

2018-2019 学年第一学期

《大数据分析与应用》期末课程项目

天线宝宝搜索引擎

组 长：陈婉月

组 员：杨敏 范德宝

学 院：计算机学院

任课教师：张华平

完成时间：2018 年 12 月 15 日

[摘 要] 随着大数据技术的不断发展, 构建智能化搜索引擎是满足现代网络应用的重要举措。文章是关于我们自己搭建一个新闻搜索引擎过程的相关介绍, 包括爬取新闻网站, 抽取新闻的主体内容, 得到结构化的 xml 数据。然后一方面使用内存式单遍扫描索引构建方法 (SPIMI) 构建倒排索引, 另一方面根据 KNN 算法和余弦相似度实现推荐功能。最后利用 BM25 公式计算给定关键词下的文档相关性评分, 其公式打分结合时间因素得到热度评分, 根据评分给出排序结果。

[关键词] 搜索引擎 爬虫 索引 推荐 BM25

目录

| | |
|--------------------|------|
| 1 引言..... | 2 - |
| 2 相关技术..... | 3 - |
| 2.1 网络爬虫..... | 3 - |
| 2.2 建立索引..... | 3 - |
| 2.3 搜索匹配..... | 3 - |
| 2.4 推荐算法..... | 3 - |
| 3 系统设计..... | 4 - |
| 3.1 系统类设计..... | 4 - |
| 3.2 系统模块顺序图设计..... | 4 - |
| 3.3 系统活动图设计..... | 5 - |
| 3.4 实现技术设计..... | 6 - |
| 3.5 微博爬虫模块设计..... | 6 - |
| 3.6 检索模块设计..... | 7 - |
| 3.7 推荐模块设计..... | 7 - |
| 3.8 系统数据流图设计..... | 7 - |
| 4 系统实现..... | 9 - |
| 4.1 算法设计..... | 9 - |
| 4.2 系统模块具体设计..... | 11 - |
| 5 系统展示..... | 16 - |
| 5.1 搜索引擎首页..... | 16 - |
| 5.2 新闻搜索页..... | 16 - |
| 5.3 新闻展示页..... | 17 - |
| 6 系统测试..... | 18 - |
| 6.1 数据抓取测试..... | 18 - |
| 6.2 功能测试用例..... | 18 - |
| 7 总结与展望..... | 20 - |
| 参考文献..... | 21 - |

1 引言

随着社会的发展，人们对搜索引擎的要求越来越高，搜索引擎已经成为人们日常生活中不可缺割的一部分。本文重点研究如何实现一个新闻搜索引擎系统，包括使用网络爬虫来实现数据的获取，使用 SPIMI 方法建立索引表，使用 BM25 算法实现检索，使用 KNN 等方法实现推荐等方面，并且在取得了不错的检索效果。

本文的以下工作安排：

第二部分主要讲述本项目中设计的技术，并对相关技术进行的原理及背景进行了简要的介绍。

第三部分主要讲述了系统设计，包括整体的框架，使用的技术，爬虫框架，检索模和推荐模块的框架设计。

第四部分主要讲述了系统实现，包括项目中使用到的 MD25 算法，SPIMI 算法等，以及爬虫，索引构建，检索，推荐等功能的具体代码实现。

第五部分主要讲述了系统展示，包括系统界面，搜索结果等。

第六部分是测试，针对系统进行的测试结果及说明。

第七部分是总结与展望，包括对工作的总结和项目下一阶段的发展方向。

最后是参考文献。

2 相关技术

2.1 网络爬虫

网络爬虫实际上是一个基于 Web 的程序。它从一个初始的网页集出发，遍历自动的采集网络信息。当爬虫打开某个 HTML 页面后，它会分析 HTML 标记结构来获取信息，并获取指向其它页面的超级链接，然后通过既定的搜索策略选择一个要访问的站点。从理论上讲，如果为 Spider 指定个适当的初始文档集和个适当的网络搜索策略，它就可以遍历整个网络。它的性能在很大程度上影响了搜索引擎站点的规模。本项目将使用 Python 编写爬虫程序，实现数据爬取的功能。

2.2 建立索引

通常有三种索引的建立基本技术：倒排文件、后缀数组和签名文件。倒排文件在当前大多数信息获取系统中得到应用，它对于关键词的搜索非常有效。后缀数组在短语查询中具有较快的速度，但是该结构在维护上相对比较麻烦。签名文档如今已被倒排索引技术替代。处理网页的过程主要包括这几部分：文档特征向量提取、网页筛选、相关度分析、文档分类和入库操作。

2.3 搜索匹配

用户查询模块是搜索引擎和用户之间的接口。其首先获取用户查询条件并加以分析，然后访问索引数据库进行匹配后获得检索结果，然后根据设定的相关度进行降序排序处理后返回给用户。本项目在实现该模块时，采用 BM25 算法，该算法应用范围较广，并且容易实现。

2.4 推荐算法

在实际的搜索引擎中，往往都需要有推荐模块，能根据用户的输入，推荐一些相似度较高的新闻，在本项目中，我们对比了大量的算法，包括 Apriori 和 FW-growth 算法等，在实际使用过程中，考虑到文档数量较大，分词可能较多，我们决定使用 jieba 分词组件获取词频和文档长度，然后使用 KNN 算法来实现相似文档的分类，再通过余弦相似度来得到两两文本的相似度，最终输出 K 个最相似度最高的新闻。

