

Lab4 相关反馈&查询扩展 实验报告

10152130122 钱庭涵

创建 query_topics 文件夹，将 lab3 得到的 TF-IDF 和 BM25 两种方法的结果复制到该文件夹目录下。

VSM_query.py

主函数中，total_file_num 是所有文档数量。

```
31 total_file_num = 741856
```

读入 map2 字典。Map2 字典里存储了 disk12 里所有 document 文件对应的文档 ID、文档名称、文档长度，字典的结构如下：

{'fileID': {'DOC_name': 不含后缀的文档名称, 'DOC_length': 文档包含token数}, ...}

```
33 file2 = open('../lab3/map2.json', 'r')
34 docno_dict = json.load(file2)
35 file2.close()
```

创建向量空间模型 VSM 方法的结果文件。

```
37 file3 = open(u'./query_topics/10152130122_钱庭涵_VSM.res', 'w')
```

遍历 lab1 预处理得到的 query 文件，即 topic151-200，获取当前循环中，读入的 topic 文件的 title、desc、narr 标签的文本，去除头尾的空格，再去除文本中的两种无用符号、和标签文本开头的标识，防止其对检索结果产生干扰，然后按照空格拆分成多个 token 存入 query_list 列表，最后去除 query_list 中的空字符。

```
39 for a in os.walk('../lab1/code/Answer/query'):
40     document_files = a[2]
41     for f in document_files:
42         tree = ET.parse('../lab1/code/Answer/query/' + f)
43         root = tree.getroot()
44         query_title = re.split(' ', root.find('title').text.strip(' ').replace('\n', '').replace('"', ''))[6:]
45         query_desc = re.split(' ', root.find('desc').text.strip(' ').replace('\n', '').replace('"', ''))[9:]
46         query_narr = re.split(' ', root.find('narr').text.strip(' ').replace('\n', '').replace('"', ''))[5:]
47         query_list = query_title + query_desc + query_narr
48         while '' in query_list:
49             query_list.remove('')
```

query_tf 字典用于记录 query_list 中各 token 的出现次数, idf_token 字典用于记录该 token 在整个 disk12 文件中的 idf 值, doc_w 字典用于记录该 token 在该 document 中的 tf-idf 值, 其键的值为文档 ID。

```
51 query_tf = {}
52 idf_token = {}
53 doc_w = {}
```

遍历 query_list 中的每个 token, query_tf 字典做统计, 调用 query_index 函数, 获取当前 token 即 query_list[i] 在索引中的 value 值。

```
54 for i in range(len(query_list)):
55     if query_list[i] not in query_tf:
56         query_tf[query_list[i]] = 1
57     else:
58         query_tf[query_list[i]] += 1
59
60 token_dict = query_index(query_list[i])
```

跳转到 query_index 函数, 根据参数 token 的首字母, 判断应该调用哪个索引文件 (lab2 中按照首字母, 将总的索引分成了 27 个索引), 再读取该索引, 返回该 token 在索引中的 value 值, 返回的字典结构如下:

`{'rate': 总词频, 'pos': {'fileID': {'rate': 词频, 'pos': [pos1, pos2, ...]}, ...}}`

若该 token 并不在索引中, 则返回 -1。

```
8 def query_index(token):
9     if token[0] >= 'a' and token[0] <= 'z':
10         path = '../lab2/index_file/index_' + token[0] + '.json'
11     else:
12         path = '../lab2/index_file/index__other.json'
13     file1 = open(path, 'r')
14     index_dict = json.load(file1)
15     file1.close()
16     if token in index_dict.keys():
17         return index_dict[token]
18     else:
19         return -1
```

跳转回主函数，若 token_dict 为 -1，则说明该 token 并不在索引中，idf_token 中对应存储 0 值，再 continue 直接进入下一次循环。

否则，判断该 token 的 idf 值是否有计算过，若没有，则先计算其 idf 值并存储在 idf_token 字典中。

对于包含该 token 的所有文档，计算对应的 document 的 tf-idf 值，并存入 doc_w 字典中，这里 doc_w 字典的键的值为文档 ID。

```
61 if token_dict == -1:
62     idf_token[query_list[i]] = 0
63     continue
64
65 if query_list[i] not in idf_token:
66     idf_token[query_list[i]] = math.log(total_file_num / len(token_dict['pos'])) * 1.0
67
68 for docno in token_dict['pos'].keys():
69     if docno not in doc_w.keys():
70         doc_w[docno] = {}
71     doc_w[docno][query_list[i]] = token_dict['pos'][docno]['rate'] * idf_token[query_list[i]]
```

使用 query_tf 字典中的出现次数和 idf_token 字典中的 idf 值，计算得到 query 向量。

```
73 query_vector = []
74 for i in range(len(query_list)):
75     query_vector.append(query_tf[query_list[i]] / len(query_list) * 1.0 * idf_token[query_list[i]])
```

score_VSM 字典用于存放当前 topic 下不同文档的得分。对于 doc_w 字典中的所有候选文档，计算各自的 document 向量。

遍历 query_list 中所有 token，若当前 document 包含当前 token，则在 document 向量中存入对应的 tf-idf 值；若不包含，则在 document 向量中存入 0。

计算出当前 document 向量后，调用 conine_score 函数计算其与 query 向量的相似度，并得到该文档的得分。

```
77 score_VSM = {}
78 for docno in doc_w.keys():
79     doc_vector = []
80     for i in range(len(query_list)):
81         if query_list[i] in doc_w[docno]:
82             doc_vector.append(doc_w[docno][query_list[i]])
83         else:
84             doc_vector.append(0)
85     score_VSM[docno] = conine_score(query_vector, doc_vector)
```

跳转到 conine_score 函数，计算两个向量的余弦相似度并返回给 score_VSM 字典。

```
21 def conine_score(u, v):
22     fenzi = 0
23     fenmu1 = 0
24     fenmu2 = 0
25     for i in range(len(u)):
26         fenzi += u[i] * v[i]
27         fenmu1 += u[i] * u[i]
28         fenmu2 += v[i] * v[i]
29     return fenzi / (math.sqrt(fenmu1) * math.sqrt(fenmu2)) * 1.0
```

跳转回主函数，计算完当前 topic 的所有候选文档的得分之后，按照得分降序的顺序，对字典进行排序（字典本身无序，这里通过建立一个新的字典存储的方式来达到降序的效果）。

```
87     sort_tmp1 = sorted(score_VSM.items(), key=lambda x: x[1], reverse=True)
88     sort_score_VSM = {}
89     for elem in sort_tmp1:
90         sort_score_VSM[elem[0]] = elem[1]
```

按照 TOPIC_ID Q0 DOC_ID RANK SCORE RUN_ID 的格式，将结果写入结果文件，并同时输出到控制台，rank 从 1 开始排名，输出排名前 1000 的结果。

比如：151 0 WSJ870612-0080 1 0.8818492793706986 10152130122_VSM

```
92 rank = 1
93 for docno in sort_score_VSM.keys():
94     print(f[6:-4] + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) + ' ' +
95           str(sort_score_VSM[docno]) + ' 10152130122_VSM')
96     file3.write(f[6:-4] + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) +
97                ' ' + str(sort_score_VSM[docno]) + ' 10152130122_VSM\n')
98     rank += 1
99     if rank == 1001:
100         break
```

完成对当前 topic 的检索，输出成功信息，再进入下一个循环，处理下一个 topic。



```
102     print('query ' + f[:-4] + ' succeed')
```

最后关闭文件指针，完成所有检索。

```
104     file3.close()
```

Result :

使用 VSM 的检索结果存放在 query_topics 文件夹中。

 query_topics	2018/5/28 18:52	文件夹	
 10152130122_钱庭涵_VSM.res	2018/5/24 12:44	Compiled Resou...	2,925 KB

