

# Lab6 胜者表实现非精确TopK查询

10235501419 李佳亮

本次实验采用Java语言，基于向量空间模型与倒排索引，引入了胜者表结构来实现高效的非精确Top-K检索，其中胜者表结构由优先队列来维护得到。

## 实验思路

1. **分词与计算词频**：读取 `article/` 文件夹下的所有文档，去除所有stopwords并用jieba分词，每篇文档保留分词结果，并统计词项频率 `TF`。
2. **离线构建文档向量**：首先，构建整个文档集合，统计整个语料库中的词项文档频率 `DF`。词典中的所有词项个数为向量空间的维度。然后，为每篇文档构建 `TF-IDF` 向量，并同时计算并保存其L2范数，以便在后续查询中用于余弦相似度计算，避免重复计算。
3. **构建倒排索引和胜者表**：为每个词项构建倒排索引，记录包含该词项的文档列表。在此基础上，为每个词项选择 `TF` 值最高的前 `r` 篇文档加入胜者表，减少在线查询时的计算开销。
4. **在线查询**：根据给出的 `query`，把 `query` 转换为 `TF-IDF` 向量，与 `query` 中各个词的胜者表中的文档的向量做相似度运算，然后取TOP-K作为返回结果。

## 具体实现

我们采用向量空间模型，权重采用 `tf-idf` 值。

### （一）文档Document类

一个Document对象表示一个文档，其属性包括文档ID，文档内容，文档包含的各个词项的tf值（用 `Map<String, Integer>`），稀疏表示的tf-idf向量以及其L2范数的值。

#### 表示文档的tf-idf向量：

tf-idf向量用的是 `Map<String, Double>` 来存储的。文档向量有稀疏性，使用 `Map<String, Double>` 进行存储不仅有效减少了内存占用（减少了0占用的空间），也使得在计算与查询向量的余弦相似度时，仅需遍历交集词项，从而提升整体计算效率，详见（四）。

在构造方法中，完成分词与去除停用词的操作；成员方法 `computeTFIDF()` 接收各个词项的df值以及总的文档数，计算当前文档的tf-idf向量以及其L2范数。由于IDF值需要依据整个文档集合来计算，因此我们接下来要先构造一个文档集合类。

### （二）文档集合Corpus类

该类通过一个 `List<Document>` 对象维护所有的文档，用 `Map<String, Integer> dfMap` 维护所有单词的文档频率df，用于后续计算tf-idf。

成员方法 `loadDocuments()` 从文档路径中依次读取文档，对每篇文档构建 `Document` 对象，同时更新 `dfMap`。

### （三）包含胜者表的倒排索引InvertedIndex类

这个类构建倒排索引，并且根据tf值筛选出每个词项的Top-r文档加入胜者表。胜者表用 `Map<String, List<Integer>> championList` 来维护。

胜者表的构建采用由JAVA优先级队列实现的最小堆，对于每个词项，从倒排索引遍历所有出现的文档，仅仅保留tf值最大的前 `r` 个文档，时间复杂度为  $O(n \log r)$ 。

具体来说，我们对倒排索引中的每个词项term构建起一个 `PriorityQueue<Document>`，设置比较逻辑为 `Document` 中词项term的词频tf，模拟最小堆。每遇到一个文档就尝试加入堆中，如果堆的大小超过了 `r`，就用 `heap.poll()` 把tf最小的堆顶弹出；最终，堆中只剩下tf最大的 `r` 篇文档，也就是该词项的胜者表。

#### (四) 查询类QueryProcessor

处理查询query，去除query中的停用词，构造其tf-idf向量。其中，为了保证idf值有意义，idf值是根据文档集合Corpus类的 `dfMap` 计算的。

接着，遍历query中各个词的胜者表的并集（返回的结果，考虑至少包含一个查询词项的文档），使用query向量与各个文档的向量计算余弦相似度，返回余弦相似度最高的Top-K个文档。

##### 向量之间的点积计算：

前面说过向量用 `Map<String, Double>` 对象表示。计算余弦相似度时，核心是对两个 `Map<String, Double>` 对象的操作。只遍历query中出现的term（因为query的term更少），与文档向量中对应的tf-idf值相乘并累加，从而实现稀疏向量之间的点积计算。

#### (五) 精确查询类ExactSearcher

这个类用于返回精确的Top-K结果，与胜者表方法返回的非精确结果进行对比。

## 实验结果

```
输入查询： 中央 发言 小组
输入K值： 10

===非精确Top-K: ===           ===精确Top-K: ===
文档： 1819 - 相似度： 0.1434   文档： 1819 - 相似度： 0.1434
文档： 3899 - 相似度： 0.1335   文档： 3899 - 相似度： 0.1335
文档： 3936 - 相似度： 0.1205   文档： 3871 - 相似度： 0.1318
文档： 1683 - 相似度： 0.1103   文档： 3936 - 相似度： 0.1205
文档： 8218 - 相似度： 0.1087   文档： 2004 - 相似度： 0.1166
文档： 705 - 相似度： 0.0996     文档： 8581 - 相似度： 0.1158
文档： 3187 - 相似度： 0.0996   文档： 6860 - 相似度： 0.1141
文档： 873 - 相似度： 0.0848    文档： 1683 - 相似度： 0.1103
文档： 1291 - 相似度： 0.0789   文档： 680 - 相似度： 0.1097
文档： 2026 - 相似度： 0.0723   文档： 8218 - 相似度： 0.1087
```

从上图可见，用胜者表的非精 Top-K与精确Top-K的检索结果重合度很高：设置胜者表的 `r=10`，对查询“中央 发言 小组”和 `K=10`，非精确Top-10检索能检索到精确Top-10的前5个。这说明所构建的胜者表机制能够有效覆盖大部分高相关文档。并且，随着 `r` 增加，覆盖度也会增加。

## 附录 (GitHub链接)

- [Main.java](#) - 本次实验的代码