

武汉大学国家网络安全学院

实验报告

课程名称 信 息 检 索

专业年级 网 络 空 间 安 全

姓 名 杨 楚 秦

学 号 **2017301040186**

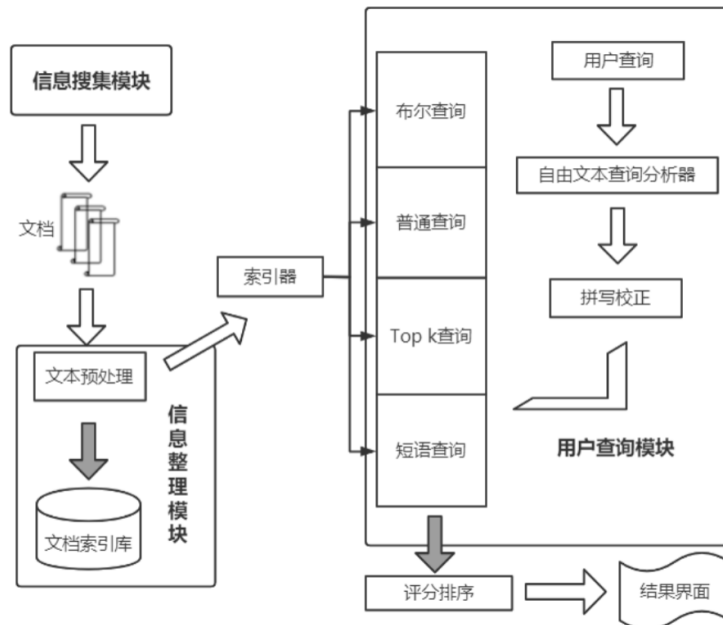
协 作 者 无

实验学期 **2020-2021** 学年 第二 学期

课堂时数 **32** 课外时数

填写时间 **2021** 年 **6** 月 **20** 日

实验介绍
<p>【实验名称】： 信息检索搜索引擎实现</p> <p>【实验目的】：</p> <p>独立编写一个命令行搜索引擎，对自由选择的文档集建立索引，实现信息检索（语言不限、功能不限）。</p> <p>基于倒排索引和向量空间模型实现具有以下功能的搜索引擎：</p> <ol style="list-style-type: none"> ①拼写矫正（判断检索关键词拼写是否正确，返回矫正后的单词） ②BOOL 查询（接受 a and/or (not) b 形式的布尔表达式搜索查询） ③普通查询（将所有查询结果按 wf-idf 排序输出） ④TOP K 查询（输出 K 个离查询最近的文档） ⑤短语查询（查询一个完整的短语/句子，精确查找） <p>【实验环境】：</p> <p>电脑配置：Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz (2712 MHz)，8G 内存</p> <p>操作系统：Microsoft Windows 10 家庭中文版（64 位）</p> <p>开发环境：python3.7.4、Pycharm2020.2.3 x64</p> <p>【参考文献】：</p> <p>【1】 https://blog.csdn.net/huanglong0438/article/details/51620498</p> <p>【2】 https://blog.csdn.net/fengjiancangyue/article/details/43746651?utm_medium=distribute.pc_relevant.none-task-blog-2%Edefault%7EBlogCommendFromBaidu%7Edefault-6.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%Edefault%7EBlogCommendFromBaidu%7Edefault-6.control</p> <p>【3】 https://github.com/9ayhub/nlp-search-engine</p> <p>【4】 http://norvig.com/spell-correct.html</p> <p>【5】 https://zhuanlan.zhihu.com/p/124885051</p> <p>【6】 https://blog.csdn.net/asialeebird/article/details/81486700</p> <p>【7】 https://nullcc.github.io/2017/05/18/%E5%80%92%E6%8E%92%E7%B4%A2%E5%BC%95%E5%9F%BA%E7%A1%80%E7%9F%A5%E8%AF%86/</p> <p>【8】 https://blog.csdn.net/weixin_29079671/article/details/111982039?spm=1001.2014.3001.5502</p>
实验内容
<p>【实验方案设计】：</p> <p>搜索引擎是“对网络信息资源进行搜集整理并提供信息查询服务的系统，包括信息搜集、信息整理和用户查询三部分”，本项目中由于自定义了信息检索的数据集，因此，信息搜集模块较略，着重信息处理模块和用户查询模块的设计和实现。</p>



信息整理模块对采集的信息进行分词、词形还原、赋权重等操作后建立索引表（倒排索引）构成索引库；搜索引擎识别用户的检索需求并提供检索服务，用户通过查询模块的四种查询方式来检索自己想要的信息。

一、系统功能设计

基于倒排索引和向量空间模型实现具有以下功能的搜索引擎：

- ①拼写矫正（判断检索关键词拼写是否正确，返回矫正后的单词）
- ②BOOL 查询（接受 a and/or (not) b 形式的布尔表达式搜索查询）
- ③普通查询（将所有查询结果按 wf-idf 排序输出）
- ④TOP K 查询（输出 K 个离查询最近的文档）
- ⑤短语查询（查询一个完整的短语/句子，精确查找）

