信息检索系统实验报告

陈朴炎 2021211138 王唯宇 2021211133 陈锦鹏 2021211137

目录

1 实验要求

1.1 基本要求

设计并实现一个信息检索系统，中、英文皆可，数据源可以自选，数据通过开源的网络爬虫获取，规模不低于 100 篇文档，进行本地存储。中文可以分词（可用开源代码），也可以不分词，直接使用字作为基本单元。英文可以直接通过空格分隔。构建基本的倒排索引文件。实现基本的向量空间检索模型的匹配算法。用户查询输入为自然语言字串，查询结果输出按相关度从大到小排序，列出相关度、文档题目、主要匹配内容、URL、文档日期等信息。最好能对检索结果的准确率进行人工评价。界面不做强制要求，可以是命令行，也可以是可操作的界面。提交作业报告和源代码。

1.2 扩展要求

鼓励有兴趣和有能力的同学积极尝试多媒体信息检索以及优化各模块算法，也可关注各类相关竞赛。自主开展相关文献调研与分析，完成算法评估、优化、论证创新点的过程。

1.3 评分标准

1、完成基本的信息检索功能且有对环境和社会可持续发展影响的考虑，系统能够正常运行，并提交源代码和实验报告：60 分；

2、完成要求的信息检索功能且有对环境和社会可持续发展影响的考虑，系统能够正常运行，并按时提交源代码和实验报告：61~70 分；

3、在 2 的基础上，且实验报告撰写认真、思路清晰、表达准确：71~80 分；

4、在 3 的基础上，支持检索结果准确率人工评价：81~90 分；

5、在 4 的基础上，融入了自己的创新性思考、优化算法或对多媒体信息检索进行了尝试：91-100 分。

2 数据处理

2.1 爬取 url

本次的实验数据我是在中国新闻网上收集的。

在即时新闻页面上可以查看往日的所有新闻，即使新闻链接为：

https://www.chinanews.com.cn/scroll-news/news1.html

页面示意图如下：

图 2-1 中国新闻网即时新闻示意图

我们可以发现 url 为

https://www.chinanews.com.cn/scroll-news/2023/1214/news.shtml

其中，https://www.chinanews.com.cn/ 为中国新闻网的基础域名，

/scroll-news/ 表示是滚动新闻页面，2023 是年份，1214 是月份和日期。

打开网页的 Elements，我们查看这个网页的 html 格式，可以看到，左端的 url 都保存在<content-left>/<content_list>下的<url>里面。

