

武汉大学国家网络安全学院

实验报告

课程名称 信 息 检 索 (跨)

专业年级 网安 21 级

姓 名 李申选

学 号 2021302181070

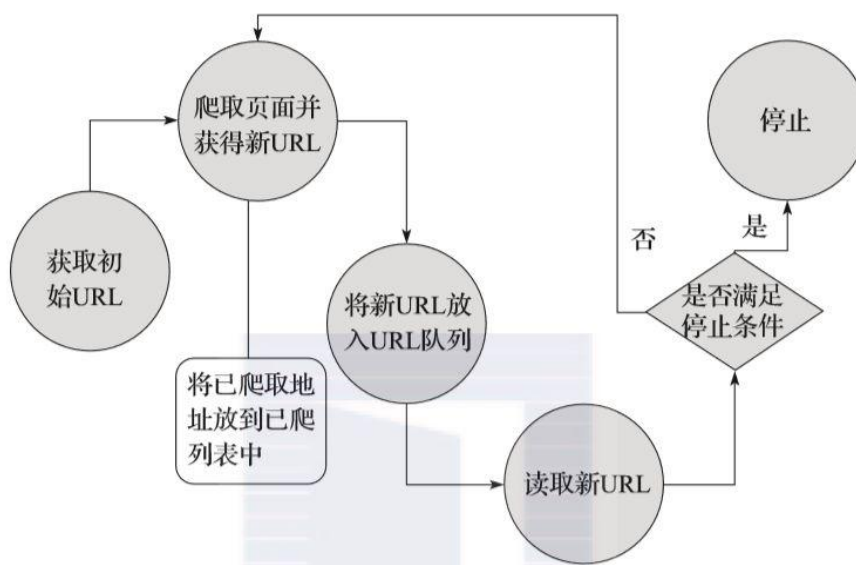
协 作 者 无

实验学期 2023-2024 学年 第二 学期

课堂时数 32 课外时数 12

填写时间 2024 年 5 月 14 日

实验介绍
<p>【实验名称】：新闻联播信息检索系统</p> <p>【实验目的】： 实现支持布尔查询和向量空间模型查询的本地新闻联播信息检索系统</p> <p>【实验环境】： Windows11 + Anaconda + PyCharm</p> <p>【参考文献】：</p>
实验内容
<p>【实验方案设计】：</p> <p>一. 实验原理</p> <p>一个本地的信息检索系统，需要有信息获取、文档预处理、倒排索引构建、布尔查询实现、向量空间模型实现等模块，下面将简单介绍各个模块的实现原理：</p> <p>(1) 信息和获取</p> <p>使用网络爬虫爬取目标网页的内容（合法使用）原理和示意图如下：</p> <ol style="list-style-type: none"> 获取初始的 URL。初始的 URL 地址可以由用户人为地指定，也可以由用户指定的某个或某几个初始爬取网页决定。 根据初始的 URL 爬取页面并获得新的 URL。获得初始的 URL 地址之后，首先需要爬取对应 URL 地址中的网页，爬取了对应的 URL 地址中的网页后，将网页存储到原始数据库中，并且在爬取网页的同时，发现新的 URL 地址，同时将已爬取的 URL 地址存放到一个 URL 列表中，用于去重及判断爬取的进程。 将新的 URL 放到 URL 队列中。在第 2 步中，获取了下一个新的 URL 地址之后，会将新的 URL 地址放到 URL 队列中。 从 URL 队列中读取新的 URL，并依据新的 URL 爬取网页，同时从新网页中获取新 URL，并重复上述的爬取过程。 满足爬虫系统设置的停止条件时，停止爬取。在编写爬虫的时候，一般会设置相应的停止条件。如果没有设置停止条件，爬虫则会一直爬取下去，一直到无法获取新的 URL 地址为止，若设置了停止条件，爬虫则会在停止条件满足时停止爬取。



此处爬取的网页结构简单，只需以下几步：

- 设置 URL：观察 URL 格式，设置爬取网页 URL 列表
- 编写代码抓取网页：使用 requests 库向目标网站发起请求，获取网页内容。
- 解析内容：通过 BeautifulSoup 库解析 HTML，提取需要的信息。
- 数据存储：将抓取到的数据存储到本地 txt 文件中。