二、数据集选择

Reuters - 21578 数据集

数据集下载链接：<http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html>

Reuters - 21578 新闻数据集是一个知名的英文标准文本集合，属于多类、多标签数据集。本项目采用 Reuters - 21578 下用 ModApte 划分方法划分的一个子集作为基准数据集，数据集的大小为 10788 篇文档，包括 90 类、7769 个训练文件以及 3019 个测试文件。

三、数据处理模块

（1）文本预处理（词形还原）

读取数据集文档，给每个文档对应一个 ID，针对每篇文档进行下面的操作：

利用 NLP 库 nltk，将文章按句子进行分割，然后对句子进行分词，然后进行词形还原。

词形还原 (Lemmatization) 就是去掉单词的词缀，提取单词的主干部分，通常提取后的单词会是字典中的单词。比如 cars 词形还原后的单词为 car，ate 词形还原后的单词为 eat。使用词形还原可以减少词语因为时态、单复数和变形等对于处理精度的影响，缩短检索时间。

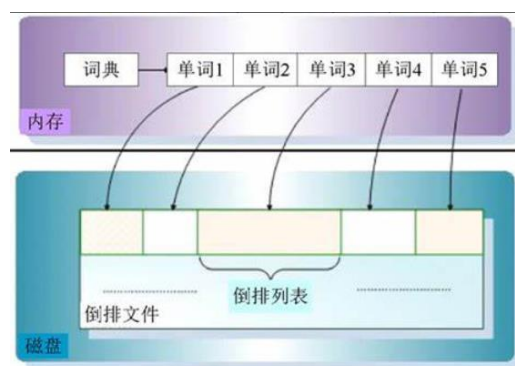
具体操作时，首先获取单词在句子中的词性，这里规定为四种（名词 NOUN、动词

VERB、形容词 ADJ、副词 ADV），使用 Parts of speech (POS) 技术实现。然后使用 nltk 模块中的 WordNet 中的函数进行词形还原，再从结果中筛选、去除不关注信息（标点符号、空字符串等），得到预处理后的每篇文档的单词列表。

(2) 建立索引

采用倒排索引的方法，建立某个单词在一组文档中的存储位置的映射。在前面，已经实现了对所有文本扫描获得单词 token，并且通过词形还原将单词变为相应的词根 term。接下来需要将词根建立为字典 dictionary，通过倒排索引建立每个 term 的文档 id 列表（倒排列表）。

对于一个搜索引擎来说，最重要的部分应该就是索引的建立，一个索引建立的好坏与否直接决定了后面搜索部分的速度和准确度。倒排索引是一种索引方法，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。通过倒排索引，可以根据单词快速获取包含这个单词的文档列表。倒排索引主要由单词词典 (Term Dictionary) 和倒排列表 (Posting List) 及倒排文件 (Inverted File) 组成。



具体操作时，首先建立两个字典 docList 和 invertedIndex，还需要引入数据集中的每篇文档对应的 ID 列表 docId。循环读取每篇文档的词根 term 列表，按照 0、1、2……形式标记每个词在文档中的位置。docList 字典以文档号为键，词根所在位置为值，反映了词根 term 在文档中的具体位置。invertedIndex 字典以词根字符串内容为键，词根的 docList 字典为值，将词根的位置信息和词根本身内容对应起来。invertedIndex 字典就是带位置信息的倒排索引，例如：

```
1 {
2   "eat":{
3     "1":[5,6,10],
4     "5":[10,20,30]
5   },
6   ...
7 }说明eat这个单词在文档1的第5、6、10位置上出现，在文档5的第10、20、30位置上出现。
```

接着要对原始形成的 invertedIndex 字典做排序处理，按照键的内容，也就是单词内容进行排序。若不排序，那么 term 排列是混乱的，检索时找出某个特定的 term 的速度很慢，因为需要全部过滤一遍才能找出特定的 term。使用 python 的 sorted 函数将单词按首字母顺序排序。

将上述处理后的倒排索引数据储存到 invertIndex.json 文件中，并且单独将 key 部分提取出来得到单词词典 wordList，储存到 wordList.json 文件中。之后每次运行程序时，需将倒排索引和单词词典加载到内存中。

(3) 构建向量空间模型

向量空间模型 VSM 是把对文本转换为空间向量，通过向量的计算来处理文本内容的模型。由于本项目的搜索引擎不仅仅是支持布尔查询这种命中式的查询，还希望能够考虑单词的出现频数、文档的频数、文档长度等综合起来的统计特征对排序结果的影响，因此需要构建向量空间模型，通过查询向量计算出文档的 wf-idf 评分进行结果的排序。这里的 wf-idf 评分方法是从 tf-idf 变化而来的，对于 tf-idf：

TF (Term Frequency) 单词频率，表示词条（关键字）在文本中出现的频率。

IDF (Inverse Document Frequency) 逆向文件频率，某一特定词语的 IDF 由总文件数目除以包含该词语的文件的数目，再将得到的商取对数得到。

*TF-IDF 实际上就是 $TF * IDF$ ，以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。*

wf-idf 的计算方法与 tf-idf 类似，wf-idf 实际上是 wf*idf，wf 相比于 tf，多了一步 log 计算的操作，目的为不让 tf 线性增长，削弱词项频率对评分的影响。用 $1+\log(tf)$ 这个值 wf 来代替原来的 tf 取值。wf-idf 评分能够在一篇文章中单词出现 n 次时，使其权重并不扩大 n 倍，更符合统计特征时的考量。

wf-idf 计算方法：

- 首先对 query 中出现的单词对应的文档列表取并集。
- 随后对 query 中出现的单词对文档进行 wf-idf 计算并评分。
- 得到所有文档对该查询的评分后再对所有文档进行排序。

