信息系统检索

向量空间模型

姓名： 蒋浩南　　学号：2012948

作业描述

给定查询文档集合（诗词txt文件），完成向量空间模型并对文档集合实现查询功能。

作业要求

1. 实现带域的查询功能。
2. 编写实验报告，主要阐述代码细节以及原理。
3. 录制演示视频，举例演示查询功能。
4. 编程语言不做限制。

实现思路

一、数据集处理、建立向量空间

（1）词袋生成、map构建

- 分别构建标题、作者和内容的词袋，实现为遍历文件。

一层循环为构建文档的词袋，需要对输入的题目进行预处理，去掉.txt，小写。并添加doc和id的map。

二层循环为构建作者和内容的词袋。第一行为作者，其余行为内容。输入文本需要预处理，提取作者名，去除停用词，小写。

- 根据生成标题、作者和内容的词袋构建IDMAP

（2）生成tf

分别构建标题、作者和内容的tf向量。

实现为类似于上一步的循环，根据上一步构建的词袋，遍历数据集，累加term。

向量形如：
term doc1 doc2
a 0 2
b 1 2
c 0 1

```
#tf向量，（term数，文档数）
tf_title=np.zeros((len(bag_words_title),num_doc))
tf_author=np.zeros((len(bag_words_author),num_doc))
tf_content=np.zeros((len(bag_words_content),num_doc))
```

（3）生成df、idf

遍历生成的tf向量，df为tf一行中不为0的文档数，idf为 log10(N/df)，其中N为文档数,df为含有对应term的文档数。

```
#df和idf向量

df_title=np.zeros((len(bag_words_title),2))
df_author=np.zeros((len(bag_words_author),2))
df_content=np.zeros((len(bag_words_content),2))

'''
其中IDF = log10( N/df)，其中N为文档数,df为含有对应term的文档数

向量形如：
term  df  idf
a    2  log10(N/2)
b    1  log10(N/1)
c    1  log10(N/1)
'''

#生成term对应的df和idf
def df(len,num_doc,tf_x,df_x):
    for i in range(len):
        count=0
        for j in range(num_doc):
            if tf_x[i][j]!=0:
                count+=1
        df_x[i][0]=count
        df_x[i][1]=math.log10(num_doc/count)
```

（4）tf-idf

分别生成文档的标题、作者、内容的tf-idf向量。

tf为log10(N+1)，减少文本长度带来的应影响。

idf调用（3）构建的向量

```
#tf-idf,(文档数，term数)
tf_idf_title=np.zeros((num_doc,len(bag_words_title)))
tf_idf_author=np.zeros((num_doc,len(bag_words_author)))
tf_idf_content=np.zeros((num_doc,len(bag_words_content)))

#生成文档对对应的tf-idf向量
def tf_idf(num_doc,len,tf_idf_x,tf_x,df_x):

    for i in range(num_doc):
        for j in range(len):
            tf_idf_x[i][j]=math.log10(tf_x[j][i]+1)*df_x[j][1]
```

(5) 序列化储存

将生成的向量序列化储存。

```
try:
    os.mkdir("pkl_dir")
except FileExistsError:
    pass

with open('pkl_dir/doc.pkl','wb') as doc:
    pkl.dump((num_doc,doc_id_map),doc)


with open('pkl_dir/term_map.pkl','wb') as tp:
    pkl.dump((title_term_id_map,author_term_id_map,content_term_id_map),tp)

with open('pkl_dir/bag_words.pkl','wb') as bw:
    pkl.dump((bag_words_title,bag_words_author,bag_words_content),bw)

with open('pkl_dir/tf.pkl','wb') as tf:
    pkl.dump((tf_title,tf_author,tf_content),tf)

with open('pkl_dir/df.pkl','wb') as df:
    pkl.dump((df_title,df_author,df_content),df)

with open('pkl_dir/tf_idf.pkl','wb') as tf_idf :
    pkl.dump((tf_idf_title,tf_idf_author,tf_idf_content),tf_idf)
```

二、查询

（1）反序列化、读取存储的向量及相关信息

（2） 输入的处理、生成检索向量的函数

分为两个函数，一个函数为对输入文本的预处理，另一个为根据预处理后的文本，生成查询向量

```
#构建输入检索的向量
#输入的预处理
def pre_input(line):
         #去除停用词
        line=line.replace(';','')
        line=line.replace(",", '')
        line=line.replace('.','')
        line=line.replace(':','')
        line=line.replace('?','')
        line=line.replace('!','')
        line=line.replace("'s",'')
        line=line.replace("'ll",'')
        line=line.replace("'",'')
        #内容预处理，小写，分词
        line=line.strip().lower().split(' ')

        return line
```

```
#构建查询向量
def query_vector(input_query,len,bag_words_x,x_term_id_map,df_x,x_query_vector):
    bag_words=pre_input(input_query)
    #查询的tf
    tf=np.zeros(len)
    for term in bag_words:
        if term in bag_words_x:
            term_id=x_term_id_map._get_id(term)
            tf[term_id]+=1

    #查询的tf-idf
    for i in range(len):
        x_query_vector[i]=math.log10(tf[i]+1)*df_x[i][1]

    return tf_idf
```

