Web信息处理与应用Lab1

> 王道宇 PB21030794
>
> 王昱 PB21030814
>
> 吴泽众 PB21030802

实验内容简介

- **Stage1：** 豆瓣数据的爬取和检索
  - 爬虫：
    - 对于电影数据，至少爬取其基本信息、剧情简介、演职员表，本次实验额外爬取的信息有：看过该电影的人数、想看电影该电影的人数
    - 对于书籍数据，至少爬取其基本信息、内容简介、作者简介，本次实验额外爬取的信息有：正在看、想看、看过该书籍的人数
    - 爬取方式：API爬取或网页爬取，本次实验采取网页爬取方式来获取html信息，并采用beautifulsoup+正则表达式匹配的方式来解析
    - 应对平台反爬措施的策略：随机化爬取的等待时间、伪造User Agent、加入cookie等等
  - 检索
    - 分词。采用jieba分词中的精确模式对book和movie的简介进行分词，并将其他包含单个词的信息（例如作品名称、作者姓名、作品类型）分别进行分词和不分词处理加入到分词结果中。
    - 去停用词处理。采用链接https://github.com/guotong1988/chinese_dictionary 中的停用词词典进行去停用词处理
    - 同义词替换。应用synonyms库中的compare函数，使用word2vector模型来替换同义词。
    - 倒排表的生成。根据分词结果，把原先以id索引词项的字典修改为以词项索引id列表的倒排表。
    - 跳表指针。在倒排表的基础上添加跳表指针，加速查询过程。
    - 压缩存储。采用可变长度编码的方式压缩id列表

    - 布尔查询。设计自底向上的文法匹配布尔表达式，并实现AND、NOT、OR三种布尔匹配的操作。

- **Stage2：** 使用豆瓣数据进行推荐

实现过程介绍

- 爬取数据集的设计

  实验的最开始本小组先调研了豆瓣读书和豆瓣电影的作品条目网页里的元素，选取其中的作品基本信息作为构建数据集的主要对象。同时还注意到了网页左下角"****人看过"和"****人想看"的数据（图1），敏锐地意识到这些信息对Stage2中推荐环节的作用，加入了数据集中。

- 请求头的构造

  调用fake_useragent库，将虚构的用户代理添加进用户代理池中，每爬取一个网页就换用用户代理池中其他User Agent，其中用户代理池中包含chrome、safari、firefox、edge四类虚假的User Agent，不容易被豆瓣平台的反爬策略监测出来。

- 网页爬虫算法结构的设计

    - 继承

      由于书籍和电影数据的爬虫具备一定程度的相似性，但要爬取的数据略有不同，因此将两者共同的操作如构造请求头、requset获取网页信息等封装在父类Spider中，再使用继承的模式让子类Book和Spider共同使用父类的方法，并且能够在子类中封装书籍和电影各自的爬虫和解析算法。

    - 解析

使用soup.find函数解析获取的html内容。在网页源代码中找到需要爬取的信息，根据其在html内容中对应的网页元素、属性和值来解析出需要的信息，下面两个表分别展现了Movie和Book解析网页的方式

Movie Parse	影片名	导演，主演等信息	简要介绍	想看的人数
网页元素	span	div	span	a
属性	property	id	class/ property	href
属性的值	v:itemreviewed	info	all hidden/ v:summary	https://movie.douban.com/subjec {movie_id}/comments?status=P

表注：简要介绍信息在所有的电影中有两种形式，一种放在property=“ v:summary”中；另一种由于信息长度过长，需要用户点击“展开按钮”来看到完整的简介，其完整的信息放在 class=“all hidden”中

Book Parse	书名	作者、出版社等信息	简要介绍	想看、在看、看过的人数	评分
网页元素	span	div	div	div	div
属性	property	id	class	id	class
属性的值	v:itemreviewed	info	intro	collector	rating_self clearfix

表注：简要介绍中的“展开全部”的处理 /TODO

- 反爬虫

由于在解析程序把解析失败的错误信息输出到了文件中（Movie_error.json和Book_error.json），所以能够在爬虫过程中发现大概三十个豆瓣的电影id找不到对应的资源。经过检查后发现部分电影和书籍只有在登录过豆瓣账号后才能访问，因此需要请求头中包含cookie信息。但测试后发现豆瓣的cookie属于session cookies只有在保持网页端页面的会话不关闭的情况下才能生效，有效期较短，所以本小组最后采用的措施是：先将不需要cookie信息的电影和书籍信息爬取，再根据报错信息找到爬取失败的id

并用小组成员的豆瓣登录cookie以及用户代理构造请求头爬取。其中，cookie的获取方式为：事先登录豆瓣读书或豆瓣电影，打开浏览器的开发者模式，输入豆瓣读书或豆瓣电影网址，此时开发者模式提供的cookie信息就已经携带有小组成员豆瓣账号登录信息了，可以进一步爬取原先被反爬虫机制限制爬取的id。但加入有效cookie信息后发现Movie部分仍有id：1309046无法在豆瓣中索引到资源，手动输入id对应网址后发现豆瓣未收录条目，猜测可能是涉及到了敏感信息而遭到封禁。

- 分词

  分词方面选取的是jieba库作为分词工具。jieba提供了两种分词模式：精确模式和全模式，分别代表唯一结果分词和多结果分词。本小组最开始采用的是全模式进行分词，但这样做产生了两个问题：一是全模式提供的多结果分词中会有大量词汇的前缀，在下一步的去同义词步骤中会被当做同义词去掉，最后只留下了词汇的前缀，而正确的词汇本身在这一步被替换；二是全模式产生的多结果会对词汇在文档中的tf的计算产生干扰，影响实验结果

- 去同义词和停用词
- 倒排表生成和跳表指针的建立
- 压缩存储和解压
- 布尔查询操作
  - AND
  - OR
  - NOT
  - AND_NOT（优化）

    根据长尾效应，大部分词项在文档中出现频率较小，一旦用NOT取补集，空间复杂度就会极大。通过优化AND后面立刻跟上的NOT这一情况可以减小程序的空间复杂度。

- 文法设计

  本次实验的布尔查询提供AND、OR、NOT 和 () 四种运算符，设计出左递归的非二义性文法来匹配查询内容。其中，E、T、F、G为非终结符，word为终结符，用于匹配查询输入的词项。

$$
\begin{aligned}
E &\rightarrow E\ OR\ T && (1)\\
&\mid T && (2)\\
T &\rightarrow T\ AND\ F && (3)\\
&\mid F && (4)\\
F &\rightarrow NOT\ G && (5)\\
&\mid F && (6)\\
G &\rightarrow (E) && (7)\\
&\mid word && (8)
\end{aligned}
$$

由于用户在布尔查询时的输入通常不会太复杂，如果将上述文法设计成LR(1)文法每次程序启动都需要先花时间构造LR分析表，但考虑到实际情况里用户的查询间断性较强，这样的文法设计可能会花更多的时间在LR分析表的生成上。本小组采用括号匹配的思想，先递归地匹配左括号，找到其对应的右括号并根据产生式(7)将括号内部的内容用非终结符E表示，直到所有括号都被拆除后再自顶向下递归地先后匹配字符串OR、AND和NOT（匹配操作符而不是文法符号，这样能够避免处理左递归文法），并用对应的布尔操作作用于词项对应的倒排表，将id列表作为结果返回