有了向量空间模型和 wf-idf 的评分机制，通过向量运算，在进行如 TOP K 一类需要对搜索结果进行排序的查询时，就能把最满足用户需求的文档放在最上面显示给用户了。

(4) 拼写矫正

拼写校正模块在搜索引擎进行搜索前，都会运行。

这一部分参考了流行的 Peter Norvig 拼写检查器的思想和代码。任务是对给定的单词，判断是否拼写正确，若单词本身拼写是正确的，则无需矫正，若不正确，则选择和它最相似的拼写正确的单词。采取概率的思路，在错误拼写 w 出现的条件下，选择所有可能的备选纠正单词 c 中概率最大的作为结果。

原文链接：<http://norvig.com/spell-correct.html>

数据集链接<http://norvig.com/big.txt>

预处理部分，利用 words() 函数把语料中的单词全部抽取出来，转成小写，并且去除单词中间的特殊符号（don't 变成 don 和 t）。

假设 c 是我们猜测用户可能想要输入的单词，而 w 是用户实际输入的错误单词，使用最小编辑距离的概念为给定单词 w 枚举所有可能正确的拼写。在函数 edits1() 中返回所有与单词 w 最小编辑距离为 1 的集合，在函数 edits2() 中返回所有与单词 w 最小编辑距离为 2 的集合。再通过函数 candidates() 中计算最小编辑距离获取最终的一个非空的拼写建议候选集 (candidate w)。

两个字符串间最小的编辑距离就是将其中一个变成另外一个时需要的最小的编辑操作次数（操作包含插入、删除、替代三种）

根据贝叶斯公式 $P(c|w)=P(w|c) * P(c) / P(w)$ ，需得到使 $P(c|w)$ 最大的 c。

- c 是编辑距离为 1 或者 2 或者 0 的词，并且词属于给定的词典。
- $P(w)$ 是每一种拼写错误可能出现的概率，对于每个单词都一样，这里认为是一个常数。

- $P(c)$ 代表了在字典中 c 出现的概率，称为语言模型，通过读入一个巨大的文本文件 big.txt 作为语料库，计算文章中出现一个正确拼写词 c 的概率 $P(c)$ 。
- $P(w|c)$ 是误差模型，即用户想要输入 w 却输入 c 的概率，需要大量样本数据和事实依据得到，简单起见用编辑距离为 1 的正确单词比编辑距离为 2 的优先级高，而编辑距离为 0 的正确单词优先级比编辑距离为 1 的高 ($P(W|C0) \gg P(W|C1) \gg P(W|C2)$) 的思路来设置。

最后，模型工作流程为：①如果输入词在词典里找到了原词，则返回；②从编辑距离为 1 的词选出频率最高的，返回；③从编辑距离为 2 的词选出频率最高的，返回。

四、用户查询模块

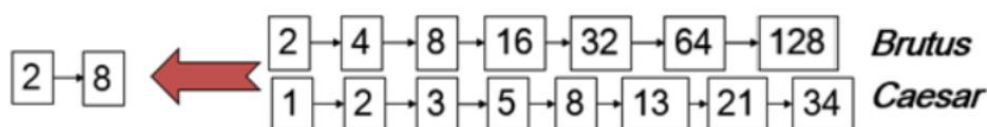
(1) BOOL 查询

布尔查询是通过 AND、OR、NOT 等逻辑操作符将检索词连接起来的查询。在本项目中，针对单个 AND、OR、NOT (A and B 、 A or B 、 A not B) 三种布尔查询进行了实现，也能处理这三种操作符连接起来的较长且复杂的查询表达式 ($(A$ and $B)$ or C 等)，但是无法处理 NOT 后再接 bool 表达式的情况。

首先对输入的搜索关键词，进行预处理、词形还原、拼写校正等工作，然后对输入的查询命令进行如下处理：

将查询表达式转为后序表达式，如“ A OR B AND C ” 转为“ A B C AND OR”，这个过程中借助栈来完成。有了后序表达式，依次对一个逻辑操作符和它的两个操作对象进行操作，以 A AND B 为例，首先在单词词典中定位 A 和 B ，得到两张倒排记录表。每个倒排记录表都有一个定位指针，两个指针同时从前往后扫描，每次比较当前指针对应倒排记录，然后移动某个或两个指针，将倒排记录表求交集。得到交集后将新倒排索引表中的值 docList 字典中的键——文档号 docId 存入结果列表中，进行输出。不同的逻辑操作符在求倒排记录表时的操作不同，如下。

- AND，求两个倒排记录表的交集
- OR，求两个倒排记录表的并集
- NOT，求两个倒排记录表的减



(2) 普通查询

普通查询是用一种评分机制对搜索结果进行排序后，按顺序返回文档，不再进行其他操作的查询方式。输入可以是一个/几个关键词或者句子。

对于搜索的输入，首先进行预处理、词形还原、拼写校正、单词去重等工作，然后对于搜索的关键词（一个或多个），首先在倒排索引中寻找键值对，返回其所在文档的 id，并且把所有出现的 id 号进行一个排序。再将所有 word 的查询结果取并，即在一个 list 里面存储所有包含至少一个关键词 word 的文档号 ID。

