# Web Lab1 邮件搜索引擎

PB17111585 张永停

## 一、算法

### I、数据处理

本部分代码见`data_process.py`

- 将每个邮件映射到一个INT数字， 映射关系存储为字典，`{Int: EmailAddress}`，存储文件为`dict.npy`
    - 代码见`convert_to_dict`函数

```
def convert_to_dict(filelist):
    i = 0
    dict = {}
    for file in filelist:
        dict[str(i)] = file
        i = i + 1
    np.save(r"../dataset/dict.npy", dict)
```

- 分词、词根化、去停用词
    - 代码见`tokenization`函数

```
def tokenization(dict):
    dict = np.load(dict, allow_pickle=True).item()
    snowball_stemmer = SnowballStemmer("english")
    stopWords = set(stopwords.words('english'))
    for w in ['!', ',', '.', '?', '-s', '-ly', '</s>', 's']:
        stopWords.add(w)
    for key, value in dict.items():
        #if int(key) < 24502:
        #    continue
        dict_fre = {}
        #print(value)
        with open(value, 'r', errors='ignore') as f:
            data = f.read()
            words = word_tokenize(data)
            #words_filter = []
            for word in words:
                if word not in stopWords:
                    if snowball_stemmer.stem(word) not in dict_fre:
                        dict_fre[snowball_stemmer.stem(word)] = 1
                    else:
                        dict_fre[snowball_stemmer.stem(word)] += 1
            path_index = target_path + key
            np.save(path_index, dict_fre)
            #t = np.load(path_index + '.npy',
allow_pickle=True).item()
            #print(t)
```

```
            #break
```

- 此部分需要注意部分邮件还有非ASCII符号，因此读入时设置为`open(value, 'r', errors='ignore')`
- 采用`nltk`进行分词
- 使用`nltk`的英文包作为停用词，同时将`'!', ',', '.', '?', '-s', '-ly', '</s>', 's'`加入停用词
- 使用`nltk`的`SnowballStemmer("english")`来进行词干提取
- 为后续方便，将每个文件的分词结果按形式`{word, frequence}`存储在`\dataset\index`下，按第一步中的Int命名

- 检索出现频率最高的1000个词
  - 代码见`get_top1k_words`

```
def get_top1k_words():
    words = {}
    for file in os.listdir(target_path):
        word = np.load(target_path + file,
allow_pickle=True).item()
        for w, f in word.items():
            if w not in words:
                words[w] = f
            else:
                words[w] += f
    words_order = sorted(words.items(), key=lambda x: x[1],
reverse=True)
    np.save(top1k_path, words_order[:1000])
    #print(words_order)
```

  - 对上一步分词的存储结果遍历检索，统计词频
  - 将前1000的词按照`{word, frequence}`存储为`output\top1k.npy`
  - 为后续检索方便，同时将`{word, index}`形式字典存储为`\output\word_index_top1k.npy`

- 建立倒排表
  - 代码见`get_invered_table`

```
def get_invered_table():
    words_table = []
    for i in range(1000):
        words_table.append([])
    words_fre = np.zeros(1000)
    top1k = get_dict_top1k()
    index_word_lastone = [None] * 1000
    for i in range(517401):
    #for file in os.listdir(target_path):
        top1k_words = {}
        word = np.load(target_path + str(i) + '.npy',
allow_pickle=True).item()
        for w, f in word.items():
```

```
            if w in top1k:
                index_word = list((top1k.keys())).index(w)
                if words_table[index_word] == []:
                    words_table[index_word].append(i)
                    index_word_lastone[index_word] = i
                else:
                    lastone = index_word_lastone[index_word]
                    words_table[index_word].append(i - lastone)
                    index_word_lastone[index_word] = i
                top1k_words[w] = f
                words_fre[index_word] += int(f)
        #if(i > 5):
        #    break
        #np.save(filter_path + str(i), top1k_words)

    #words_fre_path =
r'D:\study\USTC\2020.fall\web\lab1\output\freq.npy'
    #np.save(words_fre_path, words_fre)
    words_table_path =
r'D:\study\USTC\2020.fall\web\lab1\output\invertedtable.npy'
    np.save(words_table_path, words_table)
```

- 对第二步分词存储的文件遍历，如果文件中的词出现在top1k，则将该文件的INT值存入相应索引的列表中。建立words_table， 形式为`[[]]`，存储于`\output\invertedtable.npy`
- 这里使用临时数组`index_word_lastone`来记录每个索引上次存入的文件编号，从而在存储时，使用文件编号的间隔代替原编号，节省存储空间。
- 在该步，同时统计每个词出现的次数，并存放入`\output\freq.npy`