10152130122_钱庭涵_VSM.res ×									
1	151	0	WSJ870612-0080	1	0.8818492793706986	10152130122_VSM			
2	151	0	AP890108-0025	2	0.8780019649913248	10152130122_VSM			
3	151	0	AP880310-0051	3	0.8739929395043023	10152130122_VSM			
4	151	0	AP880519-0231	4	0.8739929395043021	10152130122_VSM			
5	151	0	WSJ910813-0096	5	0.8734792142232285	10152130122_VSM			
6	151	0	AP880509-0008	6	0.8732442345722028	10152130122_VSM			
7	151	0	WSJ870611-0139	7	0.867368220664142	10152130122_VSM			
8	151	0	AP880808-0039	8	0.8645520023498795	10152130122_VSM			
9	151	0	AP881215-0194	9	0.8611276671395669	10152130122_VSM			
10	151	0	AP880402-0095	10	0.8611276671395668	10152130122_VSM			
11	151	0	AP890920-0185	11	0.8611276671395668	10152130122_VSM			
12	151	0	AP880809-0107	12	0.8611276671395668	10152130122_VSM			
13	151	0	AP881027-0051	13	0.8611276671395667	10152130122_VSM			
14	151	0	WSJ870715-0134	14	0.8592587106458544	10152130122_VSM			
15	151	0	AP890110-0183	15	0.8580481713433941	10152130122_VSM			
16	151	0	AP890102-0126	16	0.858048171343394	10152130122_VSM			
17	151	0	AP880825-0171	17	0.8575863878964579	10152130122_VSM			
18	151	0	AP891116-0078	18	0.8572057016325153	10152130122_VSM			
19	151	0	AP880504-0183	19	0.8567640687406256	10152130122_VSM			
20	151	0	WSJ890728-0041	20	0.8564495708427922	10152130122_VSM			
21	151	0	WSJ870929-0147	21	0.8556931589624283	10152130122_VSM			
22	151	0	AP890328-0186	22	0.8556931589624281	10152130122_VSM			
23	151	0	WSJ900403-0171	23	0.8547266768093974	10152130122_VSM			
24	151	0	AP891102-0220	24	0.8545572229416581	10152130122_VSM			
25	151	0	AP890421-0165	25	0.8545359653436047	10152130122_VSM			
26	151	0	AP880921-0098	26	0.8544762411806974	10152130122_VSM			
27	151	0	AP890108-0022	27	0.854419417455674	10152130122_VSM			
28	151	0	AP880505-0259	28	0.8530926385840685	10152130122_VSM			
29	151	0	AP880312-0057	29	0.8530051101616133	10152130122_VSM			
30	151	0	AP880915-0028	30	0.8530051101616133	10152130122_VSM			

query_expansion.py

主函数中，创建需要用到的文件夹。

```
117 if (not (os.path.exists('./query_expansion_result'))):
118     os.mkdir('./query_expansion_result')
```

原始查询结果一共有三种（三种方法得到的结果），通过 input 函数获取键盘输出，输入 0 表示采用 TF-IDF 得到的结果，输入 1 表示采用 BM25 得到的结果，输入 2 表示采用 VSM 得到的结果。

```
120 query_res_list = ['TF-IDF', 'BM25', 'VSM']
121 method = input('Please enter your query method:')
122 # 0 = TF-IDF, 1 = BM25, 2 = VSM
```

根据 method 参数即选取的结果，调用 get_orignal_query 函数获取原始查询结果，将结果存入 orignal_query 字典中。

```
123 orignal_query = get_orignal_query(int(method))
```

跳转到 get_orignal_query 函数，打开 query_topics 文件夹中对应选取的原始查询结果文件，每个 topic 有 1000 条结果，按照第一个参数 topic，将每个 topic 对应的所有结果中的文档名称存入 query_dict 字典中，该字典结构为：

$\{topic_number_1: [document_name_1, document_name_2, \dots], \dots\}$

最后输出获取成功信息，再返回 query_dict 字典。

```
21 def get_orignal_query(way):
22     file2 = open('./query_topics/10152130122_钱庭涵_' + query_res_list[way] + '.res', 'r')
23     lines = file2.readlines()
24     file2.close()
25     query_dict = {}
26     count = 0
27     for topic in range(151, 201):
28         query_dict[topic] = []
29         while count < 1000:
30             res = re.split(' ', lines[count + (topic - 151) * 1000].strip('\n'))
31             query_dict[topic].append(res[2])
32             count += 1
33         count = 0
34     print('original query 10152130122_钱庭涵_' + query_res_list[way] + '.res load succeed')
35     return query_dict
```

跳转回主函数，调用 get_qrels 函数，获取标准结果答案存入 qrels 字典中。

```
124 qrels = get_qrels()
```


跳转到 get_qrels 函数，打开 lab3 中合并之后的含 topic151-200 所有标准答案的文件 qrels.res，按照第一个参数 topic，将每个 topic 对应的所有结果中的文档名称，按照第四个参数是否为 0 存入 query_dict 字典的相应位置中，参数为 0 表示为不相关文档，否则为相关文档。query_dict 字典的结构为：

$\{topic_number_1: \{ 'Cr': [doc_name_1, doc_name_2, \dots], 'Cnr': [doc_name_1, doc_name_2, \dots] \}, \dots \}$
 输出获取成功信息，再返回 qrels_dict 字典。

```
37 def get_qrels():
38     file3 = open('../lab3/qrels_for_disk12/qrels.res', 'r')
39     lines = file3.readlines()
40     file3.close()
41     qrels_dict = {}
42     for line in lines:
43         tmp_line = re.split(' ', line.strip('\n'))
44         if tmp_line[0] not in qrels_dict.keys():
45             qrels_dict[tmp_line[0]] = {'Cr': [], 'Cnr': []}
46         if tmp_line[3] == '0':
47             qrels_dict[tmp_line[0]]['Cnr'].append(tmp_line[2])
48         else:
49             qrels_dict[tmp_line[0]]['Cr'].append(tmp_line[2])
50     print('qrels.res load succeed')
51     return qrels_dict
```

跳转回主函数，进行一些参数的设定，total_file_num 是所有文档数量，top_k1 表示相关反馈中已知原始查询结果的前 top_k1 个文档是否为相关文档，alpha1、beta1、gama1 是相关反馈中向量更新时用到的参数；

top_k2 表示伪相关反馈中假定原始查询结果的前 top_k2 个文档为相关文档，alpha2、beta2、gama2 是伪相关反馈中向量更新时用到的参数。

```
126 total_file_num = 741856
127 top_k1 = 50
128 alpha1 = 1
129 beta1 = 1
130 gama1 = 1
131 top_k2 = 10
132 alpha2 = 1
133 beta2 = 0
134 gama2 = 0.15
```

读入 map2 字典，并输出读取成功信息。

```
136 file0 = open('../lab3/map2.json', 'r')
137 docno_dict = json.load(file0)
138 file0.close()
139 print('map2 load succeed')
```

创建相关反馈（RF）方法和伪相关反馈（PRF）的对应原始查询方法的结果文件。

```
141 file4 = open('./query_expansion_result/10152130122-钱庭涵_' + query_res_list[int(method)] + '_RF.res', 'w')
142 file5 = open('./query_expansion_result/10152130122-钱庭涵_' + query_res_list[int(method)] + '_PRF.res', 'w')
```

遍历 lab1 预处理得到的 query 文件，即 topic151-200，获取当前循环中，读入的 topic 文件的 title、desc、narr 标签的文本，去除头尾的空格，再去除文本中的两种无用符号、和标签文本开头的标识，防止其对检索结果产生干扰，然后按照空格拆分成多个 token 存入 query_list 列表，最后去除 query_list 中的空字符。

```
144 for a in os.walk('../lab1/code/Answer/query'):
145     document_files = a[2]
146     for f in document_files:
147         tree = ET.parse('../lab1/code/Answer/query/' + f)
148         root = tree.getroot()
149         query_title = re.split(' ', root.find('title').text.strip(' ').replace("`", "").replace("'", ""))[6:]
150         query_desc = re.split(' ', root.find('desc').text.strip(' ').replace("`", "").replace("'", ""))[9:]
151         query_narr = re.split(' ', root.find('narr').text.strip(' ').replace("`", "").replace("'", ""))[5:]
152         query_list = query_title + query_desc + query_narr
153         while '' in query_list:
154             query_list.remove('')
```

document_list 列表为 original_query 字典中对应当前 topic 的原始查询结果文档名称列表。

```
156 document_list = original_query[int(f[6:-4])]
```

采用相关反馈的方法，调用 relevance_feedback 函数，从 document_list 列表中，按照当前 topic 和 top_k1 参数，获取对应的已知相关文档列表 RF_Cr 和已知不相关文档列表 RF_Cnr，并输出获取成功信息。

```
158 RF_Cr, RF_Cnr = relevance_feedback(document_list, f[6:-4], top_k1)
159 print('RF_Cr and RF_Cnr succeed')
```

跳转到 relevance_feedback 函数，对于文档列表的前 k 个文档，根据标准结果字典 qrels 和当前 topic 的 number，将这 k 个文档对应添加到相关文档列表 Cr 和不相关文档列表 Cnr 中，最后返回 Cr 和 Cnr。

```
53 def relevance_feedback(doc_list, number, k):
54     Cr = []
55     Cnr = []
56     for doc in doc_list[:k]:
57         if doc in qrels[number]['Cr']:
58             Cr.append(doc)
59         else:
60             Cnr.append(doc)
61     return Cr, Cnr
```

