QINMIN HU

2018年5月1日

LAB 3 索引模型建立及测试

一. 实验目的

- 布尔检索
- TF-IDF & BM25
- 检索结果 & 评测

二. 实验思路

1. 布尔检索（例子A and B）
   - 根据第二次实验的倒排索引列表定位出词典中的A，B，返回倒排记录表
   - 两个倒排记录表求交集

因为布尔检索存在问题，所以跟进一步用TF-IDF & BM25改进：

2. TF-IDF & BM25
   - 给定Query q={t_1,…,t_n}
   - 在倒排索引中找到候选文档集合 $D=\bigcup_{t=1}^{n} docs\_in\_postinglist(t_i)$
   - 计算相关程度 TF-IDF 、BM25
   - 返回最终排序结果
3. 检索结果 & 评测——TREC 评测脚本(linux)
   - Installation: trec_eval目录下make命令
   - Evaluation: ./trec_eval [-q] [-m measure] qrel_file res_file >> xxx.xxx

三. 实验步骤

1. 布尔检索

1.1. get问题

```
while(True):
```

```
query_word = input("请输入查询语句：")
if query_word == "quit":
    break
result = getBooleanSearch(query_word)
```

（对输入的查询语句进行布尔查询）

```
for i in range(len(result)):
    if result[i] == 1:
        print(doc_filename[i])
```

（将查询的文件名结果返回）

1.2. 根据倒排索引列表定位出词典中的查询词语，构造矩阵

```
def makeMatrix(word):
    word_array=[0]*len(doc_filename)
    i=0
    word_key=[]
    for key in word_dic[word].keys():
        word_key.append(key)
    for doc in doc_filename:
        if doc in word_key:
            word_array[i]=1
        else:
            word_array[i]=0
        i=i+1
    return word_array
```

1.3. 对矩阵进行and、or、not计算

```
def operatorAnd(word1,word2,search_result):
    arr=[0]*len(doc_filename)
    if word1:
        arr1 = makeMatrix(word1)
        arr2 = makeMatrix(word2)
        for i in range(len(arr1)):
            arr[i] = arr1[i] & arr2[i]
    else:
        arr2 = makeMatrix(word2)
        for i in range(len(arr2)):
            arr[i] = search_result[i] & arr2[i]
    return arr
```

（A and B 的计算方法，A、B为矩阵）

Or 、not 计算方法类似：

def operatorOr(word1,word2,search_result)

def operatorNot(word1,word2,search_result)【具体见代码】

2. 以相关程度TF-IDF为标准的查询方式

2.1.对query进行处理

```
def loadQuery():
    xmldoc = ET.parse('topics_new.xml')
    for doc in xmldoc.findall('top'):
        test=''
        for s1 in doc.findall('num'):
            query_id.append(s1.text.replace("Number:","").strip())
        for s2 in doc.findall('desc'):
            test=test+s2.text.replace("Descript :","").strip()
        for s3 in doc.findall('narr'):
            test=test+s3.text.replace("Narr :","").strip()
        while '_' in test or '`' in test or '.' in test or '-' in test or '!' in test or ';' in test or ',' in test or '$' in test or '//' in test or '/' in test or '{' in test or '}' in test or '*' in test or '#' in test or '^' in test or '|' in test or '~' in test or '=' in test or '\'' in test or '+' in test or ':' in test or '?' in test:
            test = test.replace("_", "").replace('`', "").replace(".", "").replace('-', "").replace("!","").replace(";","").replace(",", "").replace("$", "").replace("//", "").replace('/', "").replace('{', "").replace('}',"").replace('*',"").replace('^', "").replace("|", "").replace('~', "").replace('=', "").replace('\'', "").replace('+',"").replace(':',"").replace('?', “”)
```

（将topics中的所有符号去除）

```
        if re.findall(r'\d+', test) or test == "":
            continue
        query.append(test.split())
```

2.2.在倒排索引中找到候选文档集合

```
def loadDic():
    f_dic=open("inverted_index.json",encoding='utf-8')
    word_list=json.load(f_dic)
    for word in word_list:
        word_dic[word[0]]=word[1]
```