然后针对这些文档，通过向量空间模型得到所有文档中单词的 wf-idf 评分，以[分数，文档排号]的方式存储在一个列表中，对此列表按照分数的高低用 python 的 sorted 函数进行排序，返回所有符合查询要求的文档的 id 和 wf-idf 分数。下面是一次普通查询的部分返回结果。


```

1 [1]BOOL [2]Overall [3]TOP K [4]Phrase [5]wildcard [6]exit
2 your choice(int):
3 2//选择普通查询
4 input the query statement:
5 happy ending//输入的查询内容
6 stemming...
7 ['happy', 'endding']
8 spelling correcting...
9 ['happy', 'ending']//可见endding输错了, 拼写校正将endding改为ending
10 doc ID: 21565 score: 2.6350//wf-idf值作为score, 21565和21006是返回的文档
11 doc ID: 21006 score: 2.6350

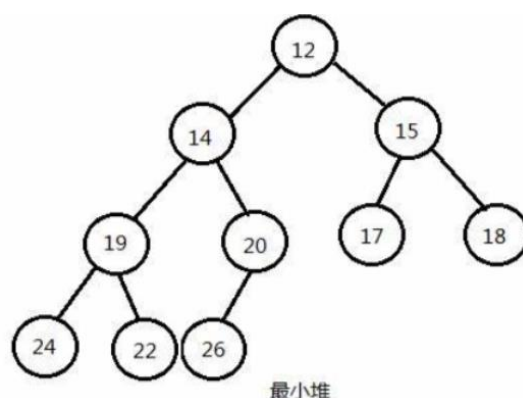
```

(3) TOP K 查询

TOP K 查询是用一种评分机制对搜索结果进行排序, 通过得分的高低将排名前 top K 的文档返回给用户。在本项目中, 将 K 设置为 20, 即返回 20 个最符合搜索条件的文档号。TOP K 查询一开始的处理和普通查询是相同的, 得到一个存储了所有的包含至少一个关键词 word 的文档号 ID 列表 List。

针对这些文档, 也是通过向量空间模型得到所有文档中单词的 wf-idf 评分, 以[分数, 文档排号]的方式存储在一个列表中。不同的是, YOP K 查询返回时文档数有限。由于搜索引擎是非常大规模的检索系统, 因此排序的选择十分重要, 利用堆排序按照 wf-idf 评分的高低对结果进行排序。采用时间复杂度为 $O(n\log k)$ 的最小堆方法, 通过堆排序建好最小堆以后进行 precDown 操作, 堆最后的保留的所有元素即为评分前 K 大的元素, 并返回其文档 id。方法步骤为:

- 取出数组的前 k 个元素, 创建长度为 k 的最小堆。
- 从 k 开始循环数组的剩余元素, 如果元素(a)比最小堆的根节点大, 将 a 设置成最小堆的根节点, 并让堆保持最小堆的特性, 否则跳过。
- 循环完成后, 最小堆中的所有元素就是需要找的最小的 k 个元素 (最符合条件的 k 个文档)。



(4) 短语查询

短语查询是在普通查询的基础上实现的, 更加精准的查询方式。由于倒排索引中保存的都是独立的词, 当查询请求是一个短语时, 普通查询实际上仍然是在索引中检索组成短语的每个词。例如在普通查询时 “World War ” 会被拆分成 “world” 和 “war” 分别进行查询并合并结果, 显然这样得到的效果并不理想, 有许多无关的信息。

短语查询“Canadian farmers”时，下左图中的文档属于检索结果，下右图则不属于。

In a report to the Canadian finance department, the tribunal said the duty is hurting Canadian farmers and food processors. The duty was imposed last year after the revenue department found U.S. corn production was subsidized.	Beet supplies have dried up at Ipswich due to a combination of very wet weather, which has prevented most farmers in the Canadian factory's catchment area from harvesting, and last week's hurricane which blocked roads.
--	--

因此需要进行短语查询，具体地，仍然采用合并算法查找符合的文档，但是，不再仅简单判断两个词是否出现在同一个文档中，还需要检查它们出现的位置情况。只有当所有的词在一篇文档中的出现顺序为前后相继时，才算是符合查询。具体步骤如下：

- 利用带位置信息的倒排索引将组成 query 短语的每个词的倒排表取交集，得到文档候选集。
- 从这些文档集中根据位置索引查找是否有匹配的短语。
- 遍历第一个词项在文档中的位置，依次检测后面的词项位置中是否包含与其匹配的位置。
- 如果 query 中前后的单词在一篇文档中的位置序号为加一的关系，那么两者是一个短语形式，反之，这篇文档不符合条件，放弃。

最后按照 query 短语出现在文档中的“标号 ID、总共出现的次数、每次出现首单词在文档中的位置”的形式输出。

五、核心算法模块说明

在初次运行程序前需下载词干还原依赖的语料库（在 SearchSystem/main.py 中已经注释掉了）下载语料库的命令为：

```
1 nltk.download("wordnet")
2 nltk.download("averaged_perceptron_tagger")
3 nltk.download("punkt")
4 nltk.download("maxnet_treebank_pos_tagger")
```

