山东大学______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

学号：201600130053	姓名：王斌	班级： 16 人工智能

实验题目： Pivoted Length Normalization VSM and BM25

实验内容：

#Homework 4: Pivoted Length Normalization VSM and BM25

任务：

- 实现 Pivoted Length Normalization VSM;
- 实现 BM25;

注意

- 改进 Postings：(docID, Freq)，不仅记录单词所在的文档 ID，也记录其在文档中的 Frequency;
- 构建 inverted index 时，记录文档的长度，以及计算 average document length (avdl)

实验环境：

Spyder+python3.6
Win10

实验过程中遇到和解决的问题：

（记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）

一、 对推特数据的处理和对 postings 等一些数据结构的增加与改善

对推特数据文本的处理与实验三相同，只保留用户名、tweet 内容以及用来作为文档 id 的 tweetid，修改 postings 的结构，添加 TF 的信息，保存每个单词在每条 tweet 中的词频信息，tf = postings[term][tweeted],打印后如下所示：

```
defaultdict(<class 'dict'>, {'arus': {'28965792812892160': 1}, 'house':
{'28965792812892160': 2}, 'rand': {'28965792812892160': 1}, 'kill':
{'28965792812892160': 1}, 'the': {'28965792812892160': 2, '28968581949558787': 1,
'28969422056071169': 1, '28971749961891840': 1, '28974904342740992': 2,
'28977078074343425': 1, '28977806142603264': 2}, 'immigration': {'28965792812892160':
1}, 'be': {'28965792812892160': 1, '28968581949558787': 1, '28971749961891840': 1,
'28976831738683393': 1}, 'unlikely': {'28965792812892160': 1}, 'to':
{'28965792812892160': 1, '28967672074993664': 1, '28968581949558787': 1,
'28974862038994945': 3, '28974904342740992': 1}, 'say': {'28965792812892160': 1,
'28976409057697792': 1}, 'http': {'28965792812892160': 1, '28967095878287360': 1,
'28967914417688576': 1, '28968479176531969': 1, '28969422056071169': 1,
'28973080491589632': 1, '28974862038994945': 1, '28974904342740992': 1,
'28976409057697792': 1, '28976831738683393': 1, '28977078074343425': 1,
'28977806142603264': 1}, 'tinyurl.com/4jrjcdz': {'28965792812892160': 1}, 'rick':
```

TF

添加其它需要的数据结构来分别记录 DF、文档长度（tweet 词数）、文档数量和文档平均长度（词数）：

```
postings = defaultdict(dict)
```

```
        document_frequency = defaultdict(int)
        document_lengths= defaultdict(int)
        document_numbers = len(document_lengths)
        avdl=0
```

文档长度（每条 tweet 词数）的数据结构如下所示：

```
defaultdict(<class 'int'>, {'28965792812892160': 22, '28967095878287360': 21,
'28967672074993664': 13, '28967914417688576': 16, '28968479176531969': 11,
'28968581949558787': 11, '28969422056071169': 26, '28971749961891840': 26,
'28973080401589632': 18, '28974862038994945': 25, '28974904342740992': 27,
'28976409057697792': 22, '28976831738683393': 23, '28977078074343425': 20,
```

文档词数长度

tweetid

并分别通过下列函数来初始化，完成数据结构的创建；

```
        get_postings_dl()
        initialize_document_frequencies()
        initialize_avdl()
```

二、 PLN_VSM 和 BM25 的具体实现应用

首先依旧是利用已有数据结构实现查询的功能，对于输入的一串语句，进行相同的 token 处理，而后为了加快检索速率，避免去遍历所有 tweet 计算每一个 F（q,d），可以先对于查询检索提取出相关的 tweetid,得到 relevant_tweetids 列表，然后利用 PLN 和 BM25 的方法分别计算他们的分数最后降序输出。