（2）文档预处理

对爬取的中文文档进行分词处理，并去除停用词，以下为原理介绍：

jieba 分词系统，主要实现三个模块：分词、词性标注、关键词抽取

其中，分词有三种模式，默认是精确模式

- 精确模式，试图将句子最精确地切开，适合文本分析；
- 全模式，把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不能解决歧义；
- 搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词；

jieba 分词主要是基于统计词典，构造一个前缀词典；然后利用前缀词典对输入句子进行切分，得到所有的切分可能，根据切分位置，构造一个有向无环图；通

过动态规划算法，计算得到最大概率路径，也就得到了最终的切分形式。

1. 构建前缀词典：

基于统计词典构造前缀词典，统计词典有三列，第一列是词，第二列是词频，第三列是词性

2. 构建有向无环图：

根据前缀词典对输入文本进行切分，比如“北”，有北、北京、北京大学三种划分方式。因此，对于每个字，可以构建一个以位置为 key，相应划分的末尾位置构成的列表为 value 的映射。

3. 最大概率路径计算

在得到所有可能的切分方式构成的有向无环图后，我们发现从起点到终点存在多条路径，多条路径也就意味着存在多种分词结果。需要计算最大概率路径。

计算最大概率路径时，jieba 采用从后往前的方式，采用动态规划计算最大概率路径。每到达一个节点，它前面的节点到终点的最大路径概率就已经计算出来。

停用词（Stop words）是指在文本处理过程中被忽略或删除的常见词汇。这些词汇通常是频繁出现的功能词或无实际意义的词语，例如介词、连词、冠词、代词等。停用词通常对于文本的含义分析没有太大贡献，且会占据大量的存储空间和计算资源。因此，在文本处理任务（如文本分类、信息检索等）中，常常会预先定义一组停用词，并在处理过程中将它们从文本中移除。