跳转回主函数，采用伪相关反馈的方法，调用 pseudo_relevance_feedback 函数，从 document_list 列表中，按照 top_k2 参数，获取对应的假定相关文档列表 PRF_Cr 和假定不相关文档列表 PRF_Cnr，并输出获取成功信息。

```
160 PRF_Cr, PRF_Cnr = pseudo_relevance_feedback(document_list, top_k2)
161 print('PRF_Cr and PRF_Cnr succeed')
```

跳转到 pseudo_relevance_feedback 函数，返回文档列表的前 k 个（即相关文档列表）和剩下的文档（即不相关文档列表）。

```
63 def pseudo_relevance_feedback(doc_list, k):
64     return doc_list[:k], doc_list[k:]
```


跳转到主函数，query_tf 字典用于记录 query_list 中各 token 的出现次数，idf_token 字典用于记录该 token 在整个 disk12 文件中的 idf 值，doc_w 字典用于记录该 token 在该 document 中的 tf-idf 值，其键的值为文档名称。

```
163     query_tf = {}
164     idf_token = {}
165     doc_w = {}
```

遍历 query_list 中的每个 token，query_tf 字典做统计，调用 query_index 函数，获取当前 token 即 query_list[i] 在索引中的 value 值。

```
166     for i in range(len(query_list)):
167         if query_list[i] not in query_tf:
168             query_tf[query_list[i]] = 1
169         else:
170             query_tf[query_list[i]] += 1
171
172     token_dict = query_index(query_list[i])
```

跳转到 query_index 函数，根据参数 token 的首字母，判断应该调用哪个索引文件 (lab2 中按照首字母，将总的索引分成了 27 个索引)，再读取该索引，返回该 token 在索引中的 value 值，返回的字典结构如下：

`{'rate': 总词频, 'pos': {'fileID': {'rate': 词频, 'pos': [pos1, pos2, ...]}, ...}}`

若该 token 并不在索引中，则返回 -1。

```
8 def query_index(token):
9     if token[0] >= 'a' and token[0] <= 'z':
10         path = '../lab2/index_file/index_' + token[0] + '.json'
11     else:
12         path = '../lab2/index_file/index__other.json'
13     file1 = open(path, 'r')
14     index_dict = json.load(file1)
15     file1.close()
16     if token in index_dict.keys():
17         return index_dict[token]
18     else:
19         return -1
```

跳转回主函数，若 token_dict 为 -1，则说明该 token 并不在索引中，idf_token 中对应存储 0 值，再 continue 直接进入下一次循环。

否则，判断该 token 的 idf 值是否有计算过，若没有，则先计算其 idf 值并存储在 idf_token 字典中。

对于包含该 token 的所有文档，按照 docno_dict 字典获取对应的文档名称，判断该文档名称是否在 document_list 列表中，即判断该文档是否为向量更新需要用到的文档。若不是，则 continue 直接进入下一次循环；若是，则计算对应的 document 的 tf-idf 值，并存入 doc_w 字典中，这里 doc_w 字典的键的值为文档名称。

```
173     if token_dict == -1:
174         idf_token[query_list[i]] = 0
175         continue
176
177     if query_list[i] not in idf_token:
178         idf_token[query_list[i]] = math.log(total_file_num / len(token_dict['pos'])) * 1.0
179
180     for docno in token_dict['pos'].keys():
181         docname = docno_dict[docno]['DOC_name']
182         if docname not in document_list:
183             continue
184         if docname not in doc_w.keys():
185             doc_w[docname] = {}
186         doc_w[docname][query_list[i]] = token_dict['pos'][docno]['rate'] * idf_token[query_list[i]]
```

使用 query_tf 字典中的出现次数和 idf_token 字典中的 idf 值，计算得到初始 query 向量 q0，输出成功信息。

```
188 q0 = []
189 for i in range(len(query_list)):
190     q0.append(query_tf[query_list[i]] / len(query_list) * 1.0 * idf_token[query_list[i]])
191 print('query vector succeed')
```

对于 doc_w 字典中的所有候选文档，计算各自的 document 向量，并存入 doc_vector_dict 字典中。

遍历 query_list 中所有 token，若当前 document 包含当前 token，则在 document 向量中存入对应的 tf-idf 值；若不包含，则在 document 向量中存入 0。

最后输出成功信息。

```
193 doc_vector_dict = {}
194 for docname in doc_w.keys():
195     doc_vector_dict[docname] = []
196     for i in range(len(query_list)):
197         if query_list[i] in doc_w[docname]:
198             doc_vector_dict[docname].append(doc_w[docname][query_list[i]])
199         else:
200             doc_vector_dict[docname].append(0)
201 print('doc vector succeed')
```

调用 query_update 函数，根据初始向量 q0、需要用到的候选文档的向量字典、相关文档列表、不相关文档列表、向量更新需要用到的三个参数，对初始向量进行更新，采用相关反馈和伪相关反馈两种方法，获得两种更新后的向量，并各自输出成功信息。

```
203 RF_qm = query_update(q0, doc_vector_dict, RF_Cr, RF_Cnr, alpha1, beta1, gama1)
204 print('RF query update succeed')
205 PRF_qm = query_update(q0, doc_vector_dict, PRF_Cr, PRF_Cnr, alpha2, beta2, gama2)
206 print('PRF query update succeed')
```

跳转到 query_update 函数，根据公式，对于 query_0 向量中的每一个值进行更新，计算出更新后的值。

若相关文档列表为空，则按照以下公式计算：

$$q_m = \alpha q_0 - \gamma \frac{1}{|D_{nr}|} \sum_{d_j \in D_{nr}} d_j$$

若不相关文档列表为空，则按照以下公式计算：

$$q_m = \alpha q_0 + \beta \frac{1}{|D_r|} \sum_{d_j \in D_r} d_j$$

若两个列表都不为空，则按照以下公式计算：

$$q_m = \alpha q_0 + \beta \frac{1}{|D_r|} \sum_{d_j \in D_r} d_j - \gamma \frac{1}{|D_{nr}|} \sum_{d_j \in D_{nr}} d_j$$

最后返回更新后的向量 query_m。

```
66 def query_update(query_0, doc_vector, Cr, Cnr, alpha, beta, gama):
67     query_m = query_0
68     for i in range(len(query_0)):
69         sum_cr = 0
70         sum_cnr = 0
71         for docname in Cr:
72             sum_cr += doc_vector[docname][i]
73         for docname in Cnr:
74             sum_cnr += doc_vector[docname][i]
75         if len(Cr) == 0:
76             query_m[i] = alpha * query_0[i] - gama / len(Cnr) * sum_cnr * 1.0
77         elif len(Cnr) == 0:
78             query_m[i] = alpha * query_0[i] + beta / len(Cr) * sum_cr * 1.0
79         else:
80             query_m[i] = alpha * query_0[i] + beta / len(Cr) * sum_cr * 1.0 - gama / len(Cnr) * sum_cnr * 1.0
81     return query_m
```

跳转回主函数，调用 new_query 函数，使用更新后的向量 qm 去计算文档得分并得到当前 topic 的最终结果，采用相关反馈和伪相关反馈两种方法。

```
208 new_query(RF_qm, doc_vector_dict, 'RF', f[6:-4])
209 new_query(PRF_qm, doc_vector_dict, 'PRF', f[6:-4])
```

跳转到 **new_query** 函数，score 字典用于存放当前 topic 下不同文档的得分，对于 doc_vector 字典中的所有文档，调用 conine_score 函数计算其文档向量与 query_m 向量的相似度，并得到该文档的得分。

```
93 def new_query(query_m, doc_vector, feedback, topic):
94     score = {}
95     for docname in doc_vector.keys():
96         score[docname] = conine_score(query_m, doc_vector[docname])
```

跳转到 **conine_score** 函数，计算两个向量的余弦相似度并返回给 score 字典。

```
83 def conine_score(u, v):
84     fenzi = 0
85     fenmu1 = 0
86     fenmu2 = 0
87     for i in range(len(u)):
88         fenzi += u[i] * v[i]
89         fenmu1 += u[i] * u[i]
90         fenmu2 += v[i] * v[i]
91     return fenzi / (math.sqrt(fenmu1) * math.sqrt(fenmu2)) * 1.0
```

跳转回 **new_query** 函数，按照得分降序的顺序，对字典进行排序（字典本身无序，这里通过建立一个新的字典存储的方式来达到降序的效果）。

```
98     sort_tmp = sorted(score.items(), key=lambda x: x[1], reverse=True)
99     sort_score = {}
100     for elem in sort_tmp:
101         sort_score[elem[0]] = elem[1]
```