## II、布尔检索

本部分代码见`bool_search.py`

- 处理输入：
  - 对输入的查询进行分词，词干提取，见`deal_search`函数

```
def deal_search(result_path):
    global word_index
    word_index = np.load(word_index_path, allow_pickle=True).item()
    origin = input("Please input query(Input only E to exit):")
    while(origin != 'E'):
        query = origin.split(' ')
        start = time.time()
        split_query = []
        snowball_stemmer = SnowballStemmer("english")
        for i, q in enumerate(query):
            # 这里默认不会查询带着'(' ')'的词
            if (len(q) > 1 and q[0] == '('):
                split_query.append('(')
                split_query.append(snowball_stemmer.stem(q[1:]))
            elif (len(q) > 1 and q[-1] == ')'):
                split_query.append(snowball_stemmer.stem(q[:-1]))
                split_query.append(')')
            else:
                split_query.append(snowball_stemmer.stem(q))
```

```
20
21          vec = bool_search(split_query)
22          result = []
23          dict = np.load(dict_path, allow_pickle=True).item()
24          # print(dict)
25          for i in range(num_file):
26              if vec[i]:
27                  result.append(dict[str(i)][48:])
28          end = time.time()
29          print("Finish! See result at %s" % result_path)
30          print("Time:", end - start)
31          with open(result_path, 'a+') as f:
32              f.write("For query: " + origin + "\n")
33              f.write("Result is \n")
34              for r in result:
35                  f.write(r)
36                  f.write("\n")
37
 f.write("==========================================================
====================\n")
38          origin = input("Please input query(Input only E to exit):")
```

- 计算
  - 本部分采用数组来模拟栈操作，以正确计算`AND`和`NOT`和`OR`以及`(` `)`， 思想借鉴于简单计算器的实现， 优先级`NOT`>`AND`>`OR`，当遇到`)`，弹出直到遇到`(`
  - 采用两个栈: 符号栈与内容栈

    上两个部分代码见`bool_search`

```
1  def bool_search(query):
2      global head_oper
3      global  head_word
4      for word in query:
5          if(word == '('):
6              head_oper += 1
7              stack_oper[head_oper] = '('
8
9          elif(word == ')'):
10             while(stack_oper[head_oper] != '('):
11                 do_bool(stack_oper[head_oper])
12             head_oper -= 1
13         elif(word == 'not'):
14             head_oper += 1
15             stack_oper[head_oper] = word
16
17         elif(word == 'and'):
18             if head_oper == -1:
19                 head_oper += 1
20                 stack_oper[head_oper] = word
21             elif stack_oper[head_oper] == 'or' or
stack_oper[head_oper] == '(':
22                 head_oper += 1
23                 stack_oper[head_oper] = word
24             elif stack_oper[head_oper] == 'and' or
stack_oper[head_oper] == 'not' :
```

```
                while(head_oper >= 0 and ([head_oper] == 'and' or stack_oper[head_oper] == 'not')):
                    do_bool(stack_oper[head_oper])
                head_oper += 1
                stack_oper[head_oper] = word
        elif (word == 'or'):
            if head_oper == -1 or stack_oper[head_oper] == '(':
                head_oper += 1
                stack_oper[head_oper] = word
            else:
                while (head_oper >= 0 and ([head_oper] == 'and' or stack_oper[head_oper] == 'not'or stack_oper[head_oper] == 'or')):
                    do_bool(stack_oper[head_oper])
                head_oper += 1
                stack_oper[head_oper] = word
        else:
            head_word += 1
            stack_word[head_word] = word_index[word]

    while(head_oper != -1):
        do_bool(stack_oper[head_oper])

    return stack_word[head_word]
```

- 由于之前在倒排表中存放的是文件间隔，故这里需要恢复文件索引，在恢复文件索引的同时，将倒排表每个索引对应文件变成bool向量

```
def get_bool_vec(word):
    result = np.zeros(num_file).astype(bool)
    contain_file = inverted_table[word]
    last_index = 0
    for f in contain_file:
        real_index = last_index + f
        result[real_index] = True
        last_index += f
    return result
```