因此，可以通过爬取这个前端元素来获取我们所需要的新闻的 url。

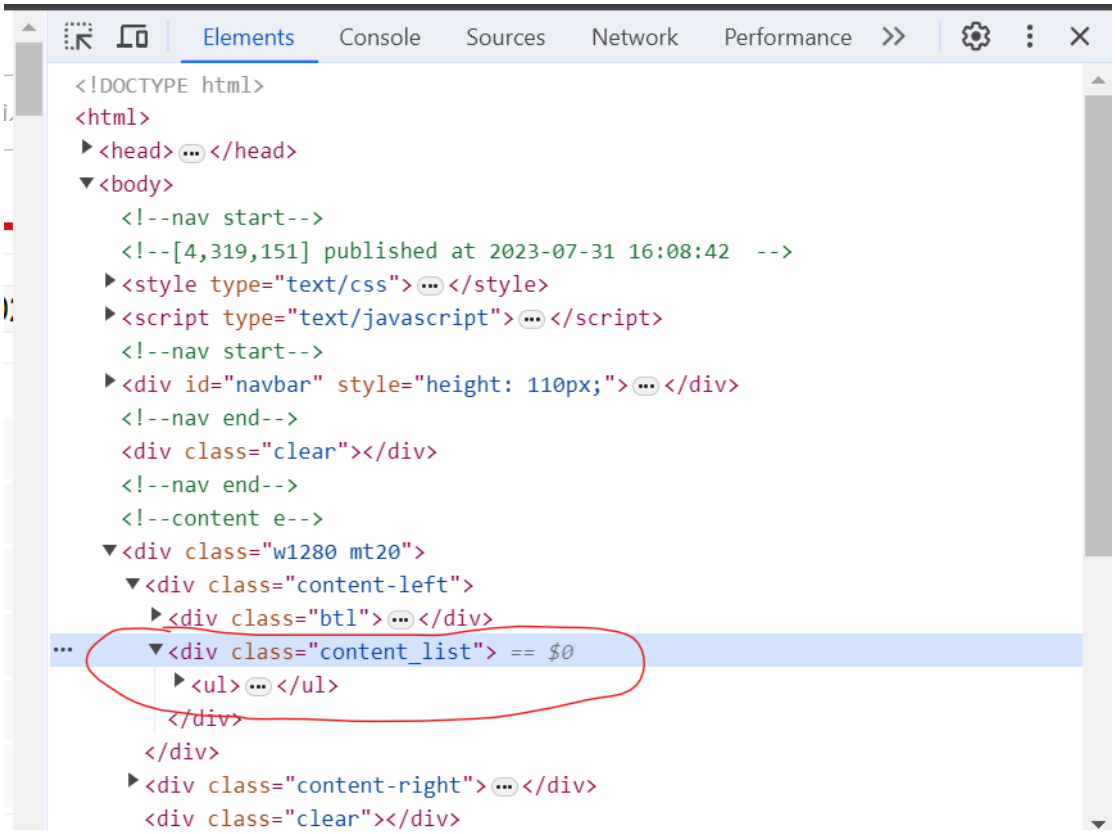


图 2-2 网页页面元素示意图

通过以上信息，我们可以使用 python 的 bs4、urllib 和 request 来爬取中国新闻网滚动新闻的 url 信息，主要操作步骤如下：

1．设置好基础的 url 部分，以及年月日列表

2．通过 request 里的 get 函数来到达某一日滚动新闻页面

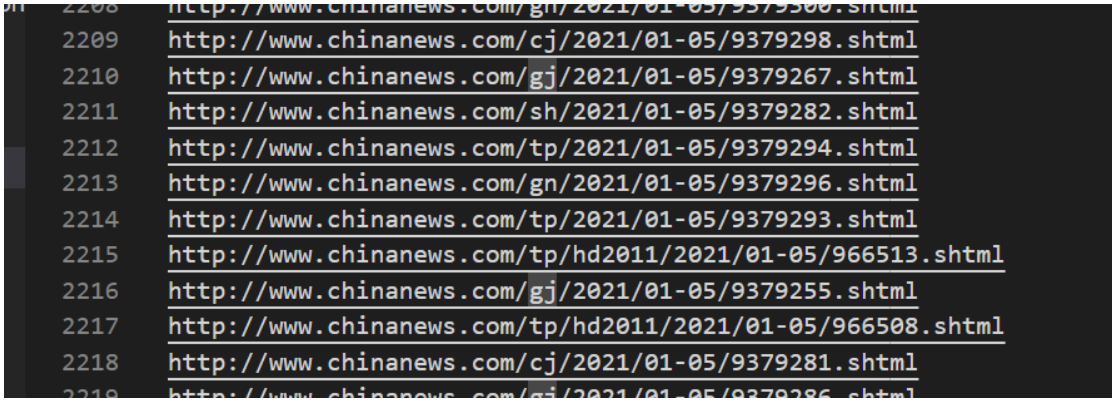
3．通过 html parser 解析 html 结构，找到 content_list 和 url 列表

4．将 url 爬取下来，放到列表中，写入到文件中

程序如下所示：

```python
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin

def crawl_news_urls(base_url, year, month, day):
    url = f"{base_url}/{year}/{month:02d}{day:02d}/news.shtml"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    news_urls = []
    for li in soup.select('.content_list ul li'):
        div_bt = li.find('div', class_='dd_bt')
        if div_bt:
            link = div_bt.a
            if link:
                news_url = urljoin(url, link['href'])
                news_urls.append(news_url)

    return news_urls

# 主程序
base_url = "http://www.chinanews.com/scroll-news"

years = [2011,2012, 2013, 2014, 2015, 2016, 2017,2018,2019,2020,2021,2022, 2023]
months = [1, 2, 3,4,5,6,7,8,9,10,11,12]
days = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]

# 爬取新闻页面的 URL
with open('./urls.txt', 'a', encoding='utf8') as file:
    for year in years:
        for month in months:
            for day in days:
                news_urls = crawl_news_urls(base_url, year, month, day)
                print(news_urls)
                for news_url in news_urls:
                    file.write(news_url + '\n')
```

图 2-3 收集 url 程序示意图



```
2209  http://www.chinanews.com/cj/2021/01-05/9379298.shtml
2210  http://www.chinanews.com/gj/2021/01-05/9379267.shtml
2211  http://www.chinanews.com/sh/2021/01-05/9379282.shtml
2212  http://www.chinanews.com/tp/2021/01-05/9379294.shtml
2213  http://www.chinanews.com/gn/2021/01-05/9379296.shtml
2214  http://www.chinanews.com/tp/2021/01-05/9379293.shtml
2215  http://www.chinanews.com/tp/hd2011/2021/01-05/966513.shtml
2216  http://www.chinanews.com/gj/2021/01-05/9379255.shtml
2217  http://www.chinanews.com/tp/hd2011/2021/01-05/966508.shtml
2218  http://www.chinanews.com/cj/2021/01-05/9379281.shtml
```

图 2-4 url 收集结果图

如上图所示，我从 2011 到 2023 年收集到了共 2000 多条数据。

代码如下：

```
1.  # 这个程序是用来爬取 类似 https://www.chinanews.com/scroll-
    news/2019/1211/news.shtml 这个网站里的 url 的
2.
3.  import requests
4.  from bs4 import BeautifulSoup
5.  from urllib.parse import urljoin
```

```
6.
7.  def crawl_news_urls(base_url, year, month, day):
8.      url = f"{base_url}/{year}/{month:02d}{day:02d}/news.shtml"
9.      response = requests.get(url)
10.     soup = BeautifulSoup(response.text, 'html.parser')
11.
12.     news_urls = []
13.     for li in soup.select('.content_list ul li'):
14.         div_bt = li.find('div', class_='dd_bt')
15.         if div_bt:
16.             link = div_bt.a
17.             if link:
18.                 news_url = urljoin(url, link['href'])
19.                 news_urls.append(news_url)
20.
21.     return news_urls
22.
23. # 主程序
24. base_url = "http://www.chinanews.com/scroll-news"
25.
26. years = [2021,2022, 2023]
27. months = [1, 2, 3,4,5,6,7,8,9,10,11,12]
28. days = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18
    , 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]
29.
30. # 爬取新闻页面的 URL
31. with open('./data/urls.txt', 'a', encoding='utf8') as file:
32.     for year in years:
33.         for month in months:
34.             for day in days:
35.                 news_urls = crawl_news_urls(base_url, year, month, da
    y)
36.                 print(news_urls)
37.                 for news_url in news_urls:
38.                     file.write(news_url + '\n')
```

