

Lab5 语言模型 实验报告

10152130122 钱庭涵

SLM_query.py

主函数中，创建需要用到的文件夹。

```
42 if (not (os.path.exists('./SLM_query_topics'))):
43     os.mkdir('./SLM_query_topics')
```

读入 map2 字典，并输出读取成功信息。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}, ...}`

```
45 file2 = open('../lab3/map2.json', 'r')
46 docno_dict = json.load(file2)
47 file2.close()
48 print('map2 load succeed')
```

alpha1 是线性插值平滑处理方式中使用的参数，alpha2 则是 Dirichlet 平滑处理方式中使用的参数，并创建采用这两种不同的平滑处理方式进行统计语言模型检索的结果文件。

```
50 alpha1 = 0.95
51 alpha2 = 0.9
52 file3 = open(u'./SLM_query_topics/10152130122_钱庭涵_SLM_Interpolate.res', 'w')
53 file4 = open(u'./SLM_query_topics/10152130122_钱庭涵_SLM_Dirichlet.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。

```
55 for a in os.walk('../lab1/code/Answer/query'):
56     document_files = a[2]
57     for f in document_files:
58         tree = ET.parse('../lab1/code/Answer/query/' + f)
59         root = tree.getroot()
60         query_title = re.split(' ', root.find('title').text.strip(' ').replace('\n', '').replace('"', "'')[6:])
61         query_desc = re.split(' ', root.find('desc').text.strip(' ').replace('\n', '').replace('"', "'')[9:])
62         query_narr = re.split(' ', root.find('narr').text.strip(' ').replace('\n', '').replace('"', "'')[5:])
63         query_list = query_title + query_desc + query_narr
64         while '' in query_list:
65             query_list.remove('')
```

score_SLM_interpolate 和 score_SLM_dirichlet 两个字典用于存放当前 topic 下两种平滑处理方式各自的不同文档的得分，effective_token_dict 用于存放每个有效的 token 对应的总词频和出现文档列表，其字典结构如下：

```
{token: {'total_rate': 总词频, 'docno_list': [docno1, docno2, ...]}, ...}
```

```
67 score_SLM_interpolate = {}
68 score_SLM_dirichlet = {}
69 effective_token_dict = {}
```

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。

```
71 for i in range(len(query_list)):
72     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。

```
7 def query_index(token):
8     if token[0] >= 'a' and token[0] <= 'z':
9         path = '../lab2/index_file/index_' + token[0] + '.json'
10    else:
11        path = '../lab2/index_file/index__other.json'
12    file1 = open(path, 'r')
13    index_dict = json.load(file1)
14    file1.close()
15    if token in index_dict.keys():
16        return index_dict[token]
17    else:
18        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，该 token 为有效 token，将其作为 key 存入 effective_token_dict 字典中，并存储该 token 的总词频和出现文档列表。

```
73 if token_dict == -1:
74     continue
75
76 effective_token_dict[query_list[i]] = {'total_rate': token_dict['rate'],
77                                         'docno_list': token_dict['pos'].keys()}
```

对于包含该 token 的所有文档，分别计算线性插值和 Dirichlet 两种平滑方式的 score 得分，得分的初始化为 1，之后一个一个相乘。

```
78 for docno in token_dict['pos'].keys():
79     if docno not in score_SLM_interpolate.keys():
80         score_SLM_interpolate[docno] = 1
81         score_SLM_dirichlet[docno] = 1
82     score_SLM_interpolate[docno] *= (
83         alpha1 * token_dict['pos'][docno]['rate'] + (1 - alpha1) * token_dict['rate'])
84     score_SLM_dirichlet[docno] *= (
85         (int(token_dict['pos'][docno]['rate'] * docno_dict[docno]['DOC_length'])
86          + alpha2 * token_dict['rate']) / (docno_dict[docno]['DOC_length'] + alpha2) * 1.0)
```

处理完当前 topic 所有 token 之后，输出语言模型成功信息。

```
87 print('LM succeedd')
```

对于当前 topic 的所有 token，对于候选文档列表中不包含当前 token 的文档，进行平滑处理计算，分别计算线性插值和 Dirichlet 两种平滑方式的 score 得分，并乘到原始得分中去。

处理完当前 topic 所有 token 之后，输出平滑处理成功信息。

