综合实训报告

Real Time News Topic Heat Analysis System and Implementation

题目：实时新闻的主题热度分析系统与实现

组长：王滢 （2023317220624）

组员：雷博琦（2023317220506）

易安麒（2023317220302）

指导老师：刘世超

中国 • 武汉

2025 年 6 月

1 项目基本信息

1.1 研究背景以及意义

1.1.1 研究背景

随着互联网的普及，信息传播速度和范围不断扩大，每天产生的新闻和信息呈爆炸式增长。海量数据使得人们难以快速获取和理解关键信息；社交媒体、新闻网站、论坛、自媒体等平台的兴起，让信息传播渠道更加多元化，公众获取新闻的方式也更加多样，实时热点更容易迅速形成；大数据处理技术的成熟，使得对海量新闻数据进行高效采集、存储、处理和分析成为可能，为实时新闻热度分析提供了技术基础。

1.1.2 研究意义

我们设计的系统可以抓取网上的新闻，提取出关键词并进行高频词展示。在这个信息海量的时代，用户可以利用我们的系统更快定位重要信息，了解实时热点；媒体可以利用我们的系统更快发现热点事件，进行报道；企业可以利用我们的系统了解人们的需求和兴趣来设计和制造产品；政府可以利用我们的系统及时了解舆论趋势，更好进行舆论管理和谣言处理。

1.2 主要任务与分工

1.2.1 主要任务

1.主题分析系统结构层次：

层次	模块	实现功能
数据采集层	新浪新闻爬虫	基于爬虫脚本，从新浪新闻网站自动化采集大规模新闻文本数据，涵盖标题、内容、发布时间、来源等关键信息。
	观察者网爬虫	实现对观察者网新闻页面的结构化解析与信息提取，构建面向多源新闻数据的高频更新采集通道。
数据存储层	新闻数据存储	采用MySQL数据库对原始新闻数据进行结构化存储与管理，确保数据的完整性与查询效率。
	新闻数据整合	对新闻表进行统计分析，生成高频词统计表、新闻来源统计表以及词文档关系表，为后续建模与可视化分析提供高质量输入数据。
	词向量库加载	加载已有高质量中文词向量库（百度百科词向

算法建模层		量），为后续语义相似度计算提供向量基础。
	对新闻内容进行jieba,TextRank分词	对新闻正文应用 Jieba 分词算法与 TextRank 排名算法，自动提取每篇新闻中最具代表性的5个核心关键词。
	每篇新闻的平均词向量计算	基于关键词在词向量库中的词向量，计算单篇新闻的平均语义向量，实现对新闻内容的向量化建模。
分析计算层	查询相关搜索排序	将用户查询内容进行分词并计算平均词向量，与数据库中每篇新闻的平均词向量计算余弦相似度，按相关度排序，返回前十条最相关的新闻结果。
	分析和可视化	利用 ECharts 实现多维数据可视化，包括媒体来源分布、热度对比、关键词权重等，提升分析直观性与用户交互体验。
前端应用层	前端可视化交互	通过 HTML、CSS 与 JavaScript 构建响应式前端界面，结合 Flask 框架完成前后端数据联动，实现关键词查询、热度分析等功能模块的可视化操作入口。

2.**主题系统主要功能：**

（1）使用多线程和丰富的代理池爬取新浪新闻的观察者网新闻，并将爬取的新闻数据存储到mysql数据库中。

（2）加载中文高质量词向量库，利用jieba分词算法和textrank排序算法每篇文章提取5个关键词进行新闻内容向量化建模，将用户查询内容进行分词并计算平均词向量，与数据库中每篇新闻的平均词向量计算余弦相似度，按相关度排序，返回前十条最相关的新闻结果。

（3）利用 ECharts 实现多维数据可视化，生成词云、热度分析图、来源饼状图、质量分析图和前20个高频词汇图。

（4）通过 HTML、CSS 与 JavaScript 构建响应式前端界面，结合 Flask 框架完成前后端数据交互，实现关键词查询、热度分析等功能模块的可视化操作入口。

1.2.2 分工

王滢	数据爬虫部分、新闻数据整合、搜索功能（时间选择）、前端页面设计和后端连接、实验报告撰写

雷博琦	数据爬虫部分、倒排表建立、使用jieba TextRank 提取关键词、相关度查询、前端页面设计和后端连接、实验报告撰写
易安麒	数据爬虫部分、词云设计、词到文档、实验报告撰写