3 系统设计

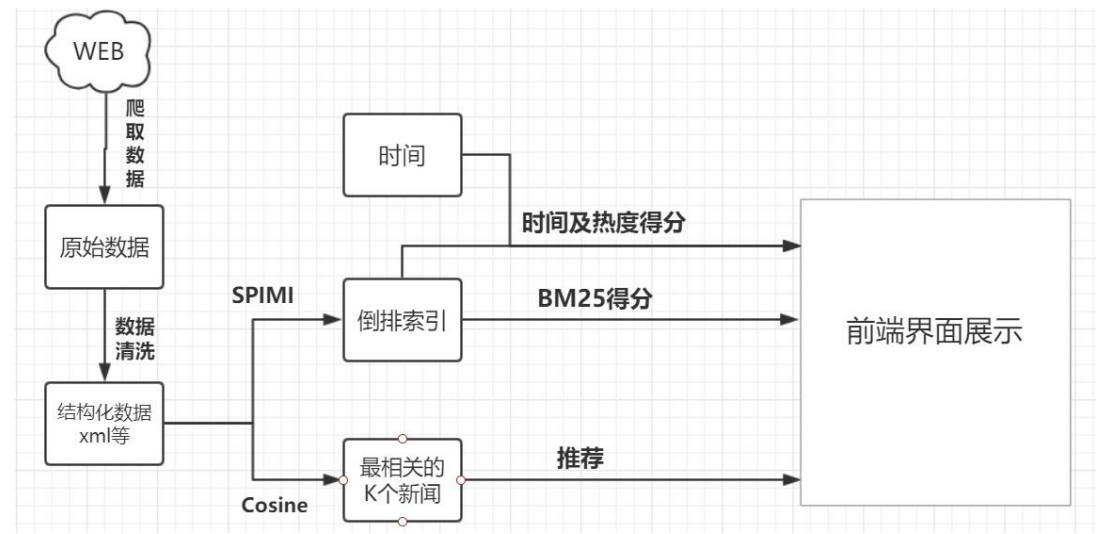


图 3.1 系统框架图

本图为系统框架图，首先使用爬虫从指定的网站上爬起新闻数据，得到原始数据，然后对原始数据进行数据清洗，包括过滤图片，广告，视频等，以此得到结构化的数据，最后使用 cosine，SPIMI 等得到检索出的信息和推荐新闻。

3.1 系统类设计

TextClear 类负责数据格式化存储。数据抓取类负责对网络微博数据进行实时抓取。数据分析类负责对抓取的数据进行格式分析等操作。URL 处理类负责对抓取的数据中重复的子节点 URL 进行过滤筛选等。页面显示类负责对抓取的数据进行可视化显示等。

