向量空间模型实验报告

计算机科学与技术　　2012679　王娇妹

实验内容

1. 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。
2. 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。

重点问题

- 不同域的存储，查询时的组合
  由于需要实现用户自定义域的查询，不同的查询需要的数据可能不一样，所以诗名、作者、诗句三部分不能存到同一个变量中。但还需要根据自定义域将不同部分组合到一起进行查询，诗名、作者和诗句的存储应该是有联系的。
  我定义了一个`Doc类`，有三个成员（string类型）来存储一个文档的不同部分。
  在查询时，根据自定义域，对一个Doc实例中的成员变量进行拼接，存到一个新的字符串中。
- 文档处理
  读取文档后要去掉字符串中的标点符号、回车，统一为小写，还要对词项进行排序。
  借助`string.punctuation`和字符串的`replace()`函数可以替换掉标点符号、回车。
  统一小写使用字符串的`lower()`函数。
  词项的排序需要在确定了查询域之后进行。确定查询域后，将每个文档中查询需要的部分拼接为一个字符串。去重操作可以通过词项列表转换为集合进行，排序使用`sorted()`函数。
- tf 、idf
  tf 是某个词项在某个文档中出现的次数，再进行对数处理。
  通过一个二维数组来保存 tf ，每一行对应一个词项，列数 = 文档数 + 1，第一列存储查询，剩余的存储文档。
  idf 是每个词项在整个文档集合的逆向文件频率。
  遍历 tf 的每一行，可以得到某个词项在所有文档中出现的次数（注意跳过第一列），然后进行对数处理，结果使用一维数组来保存。

流程分析和关键代码

1、读取文档

构建一个Doc类，用来存储某个文档的词项，它有三个成员变量，分别是`poem_name`、`author`和`content`，对应三个域，并将域中的内容以字符串的形式保存。

```
class Doc:
    poem_name = None
    author = None
    content = None
    def __init__(self, poem_name):
        self.poem_name = poem_name
```

- 诗名
  诗名就是文档的名字。
  借助`os.listdir()`函数，顺序读取所有的文档名，存到列表中，并计算文档总数，便于后面索引文档。
  这个列表中的文档名会存到对应`Doc实例`的成员变量`poem_name`中，这些`Doc实例`也都存到一个列表中。

```
if os.path.exists(path):
    # 读取文档名（str类型）到列表，计算文档数量
    file_name_list = os.listdir(path)
    file_num = len(file_name_list)
else: # 路径不存在
    print('Error: this path not exist.')
```

这一步骤读取的文档名带有.txt后缀，通过`切片`去掉。

```
# 用doc列表来保存每个文档的class实例（包括诗名、作者、诗句）
doc = []
# 读文档的诗名（poem_name）
for i in range(file_num):
    doc.append(Doc(file_name_list[i]))
    # remove ".txt"
    doc[i].poem_name = doc[i].poem_name[:-4]
```

- 作者
  作者部分位于txt文档内容的第一行。
- 诗句
  诗句是txt文档除第一行以外的部分。

```
i = 0
for file in os.listdir(path):
    file_path = os.path.join(path, file)
    with open(file_path, 'r') as f:
        # 读取文档的第一行 author部分
        doc[i].author = f.readline()
        # 读取文档的剩余部分 content部分
        lines = f.readlines()
        doc[i].content = str('')
        for line in lines:
            doc[i].content = doc[i].content + line
```

处理文档

读取的内容带有标点符号，大小写不统一，需要对其进行处理。
诗名部分只需要统一小写。作者、诗句部分还需要去掉标点符号，使用字符串的`replace()`函数将其替换为空。

```
# 去掉换行符
doc[i].author = doc[i].author.replace('\n', '')
doc[i].content = doc[i].content.replace('\n', ' ')
# 去掉标点符号
for c in string.punctuation:
   doc[i].content = doc[i].content.replace(c, '')
   doc[i].author = doc[i].author.replace(c, '')
# 单词统一为小写
doc[i].poem_name = doc[i].poem_name.lower()
doc[i].author = doc[i].author.lower()
doc[i].content = doc[i].content.lower()
```

根据组合域汇总文档各部分

通过命令行交互，确定用户选择的查询域。根据查询域，将文档的不同部分拼接为新的字符串。
为防止前一部分的结尾词项跟后一部分的起始词项连在一起，在中间添加一个空格。这可能导致合并后的字符串根据空格分割得到的列表，出现多余的`''`项，可以通过列表的`remove()`操作去除。

```
def select_query_scope(doc, file_num, poem_name_query, author_query, content_query):
   new_doc = []
   for i in range(file_num):
       new_doc.append('')
       if poem_name_query:
           new_doc[i] = new_doc[i] + ' ' + doc[i].poem_name
       if author_query:
           new_doc[i] = new_doc[i] + ' ' + doc[i].author
       if content_query:
           new_doc[i] = new_doc[i] + ' ' + doc[i].content
   return new_doc
```

计算 tf 和 idf

- tf

```
def cal_tf(term, a_doc):
   # term 在 a_doc中出现的次数
   count = 0
   a_doc = a_doc.split(' ')
   for t in a_doc:
       if t == term:
           count += 1
   return math.log(count + 1, 10)
```

- idf

```
# 某个词项出现的文档数量， log（N/idf）
for k in range(terms_num):
   # tf_2D_arr数组中每一行，非零元素的个数，不算第一列的（那是查询的词项）
   idf_arr[k] = math.log(file_num/np.count_nonzero(tf_2D_arr[k][1:]), 10)
```

余弦相似度

对于查询和文档，向量中的每一项用 $tf * idf$ 来表示。

```
# 查询向量
query_vector = np.array([])
for i in range(terms_num):
   query_vector = np.append(query_vector, idf_arr[i] * tf_2D_arr[i][0])
# 文档的查询向量，都存到一个列表中
docs_vector = []
for doc_num in range(file_num):
   doc_vector = np.array([])
   for i in range(terms_num):
       doc_vector = np.append(doc_vector, idf_arr[i] * tf_2D_arr[i][doc_num + 1])
       docs_vector.append(doc_vector)
```

根据余弦相似度的公式来计算。其中`numpy.dot()`函数可以计算两个向量的内积，`numpy.linalg.norm()`函数返回向量的模。

```
# 计算余弦相似度Cosine similarity
docs_cos_sim = []
for i in range(file_num):
  cos_sim = query_vector.dot(docs_vector[i]) / (np.linalg.norm(query_vector) * np.linalg.norm(d
  docs_cos_sim.append(cos_sim)
```

每个文档关于查询的余弦相似度都存在列表`docs_cos_sim`中，使用`max()`函数可以得到最大的余弦相似度，然后遍历`docs_cos_sim`，就可求得余弦相似度最高的文档。