

Lab3 检索模型 实验报告

10152130122 钱庭涵

boolean_query.py

主函数中，创建需要用到的文件夹。

```
17 if (not (os.path.exists('./boolean_query_topics'))):
18     os.mkdir('./boolean_query_topics')
```

获取键盘输入，即查询次数和布尔检索的布尔表达式，布尔表达式中，用到的词为 topic.151-200 中出现的、预处理过的 token (token 中若有字母则全为小写)，逻辑运算使用的均为大写的 AND、OR、NOT。

```
20 time = input('Please enter your query number:')
21 print('Please enter your boolean query:')
22 boolean_str = input()
23 print()
```

读取 lab2 中建立的 map, 该 map 是一个文件名与文件 ID 的映射, 文件名为字典的 key, 文件 ID 为字典的 value, ID 从 0 开始标记。

```
25 file2 = open('../lab2/map.json', 'r')
26 docno_dict = json.load(file2)
27 file2.close()
```

str_list 为布尔表达式按空格拆分后的字符串列表, posting_list 为包含第一个 token 的文档 ID 列表, boolean_way 为两个 token 之间的逻辑运算方式, 初始化为空字符串。

```
29 str_list = re.split(' ', boolean_str)
30 posting_list = query_index(str_list[0])
31 boolean_way = ''
```

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, 返回包含该 token 的文档 ID 列表。

```
6 def query_index(token):
7     if token[0] >= 'a' and token[0] <= 'z':
8         path = '../lab2/index_file/index_' + token[0] + '.json'
9     else:
10        path = '../lab2/index_file/index__other.json'
11    file1 = open(path, 'r')
12    index_dict = json.load(file1)
13    file1.close()
14    docno_list = [key for key in index_dict[token]['pos'].keys()]
15    return docno_list
```

跳转回主函数，str_list[0]一定是一个 token，因此从列表中第二个元素开始遍历。
若 str_list[i]为'AND'或'OR'，则逻辑运算方式 boolean_way 为相应的'AND'或'OR'；
若 str_list[i]为'NOT'，则更新 boolean_way，连接上一个' NOT'；
若都不是，则 str_list[i]为一个 token。

```
32 for i in range(1, len(str_list)):
33     if str_list[i] == 'AND' or str_list[i] == 'OR':
34         boolean_way = str_list[i]
35     elif str_list[i] == 'NOT':
36         boolean_way += ' NOT'
37     else:
```

当 str_list[i]为一个 token 时，先获得包含该 token 的文档 ID 列表并存入 posting_list2，然后开始进行逻辑运算。

若逻辑运算方式为'AND'，则取上次计算后得到的文档 ID 列表 posting_list 与本次的 posting_list2 的交集，存入 posting_list_tmp；

若逻辑运算方式为'OR'，则先将 posting_list 与 posting_list2 合并到一个列表，存入 posting_list_tmp0，再将 posting_list_tmp0 中的每个元素都转换成整型，存入新的列表 posting_list_tmp1，调用 sort 方法进行升序排序，再遍历整个 posting_list_tmp1，若找到重复元素，则去除列表中上一次读取到的该元素。完成去重之后，再将其中的每个元素转换成字符串型，存入列表 posting_list_tmp；

若逻辑运算方式为'AND NOT'，则取 posting_list 中、不在 posting_list2 中的元素，存入列表 posting_list_tmp；

若逻辑运算方式为'OR NOT'，则通过几个暂存列表，取 posting_list 中的、或者不在 posting_list2 中的元素，存入列表 posting_list_tmp；

最后再将 posting_list_tmp 赋值给本次循环得到的结果文档 ID 列表 posting_list，再进行下一次循环。

```
38 posting_list2 = query_index(str_list[i])
39 if boolean_way == 'AND':
40     posting_list_tmp = [docno for docno in posting_list if docno in posting_list2]
41 elif boolean_way == 'OR':
42     posting_list_tmp0 = posting_list + posting_list2
43     posting_list_tmp1 = [int(elem) for elem in posting_list_tmp0]
44     posting_list_tmp1.sort()
45     i = 1
46     while (i < len(posting_list_tmp1)):
47         if posting_list_tmp1[i] == posting_list_tmp1[i - 1]:
48             posting_list_tmp1.remove(posting_list_tmp1[i - 1])
49         else:
50             i += 1
51     posting_list_tmp = [str(elem) for elem in posting_list_tmp1]
52 elif boolean_way == 'AND NOT':
53     posting_list_tmp = [docno for docno in posting_list if docno not in posting_list2]
54 else:
55     docno_list = [value for value in docno_dict.values()]
56     list_tmp = [docno for docno in posting_list2 if docno not in posting_list]
57     posting_list_tmp = [docno for docno in docno_list if docno not in list_tmp]
58 posting_list = posting_list_tmp
```

