信息检索与数据挖掘

实验报告-实验二

班　　级　智能
学　　号　201900150221
姓　　名　张进华

时　　间　2021 年 10 月 13 日

计算机科学与技术学院信息检索与数据挖掘课程实验报告

实验题目：Ranked retrieval model		学　　号　　：201900150221
日期：10-13	班级：　19 智能	姓名：　张进华
Email：zjh15117117428@163.com		

实验目的：

实现基本的排名检索模型

实验内容：

– 在Experiment1的基础上实现最基本的Ranked retrieval model

• Input：a query (like Ron Weasley birthday)

• Output: Return the top K (e.g., K = 100) relevant tweets.

• Use SMART notation: lnc.ltn

• Document: logarithmic tf (l as first character), no idf and cosine normalization

• Query: logarithmic tf (l in leftmost column), idf (t in second column), no normalization

• 改进Inverted index

• 在Dictionary中存储每个term的DF

• 在posting list中存储term在每个doc中的TF with pairs (docID, tf) • 选做

• 支持所有的SMART Notations

实验原理分析：

1. 对 tf-idf 的详细理解

TF-IDF（term frequency－inverse document frequency）是一种用于信息检索与数据挖掘的常用加权技术。TF 是词频(Term Frequency)，IDF 是逆文本频率指数(Inverse Document Frequency)。TF-IDF 的主要思想是：如果某个词或短语在一篇文章中出现的频率 TF 高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

$$\text{tf} = \frac{\text{某个词在文章中出现的次数}}{\text{该文章的总词数}}$$

n i,j 表示的是第 i 个词在文档 j 中出现的总次数,分母表示在文档 j 中其它的所有

的词数。使用上面的公式我们就可以计算出词频。

$$\mathrm{idf_i} = \lg \frac{|D|}{|\{j : t_i \in d_j\}|}$$

$$\text{逆文档频率}(idf) = \log(\frac{\text{语料库中文档总数}}{\text{包含该词的文档总数}+1})$$

使用如上公式可以计算出逆文档频率 idf，所谓 tf-idf=tf*idf。对 tf 和 idf 有了一定的理解后，开始进行实验，选择 python 语言。

2. 排序检索模型

在排序检索模型中，系统根据文档与 query 的相关性排序返回文档集合中的文档，而不是简单地返回所有满足 query 描述的文档集合。
自由文本查询：用户 query 是自然语言的一个或多个词语而不是由查询语言构造的表达式。总体上，排序检索模型中有布尔查询和自由文本查询两种方式，但是实际中排序检索模型总是与自由文本查询联系在一起，反之亦然。

3. 评分标准

引用课堂里的 PPT 页面，对于 query,计算 query 中 term 的 tf,df 及 idf 后计算出权重 query_wtq,而对于 document,则需计算出每个 term 的 tf,并进行归一化处理，然后得到 term 对 document 的权重 doc_wtd，将每个 term 的 query_wtq 与 doc_wtd 相乘累加，就可得到 query 对该 document 的评分，最后输出前十个 document
各种评分标准计算公式如下：

tf-idf weighting has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1+\log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5+\frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0,\log\frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases}1 & \text{if } \mathrm{tf}_{t,d}>0\\ 0 & \text{otherwise}\end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha<1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

SMART notation for tf-idf variants.
Here *CharLength* is the number of characters in the document.

示例如下：

tf-idf example: lnc.ltc

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	2.04

实验软件和硬件环境：

Visual studio Code python 3.9.7

Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz

实验步骤：

步骤一 数据集处理

在 Dictionary 中存储每个 term 出现的 document_id,其长度就代表 term 出现的文档数目 df，类型为字典类型，处理过程类似于实验 1 的倒排索引表，实现效果如下：

```
F:\IR\exp1\venv\Scripts\python.exe F:/IR/exp2/exp2.py
arus ['28965792812892160']
http ['28965792812892160', '28967095878287360', '28967914417688576', '28968479176531969'
arizona-style ['28965792812892160']
bill ['28965792812892160', '29009276655636480']
people ['28965792812892160']
the ['28965792812892160', '28968581949558787', '28969422056071169', '28971749961891840',
may ['28965792812892160']
mariah ['28965792812892160']
```