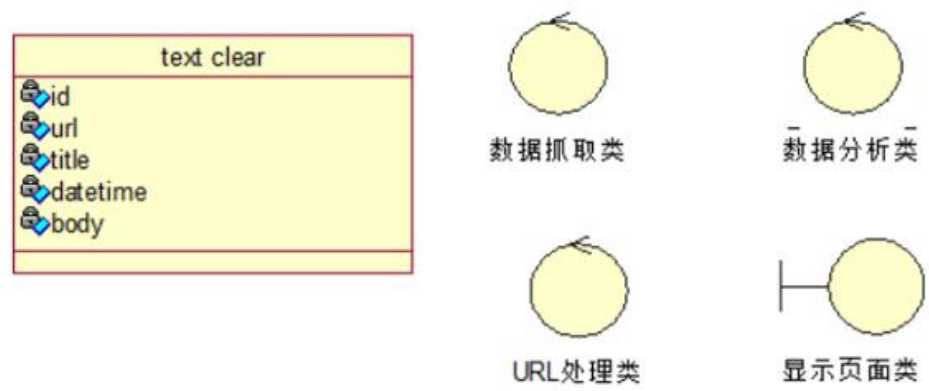


图 3.2 系统类实现设计

3.2 系统模块顺序图设计

我们根据系统实现的具体过程，画出了系统模块的顺序图，如图 3.3 所示。

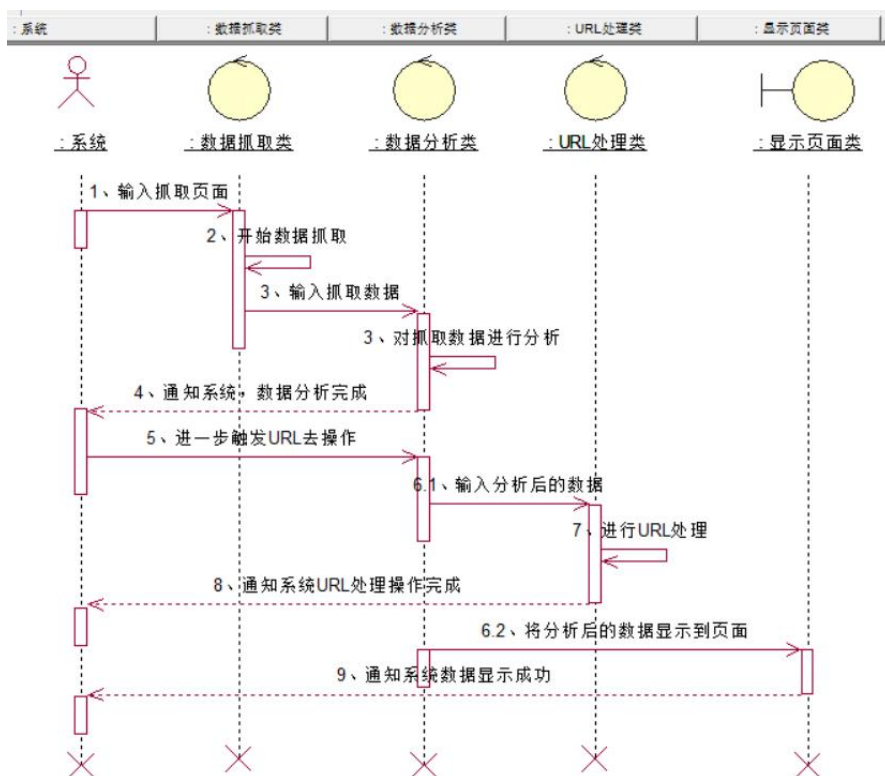


图 3.3 系统顺序图实现设计

3.3 系统活动图设计

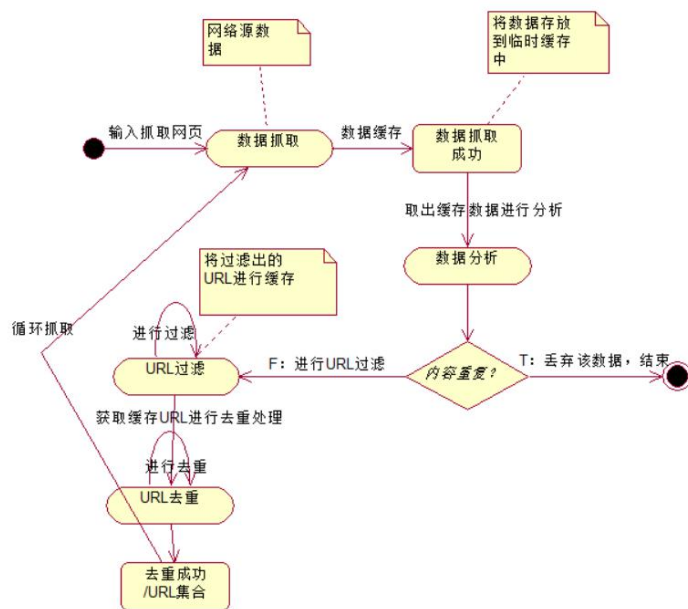


图 3.4 系统活动图实现设计

3.4 实现技术设计

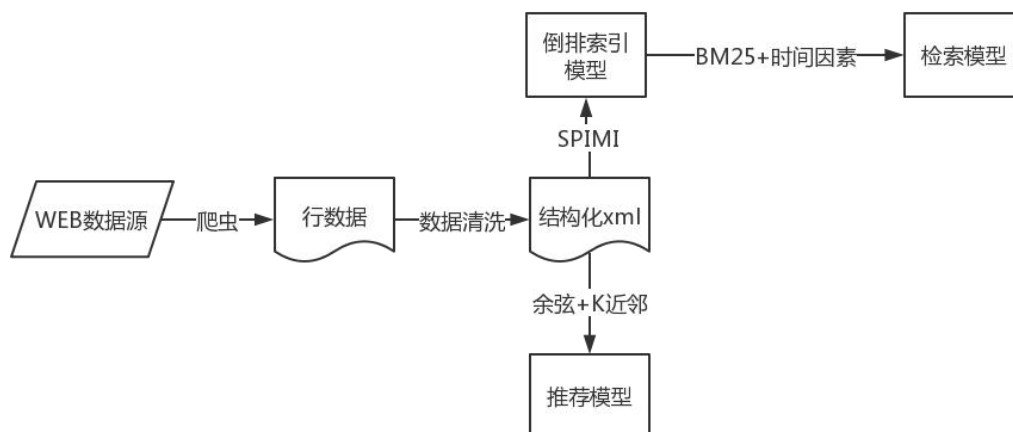


图 3.5 技术实现设计

本图展示了整个系统的设计流程和使用到的技术，我们在获取数据的时候使用了爬虫，随后对数据进行了清洗，得到了以 xml 存储的结构化数据，然后使用 jieba 分词等工具，sklearn 函数和 pairwise distances 函数来得到最相似的五篇新闻 id，以此同时，我们将 xml 使用 SPIMI 算法来构建倒排索引，随后使用 BM25 等算法实现检索。

3.5 微博爬虫模块设计

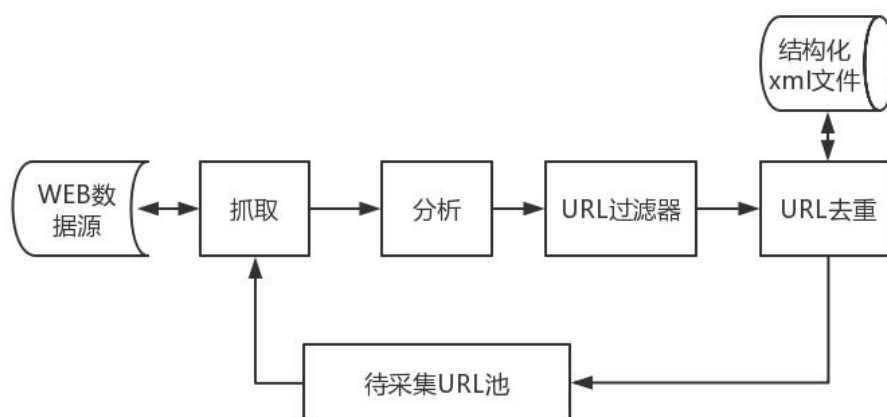


图 3.6 爬虫框架图

爬虫执行过程为“抓取->分析->得到新的 URL->再抓取->再分析”的循环，我们在背景中介绍了爬虫相关的技术和实现爬虫工具，但是在本次作业中，我们决定自己用 python 来实现爬虫，具体的编码会在详细设计中介绍。