（word_dic: 单词的倒排索引字典）

```
    for e in word_dic.keys():
        key.append(e)
```

2.3.计算指标

公式：

$$Score(q,d) = \sum_{t \in q} tf_{t,d} * idf(t)$$

$tf_{t,d}$ term t在doc d出现的次数

$$idf(t) = \log \frac{N}{n_t}$$

```
def getDocLength(dirname,filename):
    if filename != '.DS_Store':
        xmldoc = ET.parse(dirname + '/'+filename)
        for doc in xmldoc.findall('DOC'):
            doc_list=[]
            for s1 in doc.findall('TEXT'):

                if s1.text==None:
                    doc_list.append('')
                else:
                    for word in s1.text.split():
                        while '_' in word or '`' in word or '.' in word or '-' in word or '!' in word or ';' in word or ',' in word or '$' in word or '//' in word or '/' in word or '{' in word or '}' in word or '*' in word or '#' in word or '^' in word or '|' in word or '~' in word or '=' in word or '\'' in word or '+' in word or ':' in word or '?' in word:
                            word = word.replace("_", "").replace('`', "").replace(".", "").replace('-', "").replace("!","").replace(";", "").replace(",", "").replace("$", "").replace("//", "").replace('/', "").replace('{',"").replace('}', "").replace('*', "").replace('^', "").replace("|", "").replace('~', "").replace('=',"").replace('\'', "").replace('+', "").replace(':', "").replace('?', "")
                        if re.findall(r'\d+', word) or word == "":
                            continue
                        doc_list.append(word)
            for s2 in doc.findall('DOCNO'):
                docno=s2.text
            doc_filaname[docno]=len(doc_list)
def loadDocument(dirname):
    for parent,dirnames,filenames in os.walk(dirname):
        for e in filenames:
            getDocLength(dirname,e)
```

（获取document文档的信息）

```
if __name__ == '__main__':
    for dirname in dirlist:
        loadDocument(dirname)
    count = 0
    for doc in doc_filaname:
        count=count+doc_filaname[doc]
```

```
        for key,value in word_dic.items():
            word_doc_freq[key]=len(word_dic[key])
```

（word_doc_freq: 出现t的文档数目）

```
        N = len(doc_filaname)
```

（N: 文档集的大小）

```
        avgLen=count/N
    def getBM25(words,doc_id):
        scoreArr={}
        for file in doc_filaname.keys():
            s=0
            for word in words:
                if word in word_doc_freq.keys():
                    idf=math.log((N-word_doc_freq[word]+0.5)/
(word_doc_freq[word]+0.5))
                else:
                    idf = math.log((N - 0+ 0.5) / (0+ 0.5))
                if word in word_dic.keys():
                    if file in word_dic[word].keys():
                        freq = word_dic[word][file]
                        s = s + idf * freq * 2.5 / (freq + 1.5 *
(0.25 + 0.75 * doc_filaname[file] / avgLen))
                else:
                    s= s + 0
            scoreArr[file]=s
        score[doc_id]=scoreArr
```

（对文件按公式计算TF-IDF查询的score）

2.4.排序、按照顺序输出查询结果写入文件

```
if __name__ == '__main__':
    f = open('10152130138_丁婉宁_tfidf.res', 'w')
    for key,value in score.items():
        valueSort=sorted(value.items(),key=lambda
d:d[1],reverse=True)
```

（按照计算出的score排序）

```
        t=0
        for key1, value1 in valueSort:
            f.write(key+' '+'0'+' '+key1+' '+str(t)+' '+str(value1)+'
‘+'10152130138_tfidf'+'\n')
```

（将排好序的查询结果写入10152130138_丁婉宁_tfidf.res文件）

```
            t=t+1
    f.close()
```

