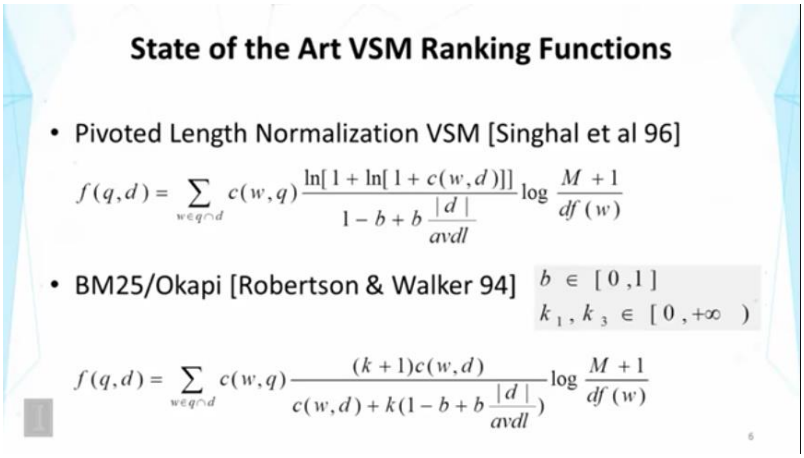
实验四 Pivoted Length Normalization VSM and BM25

本次实验是在实验三布尔检索的基础上进行改进，对文档集中每个出现检索词的文档计算相似度评分，并将 doc_id 按其相似度从大到小的顺序输出。两种相似度评分公式如下：



1. 建立 inverted index

读取 tweets 数据集、截取正文、分词等预处理等步骤与实验三一致，在建立 inverted index 时，由于在计算 query 与文档相似度时需要用到 TF 与 DF, 因此建立倒排索引时仍然使用实验三中将字典的 value 设为集合的方法不再合适，这里字典中每个 word 的 value 设为一个列表，它的元素为包含此 word 的每个 doc_id 及其 c(w,d) 组成的列表。

```
#建立 inverted index
dict1 = {}
label = 0
for word_list in textword:
    for word in word_list:
        if word not in dict1:
            dict1[word] = []
            dict1[word].append([label, word_list.count(word)])
        elif label != dict1[word][-1][0]:
            dict1[word].append([label, word_list.count(word)])
    label = label + 1
```

2. 检索、打分

采用与实验三类似的思路，以两种方法的缩写“PLNVSM”，“BM25”作为输入的标识符，采取相应的公式进行计算。对 query 进行预处理，得到包含每个检索 word 的列表，对字典中 word 对应的每个文档进行打分（参数 b = 0.5，k = 2），将列表

score_list 作为 query 的打分结果，其每个元素为 doc_id 及其分数。

```
#检索、打分
while True:
    print("begin")
    query = input()
    if query == '':
        break
    if 'PLNVSM:' in query:
        query_word = filter(query[7:])  # 注意 query 也需要进行预处理
    if 'BM25:' in query:
        query_word = filter(query[5:])
    print('query 预处理: ', query_word)
    score_list = []
    for word in query_word:
        for d in dict1[word]:
            if 'PLNVSM:' in query:
                sim = query_word.count(word) * log(1 + log(1+d[1], math.e), math.e) / \
                      (1 - b + b * len(textword[d[0]])/avdl) * log((M+1)/len(dict1[word]), 2)
            if 'BM25:' in query:
                sim = query_word.count(word) * (k+1)*d[1]/(d[1]+k*(1-b+b*len(textword[d[0]])/avdl)) \
                      * log((M+1)/len(dict1[word]), 2)
            judge = True
            for i in score_list:
                if i[0] == d[0]:
                    judge = False
                    i[1] = i[1] + sim
            if judge:
                score_list.append([d[0], sim])
```

