

向量空间模型实验报告

学号：2011743

姓名：高祎珂

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。实现带域的查询功能，具体为（诗名、作者、诗句）三个域，要求实现自定义组合域中的查询。如只在歌名中进行检索，或者只在歌名和歌词中进行检索。

实验步骤

为实现带域的查询功能，我们需要在不同的域上建立向量空间模型，对于每个向量空间模型建立的步骤都为：

1. 将每个文档相关域内容存入list，并对其中内容进行处理，如改变大小写，去掉一些标点符号等。
2. 为文本内容建立词袋，并将词项排序
3. 对于每个文本，建立tf（词频），这里使用的是词典类型
4. 接着计算每个的文档频率IDF，将起与上步得到的tf做乘法，得到tf-idf
5. 输入查询，将查询按照通向方法构建，得到查询在对应域中的tf-idf，不同的域去找不同的tf-idf，计算查询向量与不同文本对应向量的预先距离，按照计算距离的大小顺序进行排序，输出不为0的结果

代码展示

读出诗名作者和内容

```
titles=os.listdir('dataset')
doc_paths=[os.path.join('dataset',title)for title in titles]
authors=[]
contexts=[]
for i in range(len(doc_paths)):
    with open(doc_paths[i]) as f:
        # for line in f.readlines:
        authors.append(f.readline())
        contexts.append(f.read())
```

数据预处理

```
# 诗名去掉后面的.txt
for i in range(len(titles)):
    titles[i]=titles[i].strip('.txt')
titles = [x.lower() for x in titles]
for i in range(len(authors)):
    authors[i]=authors[i].strip('\n')
    authors[i]=authors[i].strip('Author: ')
authors = [x.lower() for x in authors]
# 使用string.punctuation去掉符号，因为“'s”这种代词格式是有意义的，所以不删除“ ' ”
remove_punct_map = dict.fromkeys(map(ord, re.sub('\'', '', string.punctuation)))
for i in range(len(contexts)):
    contexts[i]=contexts[i].replace('\n', ' ')
```

```

new_str=""
for word in contexts[i].split(' '):
    new_str+=word.translate(remove_punct_map)
    new_str=new_str+' '
contexts[i]=new_str
# 改变大小写
contexts = [x.lower() for x in contexts]

```

建立词袋，并对词项进行排序

```

# 作者词袋
author_wordset=set(authors[0].split(' '))
for i in range(len(authors)-1):
    author_wordset=set(author_wordset).union(authors[i+1].split(' '))
# 字符串排序函数，返回list
author_wordset=sort_str(author_wordset)

# 标题词袋
title_wordset=set(titles[0].split(' '))
for i in range(len(titles)-1):
    title_wordset=set(title_wordset).union(titles[i+1].split(' '))
title_wordset=sort_str(title_wordset)

# 正文词袋
text_wordset=set(contexts[0].split(' '))
for i in range(len(contexts)-1):
    text_wordset=set(text_wordset).union(contexts[i+1].split(' '))
text_wordset=sort_str(text_wordset)

```

TF-IDF计算

```

#2.统计词项tj在文档Di中出现的次数，也就是词频。
def computeTF(wordSet,area_list):
    tf=[]
    for i in range(len(area_list)):
        tf.append(dict.fromkeys(wordSet, 0))
    # print(tf[0])
    for i in range(len(area_list)):
        split=area_list[i].split(' ')
        for word in split:
            if word in wordSet:
                tf[i][word] += 1
    return tf

```

#3. 计算逆文档频率IDF

```
def computeIDF(tfList):
    idfDict = dict.fromkeys(tfList[0],0) #词为key, 初始值为0
    # print("1",idfDict)
    N = len(tfList) #总文档数量
    for tf in tfList: # 遍历字典中每一篇文章
        for word, count in tf.items(): #遍历当前文章的每一个词
            if count > 0 : #当前遍历的词语在当前遍历到的文章中出现
                idfDict[word] += 1 #包含词项tj的文档的篇数df+1
    for word, Ni in idfDict.items(): #利用公式将df替换为逆文档频率idf
        idfDict[word] = math.log10(N/Ni) #N,Ni均不会为0
    return idfDict #返回逆文档频率IDF字典
```