```
89 for token in effective_token_dict.keys():
90     for docno in score_SLM_interpolate.keys():
91         if docno not in effective_token_dict[token]['docno_list']:
92             score_SLM_interpolate[docno] *= ((1 - alpha1) * effective_token_dict[token]['total_rate'])
93             score_SLM_dirichlet[docno] *= (
94                 (alpha2 * effective_token_dict[token]['total_rate']) / (
95                     docno_dict[docno]['DOC_length'] + alpha2) * 1.0)
96 print('smoothing succeed')
```

调用 get_result 函数，得到当前 topic 下两种平滑处理方式各自的结果。

```
98 get_result(f[6:-4], score_SLM_interpolate, 'Interpolate')
99 get_result(f[6:-4], score_SLM_dirichlet, 'Dirichlet')
```

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）。

```
20 def get_result(topic, score, method):
21     sort_tmp = sorted(score.items(), key=lambda x: x[1], reverse=True)
22     sort_score = {}
23     for elem in sort_tmp:
24         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 WSJ880930-0141 1 2.524727748037604e-114 10152130122_SLM_Interpolate

```
26 rank = 1
27 for docno in sort_score.keys():
28     print(topic + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) + ' ' +
29           str(sort_score[docno]) + ' 10152130122_SLM_' + method)
30     if method == 'Interpolate':
31         file3.write(topic + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) + ' ' +
32                     str(sort_score[docno]) + ' 10152130122_SLM_' + method + '\n')
33     else:
34         file4.write(topic + ' 0 ' + docno_dict[docno]['DOC_name'] + ' ' + str(rank) + ' ' +
35                     str(sort_score[docno]) + ' 10152130122_SLM_' + method + '\n')
36 rank += 1
37 if rank == 1001:
38     break
```

完成采用平滑处理方式对当前 topic 的检索, 输出成功信息。

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

跳转回主函数, 完成采用两种平滑处理方式对当前 topic 的检索, 再进入下一个循环, 处理下一个 topic。




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

最后关闭文件指针, 完成所有平滑处理后语言模型的检索。

```
103 file3.close()
104 file4.close()
```

Result :

使用两种平滑处理方式的语言模型检索后的结果存放在 SLM_query_topics 文件夹中。

 SLM_query_topics	2018/6/9 16:23	文件夹	
 10152130122_钱庭涵_SLM_Dirichlet.res	2018/6/9 15:07	Compiled Resou...	3,609 KB
 10152130122_钱庭涵_SLM_Interpolate.res	2018/6/9 15:07	Compiled Resou...	3,718 KB

10152130122_钱庭涵_SLM_Interpolate.res					
1	151	0	WSJ880930-0141	1	2.524727748037604e-114 10152130122_SLM_Interpolate
2	151	0	FR88602-0055	2	1.6666252237711734e-114 10152130122_SLM_Interpolate
3	151	0	DOE1-01-0526	3	3.7395171802419172e-115 10152130122_SLM_Interpolate
4	151	0	FR891011-0072	4	1.4330161750501042e-115 10152130122_SLM_Interpolate
5	151	0	FR89516-0090	5	1.2475738458554952e-115 10152130122_SLM_Interpolate
6	151	0	WSJ870116-0113	6	8.251390355606292e-116 10152130122_SLM_Interpolate
7	151	0	FR88926-0058	7	7.590338303801714e-116 10152130122_SLM_Interpolate
8	151	0	ZF208-154-986	8	5.548607996002938e-116 10152130122_SLM_Interpolate
9	151	0	AP880622-0019	9	3.943787261525208e-116 10152130122_SLM_Interpolate
10	151	0	DOE1-90-1299	10	3.813748475741295e-116 10152130122_SLM_Interpolate
11	151	0	FR89317-0033	11	1.2546966140166016e-116 10152130122_SLM_Interpolate
12	151	0	WSJ880628-0109	12	7.81988209566515e-117 10152130122_SLM_Interpolate
13	151	0	DOE2-29-0674	13	7.508868671248278e-117 10152130122_SLM_Interpolate
14	151	0	ZF108-335-326	14	6.717437190523132e-117 10152130122_SLM_Interpolate
15	151	0	FR88122-0124	15	5.671601513175356e-117 10152130122_SLM_Interpolate