3. 排序、输出

按 score_list 的每个元素的打分结果高低顺序输出对应的 doc_id，用 Ron Weasley Birthday 进行测试，两种方法的结果如下：

```
D:\Anaconda3\python.exe C:/Users/hp/PycharmProjects/IR/IR.4.py
avdl =  11.757299986905853
begin
PLNVSM:Ron Weasley Birthday
query预处理:  ['ron', 'weasley', 'birthday']
文档评分:  [20.460633892762573, 20.460633892762573, 20.460633892762573, 20.460633892762573, 20.460633892762573, 19.565426864253077, 19.239635646699636, 19
对应doc_id:  [16745, 16787, 16938, 17011, 17023, 16745, 16787, 16938, 17011, 17023, 16745, 16787, 16938, 17011, 17023, 16745, 16787, 16938, 17011, 17023,
enter a doc_id to browse:
-1
begin
BM25:Ron Weasley Birthday
query预处理:  ['ron', 'weasley', 'birthday']
文档评分:  [35.25533328744369, 34.62665859906343, 33.59534559678648, 33.37775577756306, 33.37775577756306, 33.37775577756306, 33.37775577756306, 33.377755
对应doc_id:  [16792, 17069, 17022, 16745, 16787, 16938, 17011, 17023, 16745, 16787, 16938, 17011, 17023, 16745, 16787, 16938, 17011, 17023, 16745, 16787,
enter a doc_id to browse:
```

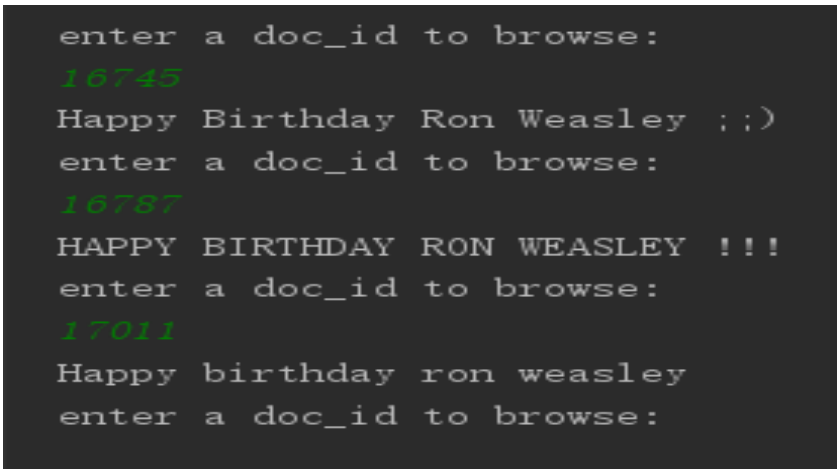
两种方法排序结果差别不大，互相验证，较为合理。

4. 浏览得到的文档

为了查看并验证检索得到的文档排序是否合理，可以对文档进行查询。

```
#文档查询
while True:
    print('enter a doc_id to browse:')
    doc_id = int(input())
    if doc_id == -1:
        break
    print(tweets_text[doc_id])
```

在搜索 Ron Weasley Birthday 完成后，查询排名靠前的 doc 的内容，发现内容简单一致：



这也解释了为何会有许多打分高且相等的 doc 的现象。

5. 检索评价

在完成检索任务后，需要对检索效果进行评价，评价的主要方法是对 queryId 为 171 到 225 的检索内容进行检索，将检索结果与 groundtruth 进行对比，计算评价指标 MAP，NDCG. 因此输入的应是 queryId (171~225)，输出结果不再是 doc_id，而是 tweetId，并且需要按指定格式输出。因此对之前的的检索过程进行以下改进：

1. 引入 tweetId

   之前读取 tweet 时用的是文本读取，而在后续评价时需要用到 tweetId，这里改为用 json 格式读取；
2. 检索并对结果进行排序

   下载 TREC 2014 test topics 文档并将其 queryId 与检索内容对应写入 querynumtoquery.txt,接着读入这个文档，并将其处理为字典 num_query.

   依次对 queryID（171~225），根据字典 num_query 检索，将返回的 tweetId 按其评分从高到低排序，由于评价时最多只取 100 个检索返回结果进行评价，这里最多只返回前 100 个 tweetId.
3. 写 result

   将上一步得到的结果按每行 queryId，tweetId 的格式依次写入 result.txt.
4. 评价

   将得到的 result.txt 及 groudtruth 文档 qrels.txt 输入 eval_hw4.py 得到评价结果如下:

	MAP	NDCG
PLNVSM	0.5270	0.6824
BM25	0.5216	0.6778