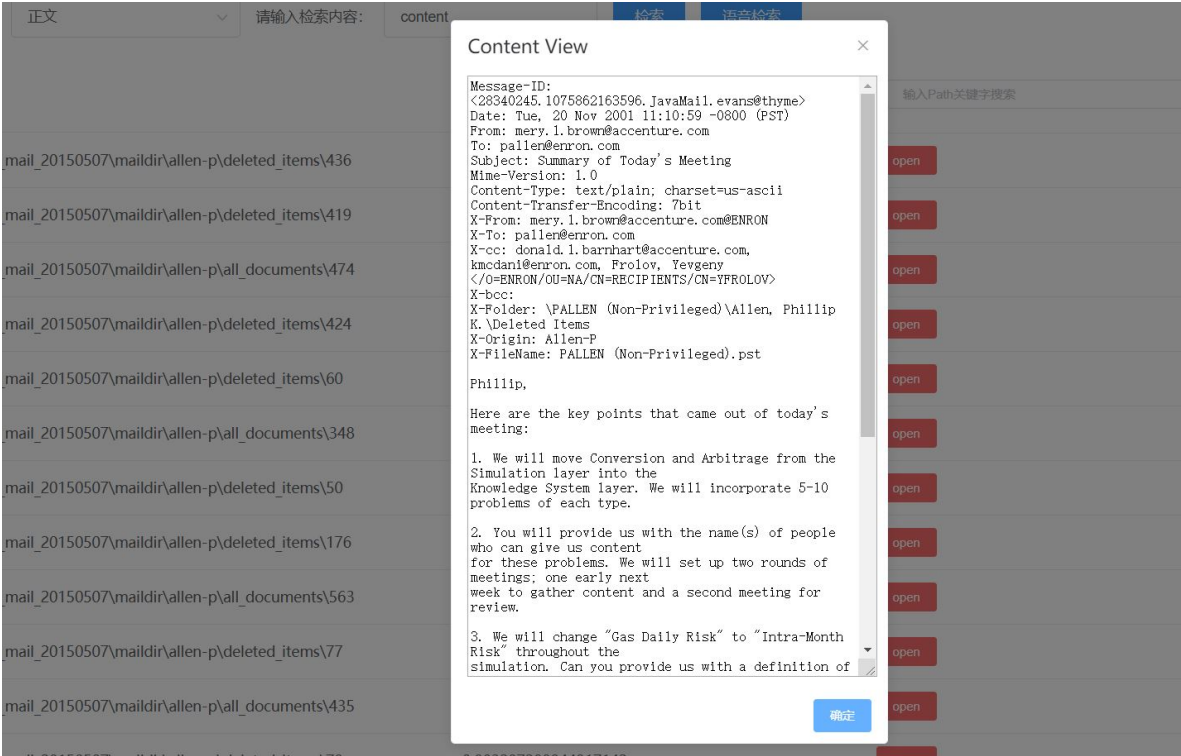
四 GUI交互搭建【拓展内容】

基于elementUI实现前端页面的构建，利用flask构建后端内容，对Python相关函数进行调用，实现数据交互和检索查询功能整合实现。

- 前端界面



- 查询显示文本



参考资料：

- [MIME笔记.阮一峰的网络日志.](#)
- [nltk官方文档](#)
- [python文档-了解相关python基础](#)
- [elementUI文档](#)
- [flask中文文档](#)
- [百度智能云语音识别文档](#)