信息检索与数据挖掘

实验报告-实验二

山东大学 SHANDONG UNIVERSITY

班	级	智能
学	号	201900161140
姓	名	张文浩
时	间	2021 年 9 月 29 日

实验内容

Ranked retrieval model

任务：

- 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.ltc
  - 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

Deadline: 2021.11.9

实验环境：

win10+spyder（anaconda）

实验步骤

我实现了实验选做部分。即支持所有的 smart Notations。

第一步：

先读取一遍数据集 tweets.txt 来获得 df0 数组，这个 df0 数组的作用是为了后面在计算 W 的时候获得每个 term 对应的 df 用的。

```python
#为提前获得df而先读一遍文件的函数
def get_df():
    global doc_tot
    f = open(r"E:\datamining\exp1\tweets.txt")
    lines = f.readlines()
    uselessword = [",","'","'s","“","”","-","’’","—"]
    #一行一行处理
    for line in lines:
        doc_tot += 1
        #大写转小写
        line = line.lower()
        line = json.loads(line)
        #处理当前行得到tweetidid和text文本
        tweetid = Word(line['tweetid'])
        text = TextBlob(line['text']+" "+line['username'])
        texts = []
        for word in text:
            if word in uselessword:
                continue
            temp = Word(word)
            temp = temp.lemmatize("v") #词形还原
            texts.append(temp)

        unique_terms=set(texts)
        for term in unique_terms:
            df0[term][tweetid]=1
```

第二步：

计算 Wt_d。一行一行读入数据，对文本进行预处理操作，与前面相同，这里不做赘述。经过预处理后得到了这一个文本的 tweetid 和这个文本里包含的单词的集合 texts。

计算每个单词出现在当前 document 的次数

```python
doc_num={} #记录这个词出现的次数
for term in texts:
    if term in doc_num.keys():
        doc_num[term]+=1
    else:
        doc_num[term]=1
```

下面可以正式开始计算 Wt_d 了

根据输入的 smart 判断用什么方式计算 Wt_d。参考下方表格

tf-idf weighting has many variants

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

先计算 Term frequency 部分，如果输入的第 1 个字符是 n 就不用操作，直接使用出现次数 tf 即可；如果第一个字符是 l，就是用 logarithm 的方法……

```
#n/l/a/b/L
if(smart[0]=='n'):
    pass
if(smart[0]=='l'):
    for term in doc_num.keys():
        doc_num[term]=math.log(doc_num[term])+1
if(smart[0]=='a'):
    maxx=0
    for term in doc_num.keys():
        if maxx<doc_num[term]:
            maxx=doc_num[term]
    for term in doc_num.keys():
        doc_num[term]=0.5+(0.5*doc_num[term])/maxx
if(smart[0]=='b'):
    for term in doc_num.keys():
        doc_num[term]=1
if(smart[0]=='L'):
    ave=0;
    for term in doc_num.keys():
        ave+=doc_num[term]
    ave/=len(doc_num)
    for term in doc_num.keys():
        doc_num[term]=(1+math.log(doc_num[term]))/(1+math.log(ave))
```

现在第一个 term frequency 部分计算完毕，下面计算 document frequency 部分，根据读入的 smart 字符串的第二个字符判断用什么方法加上 document frequency 部分的权重。计算方式根据表格填写即可，实现如下：

```
#n/t/p
if(smart[1]=='n'):
    pass
if(smart[1]=='t'):
    for term in doc_num.keys():
        doc_num[term]*=math.log(doc_tot/len(df0[term]))
if(smart[1]=='p'):
    for term in doc_num.keys():
        doc_num[term]*=max(0,math.log((doc_tot-len(df0[term]))/len(df0[term])))
```

现在表格前两列都计算完毕，最后判断是否需要归一化，表格中 normalization 我只实现了前两种，即不归一化和 cosine 归一化，因为后两种没学过，也没大看懂，就没实现。

```
#归一化
#n/c
if(smart[2]=='n'):
    pass
if(smart[2]=='c'):
    sum=0
    for term in doc_num.keys():
        sum=sum+doc_num[term]*doc_num[term]
    sum=1.0/math.sqrt(sum)
    for term in doc_num.keys():
        doc_num[term]=doc_num[term]*sum
```