按照 TOPIC_ID Q0 DOC_ID RANK SCORE RUN_ID 的格式，将结果写入对应的结果文件，并同时输出到控制台，rank 从 1 开始排名，输出排名前 1000 的结果。

比如：151 0 AP880310-0051 1 0.8985208621113493 10152130122_TF-IDF_RF

```
103 rank = 1
104 for docname in sort_score.keys():
105     print(topic + ' 0 ' + docname + ' ' + str(rank) + ' ' +
106           str(sort_score[docname]) + ' 10152130122_' + query_res_list[int(method)] + '_' + feedback)
107     if feedback == 'RF':
108         file4.write(topic + ' 0 ' + docname + ' ' + str(rank) + ' ' + str(sort_score[docname])
109                   + ' 10152130122_' + query_res_list[int(method)] + '_RF\n')
110     else:
111         file5.write(topic + ' 0 ' + docname + ' ' + str(rank) + ' ' + str(sort_score[docname])
112                   + ' 10152130122_' + query_res_list[int(method)] + '_PRF\n')
113     rank += 1
```

完成采用当前反馈方法对当前 topic 的检索，输出成功信息。

```
115     print('query Topic_' + topic + ' ' + feedback + ' succeed')
```

跳转回主函数，完成采用两种反馈方法对当前 topic 的检索，再进入下一个循环，处理下一个 topic。








```
211     print('query ' + f[: -4] + ' succeed')
```

最后关闭文件指针，完成所有反馈后的检索。

```
213 file4.close()
214 file5.close()
```

Result :

对三种原始查询结果使用相关反馈和伪相关反馈的检索结果存放在 query_expansion_result 文件夹中。

 query_expansion_result	2018/5/28 18:52	文件夹	
 10152130122_钱庭涵_BM25_PRf.res	2018/5/26 10:27	Compiled Resou...	3,180 KB
 10152130122_钱庭涵_BM25_Rf.res	2018/5/26 10:27	Compiled Resou...	3,132 KB
 10152130122_钱庭涵_TF-IDF_PRf.res	2018/5/26 10:28	Compiled Resou...	3,277 KB
 10152130122_钱庭涵_TF-IDF_Rf.res	2018/5/26 10:28	Compiled Resou...	3,229 KB
 10152130122_钱庭涵_VSM_PRf.res	2018/5/26 10:27	Compiled Resou...	3,120 KB
 10152130122_钱庭涵_VSM_Rf.res	2018/5/26 10:27	Compiled Resou...	3,071 KB

10152130122_钱庭涵_TF-IDF_Rf.res x									
1	151	0	AP880310-0051	1	0.8985208621113493	10152130122_TF-IDF_Rf			
2	151	0	AP880519-0231	2	0.8985208621113492	10152130122_TF-IDF_Rf			
3	151	0	AP880229-0107	3	0.8776905989838346	10152130122_TF-IDF_Rf			
4	151	0	AP890630-0155	4	0.8776905989838346	10152130122_TF-IDF_Rf			
5	151	0	ZF108-132-636	5	0.8776905989838346	10152130122_TF-IDF_Rf			
6	151	0	AP890817-0099	6	0.8776905989838345	10152130122_TF-IDF_Rf			
7	151	0	DOE2-49-0104	7	0.8776905989838345	10152130122_TF-IDF_Rf			
8	151	0	AP880228-0064	8	0.8776905989838344	10152130122_TF-IDF_Rf			
9	151	0	AP890326-0014	9	0.8776905989838344	10152130122_TF-IDF_Rf			
10	151	0	AP891112-0020	10	0.8776905989838344	10152130122_TF-IDF_Rf			

10152130122_钱庭涵_TF-IDF_PRf.res x									
1	151	0	AP880310-0051	1	0.8985208621113493	10152130122_TF-IDF_PRf			
2	151	0	AP880519-0231	2	0.8985208621113492	10152130122_TF-IDF_PRf			
3	151	0	AP880229-0107	3	0.8776905989838346	10152130122_TF-IDF_PRf			
4	151	0	AP890630-0155	4	0.8776905989838346	10152130122_TF-IDF_PRf			
5	151	0	ZF108-132-636	5	0.8776905989838346	10152130122_TF-IDF_PRf			
6	151	0	AP890817-0099	6	0.8776905989838345	10152130122_TF-IDF_PRf			
7	151	0	DOE2-49-0104	7	0.8776905989838345	10152130122_TF-IDF_PRf			
8	151	0	AP880228-0064	8	0.8776905989838344	10152130122_TF-IDF_PRf			
9	151	0	AP890326-0014	9	0.8776905989838344	10152130122_TF-IDF_PRf			
10	151	0	AP891112-0020	10	0.8776905989838344	10152130122_TF-IDF_PRf			

10152130122_钱庭涵_BM25_RF.res ×						
1	151	0	AP880310-0051	1	0.8901382674886976	10152130122_BM25_RF
2	151	0	AP880519-0231	2	0.8901382674886976	10152130122_BM25_RF
3	151	0	AP880229-0107	3	0.8803883056174538	10152130122_BM25_RF
4	151	0	AP890630-0155	4	0.8803883056174538	10152130122_BM25_RF
5	151	0	DOE2-49-0104	5	0.8803883056174538	10152130122_BM25_RF
6	151	0	AP890326-0014	6	0.8803883056174537	10152130122_BM25_RF
7	151	0	AP880228-0064	7	0.8803883056174536	10152130122_BM25_RF
8	151	0	AP890817-0099	8	0.8803883056174536	10152130122_BM25_RF
9	151	0	AP891112-0020	9	0.8803883056174536	10152130122_BM25_RF
10	151	0	ZF108-132-636	10	0.8803883056174536	10152130122_BM25_RF

10152130122_钱庭涵_BM25_PRF.res						
1	151	0	AP880310-0051	1	0.8901382674886976	10152130122_BM25_PRF
2	151	0	AP880519-0231	2	0.8901382674886976	10152130122_BM25_PRF
3	151	0	AP880229-0107	3	0.8803883056174538	10152130122_BM25_PRF
4	151	0	AP890630-0155	4	0.8803883056174538	10152130122_BM25_PRF
5	151	0	DOE2-49-0104	5	0.8803883056174538	10152130122_BM25_PRF
6	151	0	AP890326-0014	6	0.8803883056174537	10152130122_BM25_PRF
7	151	0	AP880228-0064	7	0.8803883056174536	10152130122_BM25_PRF
8	151	0	AP890817-0099	8	0.8803883056174536	10152130122_BM25_PRF
9	151	0	AP891112-0020	9	0.8803883056174536	10152130122_BM25_PRF
10	151	0	ZF108-132-636	10	0.8803883056174536	10152130122_BM25_PRF

10152130122_钱庭涵_VSM_RF.res						
1	151	0	AP890108-0025	1	0.8880491945929255	10152130122_VSM_RF
2	151	0	WSJ910813-0096	2	0.8866010167076025	10152130122_VSM_RF
3	151	0	AP880509-0008	3	0.8855105673485562	10152130122_VSM_RF
4	151	0	WSJ870612-0080	4	0.8820264015493584	10152130122_VSM_RF
5	151	0	AP880808-0039	5	0.870393361559726	10152130122_VSM_RF
6	151	0	AP880519-0231	6	0.8687058337960273	10152130122_VSM_RF
7	151	0	AP880310-0051	7	0.8687058337960272	10152130122_VSM_RF
8	151	0	WSJ900403-0171	8	0.8676955881523126	10152130122_VSM_RF
9	151	0	AP891116-0078	9	0.8673236464980718	10152130122_VSM_RF
10	151	0	AP890108-0022	10	0.8671915566950322	10152130122_VSM_RF

10152130122_钱庭涵_VSM_PRF.res ×						
1	151	0	AP890108-0025	1	0.8880491945929255	10152130122_VSM_PRF
2	151	0	WSJ910813-0096	2	0.8866010167076025	10152130122_VSM_PRF
3	151	0	AP880509-0008	3	0.8855105673485562	10152130122_VSM_PRF
4	151	0	WSJ870612-0080	4	0.8820264015493584	10152130122_VSM_PRF
5	151	0	AP880808-0039	5	0.870393361559726	10152130122_VSM_PRF
6	151	0	AP880519-0231	6	0.8687058337960273	10152130122_VSM_PRF
7	151	0	AP880310-0051	7	0.8687058337960272	10152130122_VSM_PRF
8	151	0	WSJ900403-0171	8	0.8676955881523126	10152130122_VSM_PRF
9	151	0	AP891116-0078	9	0.8673236464980718	10152130122_VSM_PRF
10	151	0	AP890108-0022	10	0.8671915566950322	10152130122_VSM_PRF

synonym_expansion.py

主函数中，创建需要用到的文件夹。

```
78 if (not (os.path.exists('./synonym_expansion_result'))):
79     os.mkdir('./synonym_expansion_result')
```

total_file_num 是所有文档数量，total_file_len 是所有文档中 TEXT 便签的文本中 token 的总数。

```
81 total_file_num = 741856
82 total_file_len = 189470967
```