在 posting list 中存储 term 在每个 doc 中的 TF with pairs (docID, tf)，类型为字典，实现效果如下：

```
the [['28965792812892160', 2], ['28968581949558787', 1], ['28969422056071169', 1],
to [['28965792812892160', 1], ['28967672074993664', 1], ['28968581949558787', 1], [
rick [['28965792812892160', 1]]
http [['28965792812892160', 1], ['28967095878287360', 1], ['28967914417688576', 1],
arus [['28965792812892160', 1]]
house [['28965792812892160', 2]]
say [['28965792812892160', 1], ['28976409057697792', 1], ['28979992025104385', 1],
pas [['28965792812892160', 1], ['28974862038994945', 1]]
```

步骤二 代码重要部分解释

首先是输入 query,将其每个单词进行词性还原、单复数转换后返回 term 列表，判断 query 中的 term 是否在文档中出现过，若该 term 未在任何一个文档中出现，说明该 term 对于查询无效，将其剔除。若到最后，query 中所有的 term 都未在任何一个文档中出现，即 unique 为空，返回没有匹配文档的提示，结束查询。

```
unique_query = set(query)
temp = []
for term in unique_query:
    if len(Dictionary[term]) == 0:  # 这个term未出现在出现在文档中
        temp.append(term)
for term in temp:
    unique_query.remove(term)

# 所有term都未出现
if len(unique_query) == 0:
    print("no relevant document!")
    return
```

然后要对每个 document 打分，但是此时只需对与 query 中 term 相关的 document 打分，所以根据 term，将 Dictionary[term]的 document_id 加入相关文档列表 relevant_tweetids,并进行去重，若最后 relevant_tweetds 为空，则返回提示消息后结束查询。

```
# 相关文档
relevant_tweetids = []
for term in unique_query:
    for i in Dictionary[term]:
        relevant_tweetids.append(i)
relevant_tweetids = set(relevant_tweetids)

print("A total of " + str(len(relevant_tweetids)) + " related tweetid!")
if not relevant_tweetids:
    print("No tweets matched any query terms for")
```

先计算 query 中每个 term 的 tf,df,idf 以及 wtq,计算公式根据上图所示，实现代码如下：

```
query_tf = defaultdict(dict)
query_df = defaultdict(dict)
query_idf = defaultdict(dict)
query_wtq = defaultdict(dict)

for term in unique_query:
    # query中每个term的tf
    tf_raw = query.count(term)
    query_tf[term] = 1 + math.log10(tf_raw)
    # query中term的df
    query_df[term] = len(Dictionary[term])
    # query中term的idf
    query_idf[term] = math.log10(numDocument / query_df[term])
    # query的Wtq
    query_wtq[term] = query_tf[term] * query_idf[term]
```

比如查询语句为“we are happy”，在测试集上可以看到 happy 未在任何文档中出现，而 we 和 are 的 wtq 可计算出

```
please input query: we are happy
A total of 4 related tweetid!
are 1.3152704347785915
we 1.792391689498254
```

然后遍历所有的相关文档，计算 query 中每个 term 的 tf，得到 doc_tf,然后进行归一化，得到 doc_wtd,公式如上图所示，实现代码如下：

```
# 遍历所有的相关文档
for doc in relevant_tweetids:
    doc_tf = defaultdict(dict)
    doc_wtd = defaultdict(dict)
    normalizer = 0
    for term in unique_query:
        for item in postingsList[term]:
            if item[0] == doc:
                tf_raw = item[1]
                doc_tf[term] = 1 + math.log10(tf_raw)
                normalizer += (doc_tf[term] ** 2)
            else:
                doc_tf[term] = 0
    # 归一化
    for term in unique_query:
        doc_wtd[term] = doc_tf[term] / math.sqrt(normalizer)
```

比如还是如上面一样，查询语句为“we are happy”，在测试集上可以看到 happy 未在任何文档中出现，而 we 和 are 的 wtd 可计算出

```
please input query: we are happy
A total of 4 related tweetid!
we 0.0
are 0.0
we 1.0
are 0.0
we 0.0
are 1.0
we 0.0
are 0.0
```