3.6 检索模块设计

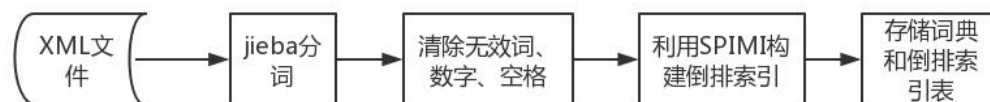


图 3.7 检索模块框架图

检索模块是将爬虫模块得到的 xml 文件进行处理，最后能输正确的检索结果。该模块的流程大致是先将文档进行分词，然后清除无效词，之后使用 SPIMI 构建排序索引，并将词典和倒排记录表存储，在后再使用 BM25 公式按照新闻的得分高低进行输出。

3.7 推荐模块设计

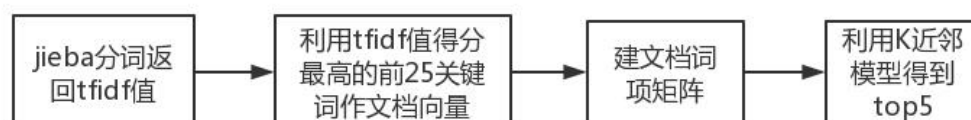


图 3.8 推荐模块框架图

将需要查询的句子先分词，然后去查询词典和记录表，建立文档词项矩阵，最后使用 KNN 得到最相近的五个推荐新闻。

3.8 系统数据流图设计

3.8.1 顶层设计

顶层设计中，系统根据用户输入待搜索关键字，将待搜索数据传输到系统中进行处理，最终系统输出符合用户搜索匹配的内容和相关推荐的数据。

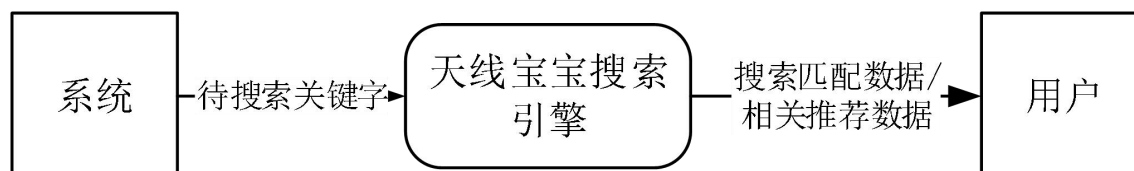


图 3.9 顶层设计流程图

3.8.2 一层设计

一层设计中将系统内部数据流程进行分解，主要包括搜索数据处理和搜索数据规整两个主要的大模块。

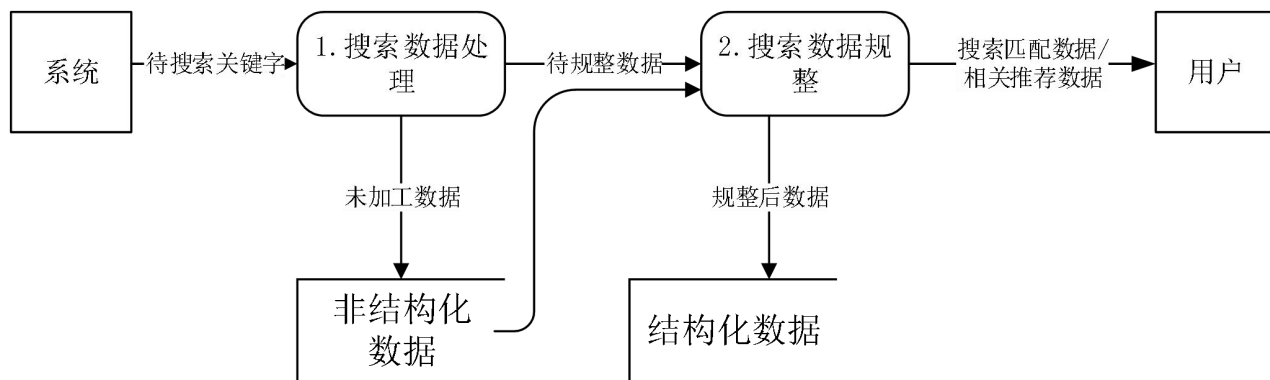


图 3.10 一层设计流程图

3.8.3 二层设计

二层设计中将系统内部的搜索数据处理和搜索数据规整两个主要的大模块进一步分解，搜索数据处理模块包括了对数据校验和数据爬取两个主要的数据流向模块，搜索数据规整主要包括数据规整处理、搜索数据匹配和新闻推荐三个主要的数据流向模块。