本项目中使用的停用词来自于 <https://github.com/goto456/stopwords>

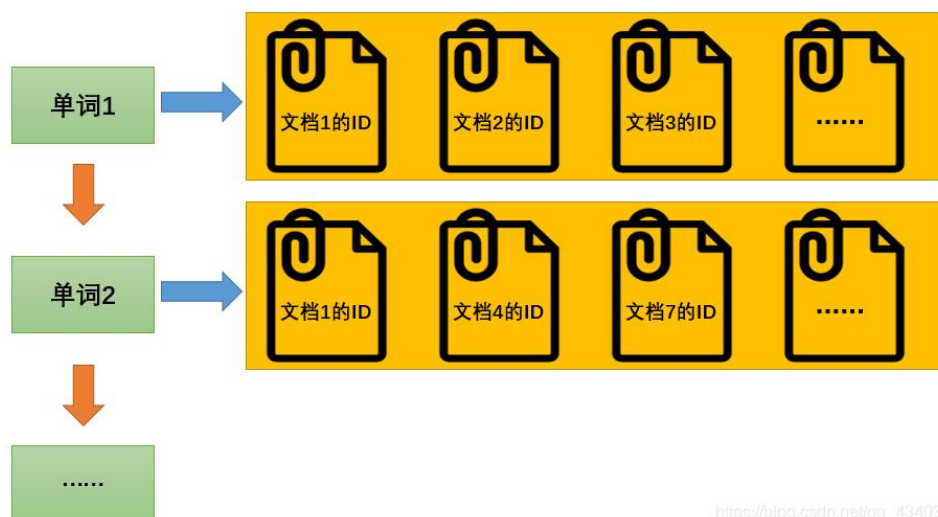
（3）倒排索引构建

正向索引如下：



https://blog.csdn.net/qq_43403025

对每篇文档构建一个索引，记录文中出现的词项及其频率，在日常的以词项为检索条件的搜索中很费时费力，因此，以词项为索引反向构建索引，如下：



https://blog.csdn.net/qq_43403025

此时我们进行布尔查询，只需对各索引对应的文档列表进行逻辑操作即可

(4) 布尔查询

布尔查询是指利用 AND, OR 或者 NOT 操作符将词项连接起来的查询，查询结果只有真和假两种

布尔查询在倒排表上的操作

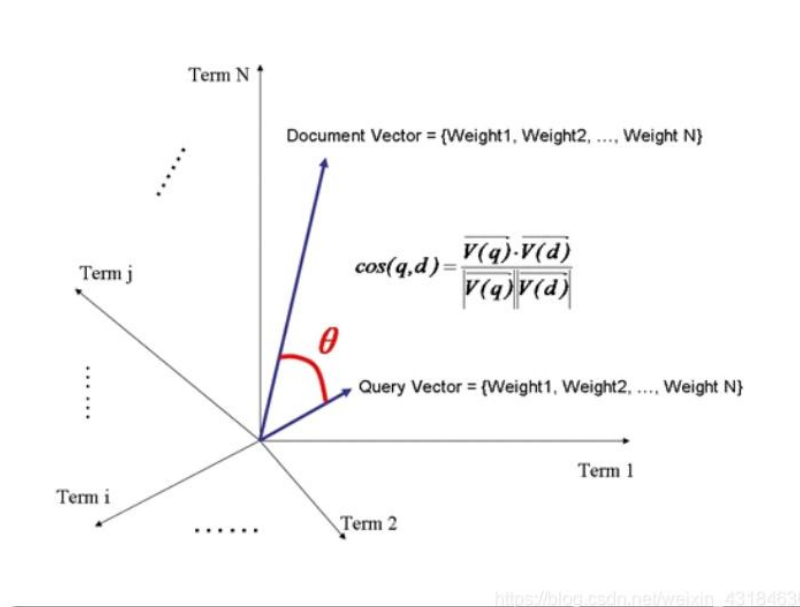
AND (Brutus AND Caesar)：两个倒排表的交集

OR (Brutus OR Caesar)：两个倒排表的并集

NOT (Brutus AND NOT Caesar)：两个倒排表的减集

(5) 向量空间模型

向量空间模型（VSM: Vector Space Model）由 Salton 等人于 20 世纪 70 年代提出，并成功地应用于文本检索系统。VSM 概念简单，把对文本内容的处理简化为向量空间中的向量运算，并且它以空间上的相似度表达语义的相似度，直观易懂。当文档被表示为文档空间的向量，就可以通过计算向量之间的相似性来度量文档间的相似性。文本处理中最常用的相似性度量方式是余弦距离。我们认为两个向量之间的夹角越小，相关性越大。所以我们计算夹角的余弦值作为相关性的打分，夹角越小，余弦值越大，打分越高，相关性越大，原理如下图：



二. 实验步骤

1. 从 <http://mrwlb.com> 中爬取近 30 天的新闻数据

首先观察并构造目标 URL 列表，爬取最近三十天的数据

```
# 基本配置
base_url = "http://mrwlb.com/"
output_dir = "../News_Database"
days_to_fetch = 30
user_agent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3"
```

```
def fetch_news_for_date(date):
    year = date.year
    month = date.month
    day = date.day
    chinese_date = date.strftime("%Y年%m月%d日")
    url = f"{base_url}/{year}/{month}/{day}/{chinese_date}新闻联播文字版/"
    headers = {"User-Agent": user_agent}
    print(f"Fetching news from URL: {url}") # Debug: print the URL
```

此处的 `user_agent` 用户代理是为了防止同一用户重复请求而被拒绝

接下来使用 `requests` 库向目标网站发起请求，获取网页内容，然后通过 `BeautifulSoup` 库解析 HTML，提取目标内容

```
27 response = requests.get(url, headers=headers)
28 if response.status_code == 200:
29     soup = BeautifulSoup(response.content, features='html.parser')
30     content_div = soup.find(name='div', class_='entry-content')
31     if content_div:
32         return content_div.get_text(strip=True)
33 else:
34     print(f"Error: Received status code {response.status_code} for URL: {url}") # Debug: print the status code
35     print(f"Response content: {response.content.decode('utf-8')}") # Debug: print the response content
36 return None
```