- 对于布尔运算，直接对bool向量进行逻辑运算。代码见`do_bool`、`get_and`等

```
def do_bool(oper):
    global head_word
    global  head_oper
    global  result
    word2 = stack_word[head_word]

    if oper == 'not':
        stack_word[head_word] = get_not(word2)

    if oper == 'and':
        word1 = stack_word[head_word - 1]
        head_word -= 1
        stack_word[head_word] = get_and(word1, word2)
    if oper == 'or':
        word1 = stack_word[head_word - 1]
        head_word -= 1
```

```
            stack_word[head_word] = get_or(word1, word2)

        head_oper -= 1
        return
```

```
def get_and(word1, word2):
    vec1 = word1
    vec2 = word2
    if not (type(word1) == type(np.array([0]))):
        vec1 = get_bool_vec(word1)
    if not (type(word2) == type(np.array([0]))):
        vec2 = get_bool_vec(word2)
    return vec1 & vec2
```

- 结果存储
  - 检索结果存储在`\output\bool_search_result.txt`中

## III、语义检索

- 输入处理，本部分输入处理较为简单，只需要对输入的查询进行分词，去停用词，词根提取

```
def deal_search(result_path):
    global tfidf_table
    tfidf_table = np.load(tfidf_path)
    dict = np.load(dict_path, allow_pickle=True).item()
    global word_index
    global  word_idf
    word_index = np.load(word_index_path, allow_pickle=True).item()
    word_idf = np.load(top1k_idf_path)
    global tfidf_norm
    tfidf_norm = np.load(tfidf_norm_path)
    snowball_stemmer = SnowballStemmer("english")
    stopWords = set(stopwords.words('english'))
    for w in ['!', ',', '.', '?', '-s', '-ly', '</s>', 's']:
        stopWords.add(w)

    origin = input("Please input query(Input only E to exit):")
    while(origin != 'E'):
        start = time.time()
        query = {}
        words = word_tokenize(origin)
        # words_filter = []
        num_query_words = 0
        for word in words:
            if word not in stopWords:
                num_query_words += 1
                if snowball_stemmer.stem(word) not in query:
                    query[snowball_stemmer.stem(word)] = 1
                else:
                    query[snowball_stemmer.stem(word)] += 1
        #print(query)
        top10_index = semantic_search(query, num_query_words)
        end = time.time()
        print("Finish! See result at %s" % result_path)
```

```
        print("Time:", end - start)
        with open(result_path, 'a+') as f:
            f.write("For query: " + origin + "\n")
            f.write("Result is \n")
            for index in top10_index:
                f.write(dict[str(index)][48:])
                f.write("\n")

 f.write("============================================================
=================\n")
        origin = input("Please input query(Input only E to exit):")
```

- tf-idf计算
  - 由于矩阵很大，故提前计算好矩阵的tf-idf并存放于硬盘中

```
def cac_tfidf():
    top1k = np.load(top1k_path, allow_pickle=True)

    words_fre = np.zeros(num_index)
    inverted_table = np.load(inverted_path, allow_pickle=True)
    for i in range(num_index):
        words_fre[i] = len(inverted_table[i])

    dict_top1k = {}
    for i in range(num_index):
        dict_top1k[top1k[i][0]] = i


    for i in range(num_file):
        word = np.load(filter_path + str(i) + '.npy',
allow_pickle=True).item()
        for w, f in word.items():
            if f == 0:
                tf = 0
            else:
                tf = 1 + np.log10(f)
            index = dict_top1k[w]
            idf = np.log10(num_file / words_fre[index])
            tfidf_table[index, i] = tf * idf
    np.save(tfidf_path, tfidf_table)
```

  - 同理，对每个关键词计算idf，并存入内存

```
def cac_all_idf():
    top1k = np.load(top1k_path, allow_pickle=True)
    words_fre = np.zeros(num_index)
    inverted_table = np.load(inverted_path, allow_pickle=True)
    for i in range(num_index):
        words_fre[i] = len(inverted_table[i])
    words_idf = np.zeros(num_index)
    for i in range(num_index):
        words_idf[i] = np.log10(num_file / words_fre[i])
    np.save(top1k_idf_path, words_idf)
```

  - 同时，为了节省时间，提前计算好tf-idf每个向量的模长并存储

```
def cac_tfidf_norm():
    tfidf_table = np.load(tfidf_path)
    result = np.linalg.norm(tfidf_table, axis=0)
    np.save(tfidf_norm_path, result)
```