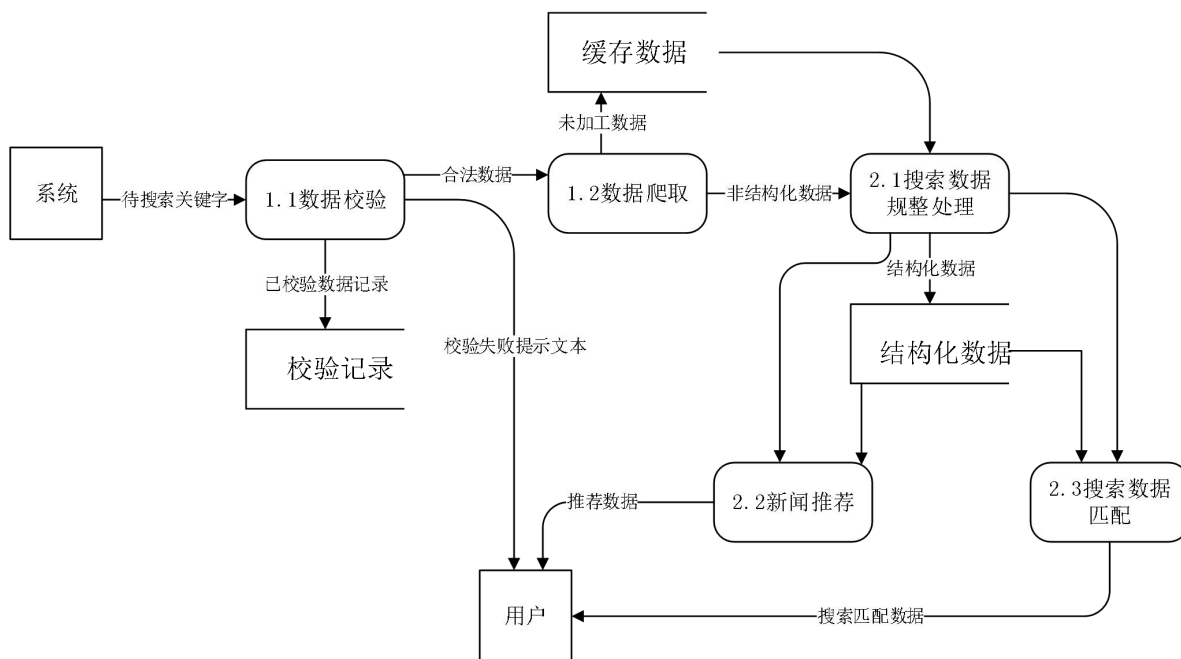


图 3.11 一层设计流程图

4 系统实现

4.1 算法设计

本小节主要介绍系统中使用到的算法，上文已经介绍了系统的整体框架，可以看到，本项目的算法设计主要集中在检索模块，下面会分别介绍用于构建倒排索引的 SPIMI 方法，用于检索的 BM25 算法。

4.1.1 SPIMI 方法

本项目中使用的索引方式为倒排索引，该索引方式弥补了布尔检索不记录词频的缺陷，

在扫描文档时，记录了词频 tf ，同时也记录了文档长度 ld 和某个词项在不同文档中出现的次数 df 。

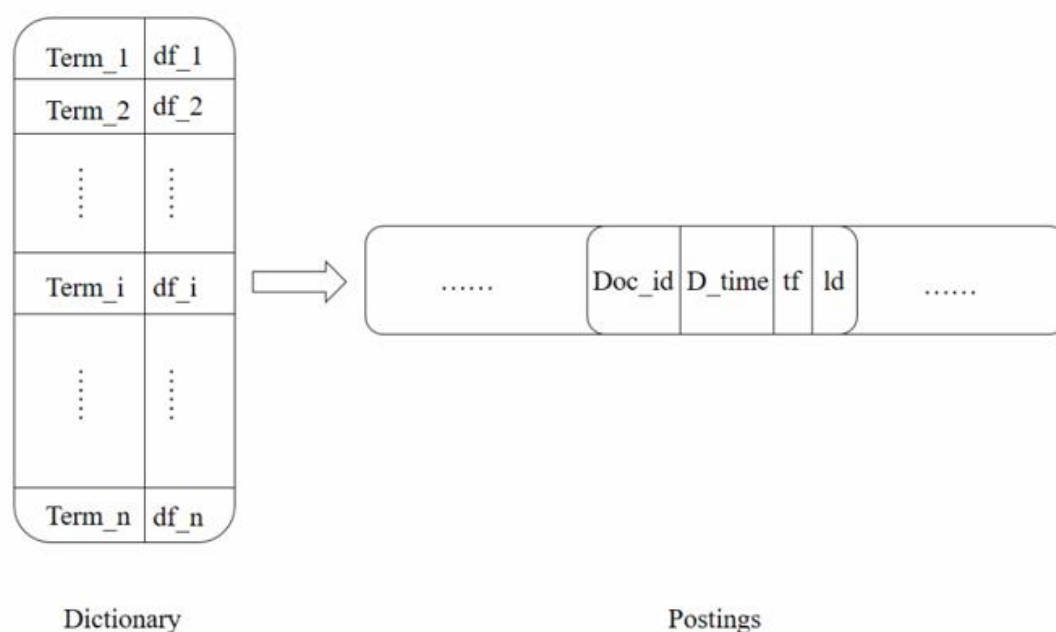


图 4.1 倒排索引结构图

本图是倒排索引的结构图，分为左半部分的词典和右半部分的倒排记录表，词典中存放的是新闻中所有不同的词项，倒排记录表中存放的是出现某个词项的文档信息，包括词频和文档长度。

在构建倒排索引之前，我们需要处理三个关键的问题：

1. 得到一篇文档中的所有词项。因为本项目面对的是中文网站，所以我们使用 jieba 中文分词组件来实现对文档的分词。

2. 去除某些停用词。我们需要去除某些不能很好划分文档的词，也就是几乎所有文档都会出现的词，这里我们在 jieba 分词完成之后进行去除。

3. 倒排记录表存储。本项目采用邻接记录表的方式来存储数据。

在处理完以上的问题后，欧文需要使用具体的算法来实现倒排索引，本项目采用 SPIMI 方法，即内存式单遍扫描索引构建方法，该方法的核心思想是将每一篇新闻分词，如果出现新的词则插入到词典中，否则将该文档的信息追加到词项对应的记录表中，下图是 SPIMI 的伪代码，具体实现的代码在后文中会介绍。

```

SPIMI-INVERT(token_stream)
1  output_file = NEWFILE()
2  dictionary = NEWHASH()
3  while (free memory available)
4  do token ← next(token_stream)
5     if term(token) ∉ dictionary
6         then postings_list = ADDTODICTIONARY(dictionary, term(token))
7         else postings_list = GETPOSTINGSLIST(dictionary, term(token))
8     if full(postings_list)
9         then postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
10    ADDTOPOSTINGSLIST(postings_list, docID(token))
11 sorted_terms ← SORTTERMS(dictionary)
12 WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
13 return output_file

