

苏州大学实验报告

院、系	计算机学院	姓名	赵鹏	学号	2127405037
课程名称	信息检索综合实践				
指导教师	李正华	实验完成日期		2022/5/9	

实验名称： 网页排序

一. 实验目的

- 1.学习网页排序相关知识。
- 2.基于向量空间模型对检索出的网页相关性进行排序。

二. 实验内容

- 1.编写代码改造倒排索引，提前处理出向量空间排序所需的 W_{td} 、 idf_t 、 $|d_i|$ ，并在排序时直接利用该数据进行排序。
- 2.输入：已分词或未分词的查询语句
- 3.输出：排序后的查询结果，按如下格式输出到文件：
文件名 1 (余弦相似度 Sim)
句子 1
句子 2

文档名 2
...

三. 解决思路（如流程图或伪代码）、遇到的问题 and 解决方法、运行结果

解决思路：1.改写作业五代码，重新生成倒排索引。

具体思路为：在以此处理每个文档进行最大匹配分词时，统计文档中的每个词在文档中出现的次数。在计数结束后根据词语在文档中出现次数计算 W_{td} 和 $|d_i|$ ，将 $|d_i|$ 存储在以文档名为键，以 $|d_i|$ 为值的字典中，在所有文档处理完成后再计算每个词语的 idf_t 。最后将 $|d_i|$ 以字典的形式输出到索引文件第一行。从第二行开始输出每词语和对应文档及文档对应的 W_{td} 。

改写后的索引文件格式如下：

```
{文档 1:|d1|,文档 2: |d2|, 文档 3:|d3|...}  
词语 1 [ $idf_t$ ,[文档 1,  $W_{td}$ ],[文档 2,  $W_{td}$ ],[文档 3,  $W_{td}$ ]...]  
词语 2 [ $idf_t$ ,[文档 1,  $W_{td}$ ],[文档 2,  $W_{td}$ ],[文档 3,  $W_{td}$ ]...]  
词语 3 [ $idf_t$ ,[文档 1,  $W_{td}$ ],[文档 2,  $W_{td}$ ],[文档 3,  $W_{td}$ ]...]
```

第一行为存储的文档 $|d_i|$,第二行开始为改写后的倒排索引。

下图为改写后的部分倒排文件：

```
['一字入公门，九牛拔不出.txt': 15.322255337674266, '一把抓了两头弗露.txt': 8.697039908804793, '一马难将两靽鞢.txt': 8.2291082934  
字 [0.8664610916297824, ['一字入公门，九牛拔不出.txt', 2.342422680822206], ['三白.txt', 1.0], ['乌白.txt', 1.0], ['伯伦.txt', 1.  
入 [1.6020599913279623, ['一字入公门，九牛拔不出.txt', 1.6020599913279623], ['凹洼.txt', 1.0], ['孤拔.txt', 1.0], ['巍昂.txt', 1.  
公门 [1.6989700043360185, ['一字入公门，九牛拔不出.txt', 1.6020599913279623], ['孤拔.txt', 1.0], ['杆拔.txt', 1.0], ['拓拔.txt',  
九 [1.5228787452803374, ['一字入公门，九牛拔不出.txt', 1.6020599913279623], ['伯箇.txt', 1.0], ['伯嗜.txt', 1.0], ['八关戒.txt',  
牛 [1.2518119729937995, ['一字入公门，九牛拔不出.txt', 1.6989700043360187], ['八乡.txt', 1.0], ['八伯.txt', 1.0], ['八刀.txt', 1.  
拔不出 [1.721246399047171, ['一字入公门，九牛拔不出.txt', 1.6989700043360187], ['孤拔.txt', 1.0], ['杆拔.txt', 1.0], ['拓拔.txt'  
解释 [0.0, ['一字入公门，九牛拔不出.txt', 1.4771212547196624], ['一把抓了两头弗露.txt', 1.4771212547196624], ['一马难将两靽鞢.tx  
词语 [0.0, ['一字入公门，九牛拔不出.txt', 1.4771212547196624], ['一把抓了两头弗露.txt', 1.4771212547196624], ['一马难将两靽鞢.tx  
汉字 [0.0, ['一字入公门，九牛拔不出.txt', 1.0], ['一把抓了两头弗露.txt', 1.0], ['一马难将两靽鞢.txt', 1.0], ['七拱八翘.txt', 1.0  
部首 [0.0, ['一字入公门，九牛拔不出.txt', 1.0], ['一把抓了两头弗露.txt', 1.0], ['一马难将两靽鞢.txt', 1.0], ['七拱八翘.txt', 1.0  
检索 [0.0, ['一字入公门，九牛拔不出.txt', 1.3010299956639813], ['一把抓了两头弗露.txt', 1.3010299956639813], ['一马难将两靽鞢.tx  
拼音 [0.0, ['一字入公门，九牛拔不出.txt', 1.3010299956639813], ['一把抓了两头弗露.txt', 1.3010299956639813], ['一马难将两靽鞢.tx  
俗谚 [2.6989700043360183, ['一字入公门，九牛拔不出.txt', 1.0], ['一把抓了两头弗露.txt', 1.0]]  
谓 [1.080921907623926, ['一字入公门，九牛拔不出.txt', 1.4771212547196624], ['一把抓了两头弗露.txt', 1.0], ['三白.txt', 1.0], ['一  
一张 [2.6989700043360183, ['一字入公门，九牛拔不出.txt', 1.0], ['一器张.txt', 1.0]]  
状纸 [2.9999999999999996, ['一字入公门，九牛拔不出.txt', 1.0]]  
送进 [2.9999999999999996, ['一字入公门，九牛拔不出.txt', 1.0]]  
衙门 [2.397940008672037, ['一字入公门，九牛拔不出.txt', 1.0], ['六案孔目.txt', 1.0], ['柏国.txt', 1.0], ['霸门.txt', 1.301029995  
便 [1.2924298239020635, ['一字入公门，九牛拔不出.txt', 1.0], ['万安.txt', 1.0], ['刹把.txt', 1.0], ['反把.txt', 1.0], ['奶扁.txt  
身 [1.494850021680094, ['一字入公门，九牛拔不出.txt', 1.0], ['不嘎.txt', 1.0], ['八识.txt', 1.0], ['凹答.txt', 1.0], ['呵辱.txt'  
遭 [1.853871964321762, ['一字入公门，九牛拔不出.txt', 1.0], ['乐呵.txt', 1.0], ['八屯.txt', 1.0], ['呵卫.txt', 1.0], ['呵怒.txt'  
讼 [2.045757490560675, ['一字入公门，九牛拔不出.txt', 1.0], ['器张.txt', 1.0], ['器阔.txt', 1.0], ['氛器.txt', 1.0], ['腰板.txt'  
累 [1.455931955649724, ['一字入公门，九牛拔不出.txt', 1.0], ['伯嗜.txt', 1.0], ['滞暗.txt', 1.0], ['熬锅.txt', 1.0], ['百兽.txt'  
无从 [2.5228787452803374, ['一字入公门，九牛拔不出.txt', 1.0], ['无巴壁.txt', 1.0], ['无霸.txt', 1.0]]
```