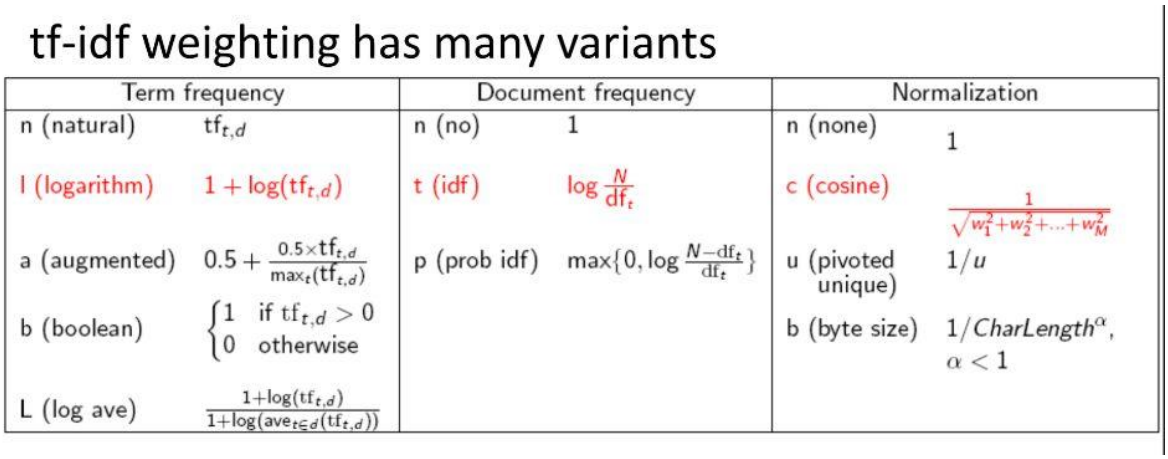
现在当前行的 Wt_d 计算完毕，把结果存到 Wt_d 字典中即可：

```
unique_terms=set(texts)
for term in unique_terms:
    Wt_d[term][tweetid]=doc_num[term]
```

OK，到此为止，我们的 Wt_d 就计算完毕了。下面一步可以计算 Wt_q 了

第三步：

计算 Wt_q



tf-idf weighting has many variants

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

计算方式与 Wt_d 完全相同，我再复述一遍。

先算出每个 term 在 query 中出现的次数即 tf，在根据输入的 smart 字符串的第 5 个字符判断计算 term frequency 的方法：

```
#n/l/a/b/L
if(smart[4]=='n'):
    pass
if(smart[4]=='l'):
    for term in Wt_q.keys():
        Wt_q[term]=math.log(Wt_q[term])+1
if(smart[4]=='a'):
    maxx=0
    for term in Wt_q.keys():
        if maxx<Wt_q[term]:
            maxx=Wt_q[term]
    for term in Wt_q.keys():
        Wt_q[term]=0.5+(0.5*Wt_q[term])/maxx
if(smart[4]=='b'):
    for term in Wt_q.keys():
        Wt_q[term]=1
if(smart[4]=='L'):
    ave=0;
    for term in Wt_q.keys():
        ave+=Wt_q[term]
    ave/=len(Wt_q)
    for term in Wt_q.keys():
        Wt_q[term]=(1+math.log(Wt_q[term]))/(1+math.log(ave))
```

现在第一个 term frequency 部分计算完毕，下面计算 document frequency 部分，根据读入的 smart 字符串的第 6 个字符判断用什么方法加上 document frequency 部分的权重。计算方式根据表格填写即可，实现如下：

```
#n/t/p
if(smart[5]=='n'):
    pass
if(smart[5]=='t'):
    for term in Wt_q.keys():
        Wt_q[term]*=math.log(doc_tot/len(df0[term]))
if(smart[5]=='p'):
    for term in Wt_q.keys():
        Wt_q[term]*=max(0,math.log((doc_tot-len(df0[term]))/len(df0[term])))
```