注意此处的 `class`，应和爬取的内容的标签一致，可在网页源码中找到

```
118 <div class="entry-content">
119 <p><strong>20240522今日新闻联播主要内容:</strong></p>
120 <ul>
121 <li>习近平向越南新任国家主席苏林致贺电</li>
122 <li>习近平向第14届中美旅游高层对话开幕致信</li>
123 <li>【新思想引领新征程】厚植绿色底色建设美丽中国</li>
124 <li>1—4月铁路运输业增动力添活力</li>
125 <li>4月份机械工业运行稳定向好</li>
126 <li>三大粮食作物两种农业保险政策在全国全面实施</li>
127 <li>第二十届文博会各项工作准备就绪</li>
128 <li>国台办：台湾地区领导人“5·20”讲话是彻头彻尾的“台独自白”</li>
129 <li>台湾各界批评台湾地区领导人“5·20”讲话严重损害两岸关系和平前景</li>
130 <li>国内联播快讯</li>
131 <li>多国政府要重申坚定奉行一个中国原则</li>
132 <li>伊朗已故总统莱希的遗体告别仪式在德黑兰举行</li>
133 <li>以色列在加沙地带多地和约旦河西岸展开军事行动</li>
134 <li>联合国举行“国际茶日”庆祝活动</li>
135 <li>国际联播快讯</li>
136 </ul>
137 <p><span style="color:red;">以下为详细的文字版全文:</span></p>
138 <p><strong>习近平向越南新任国家主席苏林致贺电</strong></p>
139 <p>5月22日，国家主席习近平致电苏林，祝贺他就任越南社会主义共和国主席。</p>
140 <p>习近平指出，中越是山水相连的社会主义友好邻邦。去年我对越南进行国事访问，同阮富仲总书记共同宣布构建具有战略意义的中越命运共同体，开辟1
141 <p><strong>习近平向第14届中美旅游高层对话开幕致信</strong></p>
142 <p>5月22日，国家主席习近平向第14届中美旅游高层对话开幕致信。</p>
143 <p>习近平指出，今年是中美建交45周年。中美关系的根基由人民浇筑，中美关系的大门由人民打开，中美关系的故事由人民书写，中美关系的未来也必将由
144 <p>习近平强调，旅游是中美两国人民交往交流、相知相近的重要桥梁。我们热情欢迎美国游客来华旅行，结识中国朋友、体验中华文化、游览美丽山水，共
145 <p>第14届中美旅游高层对话当日在陕西省西安市开幕，主题为“旅游促进中美人文交流”，由中国文化和旅游部、陕西省人民政府、美国商务
```

然后将爬取的内容保存到本地文件中

```
def save_news(date, content):
    file_name = date.strftime("%Y-%m-%d") + ".txt"
    file_path = os.path.join(output_dir, file_name)
    with open(file_path, 'w', encoding='utf-8') as file:
        file.write(content)
```

综合如下：

```
# 爬取最近30天的新闻
current_date = datetime.now()
for i in range(days_to_fetch):
    target_date = current_date - timedelta(days=i)
    news_content = fetch_news_for_date(target_date)
    if news_content:
        save_news(target_date, news_content)
        print(f"Saved news for {target_date.strftime('%Y-%m-%d')}")
    else:
        print(f"No news found for {target_date.strftime('%Y-%m-%d')}")
# 增加随机间隔以避免被检测为爬虫
time.sleep(random.uniform(a=1, b=3))

print("News fetching completed.")
```

其中，增加随机时间间隔以反爬虫

2.数据预处理（分词，去除停用词）及其可视化

停用词列表

	search_engine.py	main.py	document_processor.py	stop_words.txt ×
1707	摘要			
1708	非但			
1709	非常			
1710	非徒			
1711	非得			
1712	非特			
1713	非独			
1714	靠			
1715	顶多			

加载停用词表

```
def load_stop_words(self, stop_words_path):
    """
    加载停用词列表。
    :param stop_words_path: 停用词列表文件路径
    :return: 停用词列表
    """
    stop_words = []
    if stop_words_path:
        with open(stop_words_path, 'r', encoding='utf-8') as file:
            for line in file:
                stop_words.extend(line.strip())
    return stop_words
```

加载文档


```
def load_documents(self):
    """
    加载目录中的所有TXT文档。
    """
    for doc_id, filename in enumerate(os.listdir(self.directory)):
        if filename.endswith('.txt'):
            filepath = os.path.join(self.directory, filename)
            with open(filepath, 'r', encoding='utf-8') as file:
                content = file.read()
                self.documents.append(content)
            self.doc_names[doc_id] = filename # 保存文档名
```

对文档进行预处理，分词并去除标点和停用词

```
def preprocess_documents(self):
    processed_documents = []
    for doc in self.documents:
        # 使用结巴分词进行中文分词
        words = jieba.cut(doc)
        # 去除停用词和非字母字符
        filtered_words = [word for word in words if word.isalpha() and word not in self.stop_words]
        processed_documents.append(' '.join(filtered_words))
    self.documents = processed_documents
```

