简易信息检索系统实现

——通过倒排索引实现布尔检索

刘子晨 1610914

Abstract

我所需要实现检索的系统为一个歌曲的数据库，该数据库存储了一定量中文歌曲的歌名和歌词。通过 jieba 分词工具，将数据库中所有歌曲的歌名和歌词分解为有单个词义中文词组或短语，而后为这些歌词和歌名分别建立倒排索引。

本实验数据库采用 MySQL，后端使用 Python 的 Web 框架 Bottle，前端使用 Vue.js 框架进行展示。

线上地址为：http://47.95.200.233/search-homework/

你所需要的源码、介绍、展示、实验数据，均可在线上地址进行查看和下载。

1. 引言

信息检索（Information Retrieval）是用户进行信息查询和获取的主要方式，是查找信息的方法和手段。狭义的信息检索仅指信息查询（Information Search）。即用户根据需要，采用一定的方法，借助检索工具，从信息集合中找出所需要信息的查找过程。 [1]

布尔检索是针对布尔查询的检索，布尔查询是指利用 AND，OR 或 NOT 操作符将词项连接起来的查询。

由于数据库中每条数据需要检索的只有两个属性，所以分别对其建立倒排索引表，通过建立的倒排索引表进行查找。

2. 实验环境与数据集介绍

本实验数据库采用 MySQL。Web 框架采用 MVC 架构和前后端分离技术，后端使用 Python 的 Web 框架 Bottle，前端使用 Vue.js 框架进行展示。

本实验代码运行于阿里云服务器，服务器采用 LAMP 环境（Ubuntu16.04 Apache PHP5.5），并安装有 Python3.6，python 包：bottle、jieba、json、pymysql。以上 python 包均可通过 pip3 进行安装。

实验数据来源于网络，包含歌曲名称、歌词和 ID，我将其通过 python 脚本导入数据库，保存歌曲的表总共有 4 个属性：（歌曲 ID，歌名，歌词，倒排索引 ID）。总共包括中文歌 99 首，编码方式为 UTF-8。实验数据可以在线上查看和下载 json 格式文件。

3. 实现倒排索引表

在本实验为数据库中的两个属性，歌词和歌名分别建立了倒排索引表，两个表的实现方式相同，所以在介绍时将以歌词为例进行介绍，为歌名所建倒排索引不在进行描述。

实现倒排索引表的整个过程非常复杂，在实验过程中，我通过多个模块实现，下面我将分模块讲述我的实现思路。

读取数据库信息

由于是为数据库中的信息建表，所以我们需要必然需要进行数据库操作，所以我们建立一个基础类 Database，里面使用 python 的 pymysql 包连接数据库，并建立可以用来操控数据库的游标：

```
class Database:
    db = pymysql.connect("xx.xx.xx.xx"
    , "xx", "xx", "search_homework")
    cursor = db.cursor()
```

上述代码为链接实验所用 search_homework 数据库，隐去了隐私信息。

需要获取某个 ID 歌曲信息时，只需要使用游标对象配合 sql 语句即可：

```
# 获取某个ID的歌曲信息
def getinfo(i):
    sql = "SELECT * from song_list " \
        "where song_list.ID = '%d'" % i
    Database.cursor.execute(sql)
    data = Database.cursor.fetchall()
    return data
```

处理歌词信息

首先我们创建 song 类，在该类中进行文本信息的处理：

```
class song:
    def __init__(self, id, text):
        self.id = id
        self.lyric = text
```

由于本实验的数据集是中文歌，所以歌词信息处理需要用到中文分词工具，本实验所用的分词工具为 python 的 jieba 包，调用 jieba 中 cut_for_search 函数将文本分解为词组：

```
#用jieba将文本分解
def getwords(self):
 return jieba.cut_for_search(self.lyric)
```

由于 jieba 只是将文本尽行分解处理，并不会出重或者去除停用词，所以需要代码对于 jieba 返回的列表再次进行处理，得到我们需要的文本所含词组的无停用词、不重合列表：

```
def getwordlist(self):
    w_list = ()
    words = self.getwords()
    for word in words:
    if (word in stopwords) \
                or (word in w_list):
        continue
        w_list += (word,)
    return w_list
```

通过这一函数可以得到当前歌曲对象文本信息的无停用词、无重复词项集合。

倒排索引结构

- 词项结构：
  倒排索引表为词项结构的列表，每个词项的结构如下：

  ```
  class Node:
      def __init__(self, word, id):
          self.word = word
          self.id = [id]
          self.df = 1
  ```

  创建一个词项时，必然存在一个包含此词项的文档，不然不会遇见这个词组，所以创建时该词项的倒排记录表会包含一个最先遇到该词的文档 ID，且此词项的 df 值为 1。当遇到包含相同词组的文档时调用：

  ```
  def addID(self, Sid):
          if self.id.count(Sid) == 0:
      self.id.append(Sid)
      self.df += 1
  ```