```

图 4.2 SPIMI 伪代码图

4.1.2 BM25 算法

BM25 算法是基于概率的检索算法，下图是 BM25 算法的公式。

$$BM25_{score}(Q, d) = \sum_{t \in Q} w(t, d)$$

$$w(t, d) = \frac{qt f}{k_3 + qt f} \times \frac{k_1 \times tf}{tf + k_1(1 - b + b \times l_d / avg_l)} \times \log_2 \frac{N - df + 0.5}{df + 0.5}$$

图 4.3 BM25 公式图

公式中变量含义：

qtf: 查询中的词频

tf: 文档中的词频

l_d : 文档长度

avg_l: 平均文档长度

N: 文档数量

df: 文档频率

b, k_1 , k_3 : 可调参数

第一个公式是用于计算一个文档 d 对于查询 Q 的得分，第二个公式是用于计算某个词项 t 在文档 d 中的得分。

根据 BM25 公式，能很快的得到对于不同文档 t 对查询 Q 的得分情况，然后根据得分高低排序给出结果。后文会介绍其具体实现的代码。

4.2 系统模块具体设计

本节会介绍系统类和代码实现，包括爬虫部分的关键代码实现，检索模块的实现和推荐模块的实现。

4.2.1 爬虫的具体实现

因为需求中提出的是对搜狐娱乐进行数据爬取，所以先看搜狐娱乐新闻的页面



图 4.4 搜狐娱乐网页图

在通过浏览器查看了其页面代码后，我们设计了下图结构的 xml 文件，用于存储结构化的数据。

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <doc>
3     <id></id>
4     <url></url>
5     <title></title>
6     <datetime></datetime>
7     <body></body>
8 </doc>
```

图 4.5 数据存储类设计

下面是 python 实现的爬虫

```

def get_news_pool(root, start, end):
    news_pool = []
    for i in range(start, end, -1):
        page_url = ''
        if i != start:
            page_url = root + '_%d.shtml'%(i)
        else:
            page_url = root + '.shtml'
        try:
            response = urllib.request.urlopen(page_url)
        except Exception as e:
            print("-----%s: %s-----"%(type(e), page_url))
            continue
        html = response.read()
        soup = BeautifulSoup(html)
        td = soup.find('td', class_ = "newsblue1")
        a = td.find_all('a')
        span = td.find_all('span')
        for i in range(len(a)):
            date_time = span[i].string
            url = a[i].get('href')
            title = a[i].string
            news_info = ['2016-' + date_time[1:3] + '-' + date_time[4:-1] + ':' + date_time[-2:],
                        url, title]
            news_pool.append(news_info)
    return(news_pool)

```

图 4.6 爬虫实现（1）

上图是实现对网站的所有链接列表的获取。

```

def crawl_news(news_pool, min_body_len, doc_dir_path, doc_encoding):
    i = 1
    for news in news_pool:
        try:
            response = urllib.request.urlopen(news[1])
        except Exception as e:
            print("-----%s: %s-----"%(type(e), news[1]))
            continue
        html = response.read()
        soup = BeautifulSoup(html, "lxml") # http://www.crummy.com/software/BeautifulSoup/bs4/doc.zh/
        try:
            body = soup.find('div', class_ = "text clear").find('div').get_text()
        except Exception as e:
            print("-----%s: %s-----"%(type(e), news[1]))
            continue
        if '//' in body:
            body = body[:body.index('//')]
        body = body.replace(" ", "")
        if len(body) <= min_body_len:
            continue
        doc = ET.Element("doc")
        ET.SubElement(doc, "id").text = "%d"%(i)
        ET.SubElement(doc, "url").text = news[1]
        ET.SubElement(doc, "title").text = news[2]
        ET.SubElement(doc, "datetime").text = news[0]
        ET.SubElement(doc, "body").text = body
        tree = ET.ElementTree(doc)
        tree.write(doc_dir_path + "%d.xml"%(i), encoding = doc_encoding, xml_declaration = True)
        i += 1

```

图 4.7 爬虫实现（2）

上图是按照指定的 xml 格式获取所有的新闻信息。

4.2.2 倒排索引模型

下面简单的展示了一下创建索引的代码，包括上文讲述的字典和记录表，部分的功能和原理在上文中已经说明，这里就只展示代码

```
class Doc:
    docid = 0
    date_time = ''
    tf = 0
    ld = 0
    def __init__(self, docid, date_time, tf, ld):
        self.docid = docid
        self.date_time = date_time
        self.tf = tf
        self.ld = ld
    def __repr__(self):
        return(str(self.docid) + '\t' + self.date_time + '\t' + str(self.tf) + '\t' + str(self.ld))
    def __str__(self):
        return(str(self.docid) + '\t' + self.date_time + '\t' + str(self.tf) + '\t' + str(self.ld))
```

图 4.8 字典的实现

上图是定义的字典类，用于实现倒排索引中的字典创建。

```
def construct_postings_lists(self):
    config = configparser.ConfigParser()
    config.read(self.config_path, self.config_encoding)
    files = listdir(config['DEFAULT']['doc_dir_path'])
    AVG_L = 0
    for i in files:
        root = ET.parse(config['DEFAULT']['doc_dir_path'] + i).getroot()
        title = root.find('title').text
        body = root.find('body').text
        docid = int(root.find('id').text)
        date_time = root.find('datetime').text
        seg_list = jieba.lcut(title + '。' + body, cut_all=False)

        ld, cleaned_dict = self.clean_list(seg_list)