再根据 smart 的第 7 个字符判断是否需要对 Wt_d 进行归一化操作。

```
#归一化
#n/c
if(smart[6]=='n'):
    pass
if(smart[6]=='c'):
    sum=0
    for term in Wt_q.keys():
        sum=sum+Wt_q[term]*Wt_q[term]
    sum=1.0/math.sqrt(sum)
    for term in Wt_q.keys():
        Wt_q[term]=Wt_q[term]*sum
```

OK，现在 Wt_d 和 Wt_q 都有了，下一步可以计算 score 了

第四步：

每个 doc 的 score 就是每个 term 的 Wt_d 和 Wt_q 乘积的和。实现比较简单。

最后根据得分进行排序，方便下一步找到得分最大的 k 个 document

```
for term in query:
    if term in Wt_d:
        for doc in Wt_d[term]:
            if doc in doc_score.keys():
                doc_score[doc]+=Wt_d[term][doc]*Wt_q[term]
            else:
                doc_score[doc]=Wt_d[term][doc]*Wt_q[term]
doc_score_sorted=sorted(doc_score.items(),key=lambda x:x[1],reverse=True)
return doc_score_sorted
```

第五步：

读取输入的 query，进行与文本相同的预处理操作，然后根据 Wt_d 查看有多少与这个 term 相关的 document

```
n=int(input("请输入想得到的最相关的doc的数量:    "))
if (n==0):
    sys.exit()
query = myinput(input("请输入要查询词,退出请键入exit:  "))
ans = []
if len(query)==0:
    sys.exit()
ans_num = Union([set(Wt_d[term].keys()) for term in query])
print("一共有"+str(len(ans_num))+"条相关doc")
```

在对得到的相关的 tweetid 进行了一个取并集的操作，Union 函数里面调用了 reduce 函数

```
#取并集函数
def Union(sets):
    return reduce(set.union, [s for s in sets])
```

reduce 函数的作用和使用方法我放到实验总结里面了。

第六步：

调用函数，进行计算，最后输出 score 得分最高的 n 个 tweetid 即可。

```
print("与输入相关性最大的"+str(n)+"个doc
scores=compute_score(query)
i = 1
for (id, score) in scores:
    if i<=n:#返回前n条查询到的信息
        ans.append(id)
        print(str(score) + ": " + id)
        i = i + 1
    else:
        break
```

实验结果

测试：

使用计算机制 lnc.ltn 查询 I love you

即 document: logarithm tf 、 no idf 、 cosine normalization

query: logarithm tf 、 idf 、 no cosine normalization

```
请输入计算机制，格式请标准如：lnc.ltn: lnc.ltn

请输入想得到的最相关的doc的数量：  10

请输入要查询词,退出请键入exit:  i love you
一共有3584条相关doc
与输入相关性最大的10个doc的score及其tweetid分别
是:
4.0241263968150776: 301797008757383168
3.4739506296680225: 29257299906269184
3.1508926174333287: 32664801981243392
3.136135691575955: 30249363020189696
3.136135691575955: 315752938758864896
3.132460174393614: 30408437502312449
3.08464345532796: 626198996592250880
3.013437320264852: 624789576234745856
2.9059060283520677: 297854975344791553
2.862887102788692: 31941477105934336
```

使用计算机制 anc.lpn 查询 I love you

即 document： augmented tf 、 no idf 、 cosine normalization

query： logarithm tf 、 prob idf 、 no cosine normalization

```
请输入计算机制，格式请标准如：lnc.ltn: anc.lpn

请输入想得到的最相关的doc的数量：  10

请输入要查询词,退出请键入exit：  i love you
一共有3584条相关doc
与输入相关性最大的10个doc的score及其tweetid分别是:
3.3736924061957874: 301797008757383168
3.250830738307878: 29257299906269184
3.111007571771696: 30408437502312449
3.094839839416815: 30249363020189696
3.094839839416815: 315752938758864896
3.0444823989761676: 626198996592250880
2.8414536480684056: 624789576234745856
2.826732052190329: 32664801981243392
2.825189319857093: 31941477105934336
2.6986452137326835: 34397602740961281
```