读入 map2 字典，并输出读取成功信息。

```
84 file2 = open('../lab3/map2.json', 'r')
85 docno_dict = json.load(file2)
86 file2.close()
87 print('map2 load succeed')
```

avg_file_len 是平均文本长度，并输出到控制台。

```
89 avg_file_len = total_file_len / total_file_num * 1.0
90 print('avg_file_len = ' + str(avg_file_len)) # 255.40127329293017
```

后面单词预处理需要用到的词性归并和词干还原的方法。

```
92 lemmatizaer = WordNetLemmatizer()
93 porter_stemmer = PorterStemmer()
```

k 和 b 是在 BM25 方法中用来归一化约束的，防止某个词的词频过大，在这里取 k 为 1.5，b 为 0.75，并创建 TF-IDF、BM25、VSM 三种不同排序方式的结果文件。

```
95 k = 1.5
96 b = 0.75
97 file3 = open(u'./synonym_expansion_result/10152130122_钱庭涵_TF-IDF_synonym.res', 'w')
98 file4 = open(u'./synonym_expansion_result/10152130122_钱庭涵_BM25_synonym.res', 'w')
99 file5 = open(u'./synonym_expansion_result/10152130122_钱庭涵_VSM_synonym.res', 'w')
```

遍历 lab1 预处理得到的 query 文件，即 topic151-200，获取当前循环中，读入的 topic 文件的 title、desc、narr 标签的文本，去除头尾的空格，再去除文本中的两种无用符号、和标签文本开头的标识，防止其对检索结果产生干扰，然后按照空格拆分成多个 token 存入 query_list0 列表，最后去除 query_list0 中的空字符。

```
101 for a in os.walk('../lab1/code/Answer/query'):
102     document_files = a[2]
103     for f in document_files:
104         tree = ET.parse('../lab1/code/Answer/query/' + f)
105         root = tree.getroot()
106         query_title = re.split(' ', root.find('title').text.strip(' ').replace('\n', '').replace("'", ""))[6:]
107         query_desc = re.split(' ', root.find('desc').text.strip(' ').replace('\n', '').replace("'", ""))[9:]
108         query_narr = re.split(' ', root.find('narr').text.strip(' ').replace('\n', '').replace("'", ""))[5:]
109         query_list0 = query_title + query_desc + query_narr
110         while '' in query_list0:
111             query_list0.remove('')
```

调用 query_synonym_expansion 函数，对 query_list0 进行同义词扩展，再返回扩展后的 query_list 列表。

```
113 query_list = query_synonym_expansion(query_list0)
```

跳转到 query_synonym_expansion 函数，对于 word_list 中每一个词，先将这个词本身加入到 tmp_list 列表中，再使用 wordnet 模块获取该词的第一个词性的同义词列表；

如果这个列表不为空，则对应这个列表中所有同义词，先将其转换为小写，判断这个同义词是否为该词本身，若相等，则 continue 进行下一个同义词的循环；若不想等，再判断这个同义词是否含有 '-' 和 '_' 符号。

若含有，则为连接词，continue 进行下一个同义词的循环；若不含有，再判断这个同义词是否为停用词。

若不是停用词，则对这个同义词进行词性归并和词干还原，再判断 tmp_list 中是否含有这个预处理之后的同义词，若不含有，则将其添加到 tmp_list 中。

最后返回 tmp_list 列表。

```
35 def query_synonym_expansion(word_list):
36     tmp_list = []
37     for word in word_list:
38         tmp_list.append(word)
39         synsets = wn.synsets(word)
40         if len(synsets) != 0:
41             for name in synsets[0].lemma_names():
42                 name = name.lower()
43                 if name == word:
44                     continue
45                 if (name.find('-') != -1) or (name.find('_') != -1):
46                     continue
47                 if name not in stopwords.words('english'):
48                     name_preprocess = porter_stemmer.stem(lemmatizaer.lemmatize(name))
49                     if name_preprocess not in tmp_list:
50                         tmp_list.append(name_preprocess)
51     return tmp_list
```

跳转回主函数，score_tf_idf 和 score_bm25 两个字典用于存放当前 topic 下两种方法各自的不同文档的得分，query_tf 字典用于记录 query_list 中各 token 的出现次数，idf_token 字典用于记录该 token 在整个 disk12 文件中的 idf 值，doc_w 字典用于记录该 token 在该 document 中的 tf-idf 值，其键的值为文档 ID。

```
115 score_tf_idf = {}
116 score_bm25 = {}
117 query_tf = {}
118 idf_token = {}
119 doc_w = {}
```

遍历 query_list 中的每个 token，query_tf 字典做统计，调用 query_index 函数，获取当前 token 即 query_list[i] 在索引中的 value 值。

```
120         for i in range(len(query_list)):
121             if query_list[i] not in query_tf:
122                 query_tf[query_list[i]] = 1
123             else:
124                 query_tf[query_list[i]] += 1
125
126         token_dict = query_index(query_list[i])
```

跳转到 query_index 函数，根据参数 token 的首字母，判断应该调用哪个索引文件 (lab2 中按照首字母，将总的索引分成了 27 个索引)，再读取该索引，返回该 token 在索引中的 value 值，返回的字典结构如下：

$\{\text{'rate': 总词频, 'pos': \{\text{'fileID': \{\text{'rate': 词频, 'pos': [pos1, pos2, \dots]}\}, \dots\}}\}$

若该 token 并不在索引中，则返回 -1。

```
12 def query_index(token):
13     if token[0] >= 'a' and token[0] <= 'z':
14         path = '../lab2/index_file/index_' + token[0] + '.json'
15     else:
16         path = '../lab2/index_file/index__other.json'
17     file1 = open(path, 'r')
18     index_dict = json.load(file1)
19     file1.close()
20     if token in index_dict.keys():
21         return index_dict[token]
22     else:
23         return -1
```

跳转回主函数，若 token_dict 为 -1，则说明该 token 并不在索引中，idf_token 中对应存储 0 值，再 continue 直接进入下一次循环。

否则，计算 TD-IDF 和 BM25 两种方法中需要用到的 idf 值。

再判断该 token 的 idf 值是否有计算过，若没有，则计算其 idf 值并存储在 idf_token 字典中。

对于包含该 token 的所有文档，分别计算 TF-IDF 和 BM25 两种方法的 score 得分，得分的初始化为 0，之后一个一个叠加。并计算对应的 document 的 tf-idf 值，并存入 doc_w 字典中，这里 doc_w 字典的键的值为文档 ID。

```
127     if token_dict == -1:
128         idf_token[query_list[i]] = 0
129         continue
130
131     idf_t = math.log(total_file_num / len(token_dict['pos']) * 1.0)
132     idf_qi = math.log((total_file_num - len(token_dict['pos']) + 0.5) /
133                     (len(token_dict['pos']) + 0.5) * 1.0)
134     if query_list[i] not in idf_token:
135         idf_token[query_list[i]] = math.log(total_file_num / len(token_dict['pos']) * 1.0)
136
137     for docno in token_dict['pos'].keys():
138         if docno not in score_tf_idf.keys():
139             score_tf_idf[docno] = 0
140             score_bm25[docno] = 0
141             doc_w[docno] = {}
142             score_tf_idf[docno] += token_dict['pos'][docno]['rate'] * idf_t
143             score_bm25[docno] += idf_qi * token_dict['pos'][docno]['rate'] * (k+1) / \
144                                 (token_dict['pos'][docno]['rate'] + k *
145                                 (1-b + b * docno_dict[docno]['DOC_length'] / avg_file_len * 1.0)) * 1.0
146             doc_w[docno][query_list[i]] = token_dict['pos'][docno]['rate'] * idf_token[query_list[i]]
```

使用 query_tf 字典中的出现次数和 idf_token 字典中的 idf 值，计算得到 query 向量。

```
148 query_vector = []
149 for i in range(len(query_list)):
150     query_vector.append(query_tf[query_list[i]] / len(query_list) * 1.0 * idf_token[query_list[i]])
```

score_VSM 字典用于存放当前 topic 下不同文档的得分。对于 doc_w 字典中的所有候选文档，计算各自的 document 向量。

遍历 query_list 中所有 token，若当前 document 包含当前 token，则在 document 向量中存入对应的 tf-idf 值；若不包含，则在 document 向量中存入 0。