2.2 爬取新闻数据

打开收集到的某个 url，里面的网页结构如图所示：

图 2-5 新闻结构示意图

可以看到，新闻的主体内容是在<body>/<main_content w1280>/<con_left>/<content_maincontent_content>下的<left_zw>

因此我们可以通过爬取 left_zw 里<p>中的文字信息来获取新闻内容。

具体步骤如下：

1. 打开 urls 文件，获取 urls 列表

2. 通过 request.get 获取网页结构，并通过 html parser 解析

3. 查找 left_zw 类，将<p>中的文本扒下来

4. 保存到文件中

代码如下：

```
1.  from urllib.parse import urlsplit
2.  import requests
3.  from bs4 import BeautifulSoup
4.  import os
5.  import threading
6.
7.
8.  def fetch_and_save(urls, count):
9.      try:
```

```
        for url in urls:
            if not url.startswith("http://www.chinanews.com/"):
                continue
            parsed_url = urlsplit(url)
            # base_url = f"{parsed_url.scheme}://{parsed_url.netloc}"

            path_parts = parsed_url.path.split('/')
            year = path_parts[2]
            if not year.isdigit() :
                continue
            type = path_parts[1]
            count += 1
            filepath = f'./data/news/'
            if not os.path.exists(filepath):
                os.makedirs(filepath)
            filename = f'{filepath}{str(count)}.txt'
            if os.path.exists(filename):
                continue

            # 开始获取文本信息
            try:
                response = requests.get(url, timeout=5)
            except requests.Timeout:
                print("{} timeout".format(count))
                continue
            except requests.exceptions.RequestException as e:
                print(f"请求发生错误: {e}")

            response.encoding = 'utf-8'
            soup = BeautifulSoup(response.text, 'html.parser')

            # 查找 left_zw 类（用来保存新闻内容的）
            left_zw = soup.find('div', class_='left_zw')
            if left_zw:
                # 提取文本内容
                news_contents = url+'\n'
                paragraphs = left_zw.find_all('p')
                for paragraph in paragraphs:
                    news_contents += paragraph.text.strip()

                # 保存到文件
```

```
                with open(filename, 'w', encoding='utf-8', errors='ignore') as f:
                    f.write(news_contents)
                print("完成: {} 的读取".format(count))
    except  Exception as e:
        print(f"{count} 发生错误: {e}")


if __name__ == "__main__":
    urls = open('./data/urls.txt').read().split('\n')
    base_count=0
    chunk_size = 100
    finished_num = 0
    threads = []
    try:
        for i in range(base_count, len(urls), chunk_size):
            base = i + finished_num
            thread = threading.Thread(target=fetch_and_save, args=(urls[base:i+chunk_size], base))
            threads.append(thread)
            thread.start()
            print("开启线程: {}".format(i))

        # 等待所有线程完成
        for thread in threads:
            thread.join()
    except Exception as e:
        print("出现问题:{}".format(e))
```

2019 年之后的文本数据是 UTF-8 编码的，在爬取中得注意鉴别。

在代码里，我通过多线程爬取，开了 20 个线程同时爬取数据。

2.3 分词

分词的步骤如下：

1．读取文件到列表中

2．对于每一个文件，使用 cut 函数进行分词

3．给分词后的每个词后加空格

4．判断是否是名词

5.组合成一个字符串，写入到新的文件中

代码如下：

```
import os
import jieba.posseg as pseg
import threading

def is_all_chinese(strs):
    for _char in strs:
        if not ('\u4e00' <= _char <= '\u9fa5'):
            return False
    return True

def process_text(doc):
    words = pseg.cut(doc)
    word_str = ''
    first = True
    for word in words:
        now_word = str(word.word)
        # 判断词是否是名词，且不是人名，同时不在停用词表中
        if 'n' in word.flag :
            if first:
                first = False
            else:
                word_str += ' '
            word_str += word.word
    return word_str

def process_category(base_dir):
    dir_path = os.path.join(base_dir)
    count = 0
    out_dir = "./data/splice/"
    os.makedirs(out_dir, exist_ok=True)  # 确保文件夹存在，如果不存在则创建
    for _, _, files in os.walk(dir_path):
        for f in files:
```

```
            out_path = os.path.join('./data/splice/', f)
            if os.path.exists(out_path):
                print(f"文件 {out_path} 已存在，跳过处理。")
                continue

            file_path = os.path.join(dir_path, f)
            with open(file_path, 'r', encoding='utf-8') as file_obj:
                lines = file_obj.readlines()
                if len(lines) < 2:
                    continue
                doc = ''.join(lines[1:])  # 跳过第一行 URL

            word_str = process_text(doc)
            count += 1

            with open(out_path, 'w', encoding='utf-8') as out:
                out.write(word_str)
            if(count % 100 == 0):
                print("{}".format(count))
    print(count)

if __name__ == "__main__":

    base_dir = './data/news/'
    threads = []
    try:
        thread = threading.Thread(target=process_category, args=[base_dir])
        threads.append(thread)
        thread.start()

        for thread in threads:
            thread.join()
    except Exception as e:
        print("出现问题:{}".format(e))
```