循环结束后，得到最终的文档 ID 列表，将 posting_list 中的每个元素都转换成整型，存入新的列表 posting_list_int，调用 sort 方法进行升序排序，这个排序是为了方便后面的文档名称的获取，减少后期遍历次数。

```
60 posting_list_int = [int(elem) for elem in posting_list]
61 posting_list_int.sort()
```

创建并打开结果文件，按照查询次数进行命名，第一行写入当前查询次数，第二行写入当前布尔检索的布尔表达式，第三行写入 'boolean query result:\n'，从第四行开始是得到的结果文档名称。并在控制台输出相应信息。







```
62 file3 = open('./boolean_query_topics/boolean_query_res' + str(time) + '.txt', 'w')
63 file3.write('query number: ' + str(time) + '\n')
64 file3.write('boolean query: ' + boolean_str + '\n')
65 file3.write('boolean query result:\n')
66 print('boolean query result:')
```

因为 map 中的文档是按照文档 ID 存储的，posting_list_int 中的文档 ID 已经升序排序好，所以这里只需要遍历一次字典，按照文档 ID 查询得到文档名称，去掉 '.xml' 的后缀后写入结果文件中，number 用于计数，若 posting_list_int 已经查询完毕，则跳出循环。

```
67 number = 0
68 for key,value in docno_dict.items():
69     if value == posting_list_int[number]:
70         print(key[:-4])
71         file3.write(key[:-4] + '\n')
72         number += 1
73         if number == len(posting_list_int):
74             break
```

Result :

布尔检索的结果存放在 boolean_query_topics 文件夹中，一共进行了 5 次检索，每次检索使用了不同的 AND、OR、NOT 的搭配方式。

 boolean_query_topics	2018/5/1 11:43	文件夹	
 boolean_query_res1	2018/4/28 14:12	文本文档	90 KB
 boolean_query_res2	2018/4/28 14:12	文本文档	1,953 KB
 boolean_query_res3	2018/4/28 14:13	文本文档	2,266 KB
 boolean_query_res4	2018/4/28 14:13	文本文档	73 KB
 boolean_query_res5	2018/4/28 14:22	文本文档	1,888 KB

五次检索的布尔表达式分别为：

```
query number: 1
boolean query: a AND b AND NOT c
```

```
query number: 2
boolean query: number OR problem AND NOT must
```

```
query number: 3
boolean query: project OR small OR white OR black
```

```
query number: 4
boolean query: may AND could AND document AND public
```

```
query number: 5
boolean query: plan OR and AND NOT around OR or
```

结果文件的格式例子如下：

```
1 query number: 1
2 boolean query: a AND b AND NOT c
3 boolean query result:
4 AP880213-0014
5 AP880215-0026
6 AP880216-0082
7 AP880217-0254
8 AP880217-0281
9 AP880218-0147
10 AP880221-0028
11 AP880221-0068
12 AP880222-0026
13 AP880222-0214
14 AP880222-0237
15 AP880223-0021
16 AP880223-0094
17 AP880223-0242
18 AP880223-0257
```

ranking_query.py

主函数中，创建需要用到的文件夹。

```
21 if (not (os.path.exists('./ranking_query_topics'))):
22     os.mkdir('./ranking_query_topics')
```

接下来建立一个比 lab2 的 map 有更多信息的字典。

创建 document_path 列表，添加所有 lab1 中 disk12 经过预处理后得到的结果文件所在的文件夹路径。

```
24 document_path = []
25 document_path.append('../lab1/code/Answer/document/AP')
26 document_path.append('../lab1/code/Answer/document/DOE')
27 document_path.append('../lab1/code/Answer/document/FR')
28 document_path.append('../lab1/code/Answer/document/WSJ')
29 document_path.append('../lab1/code/Answer/document/ZIFF')
```