        AVG_L = AVG_L + ld

    for key, value in cleaned_dict.items():
        d = Doc(docid, date_time, value, ld)
        if key in self.postings_lists:
            self.postings_lists[key][0] = self.postings_lists[key][0] + 1 # df++
            self.postings_lists[key][1].append(d)
        else:
            self.postings_lists[key] = [1, [d]] # [df, [Doc]]
    AVG_L = AVG_L / len(files)
    config.set('DEFAULT', 'N', str(len(files)))
    config.set('DEFAULT', 'avg_l', str(AVG_L))
    with open(self.config_path, 'w', encoding = self.config_encoding) as configfile:
        config.write(configfile)
    self.write_postings_to_db(config['DEFAULT']['db_path'])
```

图 4.9 记录表的实现

上图是创建记录表的关键函数

以上就是对系统中倒排索引的建立，在运行代码时会生成一个数据库文件，这就是构建好的索引数据库，我们能使用这个创建好的数据库取实后文的查询和推荐功能

4.2.3 BM25 算法实现

根据 BM25 公式，能计算出不同文档对查询的得分情况，并且按照得分高低排序输出结果。

```
def result_by_BM25(self, sentence):
    seg_list = jieba.lcut(sentence, cut_all=False)
    n, cleaned_dict = self.clean_list(seg_list)
    BM25_scores = {}
    for term in cleaned_dict.keys():
        r = self.fetch_from_db(term)
        if r is None:
            continue
        df = r[1]
        w = math.log2((self.N - df + 0.5) / (df + 0.5))
        docs = r[2].split('\n')
        for doc in docs:
            docid, date_time, tf, ld = doc.split('\t')
            docid = int(docid)
            tf = int(tf)
            ld = int(ld)
            s = (self.K1 * tf * w) / (tf + self.K1 * (1 - self.B + self.B * ld))
            if docid in BM25_scores:
                BM25_scores[docid] = BM25_scores[docid] + s
            else:
                BM25_scores[docid] = s
    BM25_scores = sorted(BM25_scores.items(), key = operator.itemgetter(1))
    BM25_scores.reverse()
    if len(BM25_scores) == 0:
        return 0, []
    else:
        return 1, BM25_scores
```

图 4.10 BM2.5 实现

上图展示了一个 BM25 实现代码，首先将句子分词得到所有查找词库，然后从数据库中查找该词项对应的记录表，对记录表中的所有文档用 BM25 公式进行计算得分，最后按照得分高低排序作为查询结果。

4.2.4 推荐模块的实现

推荐模块想要实现的是下图所示的效果：

经常交流健身心得，更打趣道两人是好“肌”友，戏里戏外都是“竞争对手”。在人生十分重的兄弟情实在让人羡慕，超级期待这场长情而幸福的婚礼！返回搜狐，查看更多责任编辑：

推荐阅读

[《跨越8年的新娘》终极预告 佐藤健改变爱情观](#)
[《跨越8年的新娘》定档10.19 佐藤健化身暖男](#)
[黑金经纪再添生力军 首位女演员伊然诚意出道](#)
[聚焦青年电影人 郭晓东担任电影展形象大使](#)
[郭晓东担任第五届重庆青年电影展形象大使](#)

图 4.11 推荐模块展示

该模块的核心在于度量新闻之间的相似度，取出相似度最高的 5 篇新闻作为推荐阅读的新闻。具体实现思路在上文中已经有过说明，这里不再赘述，具体实现的代码较为复杂，会放在附录中。

5 系统展示

5.1 搜索引擎首页

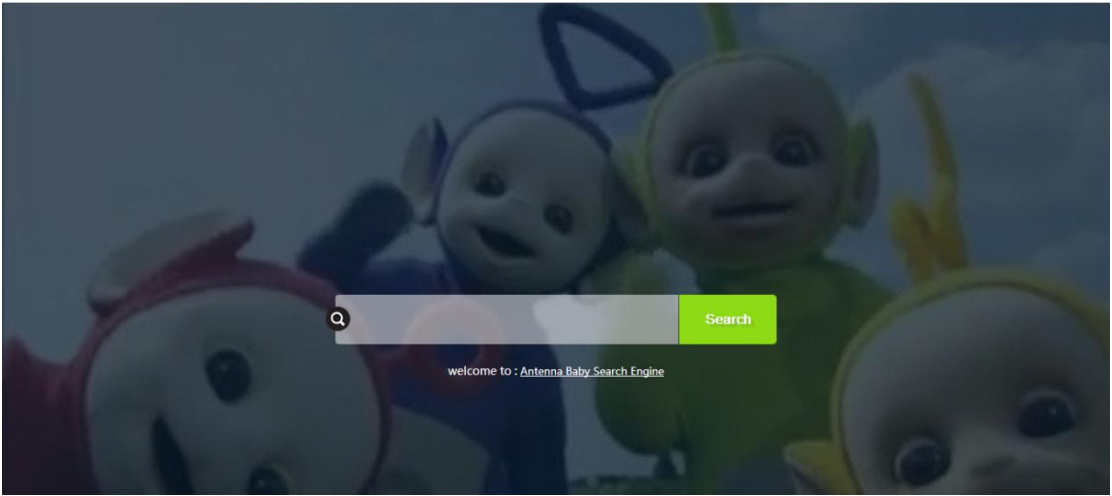


图 5.1 搜索引擎首页展示

图 5.1 是搜索引擎界面，用户在输入栏中输入需要查询的关键词，点击右侧的 Search 就能实现对关键词的搜索。

5.2 新闻搜索页



图 5.2 新闻搜索页展示

图 5.2 是新闻搜索页面，当用户在首页的搜索栏输入关键词搜索后，系统将