2.4 构建倒排索引表

2.4.1 TF-IDF

TF-IDF 有两层意思：一层是词频“Term Frequency”缩写为 TF，另一层是逆文档频率“Inverse Document Frequency”缩写为 IDF。

词频指的是某一个给定的词语在该文档中出现的频率。这个数字是对词数的标准化，以防止它偏向长的文档。

逆向档案频率是一个词语普遍重要性的度量。某一特定词语的 idf，可以由总文档数目除以包含该词语之档案的数目，再将得到的商取以 2 为底的对数得到

$$\mathrm{idf_i} = \lg \frac{|D|}{|\{j : t_i \in d_j\}|}$$

|D|为语料库中的文档总数；$|\{j:t_i \in d_j\}|$是包含词语 t_i 的文档总数目，如果词语不在语料库中，就会导致分母为零，因此一般情况下会在分母 + 1。

字词的重要性随着它在档案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

当有 TF 和 IDF 后，将这两个词相乘，就能得到一个词的 TF-IDF 的值。某个词在文章中的TF-IDF 越大,那么一般而言这个词在这篇文章的重要性会越高，所以通过计算文章中各个词的 TF-IDF，由大到小排序，排在最前面的几个词，就是该文章的关键词。

可以看到，TF-IDF 与一个词在文档中的出现次数成正比，与该词在整个语

言中的出现次数成反比。所以，自动提取关键词的算法就很清楚了，就是计算出文档的每个词的 TF-IDF 值，然后按降序排列，取排在最前面的几个词。

2.4.2 TF-IDF 算法步骤

步骤 1．计算词频：

TF = 某个词在文章中的出现次数

考虑到文字有长短之分，为了便于不同文章比较，进行“词频”标准化

TF = 某个词在文章中的出现次数 ÷ 文章总次数

步骤 2．计算逆文档频率

需要一个语料库，来模拟语言的使用环境

$$\mathrm{idf_i} = \lg \frac{|D|}{|\{j : t_i \in d_j\}|}$$

步骤 3．计算 TF-IDF

TF-IDF = TF × IDF

2.4.3 倒排索引表

倒排索引表是一种高效的数据结构，用于快速查找包含某些关键词的文档。倒排索引表的核心思想是将文档内容进行词项分解，然后记录每个词项在哪些文档中出现，从而实现快速查找。

对于每个词项，记录包含该词项的所有文档 ID 及其对应的 TF-IDF 值。倒排列表通常用一个字典来表示，字典的键是词项，值是包含该词项的文档 ID 及

权重的列表。

比如如下表：

今天	1.txt、2.txt、3.txt
新华社	888.txt、896.txt
非洲	946.txt、985.txt
中国	966.txt、955.txt

倒排索引表通过查找词，来找到包含该词的文档。比如说上表中，我查找“新华社”在哪些文档中出现，那么就能找到有 888.txt、896.txt 这两个文档。

在倒排索引表中，我们可以为表项中的那些匹配上的文档加上和该词项的相关度，相关度即为 tf-idf 的值。这样可以避免像“的”、“了”这些词频很高的词相关度被计算的很高。

在我们的代码实现中，我们先进行了词频计算，统计每个词项在每个文档中出现的次数，并进行标准化处理以防止文档长度对结果的影响。以及逆文档频率计算，统计每个词项在多少文档中出现，计算公式为 IDF = log(文档总数 / (包含该词项的文档数 + 1))。IDF 反映了词项在所有文档中的稀有程度。

并通过 TF 和 IDF 的值计算出了 TF-IDF 的值，对于每个词项和文档，计算该词项在文档中的 TF-IDF 值。TF-IDF 值用于衡量词项对文档的相对重要性，计算公式为 TF-IDF = TF * IDF。

最后通过计算好的 TF-IDF 值建立出了合理的倒排索引表，其中表项中还包含了每个文档和该词的相关度。

```
1.     def load_documents(directory):
2.         # 从指定目录加载文档
```

```
    documents = {}
    for filename in os.listdir(directory):
        if filename.endswith('.txt'):
            filepath = os.path.join(directory, filename)
            with open(filepath, 'r', encoding='utf-8') as file:
                content = file.read()
                documents[filename] = content
    return documents

def tokenize(text):
    # 分词并返回名词列表
    words = pseg.cut(text)
    return [word for word, flag in words if flag.startswith('n')]

def build_vocabulary(documents):
    # 构建词汇表
    vocabulary = set()
    for content in documents.values():
        tokens = tokenize(content)
        vocabulary.update(tokens)
    return list(vocabulary)

def compute_tf(documents):
    # 计算词频
    tf = {}
    for doc_id, content in documents.items():
        tokens = tokenize(content)
        tf[doc_id] = Counter(tokens)
    return tf

def compute_idf(documents, vocabulary):
    # 计算逆文档频率
    idf = defaultdict(int)
    num_documents = len(documents)
    for content in documents.values():
        tokens = set(tokenize(content))
        for token in tokens:
            if token in vocabulary:
                idf[token] += 1
    for token in idf:
        idf[token] = math.log(num_documents / (idf[token] + 1))
    return idf
```

```
45.
46. # 计算 tf-idf
47. def compute_tfidf(tf, idf, vocabulary):
48.     # 计算 TF-IDF 向量
49.     tfidf = defaultdict(lambda: [0.0] * len(vocabulary))
50.     token_index = {token: idx for idx, token in enumerate(vocabulary)
    }
51.     for doc_id, freqs in tf.items():
52.         for token, count in freqs.items():
53.             if token in token_index:
54.                 tfidf[doc_id][token_index[token]] = count * idf[token
    ]
55.     return tfidf
56.
57. # 构建倒排索引表
58. def build_inverted_index(tfidf):
59.     # 构建倒排索引表
60.     inverted_index = defaultdict(list)
61.     for doc_id, freqs in tfidf.items():
62.         for token, weight in enumerate(freqs):
63.             if weight > 0:
64.                 inverted_index[token].append((doc_id, weight))
65.     return inverted_index
```