核心 search 方法如下：

```
def do_search():
    query = token(input("Search query >> "))
    if query == []:
        sys.exit()
    unique_query = set(query)
    #避免遍历所有的 tweet，可先提取出有相关性的 tweetid，tweet 中包含查询的关键
词之一便可认为相关
    relevant_tweetids = Union([set(postings[term].keys()) for term
in unique_query])
    #print(relevant_tweetids)
    if not relevant_tweetids:
        print ("No tweets matched any query terms.")
    else:
        scores1 = sorted([(id,similarity_PLN(query,id))
                          for id in relevant_tweetids],
                         key=lambda x: x[1],
                         reverse=True)
        scores2 = sorted([(id,similarity_BM25(query,id))
                          for id in relevant_tweetids],
                         key=lambda x: x[1],
                         reverse=True)
```

```
        print ("<<<<<Score(PLN)--Tweeetid>>>>>")
        print("PLN 一共有"+str(len(scores1))+"条相关 tweet！")
        for (id,score) in scores1:
            print (str(score)+": "+id)
        print ("<<<<<Score(BM25)--Tweeetid>>>>>")
        print("BM25 一共有"+str(len(scores2))+"条相关 tweet！")
        for (id,score) in scores2:
            print (str(score)+": "+id)
```

其中 similarity_PLN(query,id)和 similarity_BM25(query,id)分别使用 PLN 和 BM25 的公式来计算，具体如下：

- Pivoted Length Normalization VSM [Singhal et al 96]

$$f(q,d)=\sum_{w\in q\cap d} c(w,q)\frac{\ln[1+\ln[1+c(w,d)]]}{1-b+b\frac{|d|}{avdl}}\log\frac{M+1}{df(w)}$$

对于 PLN 中的参数 b,由于每条 tweet 的次数有限，平均每条 18 个词左右（加上用户名），因此不同文档的长度不会相差太多，对于文档长度低于（或高于）平均长度的奖励 （或惩罚）比重不宜太大，经过实验发现设为 0.1 便可满足需求。

BM25/Okapi [Robertson & Walker 94]

$$b\in[0,1]$$
$$k_1,k_3\in[0,+\infty)$$

$$f(q,d)=\sum_{w\in q\cap d} c(w,q)\frac{(k+1)c(w,d)}{c(w,d)+k(1-b+b\frac{|d|}{avdl})}\log\frac{M+1}{df(w)}$$

对于 BM25 模型，基于同样原理，其比重不应太大，于是将 k 设为 1，b 同样设为 0.1.

```
def similarity_PLN(query,id):
    global postings,avdl
    fenmu =1 - 0.1 + 0.1*(document_lengths[id]/avdl)
    similarity = 0.0
    unique_query = set(query)
    for term in unique_query:
        if (term in postings) and (id in postings[term].keys()):
            #使用 ln(1+ln(C(w,d)+1))后发现相关性的分数都为负数很小
```

```
            similarity +=
(query.count(term)*(math.log(math.log(postings[term][id] + 1) + 1))
*math.log((document_numbers+1)/document_frequency[term]))/fenmu

    return similarity

def similarity_BM25(query,id):
    global postings,avdl
    fenmu =1 - 0.1 + 0.1*(document_lengths[id]/avdl)
    k = 1
    similarity = 0.0
    unique_query = set(query)
    for term in unique_query:
        if (term in postings) and (id in postings[term].keys()):
            C_wd = postings[term][id]
            #使用 ln(1+ln(C(w,d)+1))后发现相关性的分数都为负数很小
            similarity += (query.count(term)*(k+1)*C_wd*math.log
((document_numbers+1)/document_frequency[term]))/(k*fenmu+C_wd)

    return similarity
```