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。

```
31 total_file_num = 0
32 total_file_len = 0
```

遍历 disk12 预处理后的所有结果文件，计算每个文件的 TEXT 便签的文本中 token 的数量，去除文本头尾的空格，total_file_len 做统计，字典结构为

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

total_file_num 统计文档数量。

```
33 document_map = {}
34 for count in range(len(document_path)):
35     for a in os.walk(document_path[count]):
36         document_files = a[2]
37         for f in document_files:
38             tree = ET.parse(document_path[count] + '/' + f)
39             root = tree.getroot()
40             token_list = re.split(' ', root.find('TEXT').text.strip(' '))
41             total_file_len += len(token_list)
42             document_map[total_file_num] = {'DOC_name': f[:-4], 'DOC_length': len(token_list)}
43             total_file_num += 1
```

将得到的新的字典存储到 map2.json 文件中，输出成功信息和文档数量、文档总长度。

```
44 file0 = open('./map2.json', 'w')
45 json.dump(document_map, file0)
46 file0.close()
47 print('map2 succeed')
48 print('total_file_num = ' + str(total_file_num)) # 741856
49 print('total_file_len = ' + str(total_file_len)) # 189470967
```

读入 map2 字典。

```
54 file2 = open('./map2.json', 'r')
55 docno_dict = json.load(file2)
56 file2.close()
```

avg_file_len 是平均文本长度，并输出到控制台。

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

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 和 BM25 两种不同排序方式的结果文件。

```
61 k = 1.5
62 b = 0.75
63 file3 = open(u'./ranking_query_topics/10152130122_钱庭涵_TF-IDF.res', 'w')
64 file4 = open(u'./ranking_query_topics/10152130122_钱庭涵_BM25.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。

```
66 for a in os.walk('../lab1/code/Answer/query'):
67     document_files = a[2]
68     for f in document_files:
69         tree = ET.parse('../lab1/code/Answer/query/' + f)
70         root = tree.getroot()
71         query_title = re.split(' ', root.find('title').text.strip(' ').replace('`', '').replace('"', ''))[6:]
72         query_desc = re.split(' ', root.find('desc').text.strip(' ').replace('`', '').replace('"', ''))[9:]
73         query_narr = re.split(' ', root.find('narr').text.strip(' ').replace('`', '').replace('"', ''))[5:]
74         query_list = query_title + query_desc + query_narr
75         while '' in query_list:
76             query_list.remove('')
```

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。

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

```
78 score_tf_idf = {}
79 score_bm25 = {}
80 for i in range(len(query_list)):
81     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:

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

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, 那么 continue 直接进入下一次循环。

否则, 对于包含该 token 的所有文档, 使用 TF-IDF 和 BM25 两种方法, 分别计算 score 得分, 得分的初始化为 0, 之后一个一个叠加。

```
82 if token_dict == -1:
83     continue
84 idf_t = math.log(total_file_num / len(token_dict['pos']) * 1.0)
85 idf_qi = math.log((total_file_num - len(token_dict['pos']) + 0.5) /
86                 (len(token_dict['pos']) + 0.5) * 1.0)
87
88 for docno in token_dict['pos'].keys():
89     if docno not in score_tf_idf.keys():
90         score_tf_idf[docno] = 0
91         score_bm25[docno] = 0
92     score_tf_idf[docno] += token_dict['pos'][docno]['rate'] * idf_t
93     score_bm25[docno] += idf_qi * token_dict['pos'][docno]['rate'] * (k+1) / \
94                         (token_dict['pos'][docno]['rate'] + k *
95                         (1-b + b * docno_dict[docno]['DOC_length'] / avg_file_len * 1.0)) * 1.0
```