得到的结果如下所示：

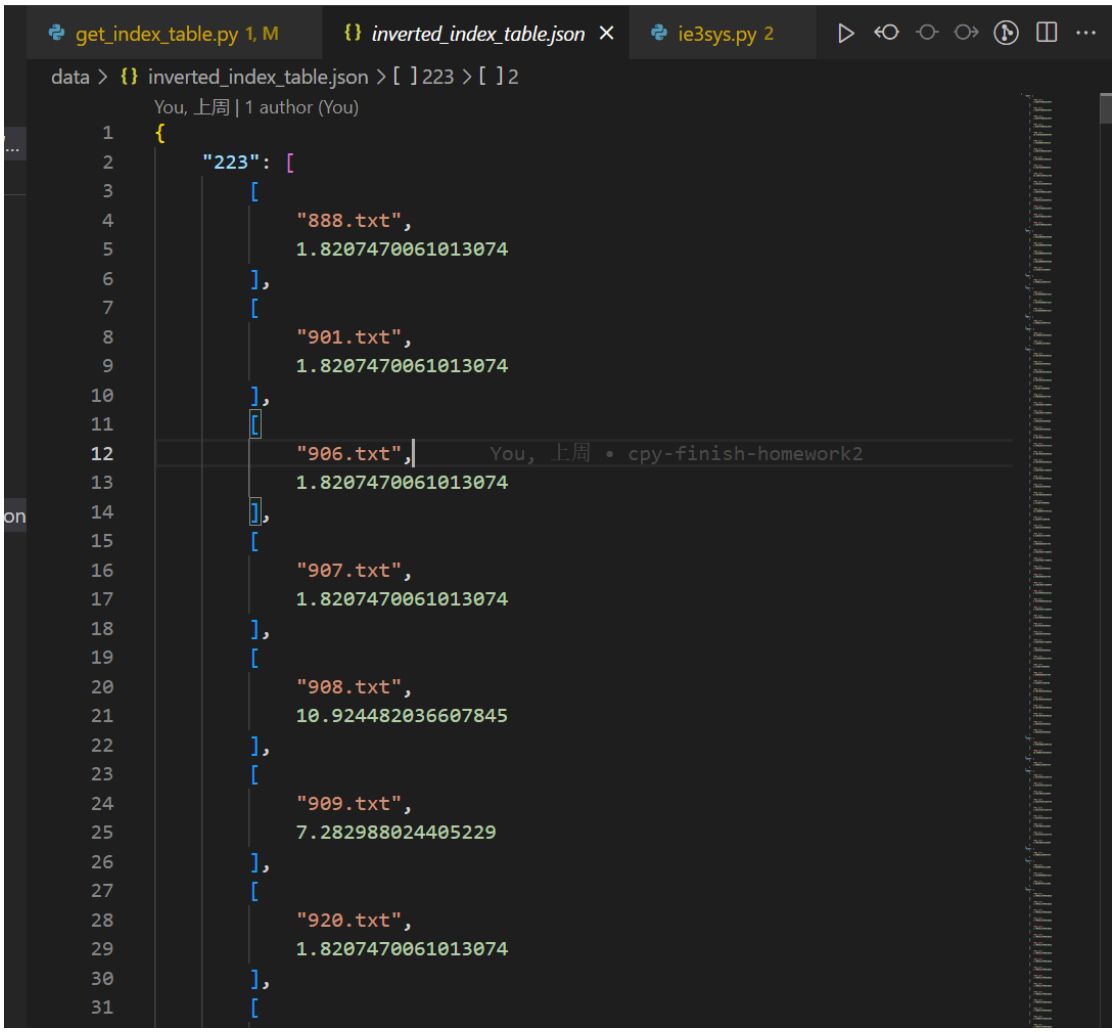


图 2-6 词汇倒排索引表示意图

其中“223”表示的词在词汇表中的索引，即表示词汇空间向量的第几维。下面的 888.txt 表示的是文档名，1.82…这串数字表示的是该词和该文档的 TF-IDF 值。

2.5 计算向量空间

2.5.1 构建词汇表

从文档集合中提取所有的词项，构成一个词汇表，如下所示：

```
1.  def build_vocabulary(documents):
2.      # 构建词汇表
```

```
    vocabulary = set()
    for content in documents.values():
        tokens = tokenize(content)
        vocabulary.update(tokens)
    return list(vocabulary)
```

图 2-7 词汇表本地存储示意图

2.5.2 构建向量

在信息检索和文本处理领域，空间向量是一种将文本表示为多维空间中的点的方法。每个维度通常对应于一个特定的特征或属性。例如，词袋模型（Bag of Words）和 TF-IDF 模型都可以将文档表示为向量，其中每个维度对应一个词汇表中的词项。