结果分析：

我们把使用不同计算机制的两个查询结果放到一起对比一下，看看查询出的得分最高的 10 个得分最高的 document 是否相同。

```
请输入计算机制，格式请标准如：lnc.ltn: lnc.ltn

请输入想得到的最相关的doc的数量：  10

请输入要查询词,退出请键入exit：  i love you
一共有3584条相关doc
与输入相关性最大的10个doc的score及其tweetid分别是:
4.0241263968150776: 301797008757383168
3.4739506296680225: 29257299906269184
3.1508926174333287: 32664801981243392
3.136135691575955: 30249363020189696
3.136135691575955: 315752938758864896
3.132460174393614: 30408437502312449
3.08464345532796: 626198996592250880
3.013437320264852: 624789576234745856
2.9059060283520677: 297854975344791553
2.862887102788692: 31941477105934336
```

```
请输入计算机制，格式请标准如：lnc.ltn: anc.lpn

请输入想得到的最相关的doc的数量：  10

请输入要查询词,退出请键入exit：  i love you
一共有3584条相关doc
与输入相关性最大的10个doc的score及其tweetid分别是:
3.3736924061957874: 301797008757383168
3.250830738307878: 29257299906269184
3.111007571771696: 30408437502312449
3.094839839416815: 30249363020189696
3.094839839416815: 315752938758864896
3.0444823989761676: 626198996592250880
2.8414536480684056: 624789576234745856
2.826732052190329: 32664801981243392
2.825189319857093: 31941477105934336
2.6986452137326835: 34397602740961281
```

通过对比发现，两个方法得出来的得分最高的两个 document 虽然得分不同，但是排名相同。后面的排名就有些变化了，左边排名第 3 的 doc 到右边排名第 8；左边排名第 6 的 doc 到右边排名第 3；左边排名第 7 的 doc 到右边排名第 6；左边排名第 8 的 doc 到右边排名第 7；左边排名第 9 的 doc 到右边排名不在前 10；左边排名第 10 的 doc 到右边排名第 9。

根据实验结果我们可以得出结论，不同的计算机制计算出来的 score 不同，排名基

本相同，但差别不大。所以在找相关度最高的 document 的时候，使用不同的相关度计算方法的结果可能会不同。

实验总结

在本次实验中，我实现了 Ranked retrieval model。实践了课堂上学习的计算 query 查询与 doc 文本之间相关度的方法，对相关性的相关知识有了更加深刻的理解。同时，在实验过程中我也遇到了一些问题，并在解决问题的过程中学习到了新的知识。

比如在最后获得一共有多少个 doc 与 query 有关的数据的时候，用到了取并集的思想，在取并集的函数 union 中，用到了 python 库 functools 中的 reduce 函数，于是在网上查找了 reduce 函数的用法，

reduce(function, sequence[, initial]) -> value

reduce 函数接受一个 function 和一串 sequence，并返回单一的值，以如下方式计算：

1.初始，function 被调用，并传入 sequence 的前两个 items，计算得到 result 并返回

2.function 继续被调用，并传入上一步中的 result，和 sequence 种下一个 item，计算得到 result 并返回。一直重复这个操作，直到 sequence 都被遍历完，返回最终结果。

我还了解了 python 中 defaultdict 和 dict 的区别

这里的 defaultdict(function_factory)构建的是一个类似 dictionary 的对象，其中 keys 的值，自行确定赋值，但是 values 的类型，是 function_factory 的类实例，而且具有默认值。比如 defaultdict(int)则创建一个类似 dictionary 对象，里面任何的 values 都是 int 的实例，而且就算是一个不存在的 key, d[key] 也有一个默认值，这个默认值是 int 类型值 0.。总之就是如果索引一个不存在的 key，dict 会报错，而 defaultdict 不会报错。

我完成了实验的选做部分，实现了所有的 smart 方法，并且通过比较使用不同的相关度计算方法查询相同的 query，对比分析结果，发现不同的计算机制计算出来的 score 不同，排名基本相同，但差别不大。所以在找相关度最高的 document 的时候，使用不同的相关度计算方法的结果可能会不同。