计算出当前 document 向量后，调用 conine_score 函数计算其与 query 向量的相似度，并得到该文档的得分。

```
152 score_VSM = {}
153 for docno in doc_w.keys():
154     doc_vector = []
155     for i in range(len(query_list)):
156         if query_list[i] in doc_w[docno]:
157             doc_vector.append(doc_w[docno][query_list[i]])
158         else:
159             doc_vector.append(0)
160     score_VSM[docno] = conine_score(query_vector, doc_vector)
```

跳转到 `conine_score` 函数，计算两个向量的余弦相似度并返回给 `score_VSM` 字典。

```
25 def conine_score(u, v):
26     fenzi = 0
27     fenmu1 = 0
28     fenmu2 = 0
29     for i in range(len(u)):
30         fenzi += u[i] * v[i]
31         fenmu1 += u[i] * u[i]
32         fenmu2 += v[i] * v[i]
33     return fenzi / (math.sqrt(fenmu1) * math.sqrt(fenmu2)) * 1.0
```

跳转回主函数，调用 `get_result` 函数，得到当前 topic 下三种方法各自的结果。

```
162 get_result(f[6:-4], score_tf_idf, 'TF-IDF')
163 get_result(f[6:-4], score_bm25, 'BM25')
164 get_result(f[6:-4], score_VSM, 'VSM')
```

跳转到 `get_result` 函数，按照得分降序的顺序，对字典进行排序（字典本身无序，这里通过建立一个新的字典存储的方式来达到降序的效果）。

```
53 def get_result(topic, score, method):
54     sort_tmp = sorted(score.items(), key=lambda x: x[1], reverse=True)
55     sort_score = {}
56     for elem in sort_tmp:
57         sort_score[elem[0]] = elem[1]
```

按照 `TOPIC_ID Q0 DOC_ID RANK SCORE RUN_ID` 的格式，将结果写入对应的结果文件，并同时输出到控制台，rank 从 1 开始排名，输出排名前 1000 的结果。

比如：151 0 AP890411-0076 1 2.224548572746962 10152130122_TF-IDF_synonym

```
59 rank = 1
60 for docno in sort_score.keys():
61     print(topic + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) + ' ' +
62           str(sort_score[docno]) + ' 10152130122_' + method + '_synonym')
63     if method == 'TF-IDF':
64         file3.write(topic + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) + ' ' +
65                    str(sort_score[docno]) + ' 10152130122_' + method + '_synonym\n')
66     elif method == 'BM25':
67         file4.write(topic + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) + ' ' +
68                    str(sort_score[docno]) + ' 10152130122_' + method + '_synonym\n')
69     else:
70         file5.write(topic + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) + ' ' +
71                    str(sort_score[docno]) + ' 10152130122_' + method + '_synonym\n')
72     rank += 1
73     if rank == 1001:
74         break
```

完成采用方法对当前 topic 的检索，输出成功信息。

```
76 print('query Topic_' + topic + ' ' + method + ' succeed')
```

跳转回主函数，完成采用三种方法+同义词扩展后对当前 topic 的检索，再进入下一个循环，处理下一个 topic。





```
166         print('query ' + f[:-4] + ' succeed')
```

最后关闭文件指针，完成所有同义词扩展后的检索。

```
168     file3.close()  
169     file4.close()  
170     file5.close()
```


Result :

使用三种方法+同义词扩展后的检索结果存放在 synonym_expansion_result 文件夹中。

 synonym_expansion_result	2018/5/28 18:53	文件夹	
 10152130122_钱庭涵_BM25_synonym.res	2018/5/27 16:12	Compiled Resou...	3,307 KB
 10152130122_钱庭涵_TF-IDF_synonym.res	2018/5/27 16:11	Compiled Resou...	3,436 KB
 10152130122_钱庭涵_VSM_synonym.res	2018/5/27 16:12	Compiled Resou...	3,317 KB














10152130122_钱庭涵_TF-IDF_synonym.res ✕									
1	151	0	AP890411-0076	1	2.224548572746962	10152130122_TF-IDF_synonym			
2	151	0	FR89517-0017	2	2.17239682879441	10152130122_TF-IDF_synonym			
3	151	0	FR891205-0022	3	2.126910745970336	10152130122_TF-IDF_synonym			
4	151	0	AP881206-0124	4	2.1155319751852	10152130122_TF-IDF_synonym			
5	151	0	AP890125-0112	5	2.0139359072509704	10152130122_TF-IDF_synonym			
6	151	0	AP890408-0056	6	1.9567597473976257	10152130122_TF-IDF_synonym			
7	151	0	AP881018-0003	7	1.880591446580549	10152130122_TF-IDF_synonym			
8	151	0	AP881002-0014	8	1.8261581941828688	10152130122_TF-IDF_synonym			
9	151	0	AP880407-0033	9	1.80287311969914	10152130122_TF-IDF_synonym			
10	151	0	AP880825-0054	10	1.7501013820870315	10152130122_TF-IDF_synonym			

10152130122_钱庭涵_BM25_synonym.res ✕									
1	151	0	FR89517-0017	1	9.41167344539938	10152130122_BM25_synonym			
2	151	0	FR891205-0022	2	8.981409774038053	10152130122_BM25_synonym			
3	151	0	AP881018-0003	3	8.57892251321101	10152130122_BM25_synonym			
4	151	0	DOE1-02-1138	4	7.607463554604731	10152130122_BM25_synonym			
5	151	0	FR88105-0022	5	6.894210984767753	10152130122_BM25_synonym			
6	151	0	FR89321-0029	6	6.699929032733861	10152130122_BM25_synonym			
7	151	0	DOE2-16-0524	7	6.515783986482806	10152130122_BM25_synonym			
8	151	0	DOE2-07-0247	8	6.289534606863489	10152130122_BM25_synonym			
9	151	0	FR891128-0028	9	6.238733547786856	10152130122_BM25_synonym			
10	151	0	FR89317-0028	10	6.238733547786856	10152130122_BM25_synonym			

10152130122_钱庭涵_VSM_synonym.res ✕									
1	151	0	WSJ870612-0080	1	0.837220708344276	10152130122_VSM_synonym			
2	151	0	AP880519-0231	2	0.8330433784753044	10152130122_VSM_synonym			
3	151	0	AP880310-0051	3	0.8304915014289348	10152130122_VSM_synonym			
4	151	0	WSJ910813-0096	4	0.8302125884773282	10152130122_VSM_synonym			
5	151	0	AP880509-0008	5	0.8244499430824295	10152130122_VSM_synonym			
6	151	0	WSJ870611-0139	6	0.8244014907714924	10152130122_VSM_synonym			
7	151	0	AP880808-0039	7	0.8240447815443568	10152130122_VSM_synonym			
8	151	0	AP881215-0194	8	0.8207808881606771	10152130122_VSM_synonym			
9	151	0	AP880809-0107	9	0.820780888160677	10152130122_VSM_synonym			
10	151	0	AP881027-0051	10	0.820780888160677	10152130122_VSM_synonym			

Trec 评测：

使用 trec_eval 进行评测的结果存放在 trec_res 文件夹中。

 trec_res	2018/5/28 18:53	文件夹	
 trec_BM25.res	2018/5/23 14:12	Compiled Resou...	1 KB
 trec_BM25_PRf.res	2018/5/26 10:33	Compiled Resou...	2 KB
 trec_BM25_Rf.res	2018/5/26 10:33	Compiled Resou...	1 KB
 trec_BM25_synonym.res	2018/5/27 16:21	Compiled Resou...	2 KB
 trec_TF-IDF.res	2018/5/23 14:12	Compiled Resou...	1 KB
 trec_TF-IDF_PRf.res	2018/5/26 10:33	Compiled Resou...	2 KB
 trec_TF-IDF_Rf.res	2018/5/26 10:32	Compiled Resou...	2 KB
 trec_TF-IDF_synonym.res	2018/5/27 16:21	Compiled Resou...	2 KB
 trec_VSM.res	2018/5/24 13:37	Compiled Resou...	1 KB
 trec_VSM_PRf.res	2018/5/26 10:33	Compiled Resou...	1 KB
 trec_VSM_Rf.res	2018/5/26 10:33	Compiled Resou...	1 KB
 trec_VSM_synonym.res	2018/5/27 16:21	Compiled Resou...	2 KB