对处理完的数据集进行可视化

生成词云：

```
def visualize_wordcloud(documents, font_path):
    """
    生成文档集的词云。
    """
    text = ' '.join(documents)
    wordcloud = WordCloud(font_path=font_path, background_color='white').generate(text)
    plt.figure(figsize=(10, 7))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.show()
```

可视化最常见的 top_n 个词项及其频率

```
def visualize_top_words(vectorizer, term_document_matrix, top_n, font_path):
    """
    可视化最常见的top_n个词项及其频率。
    """
    sum_words = term_document_matrix.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vectorizer.vocabulary_.items()]
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)[:top_n]
    words, freqs = zip(*words_freq)

    # 加载中文字体
    prop = fm.FontProperties(fname=font_path)

    plt.figure(figsize=(10, 8))
    plt.bar(words, freqs)
    plt.xticks(rotation=45, fontproperties=prop)
    plt.xlabel(xlabel='Words', fontproperties=prop)
    plt.ylabel(ylabel='Frequency', fontproperties=prop)
    plt.title(label='Top Words Frequency', fontproperties=prop)
    plt.show()
```

3.倒排索引构建及其可视化

```
def build_inverted_index(self, documents):
    """
    构建倒排索引
    """
    for doc_id, doc in enumerate(documents):
        for word in doc.split():
            self.inverted_index[word].add(doc_id)
```

打印倒排索引表

```
def visualize_inverted_index(inverted_index):
    """
    可视化倒排索引表。
    """
    print("倒排索引: ")
    for term, doc_ids in sorted(inverted_index.items()):
        print(f"词项 '{term}': 文档列表 {doc_ids}")
```

4.布尔查询

```

def search(self, query, use_boolean_search=False):
    if use_boolean_search:
        # 解析布尔查询
        query_parts = query.split()
        result_docs = None
        current_operator = None

        for part in query_parts:
            if part in ['AND', 'OR', 'NOT']:
                current_operator = part
            else:
                term_docs = set(self.inverted_index.get(part, []))
                if result_docs is None:
                    result_docs = term_docs
                else:
                    if current_operator == 'OR':
                        result_docs |= term_docs
                    elif current_operator == 'NOT':
                        result_docs -= term_docs
                    else: # Default is AND
                        result_docs &= term_docs

        return [self.doc_names[doc_id] for doc_id in result_docs] if result_docs else []

```

5. 向量空间模型

构建词项—文档矩阵

```

def build_term_document_matrix(self, documents, doc_names):
    """
    构建词项—文档矩阵
    """
    self.term_document_matrix = self.vectorizer.fit_transform(documents)
    self.doc_names = doc_names

```

使用向量空间模型进行查询

首先，使用已经训练好的 TF-IDF 向量化器（`self.vectorizer`）将用户输入的查询（`query`）转换成向量形式（`query_vector`）。将查询文本转换成与文档集中文档相同的向量空间模型，从而使得查询可以在该空间中进行。接着，使用词项-文档矩阵（`self.term_document_matrix`）与查询向量的转置（`query_vector.T`）进行矩阵乘法运算，并将结果转换为一维数组（`scores`）。这一步的目的是计算查询向量与文档集中每个文档之间的相似度得分。相似度得分通常表示为查询与文档之间的余弦相似度，这里通过 TF-IDF 向量化后的矩阵乘法来实现。然后，将文档的索引和对应的相似度得分组成元组，按照得分从高到低进行排序（`results = sorted(enumerate(scores), key=lambda x: x[1], reverse=True)`）。这样可以确保返回的结果是按照与查询相似度最高的文档开始排列的。最后从排序后的

结果中筛选出得分大于 0 的文档（即与查询至少有一定程度相似的文档），并返回这些文档的名称（`self.doc_names[i]`）和相应的得分。这一步确保了只有与查询相关的文档才会被返回给用户。

```
else:
    # 处理普通查询
    query_vector = self.vectorizer.transform([query])
    scores = (self.term_document_matrix * query_vector.T).toarray().flatten()
    results = sorted(enumerate(scores), key=lambda x: x[1], reverse=True)
    return [(self.doc_names[i], score) for i, score in results if score > 0]
```