- 检索
  - 使用矩阵计算并返回top10结果

```
def semantic_search(query, num_query_words):

    word_vec = np.zeros(num_index)

    #print(word_idf)
    for w, f in query.items():
        if w not in word_index:
            continue
        index = word_index[w]
        word_vec[index] = 1 + np.log10(f / num_query_words)
        word_vec[index] *= word_idf[index]
    #print(word_vec)

    word_vec = word_vec.reshape((1, num_index))
    #word_vec = np.array([1,2]).reshape((1,2))
    #tfidf_table = np.array([[1,2],[3,4]])
    cos = word_vec.dot(tfidf_table) / (np.linalg.norm(word_vec) *
tfidf_norm)
    cos = np.squeeze(cos)
    #print(cos)
    #print(cos.argsort()[-10:][::-1])
    top10_index = cos.argsort()[-10:][::-1]
    print(cos[top10_index])
    return top10_index
```

## IV、word2vec

- 训练
  - 本部分使用`gensim`模块来训练模型
  - 考虑到word2vec的特性，仅进行分词，并不去停用词和提取词干
  - 为了节省内存，使用字节流的形式传入模型

```
class MySentences(object):
    def __init__(self, dict):
        dict = np.load(dict, allow_pickle=True).item()
        self.dict = dict

    def __iter__(self):
        for key, value in self.dict.items():
            with open(value, 'r', errors='ignore') as f:
                data = f.read()
                words = word_tokenize(data)
                #print(words)
                #break
                yield words

```

```
def train():
    sentences = MySentences(dict_path)
    #sentences.__iter__()
    model = gensim.models.Word2Vec(sentences)
    model.save(model_path)
```

- 计算每个文件的向量
  - 将每个文件进行分词，然后传入模型得到每个词的向量，相加取平均的结果作为该文件的向量

```
def sentence_vector(model, words):
    l = 0
    v = np.zeros(100)
    for word in words:
        if word in model:
            v += model[word]
            l += 1
    if l == 0:
        return np.zeros(100)
    v /= l
    return v

def cac_file_vec():
    model = gensim.models.Word2Vec.load(model_path)
    file_vec = [[] for i in range (517401)]
    #print(model['offer'].shape)
    dict = np.load(dict_path, allow_pickle=True).item()
    for key, value in dict.items():
        with open(value, 'r', errors='ignore') as f:
            data = f.read()
            words = word_tokenize(data)

            file_vec[int(key)] = sentence_vector(model, words)

    np.save(file_vec_path, file_vec)
```

  - 同时，同语义检索部分，预先将所有向量的模计算并存入磁盘

```
def cac_vec_file_norm():
    file_vec = np.array(np.load(file_vec_path, allow_pickle=True)
[:517401])
    temp = []
    for arr in file_vec:
        temp.append(arr)
    file_vec = np.array(temp)
    #print(file_vec)
    file_vec = file_vec.transpose()
    np.save(file_vec_numpy_path, file_vec)
    #print(file_vec.shape)
    result = np.linalg.norm(file_vec, axis=0)
    #print(result)
    np.save(file_vec_norm_path, result)
```

- 检索

- 检索过程与语义检索极其相似，只是这里使用word2vec的模型得到查询向量，然后和所有文件计算余弦相似度，并取top10

```
def word2vec_search(vec):
    vec = vec.reshape((1, 100))
    cos = vec.dot(file_vec) / (np.linalg.norm(vec) * file_vec_norm)
    cos = np.squeeze(cos)
    top10_index = cos.argsort()[-10:][::-1]
    #print(cos[top10_index])
    return top10_index
```

- 输入处理
  - 输入与语义检索部分几乎完全相同，只是不去除停用词，提取词根

```
def deal_search(result_path):
    model = gensim.models.Word2Vec.load(model_path)
    global file_vec
    file_vec = np.load(file_vec_numpy_path)
    global file_vec_norm
    file_vec_norm = np.load(file_vec_norm_path)
    dict = np.load(dict_path, allow_pickle=True).item()
    origin = input("Please input query(Input only E to exit):")
    while (origin != 'E'):
        start = time.time()
        words = word_tokenize(origin)
        query = sentence_vector(model, words)

        top10_index = word2vec_search(query)
        end = time.time()
        print("Finish! See result at %s" % result_path)
        print("Time:", end - start)
        with open(result_path, 'a+') as f:
            f.write("For query: " + origin + "\n")
            f.write("Result is \n")
            for index in top10_index:
                f.write(dict[str(index)][48:])
                f.write("\n")

 f.write("============================================================
====================\n")
        origin = input("Please input query(Input only E to exit):")
```