#4. 计算tf-idf(term frequency-inverse document frequency)

```
def computeTFIDF(tflist, idfs): #tf词频,idf逆文档频率
    tfidf_list = []
    for i in range(len(tflist)):
        tf=tflist[i]
        tfidf_list.append({})
        for word, tfval in tf.items():
            tfidf_list[i][word] = tfval * idfs[word]
    return tfidf_list
```

输入查询，计算余弦相似度

```
def cos_same(query,idf,wordset,tfidf_list):
    result=[]
    if query=="":
        for i in range(len(tfidf_list)):
            result.append(0)
    query=query.strip('\n')
    new_str=""
    for word in query.split(' '):
        word = word.lower()
        new_str+=word.translate(remove_punct_map)
        new_str=new_str+' '
    # 去掉最后一个空格
    new_str=new_str.strip(' ')
    query=new_str
    query_list=[]
    query_list.append(query)
    query=query_list
    # print(query)
    tf_q=computeTF(wordset,query)
    tfidf_q=computeTFIDF(tf_q,idf)
    # print(tfidf_q)
    # 计算余弦相似度，不要分母
    for i in range(len(tfidf_list)):
        temp=0
        for word in tfidf_list[i].keys():
            temp+=tfidf_list[i][word]*tfidf_q[0][word]
        result.append(temp)
    return result
```

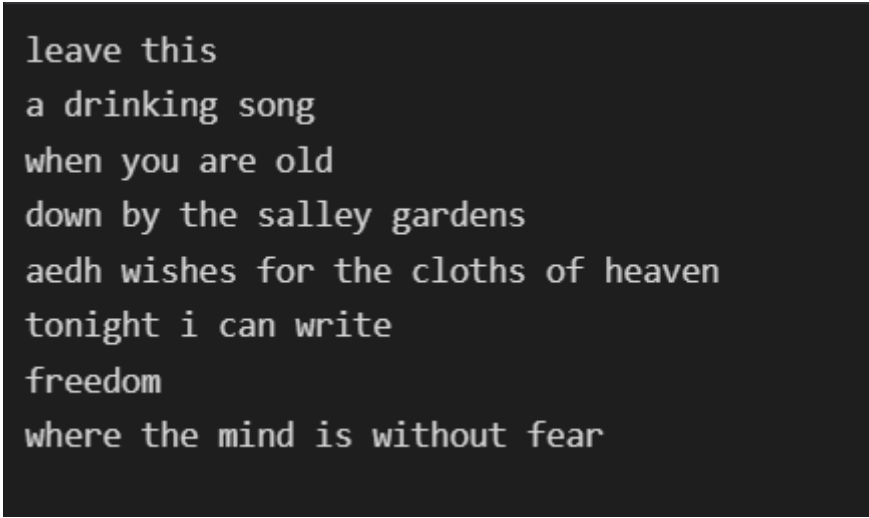
```

# 输入
query_title=input("诗名")
query_author=input("作者")
query_text=input("正文")
# 得到每个域的查询余弦相似度
title_result=cos_same(query_title,idfs_title,title_wordset,tfidf_title)
author_result=cos_same(query_author,idfs_author,author_wordset,tfidf_author)
text_result=cos_same(query_text,idfs_text,text_wordset,tfidf_text)
result=[]
for i in range(len(titles)):
    result.append(title_result[i]+author_result[i]+text_result[i])
zero_id=[]
for i in range(len(result)):
    if result[i]==0:
        zero_id.append(i)

sorted_id = sorted(range(len(result)), key=lambda k: result[k], reverse=True)
# 相似度不为0，则输出
for i in sorted_id:
    if i not in zero_id:
        print(titles[i])

```

结果展示



```

leave this
a drinking song
when you are old
down by the salley gardens
aedh wishes for the cloths of heaven
tonight i can write
freedom
where the mind is without fear

```

这是输入“诗名”为“drinking”，“作者名”为“ahsgd”，“正文”为“and singing”（为《leave this》文章里的内容），可以看到得到了8篇文章，灾后六篇正文里均有出现“and”