将每个文档表示为一个向量，向量的每个维度对应词汇表中的一个词项，向量的值对应该词项在文档中的频率。

假设我们有两个文档：文档 1：今天天气不错；文档 2：明天会下雨

构建词汇表：词汇表为：["今天"，"天气"，"不错"，"明天"，"会"，"下雨"]

计算词项的频率：文档 1 的向量为：2,1,2,0,0,0；文档 2 的向量为：0,0,0,1,1,1

构建向量：文档 1 的空间向量为：2,1,2,0,0,0；文档 2 的空间向量为：0,0,0,1,1,1

为了构造出每个文档的向量空间，我们可以使用 defaultdict 来初始化一个字典 tfidf，其中每个文档的 ID 作为键，对应的值是一个长度为词汇表大小的向量（初始值为全零向量）。defaultdict(lambda: [0.0] * len(vocabulary))确保了每个文档的向量都是长度与词汇表大小一致的全零向量。

之后构造出词汇表的索引。token_index 是一个字典，键是词汇表中的词，值是词汇表中该词的位置（即索引）。{token: idx for idx, token in enumerate(vocabulary)}通过枚举词汇表,为每个词项分配一个唯一的索引值。

然后计算出 TF-IDF 的值。遍历每个文档的词频字典 tf,tf 的结构是{doc_id: {token: count}}，其中 doc_id 是文档 ID，token 是词项，count 是该词项在文档中的出现次数。对于每个文档 doc_id 和它的词频字典 freqs：

遍历词频字典中的每个 token 及其对应的出现次数 count。如果该词项 token 在词汇表中（即 token in token_index），则：通过 token_index[token]获取该词项在词汇表中的索引位置。计算该词项的 TF-IDF 值：count * idf[token]，其中 count 是词项在文档中的出现次数，idf[token]是该词项的逆文档频率。将计算得到的 TF-IDF 值存储在 tfidf[doc_id]向量的对应位置上（通过索引位置 token_index[token]）。

最终，tfidf 字典的结构为{doc_id: [tf-idf values]}，其中每个文档

ID 对应一个 TF-IDF 向量，该向量的每个维度表示词汇表中对应词项的 TF-IDF 值。

```
    def compute_tf(documents):
        # 计算词频
        tf = {}
        for doc_id, content in documents.items():
            tokens = tokenize(content)
            tf[doc_id] = Counter(tokens)
        return tf

def compute_idf(documents, vocabulary):
    # 计算逆文档频率
    idf = defaultdict(int)
    num_documents = len(documents)
    for content in documents.values():
        tokens = set(tokenize(content))
        for token in tokens:
            if token in vocabulary:
                idf[token] += 1
    for token in idf:
        idf[token] = math.log(num_documents / (idf[token] + 1))
    return idf

# 计算 tf-idf
def compute_tfidf(tf, idf, vocabulary):
    # 计算 TF-IDF 向量
    tfidf = defaultdict(lambda: [0.0] * len(vocabulary))
    token_index = {token: idx for idx, token in enumerate(vocabulary)
}
    for doc_id, freqs in tf.items():
        for token, count in freqs.items():
            if token in token_index:
                tfidf[doc_id][token_index[token]] = count * idf[token
]
    return tfidf
```

3 系统设计及实现

3.1 系统设计

3.1.1 概述

该系统主要实现了一个基于 TF-IDF 空间向量的文档检索功能。用户可以输入查询内容，系统会根据输入查询进行相关文档的检索，并返回按相关度排序的文档列表。用户还可以对文档进行评分和查看全文。

3.1.2 模块划分

系统主要分为以下几个模块：

1．数据加载模块

2．文本处理模块

3．向量化模块

4．检索模块

5．结果显示模块

3.1.3 详细设计

（1） 数据加载

功能：从文件系统中加载所需的数据，包括文档、TF-IDF 向量、IDF 值、词汇表和倒排索引表。

设计：

load_json(filepath)：从指定路径加载 JSON 文件，返回数据。

load_documents(directory)：从指定目录加载所有文本文档，返回一个字典，其中键为文档名，值为文档内容。

（2）文本处理

功能：对文本进行分词，并提取出名词。

设计：

extract_nouns(text)：使用 jieba 库对文本进行分词，提取并返回名词列表。

（3）向量化模块

功能：将用户查询和文档转换为向量表示，用于后续的相似度计算。

设计：

query_to_vector(query， idf， vocabulary)：将用户的查询转换为向量表示。首先对查询进行分词，然后计算每个词的 TF-IDF 值，构建查询向量。

compute_cosine_similarity(vec1，vec2)：计算两个向量的余弦相似度，用于评估文档和查询之间的相似度。

（4）检索模块

功能：根据用户查询检索相关文档，并按相似度排序。

设计：

search(query，tfidf，idf，vocabulary)：执行检索操作。将查询转换为向量后，与每个文档的 TF-IDF 向量计算相似度，返回按相似度排序的文档列表。