取消注释后运行一次即可，语料库下载完成即可正常运行

若不想下载，可以直接将代码文件夹目录下的 nltk_data 文件夹（已经下载好的语料库）替换掉 user 下的 AppData/Roaming/nltk_data 文件夹。

在 SearchSystem 目录下运行命令 python main.py 即可。

(1) 词形还原

先对 query 进行分词、词性标注，再进行词性还原，使用 NLTK 当前推荐的词性标记器 pos_tag 标记给定的标记列表。

使用一个高级的数据集 wordnet 中提供的稳健的词形还原的函数进行词形还原。首先引入 wordnet 包，使用 nltk.pos_tag() 获取单词在句子中的词性。pos 值可以为 NOUN、VERB、ADJ、ADV。调用 wordnet.lemmatize() 函数进行词形还原，第一个参数为单词，第二个参数为该单词的词性，lemmatize 接受词性参数 pos 如名词、动词、形容词等，如果没有提供，默认是名词，即它将尝试找到最接近的名词，返回的结果为输入单词的词形还原后的结果。

再从结果中筛选、去除不关注信息（标点符号、空字符串等），得到预处理后的每篇文档的单词列表。

(2) 拼写校正

words() 函数把语料中的单词全部抽取出来，转成小写，并且去除单词中间的特殊符号，函数 P() 用以统计每个词在 big.txt 语料库字典中出现的频率，函数 edits1() 返回所有

与单词 w 最小编辑距离为 1 的集合, 函数 `edits2()` 返回所有与单词 w 最小编辑距离为 2 的集合, 函数 `candidates()` 计算最小编辑距离获取最终的一个非空的拼写建议候选集 (candidate w), 函数 `correction()` 返回与错误拼写最相似的拼写正确的单词。

(3) 布尔查询

根据逻辑操作符 AND、OR、NOT 的不同, 分别用 `andTwoList()`、`mergeTwoList()`、`listNotcontain()` 函数来实现倒排表 `list1` 和 `list2` 的求交集、并集和相减的操作。以 `andTwoList()` 函数为例, 有两个参数, 倒排表 1 和倒排表 2。

`rlist` 准备放倒排表结果, 初始化为 0。用 `n1` 和 `n2` 为倒排表 1 和 2 记录当前读到的文档编号, `n1`、`n2` 分别小于 `len1` 和 `len2` 时继续循环,

如果 `n1` 为索引表示的文档编号与 `n2` 为索引表示的文档编号相同, 则说明该篇文档是我们要的交集结果, 把结果添加到 `rlist`, `n1`、`n2` 加一。若 `n1` 为索引表示的文档编号与 `n2` 为索引表示的文档编号不相同, 且 `n1` 表示的文档编号小于 `n2` 表示的文档编号, 则说明要放弃 `n1` 对应的这篇文档, `n1` 加一, 接着做比较。反之 `n2` 加一……处理完后, 将 `rlist` 作为结果返回。

(4) TOP K 最小堆排序

① 先将初始的 $R[0 \cdots n-1]$ 建立成最小堆, 此时是无序堆, 而堆顶是最小元素。

② 再将堆顶 $R[0]$ 和无序区的最后一个记录 $R[n-1]$ 交换, 由此得到新的无序区 $R[0 \cdots n-2]$ 和有序区 $R[n-1]$, 且满足 $R[0 \cdots n-2].keys \geq R[n-1].key$

③ 由于交换后新的根 $R[1]$ 可能违反堆性质, 故应将当前无序区 $R[1 \cdots n-1]$ 调整为堆。然后再次将 $R[1 \cdots n-1]$ 中关键字最小的记录 $R[1]$ 和该区间的最后一个记录 $R[n-1]$ 交换, 由此得到新的无序区 $R[1 \cdots n-2]$ 和有序区 $R[n-1 \cdots n]$, 且仍满足关系 $R[1 \cdots n-2].keys \geq R[n-1 \cdots n].keys$, 同样要将 $R[1 \cdots n-2]$ 调整为堆。

④ 直到无序区只有一个元素为止。

通过堆排序建好最小堆以后进行 K 次 `precDown` 操作, 堆最后的 K 个元素即为评分前 K 大的元素, 并返回其文档 `id`。

(5) main 整体框架代码

用 LOOP 标志保证查询可以一直进行, 除非输入“退出”对应的数值 5。按照本项目的用户查询模式, 设置 1、2、3、4 四个数值分别对应布尔查询、普通查询、Top k 查询、短语查询四种查询模式。对于所有的查询, 词形还原和拼写矫正都是必不可少的。然后进入到各个分支中, 调取各自的自定义函数库来完成具体功能。布尔查询返回所有满足布尔表达式的文档标号; 普通查询按照 `wf-idf` 得分的降序排列返回文档号和得分; Top k 查询返回 20 个最符合检索要求的文档标号、`wf-idf` 得分和简要的文本内容; 短语查询按照 `query` 出现在文档中的“标号 ID、总共出现的次数、每次出现首单词在文档中的位置”的形式返回结果。

【实验结果分析】:

一、常用评价指标

对信息检索系统的效果进行评价, 可以从无序和有序两个维度考虑。本项目运行在一个大小为 10788 篇文档的数据集之上。无论是无序指标还是有序指标, 都需要提前给数据集进行打标, 例如: 对单词 A 标记每篇文档其是否出现、出现的次数、所在的位置等信息。本

项目的数据集并没有进行标注，因此没有办法从统计的角度给出评价。因此这里只是将评价的思路和方法给出。

(1) 无序评价指标

当考虑地相对简单的时候，在搜索结果中仅考虑返回的文档是否与查询相关，而不考虑这些返回文档在结果列表中的相对位置和顺序。最常用的三个指标为 P 准确率、R 召回率、F1 值。

- 准确率 (Precision) 是返回的结果中相关文档所占的比例
- 召回率 (Recall) 是返回的相关文档占有所有相关文档的比例
- F1 值是准确率和召回率的调和平均值

(2) 有序评价指标

搜索引擎返回的结果通常是有序的，因此有必要对无序评价指标进行扩展以考虑位置信息。

平均准确率 MAP (Mean Average Precision) 是近年来比较流行的评价指标，MAP 在准确率的基础上考虑了位置的因素。首先对于单个关键词的查询，其平均准确率的具体计算方法如下：

$$MAP = \frac{1}{r} \sum_{i=1}^r \frac{i}{\text{第 } i \text{ 个文档相关文档的位置}}$$

其中 r 是相关文档的总数，查询集合的平均准确率由所有单个查询的 MAP 值的平均。

总的来说，系统检索出来的相关文档在列表中越靠前，MAP 的值就越高。如果系统没有返回相关文档，则 MAP=0。

二、实际结果分析

尽管没有办法对全部的数据集进行统计评价，但是对本信息检索系统进行功能测试还是可以进行的。下面将分功能模块进行测试，分析结果。

(1) 词形还原

在代码中将输入的查询语句的词形还原结果输出（为了有代表性，分别测试了不同的单词变形形式），可以看见，四个单词都正确地被还原成了词根，词形还原模块功能完成。

```
input the query statement:
is has running fishes
stemming...
['be', 'have', 'run', 'fish']
```

(2) 拼写校正

在代码中将输入的查询语句的拼写校正结果输出（为了有代表性，分别测试了不同的单词写错的情况），可以看见，未写全的“lik”和打多了一个字母的“running”都正确地还原成了最可能的正确的单词“like”和“run”，拼写校正模块功能完成。

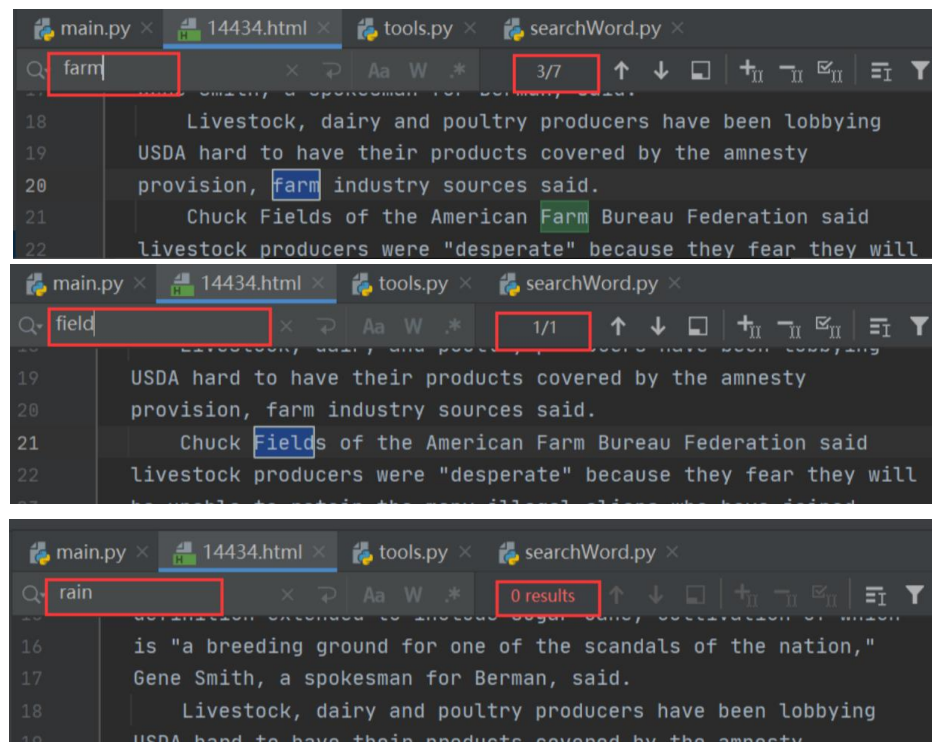
```
input the query statement:
I lik running
stemming...
['i', 'lik', 'run']
spelling correcting...
['i', 'like', 'run']
```