输入的关键词进行 jieba 分词，随后通过 BM25 算法和索引表将其与数据库中出现关键词的文档进行比较，计算出关联度和热度，最后根据关联度和热度排序来输出新闻信息。从图中实际的检索结果来看，该系统实现了较好的检索功能，能根据用户输入准备的查找出需要的新闻链接。

5.3 新闻展示页



图 5.3 新闻展示页

图 5.3 是我们打开上一步搜索到的新闻，可以看到新闻的标题，时间，链接，新闻的具体内容，还有左下角本项目实现的推荐阅读，我们从推荐阅读的新闻标题和检索的关键词可以看出系统推荐的文章和关键词之间具有较好的相似性。

6 系统测试

6.1 数据抓取测试

数据抓取需要填写的输入项和输入条件如下：

搜索关键字：100 以内字符；

表 6.1 等价类表

| 输入条件 | 有效等价类（编号） | 无效等价类（编号） |
|-------|-------------|-----------------------|
| 搜索关键字 | 100 以内字符（1） | 空（2） 大于 100 个字符（3） |

表 6.2 测试用例表

| 编号 | 输入数据 | 预期结果 | 覆盖的等价类 |
|----|-----------|--------------|--------|
| | 搜索关键字 | | |
| 1 | 成龙 | 正确 | 1 |
| 2 | 空 | 错误，搜索关键字不能为空 | 2 |
| 3 | 重复 110 字符 | 错误，超过 100 字符 | 3 |

从测试结果来看，在网页数较多，数据量较大的情况下，系统对于检索的结果跟输入需要检索的语句具有较好的精度，并且推荐模块也具有较好相关性，能满足需求。

6.2 功能测试用例

表 6.3 功能测试用例表

| | | | |
|----------|-----|--------|---------|
| 测试项目编号 | 001 | 测试项目名称 | 关键字内容搜索 |
| 测试用例编号:1 | | | |

输入：天线宝宝

预想结果：系统输出天线宝宝相关新闻信息

实验情况：和期望值一致

| | | | |
|--------|-----|--------|------|
| 测试项目编号 | 001 | 测试项目名称 | 空值搜索 |
|--------|-----|--------|------|

测试用例编号:2

输入：不输入任何内容，直接点击搜索

预想结果：提示搜索内容不能为空

实验情况：和期望值一致

| | | | |
|--------|-----|--------|--------|
| 测试项目编号 | 001 | 测试项目名称 | 异常字符搜索 |
|--------|-----|--------|--------|

测试用例编号:3

输入：输入超过 100 字符的数据

预想结果：提示搜索内容字符超出最大限制

实验情况：和期望值一致

7 总结与展望

通过本次大作业，我们学习到了爬虫和索引相关的知识，了解了国内外爬虫及搜索引擎的发展现状，调研了包括 Luncce 和 Solr 等开源项目，并且从基础原理上实现了一个自己开发的搜索引擎，整个项目包括两个部分：网页数据的爬取和搜索引擎的建立。在网页数据的爬取中，我们使用 python 来自己实现了，随后通过配置文件清除文档中的广告和，视频等数据，并且将数据以 xml 指定的格式存储，之后对所有文档进行停用词的去除，完成数据清洗。在数据清洗完成后，通过 SPIMI 方法建立倒序索引，在后再通过 BM25 公式和热度值公式计算出最终的新闻排序，与此同时，通过 KNN 等一系列方法找到相似度最高的五个新闻作为推荐新闻。

虽然我们实现了自己的搜索引擎，但该项目并不适用于大规模（网页达到亿级）的网页爬取和搜索，我们需要新的技术和体系来满足大数据的需求，这也是我们未来准备实现的方向，利用现有的大数据的平台和技术（Hadoop, Jstorm 等），实现分布式的搜索引擎。

参考文献

- [1]张亚凤. 垂直搜索引擎中关键技术的研究[D]. 长春工业大学, 2016.
- [2]王松. 垂直搜索引擎中智能爬虫系统的研究与实现[D]. 北京邮电大学, 2017.
- [3]李博文. 基于 Java 的搜索引擎的设计与实现[D]. 2016.
- [4]刘智勇. 基于 JAVA 技术搜索引擎的设计与实现[J]. 数字技术与应用, 2017(5):205-205.
- [5]俞靓亮. 基于 lucene.net 的搜索引擎在学校网站群系统中的应用[J]. 信息技术与信息化, 2016(9):27-30.
- [6]宋献民, 逢焕利, 魏姁姐. 基于 lucene 的垂直搜索引擎的研究与设计[J]. 信息技术与信息化, 2015(1):147-148.
- [7]许翰林, 王瑞, 王佳丽, et al. 基于 Lucene 的新闻垂直搜索引擎设计与实现[J]. 电脑编程技巧与维护, 2018(2).
- [8]李振宇. 基于 Lucene 的新闻搜索引擎的设计与实现[J]. 湖北农机化, 2017(6):52-52.
- [9]张晓飞, 余建桥. 基于用户兴趣模型构建与个性化搜索算法研究[J]. 电脑知识与技术, 2016, 12(18):1-4.
- [10]祁忠琪, 吕晓聪. 基于网络爬虫的搜狐网新闻搜索引擎系统的实现[J]. 数字通信世界, 2017(7).
- [11]魏茂. 搜索引擎中的网络爬虫搜索对策分析[J]. 丝路视野, 2018, (12):191.
- [12]李小三. 新闻垂直搜索引擎中文分词与网页去重的应用与研究[D]. 长安大学, 2014.
- [13]方志民, 戴洋洋, 董淑珍, et al. 新闻类垂直搜索引擎系统研究与设计[J]. 黑龙江工程学院学报, 2016, 30(6):35-37.