6.综合

首先初始化，接着加载并预处理文档，然后构建索引，进行查询，添加菜单以方便查询

```
if __name__ == '__main__':
    # 初始化文档处理器和搜索引擎
    doc_processor = DocumentProcessor(config.DIRECTORY, config.STOP_WORDS_PATH)
    search_engine = SearchEngine(stop_words=doc_processor.stop_words)

    # 加载和预处理文档
    doc_processor.load_documents()
    doc_processor.preprocess_documents()

    # 构建TF-IDF矩阵和倒排索引
    search_engine.build_term_document_matrix(doc_processor.documents, list(doc_processor.doc_names.values()))
    search_engine.build_inverted_index(doc_processor.documents)

    # 交互式查询
    while True:
        print("\n菜单:")
        print("1. 生成词云          2. 可视化最常见的30个词项及其频率")
        print("3. 可视化倒排索引      4. 进行查询          5. 退出")
        choice = input("请选择一个操作: ")

        if choice == '1':
            viz.visualize_wordcloud(doc_processor.documents, config.FONT_PATH) # 生成词云
        elif choice == '2':
            viz.visualize_top_words(search_engine.vectorizer, search_engine.term_document_matrix, top_n=30, config.FONT_PATH) # 可视化
        elif choice == '3':
            viz.visualize_inverted_index(search_engine.inverted_index) # 可视化倒排索引
        elif choice == '4':
            use_boolean_search = input("是否使用布尔查询 (AND 操作)? (y/n): ").strip().lower() == 'y'
            query = input("请输入查询词: ")
            results = search_engine.search(query, use_boolean_search)
            if results:
                if use_boolean_search:
                    print(f"找到了 {len(results)} 个包含查询词 '{query}' 的文档:")
                    for i, doc_name in enumerate(results, 1):
                        print(f"{doc_name}", end='\t')
                        if i % 4 == 0: # 每四个结果换行
                            print()
                    if len(results) % 4 != 0: # 如果结果数量不是4的倍数，需要手动换行
                        print()
                else:
                    print(f"找到了 {len(results)} 个包含查询词 '{query}' 的文档:")
                    for i, doc_name in enumerate(results, 1):
                        print(f"{doc_name}", end='\t')
                        if i % 4 == 0: # 每四个结果换行
                            print()
                    if len(results) % 4 != 0: # 如果结果数量不是4的倍数，需要手动换行
                        print()
            else:
                print("没有找到任何文档。")
        elif choice == '5':
            break
```

```














else:
    print(f"找到了 {len(results)} 个包含查询词 '{query}' 的文档:")
    for i, (doc_name, score) in enumerate(results, 1):
        print(f"{doc_name} (评分: {score:.4f})", end='\t')
        if i % 4 == 0: # 每四个结果换行
            print()
    if len(results) % 4 != 0: # 如果结果数量不是4的倍数, 需要手动换行
        print()
else:
    print(f"抱歉, 没有找到包含查询词 '{query}' 的文档。")
print()
elif choice == '5':
    print("欢迎下次使用!!!")
    break
else:
    print("无效的选择, 请重新输入。")

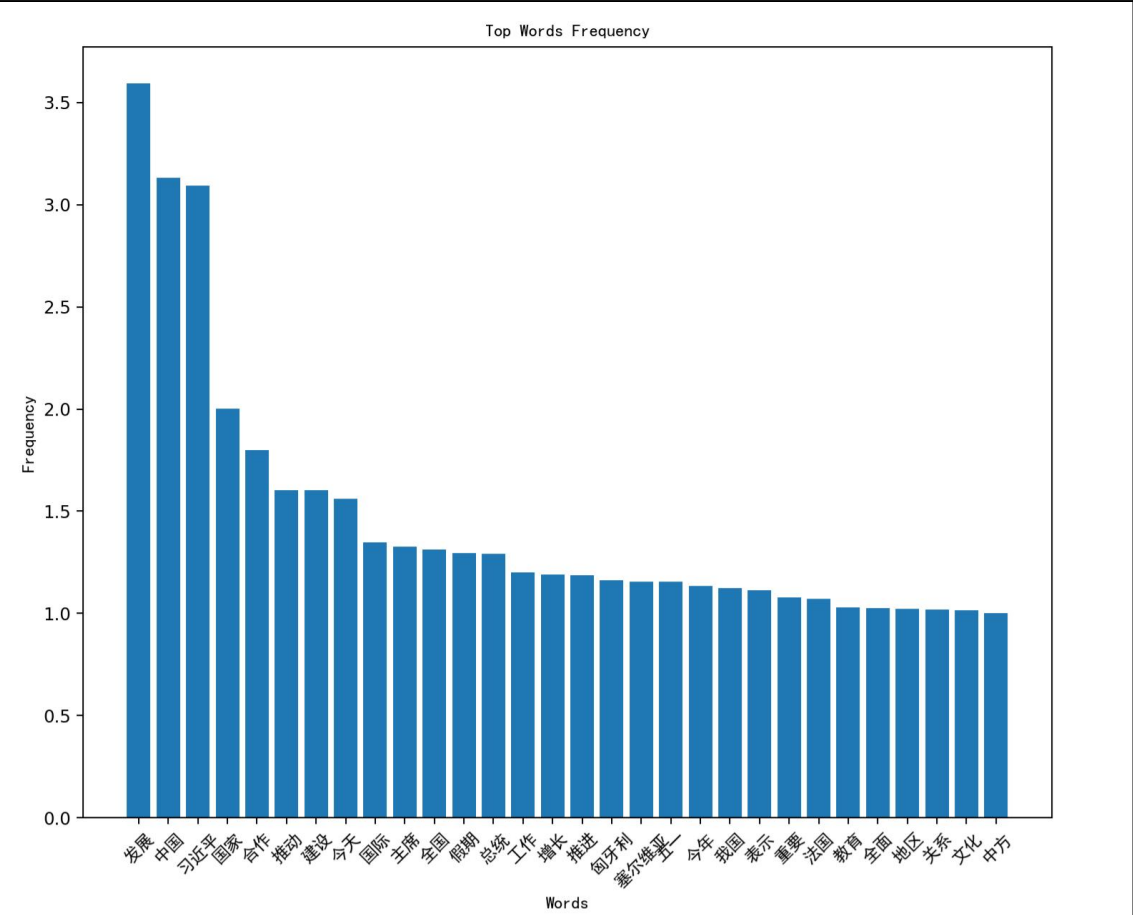
```