  为该词项的倒排记录表添加包含该词组的文档 ID 并将 df 值加一。
- 倒排索引结构：
  创建一个倒排索引时，首先建立一个空的列表，用于存储词项集合：

  ```
  class IndexList:
      def __init__(self):
          self.list = []
  ```

  将文档 ID 和词组传入，通过判断，更新倒排索引表：

  ```
  def addword(self, word, Sid):
      self.list.append(Node(word, Sid))

  def insert(self, word, Sid):
      for node in self.list:
          if word == node.word:
  ```

```
                node.addID(Sid)
                return
        self.addword(word, Sid)

def appendsong(self, words, id):
    for word in words:
        self.insert(word, id)
```

如果该词组在字典中已经存在，则直接将文档 ID 添加到该词项的倒排记录表，如果该词组在字典中没有找到，则在倒排索引表中新建立一个词项。appendsong 函数，则对于某个文档中所有包含的词组都执行更新倒排索引表函数。

最后调用之前的几个模块，对数据库中信息建立索引：

```
#更新歌曲名称和歌词的索引
@staticmethod
def updateIndex():
    lyric_index = indexlist.IndexList()
    for i in range(maxid()):
        str = getinfo(i)
        if not str == ():
            lyric = songInfo.song(str[0][2], str[0][3])
            lyric_index.appendsong(lyric.getwordlist(), lyric.getid())
    file1 = open('song_lyric_index.json', 'w+')
    file1.write(json.dumps(lyric_index.getJson()))
    file1.close()
```

图 1. 建立索引，并更新在 JSON 文件中

4. 实现布尔检索

之前建立的倒排索引是存储在一个 JSON 格式的文件中的，如果每次有查询请求就读取一编该文件或者将该文件的内容一直保存在内存中，都是不可取的方式，所以本系统将倒排索引保存在了数据库中，导入方式不是本实验的重点，所以不再过多陈述。

由于用户在进行布尔检索时会输入字符串，所以，需要有一个处理字符串所用函数：

```
#将字符串格式的id列表转换为列表
def convert_list(num_str):
    data = num_str.lstrip().split(" ")
    res = []
    for val in data:
        if val != '':
            res.append(val)
    return res
```

由于已经将倒排索引表的信息存储在数据库中，每次需要某词项的所对应的倒排记录表只需要进行一次数据库查询操作：

```
@staticmethod
def search_lyric(search_word):
    sql = "SELECT * FROM lyric_inverted_index " \
          "WHERE lyric_inverted_index.word='%s'" \
          % search_word
    Database.cursor.execute(sql)
    # 获取所有记录列表
    data = Database.cursor.fetchall()
    # return data
    if data == ():
        return []
    return convert_list(data[0][2])
```

图 2. 将所查询词项所含有的歌曲 ID 列表返回

将用户输入的布尔查询字符串处理按照布尔查询逻辑，将 AND 和 OR 分别做集合的交集和并集操作，而后再做递归：

```
#把所有运算符都当成右运算符
def deal_boolean_list(boolean_list):
    single_word_list = InvertedIndex.search_lyric(boolean_list[0])
    if len(boolean_list) > 1:
        if boolean_list[1] == 'AND':
            return list(set(single_word_list).intersection(
                set(deal_boolean_list(boolean_list[2:]))))
        elif boolean_list[1] == 'OR':
            return list(set(single_word_list).union(
                set(deal_boolean_list(boolean_list[2:]))))
    else:
        return single_word_list
```

图 3. 递归处理布尔查询逻辑

该处理方式将 AND 和 OR 两个操作符当成相同层级的操作符进行处理，而且采用右侧优先级高的处理方式。这种处理方式有待改进。

将这里处理得到的 id 列表与用户输入不想包含的词组进行并集操作得到的集合进行差处理，得到最终的 id 列表。

将最终得到的 id 列表在数据库中查询得到用户所需的歌曲列表。

5. 总结

在学习过程中，单纯的倒排索引和布尔检索这个两个抽象概念，我们都感觉非常简单，感觉只需要照着书本上的内容一步步进行操作即可，然而在实际进行实验过程中，却发现了许多之前没有预料到的问题。

比如文本编码格式不同，前后端交互时出现问题，跨域问题等等。针对倒排索引建立和布尔查询也需要考虑用户输入、数据库选项等等问题。

总之，在这次实验过程中，我加深了对于倒排索引和布尔查询的认识，也对 Web 开发有了更加深入的了解。

本次实验就实验目的而言是圆满完成了，甚至可以说是超额完成，但是这个系统仍有很多待补充的地方，还有很多可以优化的内容，我也对于它的优化有很多自己的想法，这个系统的大框架已经建成，在未来，如果有时间我希望能把我的其它想法在这个系统上得到实现。

参考文献

[1] Baike. 信息检索（一种信息技术），2017. [Online; accessed 13-December-2017]. 1