3. 以相关程度BM25为标准的查询方式，——只有在计算指标上和上述TF-IDF查询方式有区别

公式：

$$Score(q,d) = \sum_{i=1}^{n} IDF(q_i) * \frac{f(q_i,d) * (k_1 + 1)}{f(q_i,d) + k_1 * (1 - b + b * \frac{|d|}{avgdl})}$$

$$IDF(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

$$k_1 \in [1.2, 2.0]\ b = 0.75$$

```
def getTfIdf(words,doc_id):
    scoreArr={}
    for file in doc_filaname.keys():
        s = 0
        for word in words:
            if word in word_doc_freq.keys():
                idf=math.log(N/word_doc_freq[word])
            else:
                idf=0
            if word in word_dic.keys():
                if file in word_dic[word].keys():
                    freq = word_dic[word][file]/
doc_filaname[file]
                    s = s + idf * freq
                else:
                    s= s + 0
        scoreArr[file]=s
    score[doc_id]=scoreArr
```

（对文件按公式计算BM25查询的score）

4. 检索结果 & 评测

4.1.对结果处理成合适评测的格式

```
f = open('qrels.151-200','w')
fpart = []
```

```
fpart.append(open('qrels.151-200.disk1.disk2.part1','r').read())
fpart.append(open('qrels.151-200.disk1.disk2.part2','r').read())
fpart.append(open('qrels.151-200.disk1.disk2.part3','r').read())
fpart.append(open('qrels.151-200.disk1.disk2.part4','r').read())
fpart.append(open('qrels.151-200.disk1.disk2.part5','r').read())
for part in fpart:
        f.write(part)
f.close()
```

（对给出的五个part的答案进行合并，方便检测结果，计算准确率、召回率等）

4.2.进行评测

```
➜ trec_eval.9.0 ./trec_eval qrels.151-200  10152130138_丁婉宁_tfidf.res >>  test_tfidf_10152130138.txt
➜ trec_eval.9.0 ./trec_eval qrels.151-200  10152130138_丁婉宁_BM25.res >>  test_BM25_10152130138.txt
```

4.3.得到的评测结果

test_tfidf_10152130138.txt

runid	all	10152130138_tfidf
num_q	all	50
num_ret	all	33874950
num_rel	all	9805
num_rel_ret	all	8014
map	all	0.0154
gm_map	all	0.0038
Rprec	all	0.0454
bpref	all	0.0710
recip_rank	all	0.1883
iprec_at_recall_0.00	all	0.2148
iprec_at_recall_0.10	all	0.0549
iprec_at_recall_0.20	all	0.0205
iprec_at_recall_0.30	all	0.0091
iprec_at_recall_0.40	all	0.0032
iprec_at_recall_0.50	all	0.0017
iprec_at_recall_0.60	all	0.0010
iprec_at_recall_0.70	all	0.0003
iprec_at_recall_0.80	all	0.0002
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	all	0.0920
P_10	all	0.0980
P_15	all	0.0933
P_20	all	0.0930
P_30	all	0.0800
P_100	all	0.0602
P_200	all	0.0470
P_500	all	0.0300
P_1000	all	0.0207

test_BM25_10152130138.txt

runid	all	10152130138_BM25
num_q	all	50
num_ret	all	33874950
num_rel	all	9805
num_rel_ret	all	8014
map	all	0.0797
gm_map	all	0.0435
Rprec	all	0.1431
bpref	all	0.1331
recip_rank	all	0.7172
iprec_at_recall_0.00	all	0.7466
iprec_at_recall_0.10	all	0.2645
iprec_at_recall_0.20	all	0.1476
iprec_at_recall_0.30	all	0.0662
iprec_at_recall_0.40	all	0.0254
iprec_at_recall_0.50	all	0.0147
iprec_at_recall_0.60	all	0.0049
iprec_at_recall_0.70	all	0.0004
iprec_at_recall_0.80	all	0.0002
iprec_at_recall_0.90	all	0.0001
iprec_at_recall_1.00	all	0.0000
P_5	all	0.4840
P_10	all	0.4220
P_15	all	0.3720
P_20	all	0.3460
P_30	all	0.3120
P_100	all	0.1948
P_200	all	0.1321
P_500	all	0.0711
P_1000	all	0.0405