【实验结果分析】：

1. 数据获取

Saved news for 2024-04-29
 Fetching news from URL: <http://mrxmlb.com/2024/4/28/2024>年04月28日新闻联播文字版/
 Saved news for 2024-04-28
 Fetching news from URL: <http://mrxmlb.com/2024/4/27/2024>年04月27日新闻联播文字版/
 Saved news for 2024-04-27
 Fetching news from URL: <http://mrxmlb.com/2024/4/26/2024>年04月26日新闻联播文字版/
 Saved news for 2024-04-26
 Fetching news from URL: <http://mrxmlb.com/2024/4/25/2024>年04月25日新闻联播文字版/
 Saved news for 2024-04-25
 Fetching news from URL: <http://mrxmlb.com/2024/4/24/2024>年04月24日新闻联播文字版/
 Saved news for 2024-04-24
 News fetching completed.

 2024-05-10.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>
 2024-05-11.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>
 2024-05-12.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>
 2024-05-13.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>
 2024-05-14.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>
 2024-05-15.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>
 2024-05-16.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>
 2024-05-17.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>
 2024-05-18.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>
 2024-05-19.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>
 2024-05-20.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>
 2024-05-21.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>
 2024-05-22.txt	2024/5/23 14:54	TXT 文件	<u>0.1 MB</u>