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, 对两种方式的字典进行排序 (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)。

```
97 sort_tmp1 = sorted(score_tf_idf.items(), key=lambda x: x[1], reverse=True)
98 sort_score_tf_idf = {}
99 for elem in sort_tmp1:
100     sort_score_tf_idf[elem[0]] = elem[1]
101 sort_tmp2 = sorted(score_bm25.items(), key=lambda x: x[1], reverse=True)
102 sort_score_bm25 = {}
103 for elem in sort_tmp2:
104     sort_score_bm25[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 FR88115-0020 1 0.48578544927679257 10152130122_TF-IDF

```
106 rank = 1
107 for docno in sort_score_tf_idf.keys():
108     print(f[6:-4] + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) + ' ' +
109           str(sort_score_tf_idf[docno]) + ' 10152130122_TF-IDF')
110     file3.write(f[6:-4] + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) +
111                ' ' + str(sort_score_tf_idf[docno]) + ' 10152130122_TF-IDF\n')
112     rank += 1
113     if rank == 1001:
114         break
115
116 rank = 1
117 for docno in sort_score_bm25.keys():
118     print(f[6:-4] + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) + ' ' +
119           str(sort_score_bm25[docno]) + ' 10152130122_BM25')
120     file4.write(f[6:-4] + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) +
121                ' ' + str(sort_score_bm25[docno]) + ' 10152130122_BM25\n')
122     rank += 1
123     if rank == 1001:
124         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。




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

最后关闭文件指针，完成所有检索。

```
128 file3.close()
129 file4.close()
```


Result :

使用 TF-IDF 和 BM25 的检索结果存放在 ranking_query_topics 文件夹中。




 ranking_query_topics	2018/5/23 13:45	文件夹	
 10152130122_钱庭涵_BM25.res	2018/5/23 14:11	Compiled Resou...	2,921 KB
 10152130122_钱庭涵_TF-IDF.res	2018/5/23 14:11	Compiled Resou...	3,049 KB

10152130122_钱庭涵_TF-IDF.res x									
1	151	0	AP890411-0076	1	2.224548572746962	10152130122_TF-IDF			
2	151	0	FR89517-0017	2	2.17239682879441	10152130122_TF-IDF			
3	151	0	AP881206-0124	3	2.1002518459726214	10152130122_TF-IDF			
4	151	0	AP890125-0112	4	2.0139359072509704	10152130122_TF-IDF			
5	151	0	FR891205-0022	5	1.9690160774403553	10152130122_TF-IDF			
6	151	0	AP890408-0056	6	1.909933091897621	10152130122_TF-IDF			
7	151	0	AP881018-0003	7	1.880591446580549	10152130122_TF-IDF			
8	151	0	AP880407-0033	8	1.80287311969914	10152130122_TF-IDF			
9	151	0	AP881002-0014	9	1.755263876910499	10152130122_TF-IDF			
10	151	0	AP880825-0054	10	1.7501013820870315	10152130122_TF-IDF			
11	151	0	AP891208-0177	11	1.7336739424631242	10152130122_TF-IDF			
12	151	0	AP880418-0017	12	1.7307140622964947	10152130122_TF-IDF			
13	151	0	AP890827-0067	13	1.6554669284778394	10152130122_TF-IDF			
14	151	0	AP880521-0003	14	1.6259532882487882	10152130122_TF-IDF			
15	151	0	AP880620-0059	15	1.6222428911231959	10152130122_TF-IDF			
16	151	0	AP890523-0126	16	1.6066329793267984	10152130122_TF-IDF			
17	151	0	DOE1-02-1138	17	1.59179437712087	10152130122_TF-IDF			
18	151	0	AP890928-0092	18	1.5393642221734734	10152130122_TF-IDF			
19	151	0	AP891101-0115	19	1.536160091457371	10152130122_TF-IDF			
20	151	0	AP881029-0064	20	1.5251317242494857	10152130122_TF-IDF			
21	151	0	AP890125-0159	21	1.5174575821069523	10152130122_TF-IDF			
22	151	0	AP890729-0033	22	1.491744663802805	10152130122_TF-IDF			
23	151	0	AP891203-0056	23	1.4862603482610945	10152130122_TF-IDF			
24	151	0	FR88105-0022	24	1.4767620580802665	10152130122_TF-IDF			
25	151	0	AP890107-0063	25	1.4738526588063083	10152130122_TF-IDF			
26	151	0	AP890303-0051	26	1.4558601991678772	10152130122_TF-IDF			
27	151	0	FR89321-0029	27	1.4425286394240984	10152130122_TF-IDF			
28	151	0	AP890116-0047	28	1.4364258503622527	10152130122_TF-IDF			
29	151	0	DOE2-07-0247	29	1.4164148008669044	10152130122_TF-IDF			
30	151	0	AP881206-0196	30	1.4147908215243035	10152130122_TF-IDF			