trec_TF-IDF.res		
1	runid	all 10152130122_TF-IDF
2	num_q	all 50
3	num_ret	all 50000
4	num_rel	all 9805
5	num_rel_ret	all 2972
6	map	all 0.0538
7	gm_map	all 0.0132
8	Rprec	all 0.1102
9	bpref	all 0.1135
10	recip_rank	all 0.2411
11	iprec_at_recall_0.00	all 0.2937
12	iprec_at_recall_0.10	all 0.1413
13	iprec_at_recall_0.20	all 0.1203
14	iprec_at_recall_0.30	all 0.0921
15	iprec_at_recall_0.40	all 0.0610
16	iprec_at_recall_0.50	all 0.0430
17	iprec_at_recall_0.60	all 0.0127
18	iprec_at_recall_0.70	all 0.0059
19	iprec_at_recall_0.80	all 0.0026
20	iprec_at_recall_0.90	all 0.0000
21	iprec_at_recall_1.00	all 0.0000
22	P_5	all 0.0920
23	P_10	all 0.1120
24	P_15	all 0.1107
25	P_20	all 0.1190
26	P_30	all 0.1267
27	P_100	all 0.1162
28	P_200	all 0.1018
29	P_500	all 0.0799
30	P_1000	all 0.0594

trec_BM25.res		
1	runid	all 10152130122_BM25
2	num_q	all 50
3	num_ret	all 50000
4	num_rel	all 9805
5	num_rel_ret	all 1001
6	map	all 0.0075
7	gm_map	all 0.0009
8	Rprec	all 0.0306
9	bpref	all 0.0418
10	recip_rank	all 0.1412
11	iprec_at_recall_0.00	all 0.1586
12	iprec_at_recall_0.10	all 0.0268
13	iprec_at_recall_0.20	all 0.0105
14	iprec_at_recall_0.30	all 0.0055
15	iprec_at_recall_0.40	all 0.0005
16	iprec_at_recall_0.50	all 0.0005
17	iprec_at_recall_0.60	all 0.0004
18	iprec_at_recall_0.70	all 0.0000
19	iprec_at_recall_0.80	all 0.0000
20	iprec_at_recall_0.90	all 0.0000
21	iprec_at_recall_1.00	all 0.0000
22	P_5	all 0.0480
23	P_10	all 0.0480
24	P_15	all 0.0480
25	P_20	all 0.0480
26	P_30	all 0.0427
27	P_100	all 0.0374
28	P_200	all 0.0312
29	P_500	all 0.0247
30	P_1000	all 0.0200

trec_VSM.res		
1	runid	all 10152130122_VSM
2	num_q	all 50
3	num_ret	all 50000
4	num_rel	all 9805
5	num_rel_ret	all 3520
6	map	all 0.1277
7	gm_map	all 0.0268
8	Rprec	all 0.1843
9	bpref	all 0.2066
10	recip_rank	all 0.4599
11	iprec_at_recall_0.00	all 0.5125
12	iprec_at_recall_0.10	all 0.2817
13	iprec_at_recall_0.20	all 0.2252
14	iprec_at_recall_0.30	all 0.1705
15	iprec_at_recall_0.40	all 0.1350
16	iprec_at_recall_0.50	all 0.1058
17	iprec_at_recall_0.60	all 0.0896
18	iprec_at_recall_0.70	all 0.0613
19	iprec_at_recall_0.80	all 0.0385
20	iprec_at_recall_0.90	all 0.0147
21	iprec_at_recall_1.00	all 0.0000
22	P_5	all 0.3120
23	P_10	all 0.2960
24	P_15	all 0.2840
25	P_20	all 0.2800
26	P_30	all 0.2633
27	P_100	all 0.2090
28	P_200	all 0.1608
29	P_500	all 0.1062
30	P_1000	all 0.0704

本页为 TF-IDF、BM25、VSM 三种方法的原始查询结果对比

	trec_TF-IDF_RF.res	
1	runid	all 10152130122_TF-IDF_RF
2	num_q	all 50
3	num_ret	all 50000
4	num_rel	all 9805
5	num_rel_ret	all 2972
6	map	all 0.1478
7	gm_map	all 0.0499
8	Rprec	all 0.2268
9	bpref	all 0.2137
10	recip_rank	all 0.6261
11	iprec_at_recall_0.00	all 0.6813
12	iprec_at_recall_0.10	all 0.4194
13	iprec_at_recall_0.20	all 0.2938
14	iprec_at_recall_0.30	all 0.2118
15	iprec_at_recall_0.40	all 0.1364
16	iprec_at_recall_0.50	all 0.1062
17	iprec_at_recall_0.60	all 0.0553
18	iprec_at_recall_0.70	all 0.0342
19	iprec_at_recall_0.80	all 0.0105
20	iprec_at_recall_0.90	all 0.0000
21	iprec_at_recall_1.00	all 0.0000
22	P_5	all 0.4840
23	P_10	all 0.4500
24	P_15	all 0.4187
25	P_20	all 0.4130
26	P_30	all 0.3920
27	P_100	all 0.2494
28	P_200	all 0.1759
29	P_500	all 0.1023
30	P_1000	all 0.0594

	trec_TF-IDF_PRf.res	
1	runid	all 10152130122_TF-IDF_PRf
2	num_q	all 50
3	num_ret	all 50000
4	num_rel	all 9805
5	num_rel_ret	all 2972
6	map	all 0.1478
7	gm_map	all 0.0499
8	Rprec	all 0.2268
9	bpref	all 0.2137
10	recip_rank	all 0.6261
11	iprec_at_recall_0.00	all 0.6813
12	iprec_at_recall_0.10	all 0.4194
13	iprec_at_recall_0.20	all 0.2938
14	iprec_at_recall_0.30	all 0.2118
15	iprec_at_recall_0.40	all 0.1364
16	iprec_at_recall_0.50	all 0.1062
17	iprec_at_recall_0.60	all 0.0553
18	iprec_at_recall_0.70	all 0.0342
19	iprec_at_recall_0.80	all 0.0105
20	iprec_at_recall_0.90	all 0.0000
21	iprec_at_recall_1.00	all 0.0000
22	P_5	all 0.4840
23	P_10	all 0.4500
24	P_15	all 0.4187
25	P_20	all 0.4130
26	P_30	all 0.3920
27	P_100	all 0.2494
28	P_200	all 0.1759
29	P_500	all 0.1023
30	P_1000	all 0.0594

本页为对 TF-IDF 和 BM25 方法得到的原始查询结果进行相关反馈(RF)和伪相关反馈(PRF)得到的结果对比

	trec_BM25_RF.res	
1	runid	all 10152130122_BM25_RF
2	num_q	all 50
3	num_ret	all 50000
4	num_rel	all 9805
5	num_rel_ret	all 1001
6	map	all 0.0458
7	gm_map	all 0.0049
8	Rprec	all 0.0811
9	bpref	all 0.0808
10	recip_rank	all 0.4348
11	iprec_at_recall_0.00	all 0.4821
12	iprec_at_recall_0.10	all 0.1548
13	iprec_at_recall_0.20	all 0.0913
14	iprec_at_recall_0.30	all 0.0511
15	iprec_at_recall_0.40	all 0.0140
16	iprec_at_recall_0.50	all 0.0121
17	iprec_at_recall_0.60	all 0.0071
18	iprec_at_recall_0.70	all 0.0000
19	iprec_at_recall_0.80	all 0.0000
20	iprec_at_recall_0.90	all 0.0000
21	iprec_at_recall_1.00	all 0.0000
22	P_5	all 0.2880
23	P_10	all 0.2460
24	P_15	all 0.2373
25	P_20	all 0.2200
26	P_30	all 0.1993
27	P_100	all 0.1082
28	P_200	all 0.0691
29	P_500	all 0.0367
30	P_1000	all 0.0200

	trec_BM25_PRf.res	
1	runid	all 10152130122_BM25_PRf
2	num_q	all 50
3	num_ret	all 50000
4	num_rel	all 9805
5	num_rel_ret	all 1001
6	map	all 0.0458
7	gm_map	all 0.0049
8	Rprec	all 0.0811
9	bpref	all 0.0808
10	recip_rank	all 0.4348
11	iprec_at_recall_0.00	all 0.4821
12	iprec_at_recall_0.10	all 0.1548
13	iprec_at_recall_0.20	all 0.0913
14	iprec_at_recall_0.30	all 0.0511
15	iprec_at_recall_0.40	all 0.0140
16	iprec_at_recall_0.50	all 0.0121
17	iprec_at_recall_0.60	all 0.0071
18	iprec_at_recall_0.70	all 0.0000
19	iprec_at_recall_0.80	all 0.0000
20	iprec_at_recall_0.90	all 0.0000
21	iprec_at_recall_1.00	all 0.0000
22	P_5	all 0.2880
23	P_10	all 0.2460
24	P_15	all 0.2373
25	P_20	all 0.2200
26	P_30	all 0.1993
27	P_100	all 0.1082
28	P_200	all 0.0691
29	P_500	all 0.0367
30	P_1000	all 0.0200

