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：201600130050	姓名： 张涵	班级： 人工智能班

实验题目：倒排索引和布尔检索模型

实验内容：
使用课堂介绍的方法，在 tweets 数据集上构建 inverted index；
实现 Boolean Retrieval Model，使用 TREC 2014 test topics 进行测试；

实验过程中遇到的问题：

实验过程：

首先是预处理数据集。对数据集的预处理要注意一点，就是对于 tweets 与 queries 使用相同的预处理。预处理的简单过程就是：

```
#对tweets.txt进行处理，去除词性标注以、日期及一些杂质.（保留段落结构）
#输入:tweets.txt
#输出:tweets1_new.txt
def pre_file(filename):
    print("读取语料库文件%r....."%filename)
    src_data = open(filename).read()
    #去除词性标注、杂质如‘[’、‘]nt’
    des_data =re.compile(r'(\/\w+)|(\d+\-\S+)|(
\S+)').sub('',src_data)
    des_filename = "tweets_new.txt"
    print("正在写入文件%r....."%des_filename)
    open(des_filename,'w').writelines(des_data)
    print("处理完成!")

```

对文本文档进行过预处理之后，我们要对其进行构建倒排索引表，**倒排索引（英语：Inverted index）**，也常被称为反向索引、置入档案或反向档案，是一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。它是文档检索系统中最常用的数据结构。

而我在建立倒排索引表的时候，将其分为了三部分：

1. count.py 确定停词

   先得到所有词汇，然后把词汇按增序排序，输出数量最多的 250 个单词，然后对其进行处理。

```
vocabulary=sorted(vocabulary, key=itemgetter(1),reverse=True)                #按增序排序
print vocabulary[:250]
print 'drawing plot.....'
fdist.plot(120 , cumulative=False)

#output in file
file_object = open('thefile.txt', 'w')
for j in vocabulary:
    file_object.write( j[0] + ' ')
file_object.close( )
```

2. index.py 建立倒排索引

先对其进行排序，然后建立字典，进行 split 操作，并判断单词是否在字典中。

```
def output_index(result):
    #print result

    output = open('data.pkl', 'wb')
    pickle.dump(result, output)                                    # 建立字典
    output.close()
```

```
for words in txt :
    words =words.decode('utf-8').encode(sys.getfilesystemencoding())        #change string typt from utf-8 to gbk
    if result.get(words) == None :                                          #如果单词不在字典中
        result[words]=[file[x]]
    else:                                                                   #如果单词在字典中
        t=result.get(words)
        t.append(file[x])
        result[words]=t
```

3. query.py 用于查询

保存单词下标到'data.pkl'中，然后对文档中的单词进行判断，其是否在下标集中，并进行不同操作，然后对 record 进行判断分别进行不同操作，最后判断 list 是否已经存在。

```
def getdata():
    pkl_file = open('data.pkl', 'rb')                                   #下标被保存在哪 'data.pkl'
    data1 = pickle.load(pkl_file)                                           #更改格式
    #pprint.pprint(data1)
    pkl_file.close()
    return  data1
```

```
def output( result ):
    #print result
    if result == None:                                          #如果单词没有再下标集中（没有return）
        print None
        return
    if len(result) == 0 :                                       #如果单词没有再下标集中（多个return）
        print None
        return
```

```
result=[]
for k in range( 0 , len(query_list) ):
    if k==0:                                                                #如果没有list被建立
        result = data_list.get( query_list[0] )
    else:                                                                   #如果list已经被建立
        result = list( set(result).intersection(data_list.get( query_list[k] ) ) )
output( result )
```

结论分析与体会：

这次的布尔检索模型是使用 Python 写的，自己看了很多，也在网上查了很多，然后完成了这次实验，不过因为 Python 水平有限，可能实验代码还有很多可以优化的地方，以后会逐步进行优化。

此次实验也对布尔检索模型有了较为详细的了解，到目前为止，布尔模型是最常用的检索模型，因为：

1. 由于查询简单，因此容易理解 通过使用复杂的布尔表达式，可以很方便地控制查询结果 相当有效的实现方法
2. 相当于识别包含了一个某个特定 term 的文档 经过某种训练的用户可以容易地写出布尔查询式
3. 布尔模型可以通过扩展来包含排序的功能，即“扩展的布尔模型”

不过布尔检索模型也有很多局限性：

1. 布尔模型被认为是功能最弱的方式，其主要问题在于不支持部分匹配，而完全匹配会导致太多或者太少的结果文档被返回，非常刚性： “与”意味着全部； “或”意味着任何一个
2. 很难控制被检索的文档数量，原则上讲，所有被匹配的文档都将被返回很难对输出

进行排序.
3.不考虑索引词的权重，所有文档都以相同的方式和查询相匹配
4.这个过程是“data retrieval” 而不是 “information retrieval”。