接下来对每个 document 进行评分，其分数为每个 term 的 query_wtq[term]与 doc_wtd[term]乘积累加，实现代码如下：

```
# 文档评分
for term in unique_query:
    if doc not in score_document:
        score_document[doc] = query_wtq[term] * doc_wtd[term]
    else:
        score_document[doc] = score_document[doc] + query_wtq[term] * doc_wtd[term]
```

同样查询语句为“we are happy”，在测试集上可以看到对每个分档的评分如下：

```
please input query:  we are happy
A total of 4 related tweetid!
28988679422738432 1.3152704347785915
28976831738683393 0.0
28995339910389760 1.792391689498254
28979189172412416 0.0
```

最后对文档评分字典 score_document 按照评分进行排序，最后输出前 10 个相关的文档，实现代码如下：

```
# 排序输出
score = sorted(score_document.items(), key=lambda x: x[1], reverse=True)
print("Output the top 10 document_id:")
for i in range(10):
    print("tweetid:", score[i][0], "score:", score[i][1])
```

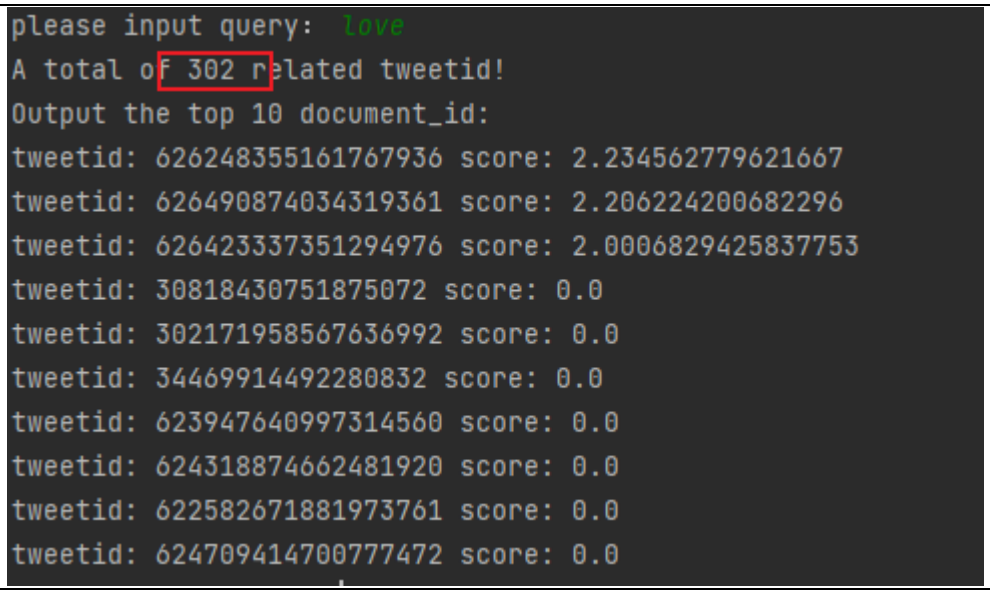
实验结果：

在训练集上进行查询

1-查询语句为“we are happy”，可以看到相关的文档有 1789 个，但是评分上真正相关的只有两个。

```
How many times do you want to check?3
please input query:  we are happy
A total of 1789 related tweetid!
Output the top 10 document_id:
tweetid: 626248355161767936 score: 2.234562779621667
tweetid: 626490874034319361 score: 2.206224200682296
tweetid: 30818430751875072 score: 0.0
tweetid: 302171958567636992 score: 0.0
tweetid: 34469914492280832 score: 0.0
tweetid: 623947640997314560 score: 0.0
tweetid: 624318874662481920 score: 0.0
tweetid: 622582671881973761 score: 0.0
tweetid: 624709414700777472 score: 0.0
tweetid: 31740806834425856 score: 0.0
```

2-查询语句为“love”，可以看到相关文档数目为 302，但评分最高的只有 3 个



结论分析与体会：

通过本次实验，复习了课上所学的知识，清楚的理解了 tf、df、idf、tf-idf 以及如何利用 wtd*wtq 来计算文档得分，比较 query 与文档的相似程度，从而确定出最相关的文档。

在计算 query 中 idf 时发现有除数为 0 的情况，及 df 为 0，这种情况就是 term 未在任何文档出出现，要特殊处理。

在计算 query 中 wtq 时并未进行归一化处理，但是在计算 document 中 tf 时要进行归一化处理。