3. 倒排索引构建及其可视化

词项 '风采': 文档列表 {9, 10, 11, 12}

词项 '风险': 文档列表 {0, 1, 3, 4, 8, 10, 11, 12, 14, 15, 16, 17, 18, 19, 21, 24, 25}

词项 '风险投资': 文档列表 {8}

词项 '风险管理': 文档列表 {21}

词项 '风雨': 文档列表 {15, 16, 19, 24, 25}

词项 '风雨同舟': 文档列表 {16, 17, 19}

词项 '风韵': 文档列表 {9}

词项 '风风雨雨': 文档列表 {4}

词项 '飓风': 文档列表 {0}

词项 '飘扬': 文档列表 {16, 17, 15}

词项 '飘香': 文档列表 {16}

词项 '飞': 文档列表 {29, 7}

词项 '飞临': 文档列表 {16}

词项 '飞入': 文档列表 {30}

词项 '飞机': 文档列表 {9, 12, 29, 7}

词项 '飞船': 文档列表 {1, 2, 3, 4, 6, 8, 9, 10}

词项 '飞花': 文档列表 {26}

词项 '飞行': 文档列表 {1, 2, 3, 6, 7, 8, 9, 11, 16, 19, 20}

词项 '飞行员': 文档列表 {9, 12}

4. 布尔查询

支持无嵌套的简单查询

是否使用布尔查询 (AND 操作)? (y/n): y

请输入查询词 : 发展 NOT 美国 AND 假期 OR 塞尔维亚

找到了 15 个包含查询词 '发展 NOT 美国 AND 假期 OR 塞尔维亚' 的文档:

2024-04-25.txt 2024-04-29.txt 2024-04-30.txt 2024-05-01.txt
2024-05-02.txt 2024-05-03.txt 2024-05-04.txt 2024-05-05.txt
2024-05-06.txt 2024-05-07.txt 2024-05-08.txt 2024-05-09.txt
2024-05-11.txt 2024-05-19.txt 2024-05-22.txt

词项 '发展': 文档列表 {0, 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}

词项 '美国': 文档列表 {0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 19, 21, 22, 24, 25, 26, 27, 28, 29, 30}

词项 '假期': 文档列表 {7, 8, 9, 10, 11, 12, 13, 14, 15, 30}

词项 '塞尔维亚': 文档列表 {3, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 27, 30}

经验证, 查询结构正确

5. 向量空间模型查询

是否使用布尔查询 (AND 操作)? (y/n): n

请输入查询词 : 黄河流域

找到了 3 个包含查询词 '黄河流域' 的文档:

2024-04-29.txt (评分: 0.0315) 2024-05-20.txt (评分: 0.0285) 2024-05-10.txt (评分: 0.0119)

词项 '黄河': 文档列表 {0, 3, 6, 10, 13, 18, 27, 28}

词项 '黄河壶口瀑布': 文档列表 {11}

词项 '黄河流域': 文档列表 {18, 28, 7}

词项 '黄海': 文档列表 {9}

词项 '黄淮': 文档列表 {18, 12, 21}

6. 项目结构

```
----Information-Retrieval-System\  
|----__init__.py  
|----news_get\  
|    |----news_get.py      // 信息获取  
|    |----__init__.py  
|----main.py              // 主函数  
|----document_processor.py // 文档处理  
|----search_engine.py     // 索引和检索  
|----visualization.py     // 可视化  
|----config.py            // 路径配置文件  
|----News_Database\  
|    |----2024-04-22.txt  
|    |----2024-04-23.txt  
|    |-----..  
|    |----2024-05-20.txt  
|    |----2024-05-21.txt  
|    |----2024-05-22.txt  
|----requirements.txt     // 项目配置  
|----stop_words.txt       // 停用词  
|----README.md            // README文件
```

【实验总结】:

通过本次信息检索实验, 巩固了之前学习到的理论知识, 也增强了实战能力, 在

这个过程中，我对之前学到的东西有了更深刻的理解，从数据集的获取，到索引的构建，一步一步搭建起了自己的信息检索系统，虽然很简陋，有很大的进步空间，但这是一个很好的开端，希望以后能继续学习相关知识，不断完善巩固这个信息检索系统。

评语及评分（指导教师）

【评语】：

评分：

日期：

附件：

实验报告说明

- 1. 实验名称：**要用最简练的语言反映实验的内容。
- 2. 实验目的：**目的要明确，要抓住重点。
- 3. 实验环境：**实验用的软硬件环境（配置）。
- 4. 实验方案设计（思路、步骤和方法等）：**这是实验报告极其重要的内容。包括概要设计、详细设计和核心算法说明及分析，系统开发工具等。应同时提交程序或设计电子版。

对于**设计型和综合型实验**，在上述内容基础上还应该画出流程图、设计思路和设计方法，再配以相应的文字说明。

对于**创新型实验**，还应注明其创新点、特色。

- 5. 实验结果分析：**即根据实验过程中所见到的现象和测得的数据，进行对比分析并做出结论（可以将部分测试结果进行截屏）。

6. 实验总结：对本次实验的心得体会，所遇到的问题及解决方法，其他思考和建议。

7. 评语及评分：指导教师依据学生的实际报告内容，用简练语言给出本次实验报告的评价和价值。