三、 进行查询测评比较

选取三条查询输入，观察两种方法各自返回的 tweetid 列表：

（1） Search query >> Ron Weasley birthday

```
<<<<<Score(PLN)--Tweeetid>>>>>          top10
PLN一共有153条相关tweet!
-2.1773496071180065: 30349168828481536
-2.180508094747723: 298402470445588480
-2.1888134452324355: 297341726752907264
-2.2046111632284724: 315884652470607872
-2.2359019193279543: 297445095383396354
-2.2359019193279543: 299653341938589696
-2.241781732846521: 307656073949618176
-2.2479923202216723: 32189228423053312
-2.257267759423584: 301143259357540353
-2.260214187107503: 31340983396335616
-2.260214187107503: 30987549543497728
-2.2725696759668703: 29540081047961602
-2.2725696759668703: 315263878726553600
-2.2725696759668703: 623864941942956032
-2.285060990183119: 30045176936271873
-2.285060990183119: 32921017177341953
-2.285060990183119: 316322634281394176
-2.29769038185145: 314330486669443072
-2.3065985926158437: 307332118504144896
-2.310460153132542: 32203309838241793
-2.3233726576515563: 33199661187600385
-2.3233726576515563: 299835089519529984
```

```
<<<<<Score(BM25)--Tweeetid>>>>>         top10
BM25一共有153条相关tweet!
-4.2008641311821355: 298402470445588480
-4.22429114440538: 315884652470607872
-4.259925718514876: 307656073949618176
-4.283210564049616: 30349168828481536
-4.294861304122313: 297341726752907264
-4.320671727575271: 307332118504144896
-4.342105058921052: 297445095383396354
-4.342105058921052: 299653341938589696
-4.354078846829904: 32189228423053312
-4.366118855177776: 31340983396335616
-4.366118855177776: 30987549543497728
-4.378225634831206: 29540081047961602
-4.378225634831206: 315263878726553600
-4.378225634831206: 623864941942956032
-4.383175259710214: 307498389086535681
-4.390399742783701: 30045176936271873
-4.390399742783701: 32921017177341953
-4.390399742783701: 316322634281394176
-4.39589360047958: 307416801489342464
-4.39589360047958: 307402142384267264
-4.39589360047958: 307592068874772480
-4.402641742241153: 314330486669443072
```

两者前十个中有 9 个相同，顺序略有不同！

（2） Search query >> Boko Haram kidnapped French tourists

top10

```
<<<<<Score(PLN)--Tweeetid>>>>>
PLN一共有281条相关tweet!
-1.6677014547692184: 303646155764551680
-1.6677014547692184: 624972649236721664
-1.6859344761020516: 317178252697313328
-1.6952013150956846: 314074841701457792
-1.6952013150956846: 302141361098981376
-1.704570588588598: 625765410433052672
-1.7140440044677268: 314253694499635520
-1.7431067644589955: 626319742245183488
-1.7431067644589955: 626495521302179840
-1.7630357293807317: 302884390445383680
-2.2465684030108037: 335543369025553600
-2.297357825087473: 335083441991188848
-2.323623550424758: 325920024869027784
-2.426334476630183: 626099113436643328
-2.44062316304116: 311455840735473664
-2.4574228132397358: 626484733564710912
-2.4574228132397358: 298484397798215681
-2.4709301837054163: 626305984911273984
-2.4709301837054163: 309061012852613l2
```

top10

```
<<<<<Score(BM25)--Tweeetid>>>>>
BM25一共有281条相关tweet!
-3.2301283068093767: 303646155764551680
-3.2301283068093767: 624972649236721664
-3.2480419060310157: 317178252697313328
-3.2570734215572306: 314074841701457792
-3.2570734215572306: 302141361098981376
-3.2661553032526554: 625765410433052672
-3.2752879736118543: 314253694499635520
-3.302995013098926: 626319742245183488
-3.302995013098926: 626495521302179840
-3.321728278145011: 302884390445383680
-4.304686090875959: 335543369025553600
-4.353239626101066: 335083441991188848
-4.377929458026776: 325920024869027784
-4.3843353018249545: 626099113436643328
-4.7212634197739956: 311455840735473664
-4.734354977242986: 626484733564710912
-4.734354977242986: 298484397798215681
-4.747519339563616: 626305984911273984
-4.747519339563616: 309061012852613l2
-4.7607571157535 2: 310554595740876 81
```