# 二、优化

## I、压缩倒排表

采用间距来代替文档ID，这部分详见 一、II、的倒排表部分

## II、word2vec

该部分详见 一、IV

结果见下

# 三、实验结果

选取`power、price、meeting、issues、offer`

## I、布尔查询

- `power and price`
  - Time: 0.26488184928894043
  - 由于结果很多，结果存放于`output/bool_search_result`中，助教可以查看~~~

```
For query: power and price
Result is
maildir\allen-p\all_documents\10
maildir\allen-p\all_documents\19
maildir\allen-p\all_documents\241
maildir\allen-p\all_documents\261
maildir\allen-p\all_documents\359
maildir\allen-p\all_documents\379
maildir\allen-p\all_documents\380
```

- `not meeting`
  - Time: 0.4627113342285156

```
For query: not meeting
Result is
maildir\allen-p\all_documents\1
maildir\allen-p\all_documents\102
maildir\allen-p\all_documents\103
maildir\allen-p\all_documents\105
maildir\allen-p\all_documents\107
maildir\allen-p\all_documents\108
maildir\allen-p\all_documents\109
```

- `not issues`
  - Time: 0.46822571754455566

```
--------------------------------------------------------------------------------
For query: not issues
Result is
maildir\allen-p\all_documents\10
maildir\allen-p\all_documents\102
maildir\allen-p\all_documents\103
maildir\allen-p\all_documents\104
maildir\allen-p\all_documents\105
maildir\allen-p\all_documents\106
maildir\allen-p\all_documents\107
maildir\allen-p\all_documents\108
```

- `issues or meeting`
  - Time: 0.3431692123413086

```
For query: issues or meeting
Result is
maildir\allen-p\all_documents\1
maildir\allen-p\all_documents\10
maildir\allen-p\all_documents\100
maildir\allen-p\all_documents\101
maildir\allen-p\all_documents\104
maildir\allen-p\all_documents\106
maildir\allen-p\all_documents\114
maildir\allen-p\all_documents\115
```

- `(power and price) and not issues or offer and meeting`
  - Time: 0.32477688789367676

```
For query: (power and price) and not issues or offer and meeting
Result is
maildir\allen-p\all_documents\10
maildir\allen-p\all_documents\19
maildir\allen-p\all_documents\241
maildir\allen-p\all_documents\261
maildir\allen-p\all_documents\359
maildir\allen-p\all_documents\379
maildir\allen-p\all_documents\380
```

## II、语义检索

- `power price`
  - Time: 0.2953352928161621
  - ```
    For query: power price
    Result is
    maildir\weldon-c\stagecoach\13
    maildir\weldon-c\all_documents\194
    maildir\weldon-c\discussion_threads\103
    maildir\lavorato-j\all_documents\390
    maildir\white-s\deleted_items\930
    maildir\white-s\deleted_items\929
    ```
- `meeting price`
  - Time: 0.2191011905670166
  - ```
    For query: meeting price
    Result is
    maildir\shively-h\calendar\13
    maildir\mckay-j\tasks\4
    maildir\rogers-b\inbox\1
    maildir\rogers-b\all_documents\1766
    maildir\campbell-l\inbox\230
    maildir\rogers-b\sent\714
    maildir\campbell-l\all_documents\1795
    maildir\campbell-l\all_documents\1258
    ```
- `issues meeting offer`
  - Time: 0.21761775016784668
  - ```
    For query: issues meeting offer
    Result is
    maildir\shackleton-s\meetings\30
    maildir\shackleton-s\all_documents\11376
    maildir\mckay-b\all_documents\65
    maildir\mckay-b\sent\40
    maildir\hyvl-d\all_documents\1015
    maildir\campbell-l\all_documents\1083
    maildir\hyvl-d\gas\hpl_customers\57
    ```
- `power offer issues`
  - Time: 0.22281527519226074
    ```

```
----------------------------------------------
For query: power offer issues
Result is
maildir\ring-r\eesirenewableenergy\22
maildir\presto-k\deleted_items\636
maildir\lavorato-j\sent_items\102
maildir\sager-e\calendar\32
maildir\mckay-b\all_documents\65
maildir\kaminski-v\deleted_items\1169
maildir\dorland-c\sent_items\573
```

- `power price meeting issues offer`
  - Time: 0.24132943153381348

```
For query: power price meeting issues offer
Result is
maildir\taylor-m\all_documents\3936
maildir\quigley-d\deleted_items\208
maildir\rogers-b\inbox\1
maildir\weldon-c\stagecoach\13
maildir\weldon-c\misc__gas_issues\3
maildir\weldon-c\misc__gas_issues\1
maildir\kean-s\heat_wave\155
```

## III、word2vec

- `power price`
  - Time: 0.09765958786010742