（5）结果显示

功能：显示检索结果，并允许用户查看文档全文和进行评分。

设计：

display_results(results, documents, query, top_n=5)：显示检索结果。包括文档的相关度、日期、URL 和匹配的句子。

main()：主程序循环，负责处理用户输入，调用检索和显示模块，并处理用户对文档的评分和查看全文操作。

3.1.4 工作流程

1．初始化

系统启动时，加载所有必要的数据文件：文档、TF-IDF 向量、IDF 值、词汇表和倒排索引表。

2．查询处理：

用户输入查询内容，系统调用方法将查询转换为向量。

3．检索

系统调用 search 方法，根据查询向量计算与每个文档向量的相似度，得到相关文档的列表并按相似度排序。

4．结果显示

系统调用 display_results 方法，显示检索到的文档列表，包括相关度、日期、URL 和匹配的句子。

5．用户交互

用户可以选择查看文档全文，或对文档进行评分。

系统更新评分数据并保存。

用户在终端输入查询内容。这一阶段的交互流程如下：

1．提示用户输入查询：

系统在终端中显示提示：“请输入搜索内容（输入 'exit' 退出）：”。

2．用户输入查询：

用户输入一个查询字符串（可以是一个或多个词语）。如果用户输入“exit”，系统会退出循环，结束程序。

系统将检索到的文档结果显示给用户。

对于每个文档，显示以下信息：

1．文档编号：用于用户选择查看或评分

2．相关度：文档与查询的相似度

3．日期：从文档 URL 中提取的日期

4．URL：文档的 URL 地址

5．匹配句子：在文档中包含查询词语的句子，并将匹配的词语标红

用户可以选择查看文档全文或对文档进行评分。具体步骤如下：

1．提示用户选择操作

系统在终端显示提示：“请输入要查看全文的文档编号或输入评分（格式：编号 评分），输入 'exit' 返回查询：”。

2．用户输入选择

用户输入一个文档编号来查看全文，或者输入“编号 评分”格式对文档进行评分。如果用户输入“exit”，系统会返回查询输入提示，允许用户输入新的查询。

3．查看全文

如果用户输入文档编号，系统会打开并显示文档的全部内容。

4．评分

如果用户输入“编号 评分”，系统会更新评分数据并保存：

5．提取文档编号和评分

更新 ratings 字典，将评分保存到对应的文档

调用 save_json 方法，将更新后的评分数据保存到文件

3.2 系统实现

3.2.1 数据加载模块

系统启动时，从磁盘加载所有必要的数据文件，包括文档内容、TF-IDF 向量、IDF 值、词汇表和倒排索引表。这些数据的加载确保了系统能够快速响应用户的查询。

数据加载函数

```
1.     def load_json(filepath):
2.      with open(filepath, 'r', encoding='utf-8') as file:
3.          data = json.load(file)
4.      print("load_inverted_index ok")
5.      return data
```

加载文档

```
def load_documents(directory):
    documents = {}
    for filename in os.listdir(directory):
        if filename.endswith('.txt'):
            filepath = os.path.join(directory, filename)
            with open(filepath, 'r', encoding='utf-8') as file:
                content = file.read()
                documents[filename] = content
    print("load document ok")
    return documents
```

3.2.2 查询处理模块

系统接收用户输入的查询，先进行分词，提取名词，并将其转换为向量表示。

提取名词函数

```
def extract_nouns(text):
    words = pseg.cut(text)
    return [word for word, flag in words if flag.startswith('n')]
```

将查询转换为向量

```
   def query_to_vector(query, idf, vocabulary):
    tokens = extract_nouns(query)
    tf = Counter(tokens)
    vector = np.zeros(len(vocabulary))
    token_index = {token: idx for idx, token in enumerate(vocabulary)
}
    for token, count in tf.items():
        if token in token_index:
            vector[token_index[token]] = count * idf.get(token, 0.0)

    return vector
```

3.2.3 检索模块

系统使用计算好的查询向量在文档库中检索相关文档，计算文档与查询的余弦相似度并排序。

计算余弦相似度

```
def compute_cosine_similarity(vec1, vec2):
    dot_product = np.dot(vec1, vec2)
    norm1 = np.linalg.norm(vec1)
    norm2 = np.linalg.norm(vec2)
    if norm1 == 0 or norm2 == 0:
        return 0.0
    return dot_product / (norm1 * norm2)
```

检索文档

```
def search(query, tfidf, idf, vocabulary):
    query_vector = query_to_vector(query, idf, vocabulary)
    scores = []
    for doc_id, doc_vector in tfidf.items():
        score = compute_cosine_similarity(query_vector, np.array(doc_vector))
        scores.append((doc_id, score))
    ranked_docs = sorted(scores, key=lambda item: item[1], reverse=True)
    return ranked_docs
```

3.2.4 显示结果

系统将检索到的文档结果显示给用户，包括文档的相关度、日期、URL 和匹配的句子。

提取日期

```
   def extract_date_from_url(url):
    match = re.search(r'/(\d{4}/\d{2}-\d{2})/', url)
    if match:
        return match.group(1).replace('/', '-')
    return "日期未知"
```

显示检索结果

```
def display_results(results, documents, query, top_n=5):
    doc_ids = []
    for idx, (doc_id, score) in enumerate(results[:top_n], 1):
        url = documents[doc_id].split('\n')[0]
        date = extract_date_from_url(url)
```

```
        print(f"{idx}. 文档: {doc_id} 相关度: {score:.4f} 日期: {date}")
        print(f"URL: {url}")

        content = documents[doc_id]
        sentences = content.split('\n')
        split_sentences = []
        for sentence in sentences:
            split_sentences.extend(sentence.split('。'))

        query_terms = set(extract_nouns(query))
        matching_sentences = [sentence for sentence in split_sentences if any(term in sentence for term in query_terms)]

        for sentence in matching_sentences[:2]:
            for term in query_terms:
                if term in sentence:
                    sentence = sentence.replace(term, f"\033[91m{term}\033[0m")
            print(f"-  {sentence.strip()}。")

        print("-" * 80)
        doc_ids.append(doc_id)

    return doc_ids
```