1	151	0	FR89517-0017	1	9.41167344539938	10152130122_BM25
2	151	0	AP881018-0003	2	8.57892251321101	10152130122_BM25
3	151	0	FR891205-0022	3	8.275627023902032	10152130122_BM25
4	151	0	DOE1-02-1138	4	7.607463554604731	10152130122_BM25
5	151	0	FR88105-0022	5	6.894210984767753	10152130122_BM25
6	151	0	FR89321-0029	6	6.699929032733861	10152130122_BM25
7	151	0	DOE2-07-0247	7	6.289534606863489	10152130122_BM25
8	151	0	FR891128-0028	8	6.238733547786856	10152130122_BM25
9	151	0	FR89317-0028	9	6.238733547786856	10152130122_BM25
10	151	0	DOE1-59-0673	10	5.892266017817334	10152130122_BM25
11	151	0	AP881029-0064	11	5.75025983796605	10152130122_BM25
12	151	0	FR881017-0021	12	5.6874060635207355	10152130122_BM25
13	151	0	AP880407-0033	13	5.666327012450536	10152130122_BM25
14	151	0	DOE2-12-0776	14	5.5640670914427375	10152130122_BM25
15	151	0	DOE2-25-1174	15	5.538529467886252	10152130122_BM25
16	151	0	AP881206-0124	16	5.4691627299973575	10152130122_BM25
17	151	0	AP880418-0017	17	5.394922475274837	10152130122_BM25
18	151	0	DOE2-12-0765	18	5.250118242648719	10152130122_BM25
19	151	0	AP891208-0177	19	5.24163992395013	10152130122_BM25
20	151	0	AP881002-0014	20	5.23535745010712	10152130122_BM25
21	151	0	AP880521-0003	21	5.100754074295208	10152130122_BM25
22	151	0	DOE2-01-0011	22	5.016437848788935	10152130122_BM25
23	151	0	DOE2-01-0012	23	5.016437848788935	10152130122_BM25
24	151	0	DOE2-08-1118	24	4.977107192265961	10152130122_BM25
25	151	0	DOE2-16-0524	25	4.977107192265961	10152130122_BM25
26	151	0	AP880525-0010	26	4.962628857467636	10152130122_BM25
27	151	0	AP890928-0092	27	4.918324355416499	10152130122_BM25
28	151	0	DOE1-15-0656	28	4.865613602819769	10152130122_BM25
29	151	0	FR89817-0045	29	4.812351148855948	10152130122_BM25
30	151	0	DOE2-54-0617	30	4.781636280746484	10152130122_BM25

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	2018/5/23 13:45	文件夹	
 trec_BM25.res	2018/5/23 14:12	Compiled Resou...	1 KB
 trec_TF-IDF.res	2018/5/23 14:12	Compiled Resou...	1 KB

	trec_TF-IDF.res	trec_BM25.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_TF-IDF.res		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

对上次报告进行修改的地方：

(上文中蓝色字体部分、修改过的代码部分的截图、ranking_query 得到的结果截图、Trec 评测的截图)

1. ranking_query.py 代码的更新
 - a) 统计 document 文件的长度并建立 map2 时，去除了文本头尾的空格，map2、total_file_len、avg_file_len 得到更新。
 - b) 读入的 query 文件时，取用了 title、desc、narr 标签的文本 (本来只用了 title)，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。
 - c) 输出最终结果的 top-1000 个。
2. 结果和评测结果截图的更新。

一些问题与讨论：

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，并对比了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，对 query 和 document 做了一些调整之后 (调整的地方见上文)，两种方法的效果都比上次要好，但 BM25 仍旧比 TF-IDF 要差，BM25 的 k 参数是经过多次测试比对结果之后才选取为 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。