(3) 布尔查询

运行时，在命令行中输入 1 则可进入布尔查询模式，输入查询指令“farm AND feild AND NOT rain”，来查找有农场和土地的关键词，同时不要有雨单词的文档。得到一篇检索结果，文档的 ID 为 14434。为了验证结果，手动去 14434 文档中查询 farm、feild、rain，发现确实是有前两个单词而没有第三个。说明布尔查询功能成功实现了。在检验的时候我发现，数据集的规模还是不够大，常常会出现 0 DOCs 的情况，这也是因为布尔模型是完全匹配的，不支持部分匹配，因此会导致太多或太少的结果被召回。

```
searching operation:
[1]BOOL [2]Overall [3]TOP K [4]Phrase [5]exit
your choice(int):
1
input the query statement:
farm AND feild AND NOT rain
stemming...
['farm', 'AND', 'feild', 'AND', 'NOT', 'rain']
spelling correcting...
['farm', 'AND', 'field', 'AND', 'NOT', 'rain']
1 DOCs :
[14434]
```

布尔查询
按照有“农场”和“土地”但是没有“雨”关键词进行查找
找到一篇符合条件



(4) 普通查询

运行时，在命令行中输入 2 则可进入普通查询模式，直接输入查询关键词“female”，得到五篇检索结果，文档的 ID 分别为 19986、13738、11160、8713 和 867，并且 wf-idf 分数越高，排列越靠前。

```

searching operation:
[1]BOOL [2]Overall [3]TOP K [4]Phrase [5]exit
your choice(int):
2 → 普通查询
input the query statement:
female
stemming...
['female']
spelling correcting...
['female']
doc ID: 19986 score: 4.3376
doc ID: 13738 score: 3.3340
doc ID: 11160 score: 3.3340 → 检索到五篇文档包含female关键词
doc ID: 8713 score: 3.3340
doc ID: 867 score: 3.3340

```

由于在对数据集进行单词词典和倒排索引建立时,已经得到了 invertIndex.json 文件,为了验证查询结果,在文件中搜索“female”,找到了它的倒排索引的信息。可以看到该单词所对应的键值对信息中确实是查询到的那 5 篇文档,并且 19986 标号的文档中该单词在 154 和 177 的标记位置上出现了两次,再结合文档的长度进行判断,其 wf-idf 的值的的确是最高的。可见,普通查询功能实现了。

```

{
  "21508": [797]},
  "feltex": {"17876": [223]},
  "female": {"867": [106], "8713": [51], "11160": [106], "13738": [130], "19986": [154, 177]},
  "fen": {"16147": [70, 77]},
  "fence": {"6086": [356], "15798": [219], "18604": [201]},
  "fend": {"3864": [337], "6665": [584], "9582": [388], "12827": [21], "13966": [217], "14504": [83], "16252": [128], "16483": [197], "17849": [239], "19411": [106]},
  "fendt": {"5684": [44, 103, 172, 201, 268]},

```

(5) Top k 查询

运行时,在命令行中输入 3 则可进入 Top k 查询模式,还是直接输入查询词“female”,得到所有的五篇检索结果(未满 k=20 的限制),文档的 ID 分别为 19986、13738、11160、8713 和 867,并且 wf-idf 分数越高,排列越靠前,并且该模式下将文档的内容也返回了一部分,便于用户直接了解检索到的文档内容。Top k 查询功能完成。

```

=====Searching System=====
searching operation:
[1]BOOL [2]Overall [3]TOP K [4]Phrase [5]exit
your choice(int):
3 → top k查询
input the query statement:
female → 查询语句
stemming...
['female']
spelling correcting...
['female']
doc ID: 19986 score: 4.338 检索文档及权重
JAPAN UNEMPLOYMENT RISES TO RECORD 3.2 PCT IN MAY
Japan's seasonally adjusted unemployment 文档内容前五行内容
rate rose to a record 3.2 pct in May, the worst level since the
government started compiling unemployment statistics in 1953,
the government's Management and Coordination Agency said.
...
doc ID: 8713 score: 3.334
IRELAND PUT ON COLORADO BEETLE ALERT
The Irish Agriculture Department issued
a Colorado beetle alert today after three of the beetles were
found in a box of parsley imported from France.
Officials said a colony of the black and amber coloured

```

(6) 短语查询

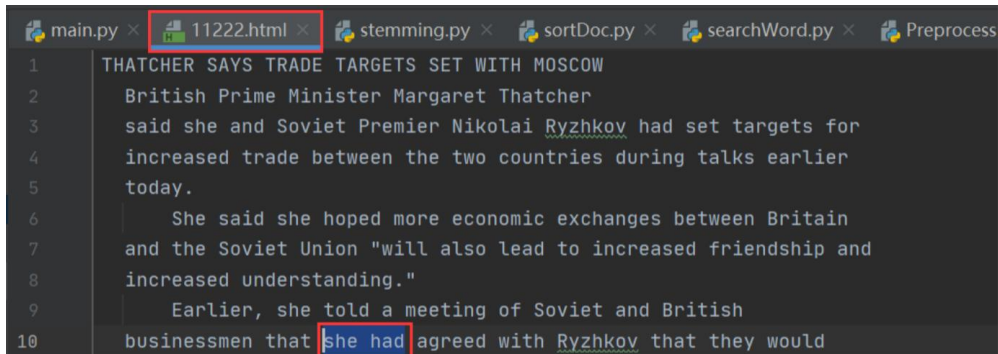
运行时，在命令行中输入 4 则可进入短语查询模式，直接输入查询 query “she has”，得到两篇检索结果，文档的 ID 分别为 5109 和 11222，并且返回了查询 query 在两篇文档中出现的次数，这里全为 1，并且还返回了短语的第一个词在文档中的位置 location。

```

=====Searching System=====
searching operation:
[1]BOOL [2]Overall [3]TOP K [4]Phrase [5]exit
your choice(int):
4 → 短语查询
input the query statement:
she has
stemming...
['she', 'have'] → 词形还原
spelling correcting...
['she', 'have']
docID: 5109 num: 1
location: [333]
docID: 11222 num: 1
location: [66]
→ 有两篇文档符合查询条件，5109和11222