两者前 10 个结果完全一样，并且顺序一致！

（3） Search query >> Chinua Achebe death

top10

```
<<<<<Score(PLN)--Tweeetid>>>>>
PLN一共有796条相关tweet!
-2.822882838686114 7: 297492784619847680
-2.867706631808294: 297385011970195459
-2.8829659104646144: 309239671177752576
-2.8829659104646144: 297324140048826368
-2.8829659104646144: 324564513141637 12
-2.8983884493366596: 625422932915957760
-2.8983884493366596: 295044157989191 70
-2.913976882616871: 302826634891894787
-2.929733901474145: 311785831826345985
-2.929733901474145: 315138896835014657
-2.929733901474145: 315124992742420482
-2.929733901474145: 298260832981250048
-2.929733901474145: 297406893687709697
-2.929733901474145: 297464804417867776
-2.929733901474145: 314150320370483201
```

top10

```
<<<<<Score(BM25)--Tweeetid>>>>>
BM25一共有796条相关tweet!
-5.5670645332139514: 297492784619847680
-5.612617275900826: 297385011970195459
-5.627967637544957: 309239671177752576
-5.627967637544957: 297324140048826368
-5.627967637544957: 324564513141637 12
-5.643402195139082: 625422932915957760
-5.643402195139082: 295044157989191 70
-5.65892164330391: 302826634891894787
-5.674526684322096: 311785831826345985
-5.674526684322096: 315138896835014657
-5.674526684322096: 315124992742420482
-5.674526684322096: 298260832981250048
-5.674526684322096: 297406893687709697
-5.674526684322096: 297464804417867776
-5.674526684322096: 314150320370483201
```

前 10 个结果完全一样，并且顺序一致！

综上比较可以发现，两种方法的效果十分接近，但具体哪一种的效率或准确率更好，还有待进一步的测评。

四、 更新用 qrels.txt 和 eval_hw4.py 对检索模型的结果进行评测：

1、利用提供的 result.txt 进行演示（baseline？）

```
query 217 , NDCG:  0.7675078383310092
query 218 , NDCG:  0.8302203434012001
query 219 , NDCG:  0.498155912259978
query 220 , NDCG:  0.5674800702438964
query 221 , NDCG:  0.9266372064962487
query 222 , NDCG:  0.5087328728028815
query 223 , NDCG:  0.9063275712084274
query 224 , NDCG:  0.3773185814513307
query 225 , NDCG:  0.9706077927297266
NDCG = 0.756819929645465
```

```
query: 217 ,AP: 0.625
query: 218 ,AP: 0.30303030303030304
query: 219 ,AP: 0.25524197520567754
query: 220 ,AP: 0.6138226621145667
query: 221 ,AP: 0.1988071570576541
query: 222 ,AP: 0.30126376980342995
query: 223 ,AP: 0.9940746736049804
query: 224 ,AP: 0.5178732378732379
query: 225 ,AP: 0.9920063553263518
MAP = 0.6148422817122279
```

刚开始由于弄混了升序和降序（sorted 函数中的 reverse 参数），导致结果很低，调整后结果如下：

1、使用 PLN 结构的结果如下：

(1) PLN result 01 (b = 0.1):

MAP = 0.4862845177853686　　NDCG = 0.6793521661615348

(2) PLN result 01 (b = 0.2):

MAP = 0.4857685995413381　　NDCG = 0.6752810089814107

B = 0.1 时有更好表现

2、使用 BM25 结构的结果如下：

(1) BM25 result 01 (k = 1, b = 0.1) :

MAP = 0.493037294021037　　NDCG = 0.6856268118707195

(2) BM25 result 02 (k = 2, b = 0.1) :

MAP = 0.476015785676824　　NDCG = 0.6648275730737171

(3) BM25 result 03 (k = 1, b = 0.2) :

MAP = 0.49094832022050855　　NDCG = 0.6836247209668984

(4) BM25 result 04 (k = 1, b = 0.3) :

MAP = 0.48967250676950935　　NDCG = 0.6807066698251555

综上可以发现 k=1, b=0.2 时有最好表现，MAP = 0.491, NDCG = 0.684

3、综合 PLN 和 BM25: MAP = 0.4902771967137613　　NDCG = 0.6827968041343199 发现并没有好多少与单独使用单一结构相近
结论分析与体会: 　　通过对 Pivoted Length Normalization VSM and BM25 的实现，对于 inverted index 模型的应用更加熟练了，对于较大规模的文本数据处理和简单检索也有了更深入的掌握。