10152130122_钱庭涵_SLM_Dirichlet.res					
1	151	0	AP880620-0259	1	2.6503885720493903e-120 10152130122_SLM_Dirichlet
2	151	0	AP890219-0004	2	2.6503885720493903e-120 10152130122_SLM_Dirichlet
3	151	0	WSJ920323-0065	3	1.4040501902968326e-120 10152130122_SLM_Dirichlet
4	151	0	AP880614-0161	4	1.1868709943411159e-120 10152130122_SLM_Dirichlet
5	151	0	AP880726-0248	5	1.1868709943411159e-120 10152130122_SLM_Dirichlet
6	151	0	AP880802-0250	6	1.1868709943411159e-120 10152130122_SLM_Dirichlet
7	151	0	AP880811-0242	7	1.1868709943411159e-120 10152130122_SLM_Dirichlet
8	151	0	AP880908-0256	8	1.1868709943411159e-120 10152130122_SLM_Dirichlet
9	151	0	AP880913-0234	9	1.1868709943411159e-120 10152130122_SLM_Dirichlet
10	151	0	AP880929-0244	10	1.1868709943411159e-120 10152130122_SLM_Dirichlet
11	151	0	AP881013-0284	11	1.1868709943411159e-120 10152130122_SLM_Dirichlet
12	151	0	AP881025-0230	12	1.1868709943411159e-120 10152130122_SLM_Dirichlet
13	151	0	AP881101-0198	13	1.1868709943411159e-120 10152130122_SLM_Dirichlet
14	151	0	AP881110-0247	14	1.1868709943411159e-120 10152130122_SLM_Dirichlet
15	151	0	AP881122-0263	15	1.1868709943411159e-120 10152130122_SLM_Dirichlet

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	2018/6/9 16:23	文件夹	
trec_SLM_Dirichlet.res	2018/6/9 15:07	Compiled Resou...	2 KB
trec_SLM_Interpolate.res	2018/6/9 15:07	Compiled Resou...	2 KB

trec_SLM_Interpolate.res		
1	runid	all 10152130122_SLM_Interpolate
2	num_q	all 50
3	num_ret	all 50000
4	num_rel	all 9805
5	num_rel_ret	all 503
6	map	all 0.0016
7	gm_map	all 0.0001
8	Rprec	all 0.0089
9	bpref	all 0.0286
10	recip_rank	all 0.0279
11	iprec_at_recall_0.00	all 0.0349
12	iprec_at_recall_0.10	all 0.0083
13	iprec_at_recall_0.20	all 0.0031
14	iprec_at_recall_0.30	all 0.0000
15	iprec_at_recall_0.40	all 0.0000
16	iprec_at_recall_0.50	all 0.0000
17	iprec_at_recall_0.60	all 0.0000
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 0.0080
23	P_10	all 0.0060
24	P_15	all 0.0040
25	P_20	all 0.0030
26	P_30	all 0.0027
27	P_100	all 0.0040
28	P_200	all 0.0086
29	P_500	all 0.0091
30	P_1000	all 0.0101

trec_SLM_Dirichlet.res		
1	runid	all 10152130122_SLM_Dirichlet
2	num_q	all 50
3	num_ret	all 50000
4	num_rel	all 9805
5	num_rel_ret	all 15
6	map	all 0.0000
7	gm_map	all 0.0000
8	Rprec	all 0.0006
9	bpref	all 0.0011
10	recip_rank	all 0.0016
11	iprec_at_recall_0.00	all 0.0018
12	iprec_at_recall_0.10	all 0.0000
13	iprec_at_recall_0.20	all 0.0000
14	iprec_at_recall_0.30	all 0.0000
15	iprec_at_recall_0.40	all 0.0000
16	iprec_at_recall_0.50	all 0.0000
17	iprec_at_recall_0.60	all 0.0000
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 0.0000
23	P_10	all 0.0000
24	P_15	all 0.0000
25	P_20	all 0.0000
26	P_30	all 0.0000
27	P_100	all 0.0004
28	P_200	all 0.0007
29	P_500	all 0.0005
30	P_1000	all 0.0003

讨论：

本实验在基于 SLM 的检索中，采用了线性插值和 Dirichlet 两种不同的平滑处理方式，并在 unigram 假设下进行检索，得到的结果准确率较低。

本想对于每个 topic，将两种平滑处理方式中需要用到的参数（线性插值需要 $\hat{P}_{MLE}(t|M_D)$ 和 $\hat{P}(t|M_C)$ ，Dirichlet 平滑需要 $tf_{t,d}$ 和 $\hat{P}_{MLE}(t|M_C)$ 和 L_D ）计算之后存储到 json 文件，即生成 100 个 json 文件，再通过调整 alpha 的取值，挨个读取所有 json 文件中的参数计算得分。

但无奈电脑内存不够，无法正常读入得到的一个 json 文件。

最后仅使用了部分 token 来进行 alpha 参数的调整，但可能由于参数选取的不好，加上 SLM 检索方式本身效果不好，导致 Dirichlet 平滑处理得到的结果的 map 值和 P_5 均为 0。