本页为对 VSM 方法得到的原始查询结果进行相关反馈 (RF) 和伪相关反馈 (PRF) 得到的结果对比

trec_VSM_RF.res		
1	runid	all 10152130122_VSM_RF
2	num_q	all 50
3	num_ret	all 50000
4	num_rel	all 9805
5	num_rel_ret	all 3520
6	map	all 0.1448
7	gm_map	all 0.0366
8	Rprec	all 0.1994
9	bpref	all 0.2206
10	recip_rank	all 0.5728
11	iprec_at_recall_0.00	all 0.6264
12	iprec_at_recall_0.10	all 0.3394
13	iprec_at_recall_0.20	all 0.2680
14	iprec_at_recall_0.30	all 0.1840
15	iprec_at_recall_0.40	all 0.1368
16	iprec_at_recall_0.50	all 0.1085
17	iprec_at_recall_0.60	all 0.0894
18	iprec_at_recall_0.70	all 0.0649
19	iprec_at_recall_0.80	all 0.0405
20	iprec_at_recall_0.90	all 0.0131
21	iprec_at_recall_1.00	all 0.0000
22	P_5	all 0.3840
23	P_10	all 0.3520
24	P_15	all 0.3413
25	P_20	all 0.3280
26	P_30	all 0.3080
27	P_100	all 0.2262
28	P_200	all 0.1679
29	P_500	all 0.1089
30	P_1000	all 0.0704

trec_VSM_PRF.res		
1	runid	all 10152130122_VSM_PRF
2	num_q	all 50
3	num_ret	all 50000
4	num_rel	all 9805
5	num_rel_ret	all 3520
6	map	all 0.1448
7	gm_map	all 0.0366
8	Rprec	all 0.1994
9	bpref	all 0.2206
10	recip_rank	all 0.5728
11	iprec_at_recall_0.00	all 0.6264
12	iprec_at_recall_0.10	all 0.3394
13	iprec_at_recall_0.20	all 0.2680
14	iprec_at_recall_0.30	all 0.1840
15	iprec_at_recall_0.40	all 0.1368
16	iprec_at_recall_0.50	all 0.1085
17	iprec_at_recall_0.60	all 0.0894
18	iprec_at_recall_0.70	all 0.0649
19	iprec_at_recall_0.80	all 0.0405
20	iprec_at_recall_0.90	all 0.0131
21	iprec_at_recall_1.00	all 0.0000
22	P_5	all 0.3840
23	P_10	all 0.3520
24	P_15	all 0.3413
25	P_20	all 0.3280
26	P_30	all 0.3080
27	P_100	all 0.2262
28	P_200	all 0.1679
29	P_500	all 0.1089
30	P_1000	all 0.0704

trec_TF-IDF_synonym.res			trec_BM25_synonym.res		
1	runid	all 10152130122_TF-IDF_synonym	1	runid	all 10152130122_BM25_synonym
2	num_q	all 50	2	num_q	all 50
3	num_ret	all 50000	3	num_ret	all 50000
4	num_rel	all 9805	4	num_rel	all 9805
5	num_rel_ret	all 2864	5	num_rel_ret	all 875
6	map	all 0.0509	6	map	all 0.0064
7	gm_map	all 0.0116	7	gm_map	all 0.0006
8	Rprec	all 0.1063	8	Rprec	all 0.0273
9	bpref	all 0.1119	9	bpref	all 0.0392
10	recip_rank	all 0.2246	10	recip_rank	all 0.1278
11	iprec_at_recall_0.00	all 0.2724	11	iprec_at_recall_0.00	all 0.1407
12	iprec_at_recall_0.10	all 0.1336	12	iprec_at_recall_0.10	all 0.0220
13	iprec_at_recall_0.20	all 0.1155	13	iprec_at_recall_0.20	all 0.0082
14	iprec_at_recall_0.30	all 0.0815	14	iprec_at_recall_0.30	all 0.0045
15	iprec_at_recall_0.40	all 0.0566	15	iprec_at_recall_0.40	all 0.0003
16	iprec_at_recall_0.50	all 0.0412	16	iprec_at_recall_0.50	all 0.0000
17	iprec_at_recall_0.60	all 0.0121	17	iprec_at_recall_0.60	all 0.0000
18	iprec_at_recall_0.70	all 0.0053	18	iprec_at_recall_0.70	all 0.0000
19	iprec_at_recall_0.80	all 0.0022	19	iprec_at_recall_0.80	all 0.0000
20	iprec_at_recall_0.90	all 0.0000	20	iprec_at_recall_0.90	all 0.0000
21	iprec_at_recall_1.00	all 0.0000	21	iprec_at_recall_1.00	all 0.0000
22	P_5	all 0.0800	22	P_5	all 0.0480
23	P_10	all 0.1040	23	P_10	all 0.0440
24	P_15	all 0.1080	24	P_15	all 0.0387
25	P_20	all 0.1160	25	P_20	all 0.0410
26	P_30	all 0.1240	26	P_30	all 0.0367
27	P_100	all 0.1080	27	P_100	all 0.0328
28	P_200	all 0.0959	28	P_200	all 0.0278
29	P_500	all 0.0749	29	P_500	all 0.0212
30	P_1000	all 0.0573	30	P_1000	all 0.0175

trec_VSM_synonym.res		
1	runid	all 10152130122_VSM_synonym
2	num_q	all 50
3	num_ret	all 50000
4	num_rel	all 9805
5	num_rel_ret	all 3681
6	map	all 0.1365
7	gm_map	all 0.0309
8	Rprec	all 0.1905
9	bpref	all 0.2105
10	recip_rank	all 0.5047
11	iprec_at_recall_0.00	all 0.5581
12	iprec_at_recall_0.10	all 0.3037
13	iprec_at_recall_0.20	all 0.2420
14	iprec_at_recall_0.30	all 0.1844
15	iprec_at_recall_0.40	all 0.1435
16	iprec_at_recall_0.50	all 0.1076
17	iprec_at_recall_0.60	all 0.0915
18	iprec_at_recall_0.70	all 0.0682
19	iprec_at_recall_0.80	all 0.0402
20	iprec_at_recall_0.90	all 0.0160
21	iprec_at_recall_1.00	all 0.0000
22	P_5	all 0.3320
23	P_10	all 0.3180
24	P_15	all 0.3013
25	P_20	all 0.2960
26	P_30	all 0.2793
27	P_100	all 0.2170
28	P_200	all 0.1629
29	P_500	all 0.1104
30	P_1000	all 0.0736

本页为使用 TF-IDF、BM25、VSM
三种方法的再进行同义词扩展
后的结果对比

Trec 评测结果对比：(map、P_5、P_10)

检索方法	map	P_5	P_10
TF-IDF	0.0538	0.0920	0.1120
TF-IDF_RF	0.1478	0.4840	0.4500
TF-IDF_PRF	0.1478	0.4840	0.4500
TF-IDF_synonym	0.0509	0.0800	0.1040
BM25	0.0075	0.0480	0.0480
BM25_RF	0.0458	0.2880	0.2460
BM25_PRF	0.0458	0.2880	0.2460
BM25_synonym	0.0064	0.0480	0.0440
VSM	0.1277	0.3120	0.2960
VSM_RF	0.1448	0.3840	0.3520
VSM_PRF	0.1448	0.3840	0.3520
VSM_synonym	0.1365	0.3320	0.3180

一些问题与讨论：

1. Lab3 中提到的问题：

按理说 BM25 的方法得到的效果应该比 TF-IDF 要好,但是我仔细检查了自己代码,并对比其他成功的同学的代码之后,并没有发现自己的代码哪里有错误, BM25 的 k 参数也是经过多次测试对比结果之后才选取为 1.5。

最终我的猜测是,可能由于预处理方法的不同,在我经过预处理之后得到的 query 和 document 下, TF-IDF 方法要比 BM25 方法效果好。

2. RF 和 PRF 的结果：

RF 和 PRF 的 top_k、alpha、beta、gama 参数都是经过多次测试对比结果之后选取的：

RF : top_k=50, alpha=1, beta=1, gama=1

PRF : top_k=10, alpha=1, beta=0, gama=0.15

猜测由于参数选取过于精确,导致 RF 和 PRF 两种方法得到的结果相同。

3. 同义词扩展的结果：

虽然我已经尽可能的选取有用的同义词,但同义词扩展之后效果还是变差了,是 topic 中文本本身造成的。