（3）查询输入、根据查询生成查询向量

分为标题、作者和内容的输入。根据需要可分别输入查询内容。无输入默认为不进行该部分的查询。

在生成查询独热码的同时，生成查询向量。

```
#输入
title_query=input("please input the title you want to search: ")
author_query=input("please input the author you want to search: ")
content_query=input("please input the content you want to search: ")

#生成输入的检索的向量
title_query_vector=np.zeros(len(bag_words_title))
```

```
author_query_vector=np.zeros(len(bag_words_author))
content_query_vector=np.zeros(len(bag_words_content))

#独热码，有相应查询，设为1，生成对应查询向量
query=[0,0,0]
if title_query!='':
    query[0]=1

 query_vector(title_query,len(bag_words_title),bag_words_title,title_term_id_map
,df_title,title_query_vector)
if author_query!='':
    query[1]=1

 query_vector(author_query,len(bag_words_author),bag_words_author,author_term_id
_map,df_author,author_query_vector)
if content_query!='':
    query[2]=1

 query_vector(content_query,len(bag_words_content),bag_words_content,content_ter
m_id_map,df_content,content_query_vector)
```

（4）计算得分、排序、输出

得分计算的函数：

```
#cos
def cosine_similarity(x,y):
    num = x.dot(y.T)
    denom = np.linalg.norm(x) * np.linalg.norm(y)
    return num / denom

#分数计算
def get_score(len,score,tf_idf_x,query):

    for i in range(len):
        score[i]=cosine_similarity(tf_idf_x[i],query)
```

计算得分。

计算过程：，比如独热码为[0,1,1]，即查询author和content。首先分别author和content的文档得分向量，同时计算权重w_sum为w_author+w_content。最终得分向量为(w_author/w_sum)*score_author*+*(w_content/w_sum)*score_content。

```
#分配权重
w_title=5
w_author=3
w_content=2
w_sum=0

#根据独热码，计算单个域的得分向量，计算对应的权重和
if query[0]==1:
    get_score(num_doc,score_title,tf_idf_title,title_query_vector)
    w_sum+=w_title
if query[1]==1:
    get_score(num_doc,score_author,tf_idf_author,author_query_vector)
```

```
    w_sum+=w_author
if query[2]==1:
    get_score(num_doc,score_content,tf_idf_content,content_query_vector)
    w_sum+=w_content
```

根据权重相加得到查询的文档分数向量

```
#文档分数向量
score=np.zeros(num_doc)
for i in range(num_doc):
    if query[0]==1:
        score[i]+=(w_title/w_sum)*score_title[i]
    if query[1]==1:
        score[i]+=(w_author/w_sum)*score_author[i]
    if query[2]==1:
        score[i]+=(w_content/w_sum)*score_content[i]
```

根据相关性得分排序，去除得分为0文档，其他文档按从大到小排序

```
sorted_score = sorted(enumerate(score), key=lambda score:score[1],reverse=True)

sorted_score_id = [score[0] for score in sorted_score if score[1]!=0]
```

排名输出：根据排序的文档id，查询文档的idmap，打印文件名

```
for i in sorted_score_id:
    str=doc_id_map._get_str(i)
    print(str)
```

附录

id_str_map

利用之前倒排索引的函数：

```
#定义IDMAP，实现id和str的转换
class IdMap:
    """Helper class to store a mapping from strings to ids."""
    def __init__(self):
        self.str_to_id = {}
        self.id_to_str = []

    def __len__(self):
        """Return number of terms stored in the IdMap"""
        return len(self.id_to_str)

    def _get_str(self, i):
        """Returns the string corresponding to a given id (`i`)."""
        ### Begin your code
        return self.id_to_str[i]
```

```
        ### End your code

    def _get_id(self, s):
        """Returns the id corresponding to a string (`s`).
        If `s` is not in the IdMap yet, then assigns a new id and returns the new
id.
        """
        ### Begin your code
        if s not in self.str_to_id:
            self.id_to_str.append(s)
            temp=self.id_to_str.index(s)
            self.str_to_id[s]=temp
            #返回新分配的id
            return temp

        #如果s在，返回已存在的id
        return self.str_to_id[s]
        ### End your code

    def __getitem__(self, key):
        """If `key` is a integer, use _get_str;
           If `key` is a string, use _get_id;"""
        if type(key) is int:
            return self._get_str(key)
        elif type(key) is str:
            return self._get_id(key)
        else:
            raise TypeError
```