```

对上述结果进行检验，打开以 11222 为 ID 的文档，人工进行浏览，可以发现确实从 0、1、2……开始数的第 66 位置上恰好是 she 所在的位置，并且是以 “she had” 的短语出现，可见，短语查询功能实现了。在实验中我也发现，虽然词组查询可以非常准确的找到所需的内容，但是也可能会一无所获，所以进一步扩充检索数据集也是很有必要的。



【实验总结】：

一、代码架构总结

/nltk_data/：所有的语料库数据文件

/corpora/：语料库，英语词典 wordnet 安装包 wordnet.zip

/SearchSystem/：搜索引擎实现的全部代码所在的文件夹，其中有多个子文件夹和子文件

/BoolSearch/：布尔查询主要代码

BoolSearchDel.py：处理 bool 表达式得到后序表达式，分操作符类别进行倒排表处理

/InvertedIndex/：倒排索引主要代码

establishIndex.py：倒排索引代码，获取单词词典、倒排列表，写入倒排文件

establishVSM.py：建立空间向量模型，写入 VSM 文件

getIndex.py：加载倒排索引文件 invertedIndex.json 和单词词典文件 wordList.json

/LanguageAnalysis/：对文本进行预处理操作的主要代码

PreprocessFile.py：获取文本内容等处理操作

stemming.py：使用 NLTK 库对文本分词、词形还原、筛选，得到单词列表

/scoreQuery/：计算查询分数并进行排序的主要代码、普通查询、TOP K 查询主要代码

getScore.py：进行 tf-idf、wf-idf 等分数的计算

sortDoc.py：进行普通查询排序、最小堆排序，返回[分数，文档号]排序结果

/Searching/：各种查询功能的基础过程代码、短语查询的主要代码

operateDocList.py：对倒排记录表进行合并、剔除等操作，主要服务于 bool 查询

SearchWord.py：各种查询功能模块的基础过程代码

/SpellingCorrect/：拼写检查主要代码

spell.py：对检索关键词进行拼写矫正

big.txt：拼写校正计算模型的语料库

tools.py：自定义的一些操作函数，包括读取数据集、形成文档 id 列表、写数据至文件等

main.py：搜索引擎整体框架代码，能够选择不同的查询模式进行信息检索

二、实验心得体会

本项目从无到有建立了一个简易的搜索引擎，由于题目要求比较宽泛，所以自由度比较大。首先我搜集资料了解了一般的检索系统索引的原理和方法，决定基于倒排索引和向量空间模型实现搜索引擎。实现的过程中主要面临的困难是因为对 python 语法的不熟练导致的在文档-句子-分词-词根的转换存储、引用传递等处理上的困难。不过当单词字典和倒排索引表生成好后，还是很有成就感的。在具体实现功能的时候，通过查阅资料和应用代码，我弄清楚了各个检索功能

的基本原理并且依葫芦画瓢进行了实现。其中涉及到最小堆等算法花费了我一定的时间。这些功能实现后在结果分析环节,我发现除了人工检查匹配效果的方法,没有一个成熟的数据集供我整体检验系统效果,感觉整个项目在评价这块存在缺陷,如果有机会和可能还需要进一步想办法解决。另外系统目前是命令行形式进行检索,未来也可以进一步发展为可视化。

评语及评分(指导教师)

【评语】:

该实验报告设计并实现了基于 TF-IDF 实现的文档检索系统,并在 Reuters - 21578 数据集上进行 precision,recall 等指标的定量评价,基本达到设计目标。虽然报告实现了包括布尔查询在内的四种查询方式,但是缺乏评价算法的实现还有横向对比。

评分: 85



日期: 2021-6-26

附件:

实验报告说明

- 1. 实验名称:** 要用最简练的语言反映实验的内容。
- 2. 实验目的:** 目的要明确,要抓住重点。
- 3. 实验环境:** 实验用的软硬件环境(配置)。
- 4. 实验方案设计(思路、步骤和方法等):** 这是实验报告极其重要的内容。包括概要设计、详细设计和核心算法说明及分析,系统开发工具等。应同时提交程序或设计电子版。

对于**设计型和综合型实验**,在上述内容基础上还应该画出流程图、设计思路和设计方法,再配以相应的文字说明。

对于**创新型实验**,还应注明其创新点、特色。

- 5. 实验结果分析：**即根据实验过程中所见到的现象和测得的数据，进行对比分析并做出结论（可以将部分测试结果进行截屏）。
- 6. 实验总结：**对本次实验的心得体会，所遇到的问题及解决方法，其他思考和建议。
- 7. 评语及评分：**指导教师依据学生的实际报告内容，用简练语言给出本次实验报告的评价和价值。