```
For query: power price
Result is
maildir\kean-s\discussion_threads\1789
maildir\kean-s\all_documents\2141
maildir\kean-s\calendar\untitled\2142
maildir\kean-s\archiving\untitled\186
maildir\dasovich-j\notes_inbox\3712
maildir\dasovich-j\all_documents\11968
maildir\dasovich-j\notes_inbox\2325
maildir\dasovich-j\all_documents\4380
maildir\dasovich-j\all_documents\4336
maildir\dasovich-j\notes_inbox\2284
----------------------------------------------------------
```

- `meeting price`
  - Time: 0.07728171348571777

```
For query: meeting price
Result is
maildir\crandell-s\inbox\rto_west\12
maildir\campbell-l\discussion_threads\1615
maildir\campbell-l\notes_inbox\567
maildir\campbell-l\all_documents\1783
maildir\jones-t\sent\5309
maildir\jones-t\all_documents\8974
maildir\hain-m\discussion_threads\633
maildir\hain-m\notes_inbox\538
maildir\hain-m\all_documents\647
maildir\steffes-j\ees_mgmt\6
```

- `issues meeting offer`
  - Time: 0.07729029655456543

```
For query: issues meeting offer
Result is
maildir\kean-s\all_documents\677
maildir\kean-s\discussion_threads\614
maildir\kean-s\archiving\untitled\1651
maildir\kean-s\calendar\untitled\677
maildir\kean-s\wto\6
maildir\hain-m\all_documents\99
maildir\kean-s\all_documents\513
maildir\hain-m\discussion_threads\1141
maildir\kean-s\sent\53
maildir\kean-s\archiving\untitled\1814
```

- `power offer issues`
  - Time: 0.07552051544189453

```
For query: power offer issues
Result is
maildir\dasovich-j\all_documents\7745
maildir\dasovich-j\notes_inbox\6124
maildir\kean-s\all_documents\1743
maildir\kean-s\discussion_threads\1488
maildir\kean-s\archiving\untitled\584
maildir\kean-s\calendar\untitled\1744
maildir\fossum-d\all_documents\616
maildir\kean-s\heat_wave\29
maildir\fossum-d\discussion_threads\369
maildir\fossum-d\notes_inbox\334
```

- `power price meeting issues offer`
  - Time: 0.07729864120483398

```
For query: power price meeting issues offer
Result is
maildir\kean-s\discussion_threads\1488
maildir\kean-s\all_documents\1743
maildir\kean-s\heat_wave\29
maildir\kean-s\archiving\untitled\584
maildir\kean-s\calendar\untitled\1744
maildir\fossum-d\all_documents\616
maildir\fossum-d\discussion_threads\369
maildir\fossum-d\notes_inbox\334
maildir\scott-s\discussion_threads\423
maildir\scott-s\all_documents\524
```

可以看到word2vec速度明显快于语义检索

# 四、运行环境与方法

- python 3.7
  - nltk、numpy、gensim
- bool检索运行方法
  - 在`bool_search.py`最下更改希望结果存储的path, 然后`python3 bool_search.py`
  - 根据提示输入bool查询
  - ```
    Please input query(Input only E to exit):(power and price) and not
     issues or offer and meeting
    Finish! See result at D:\study\USTC\2020
     .fall\web\lab1\output\bool_search_result.txt
    Time: 0.32477688789367676
    Please input query(Input only E to exit):E

    Process finished with exit code 0
    ```
- 语义检索运行方法
  - 在`semantic_search.py`最下更改希望结果存储的path, 然后`python3 semantic_search.py`
  - 根据提示输入查询
  - ```
    Please input query(Input only E to exit):offer
    [0.46981613 0.46431106 0.45986456 0.43528531 0.3991407  0.38775676
     0.3852518  0.38375876 0.3829379  0.38075948]
    Finish! See result at D:\study\USTC\2020
     .fall\web\lab1\output\semantic_search_result.txt
    Time: 0.20234274864196777
    Please input query(Input only E to exit):E
    ```
- word2vec运行方法
  - 在`word2vec.py`最下更改希望结果存储的path, 然后`python3 word2vec.py`
  - 根据提示输入查询
  - ```
    Please input query(Input only E to exit):power price meeting issues offer
    Finish! See result at D:\study\USTC\2020.fall\web\lab1\output\word2vec_search_result.txt
    Time: 0.07729864120483398
    Please input query(Input only E to exit):E

    Process finished with exit code 0
    ```
    ```