3.2.5 用户交互模块

用户可以选择查看文档全文或对文档进行评分。

```
def main():
    tfidf = load_json('./data/tfidf.json')
    idf = load_json("./data/idf.json")
    vocabulary = load_json("./data/vocabulary.json")
    documents = load_documents('./data/news')
    inverted_index = load_json('./data/inverted_index_table.json')
    ratings_filepath = './data/ratings.json'
    ratings = load_json(ratings_filepath)

    while True:
        query = input("请输入搜索内容（输入 'exit' 退出）: ")
        if query.lower() == 'exit':
```

```
            break
        results = search(query, tfidf, idf, vocabulary)
        if results:
            doc_ids = display_results(results, documents, query)
            while True:
                selection = input("请输入要查看全文的文档编号或输入评分（格式：编号 评分），输入 'exit' 返回查询：")
                if selection.lower() == 'exit':
                    break
                try:
                    if ' ' in selection:
                        doc_num, rating = selection.split()
                        doc_id = doc_ids[int(doc_num) - 1]
                        ratings[doc_id] = int(rating)
                        utils.save_json(ratings, ratings_filepath)
                        print(f"文档 {doc_id} 评分已保存：{rating}")
                    else:
                        doc_id = doc_ids[int(selection) - 1]
                        with open(f'./data/news/{doc_id}', 'r', encoding='utf-8') as file:
                            print(file.read())
                except (IndexError, ValueError):
                    print("无效输入，请输入有效的文档编号或评分。")
        else:
            print("未找到相关文档。")
```

4 系统运行效果展示

图 4-1 初始化成功界面示意图

图 4-2 用户输入搜索信息画面示意图

图 4-3 检索结果示意图

图 4-4 查看原文示意图

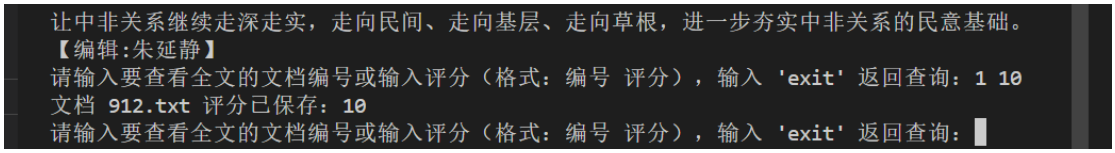


图 4-5 人工评价示意图

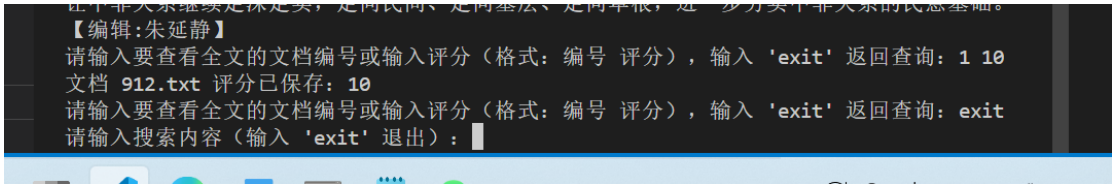


图 4-6 返回查询示意图

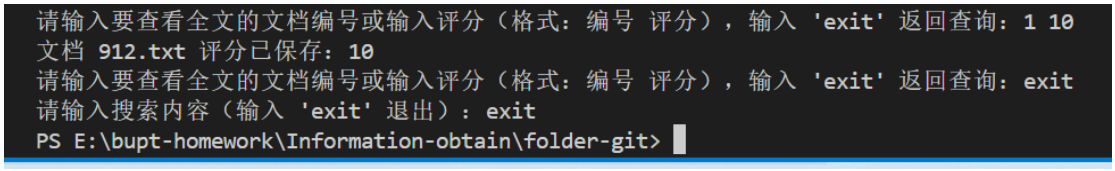


图 4-7 输入 exit 退出检索系统示意图

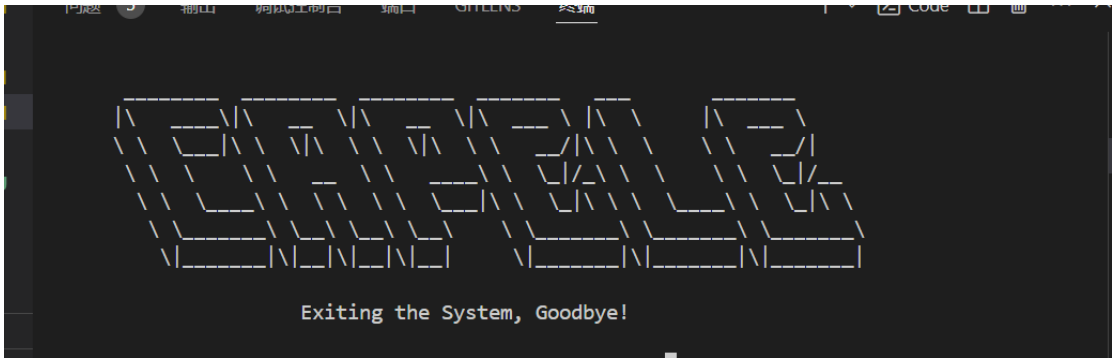


图 4-8 系统退出结束画面示意图