2.读取倒排文件并转换为多种字典。

具体方法为：1.读取第一行转化为文件名-> $|d_i|$ 的字典。

2.读取第二行开始的内容建立词语-> idf_t 、[文档名, W_{td}]的字典。

3.建立词语->文件名->列表索引的嵌套字典。在已知词语及文档后快速定位到对应列表的索引，直接获取对应 W_{td} ，提高检索速度。

3.输入语句，进行查询。

具体方法为：1.利用字符串 `split` 函数判断是否语句已经完成分词，如果未分词则利用最大匹配分词算法进行分词，并返回分词后的列表。

2.对于每个词语，获取包含该词语的对应文档并加入集合中，作为待排序文档。

3.依次处理每个文档，通过字典快速获取词语列表中每个词在文档中的 W_{td} 及 idf_t ，对于未出现的在字典键中的词语进行特殊处理，将对应的 W_{td} 和 idf_t 赋值为0。将结果存储在两个列表中，随后计算余弦值。两个列表中对对应位置相乘求和计算 \cos 的分子，通过提前建立的字典直接获取 $|d_i|$ ，对每个词语的 idf_t 平方求和再相加，最后开方并与 $|d_i|$ 相乘作为 \cos 的分母。返回分子/分母，最终的结果以[文件名,cos]格式存储在列表中列表中。

4.获取最终结果，根据 \cos 值从大到小进行排序。

5.输出结果，依次打开每个文件，遍历文件的每一行、如果一行内包含分词列表中的词，则输出该行。将所有查询结果输出到以查询语句命名的 txt 文件中。

输出格式如下：

文件名 1 (余弦相似度 Sim)

句子 1

句子 2

文档名 2 (余弦相似度 Sim)

句子 1

句子 2

...

运行结果：



问题 1: 根据单词在文档中出现的次数计算 W_{td} 时, 如果直接使用 `count` 函数计数在一些情况下会造成结果偏大的情况。例如: 在对“大学里由很多大学生”这句话统计“大学”的出现次数时, 使用 `count` 会把“大学生”里的“大学”也计算为一次。

解决方法: 在分词时对每个分词的词语进行计数, 在文档分词完毕后统一计算文档中每个词语的 W_{td}

问题 2: 在建立新的词语->[文件名, W_{td}] 字典后, 无法快速获取词语对应的文件名的 W_{td} , 如果遍历一遍会浪费大量时间。

解决方法: 建立一个能够由词语和文件名快速定位到 W_{td} 的字典, 字典中存储列表的索引。

问题 3: 在一些特殊情况下(查询词在所有文档里都出现并且不在倒排里)会出现 $|q|=0$ 的情况, 在这种情况下如果直接计算余弦值会出现除 0 的错误。

解决方法: 计算出分子分母后对分母进行一次特判, 如果出现分母为 0 则直接返回 0 作为这个文档的 \cos 值。

问题 4: 经过与同学的计算结果比对, 发现在计算 W_{td} 和 idf_t 过程中存在浮点数计算误差。

解决方法: 由于误差及其微小(例如 2.9999999999996 和 3.0), 可以直接忽略误差。

四. 实验总结

通过本次实验,学习到了网页排序的基本知识及基于向量空间模型的网页排序方法,通过改写倒排、编写代码实现了对给定语句进行分词查询,并根据向量空间模型的 \cos 值大小对查询结果进行排序。提升了自己的程序设计能力。