1.3 运行环境和开发工具

1.3.1 系统开发工具

利用 Python 实现全部的开发。

1.3.2 数据库工具

Mysql数据库。

2 项目设计与实现

2.1数据采集

2.1.1新浪新闻爬虫

基于新浪新闻的滚动新闻网址，利用 Python 的 request进行多线程爬取每条新闻的 HTML 正文，并利用beautifulsoup 对收集到的 HTML 正文进行处理，获取某一段时间内的新闻标题、时间、链接、发布来源、游览量、评论数以及新闻正文部分。

爬取步骤：

1.选取核心网址

选择新浪新闻的滚动新闻网址，完成数据爬取采集

```
def main():
    baseurl = "https://feed.mix.sina.com.cn/api/roll/get?
pageid=153&lid=2509&k=&num=50&page="
```

这是新浪新闻“国内新闻”频道的分页接口，每页会返回 50 条新闻列表（JSON格式）

2.多线程爬虫

创建 10 个线程，每个线程执行线程函数 thread_task ，多线程使用共享变量来分配不同页码，避免重复爬取，并通过线程锁保证同一时刻仅一个线程可以修改页码。

tread_task部分：

```
while True:
    with lock:
        global page_counter
        if page_counter >= 2000:
            break
        current_page = page_counter
        page_counter += 1
```

3. 获取新闻详情页

```
data = requests.get(url, headers=headers)
data_json = json.loads(data.content)
news = data_json.get("result").get("data")
```

4. 解析新闻详情页

每条新闻调用 askURL 函数，获取详情页的信息（如正文、时间、来源等）

```
page = requests.get(url, timeout=10).content
soup = BeautifulSoup(page, "lxml")
```

提取信息字段：

字段	提取方法
标题	`soup.find(class_="main-title")`
正文	`soup.find(class_="article").text` （清洗空格、尾部广告）
发布时间	`.find(class_="date-source").find_all("span")[0]`
来源	`.find(class_="date-source").a.text`
评论数、点击量	调用 `getCommentCounts()` 获取

5. 关于游览量与评论数的爬取

由于新浪新闻网的评论数与游览量是通过 JS 渲染后得到的，我们无法从新闻原 HTML 正文中直接爬取得到，经过观察可知，我们可以从该相关网址中获取相关数据：我们可以根据某一新闻获得其 channel 和 newsid 并进入对应的网址，由此便得到了需要的内容。

```
def getCommentCounts(url):
    commentURLPattern = "https://comment5.news.sina.com.cn/page/info?
version=1&format=js&channel={}&newsid={}&group=&compress=0&ie=utf-
8&oe=utf- 8&page=1&page_size=20"
    page_content = requests.get(url).content
    root = etree.HTML(page_content)

    comment_meta = root.xpath('//meta[contains(@content, "comment_channel")]/@content')
    comment_channel = None
    comment_id = None

    if len(comment_meta) > 0:
        comment_info = comment_meta[0].split(";")
        for c in comment_info:
            if "comment_channel" in c:
                comment_channel = c.split(":")[-
                1]
            elif "comment_id" in c:
                comment_id = c.split(":")[-
                1]

    commenturl = commentURLPattern.format(comment_channel, comment_id)
    comments = requests.get(commenturl)
    comments.encoding = 'utf-8'
```

最后可以得到总游览数和总评论数

2.1.2观察者新闻爬虫

核心功能模块解析：

1.请求管理模块

反爬机制:

使用随机 User-Agent（从58个游览器UA中随机选择）

请求失败重试机制（默认3次）

禁用SSL验证（verify=False）绕过某些网站证书检查

超时处理：设置30秒超时

头信息：包含标准浏览器头子段（Accept，Language等）

```
def request_with_retry(url, retries=3):
    for attempt in range(retries):
        headers = get_random_headers()  # 随机请求头
        try:
            resp = requests.get(url, headers=headers, timeout=30, verify=False)
            resp.raise_for_status()
            return resp
        except Exception as
        e:
            time.sleep(3)
```

2. url采集模块

函数：get_article_urls

```
for page in range(1,3):
    list_url = f'https://www.guancha.cn/mainnews-yw/list_{page}.shtml'
```

观察者网主页新闻是分页展示的，URL中的 list_1.shtml 、list_2.shtml 分别表示第 1、2 页。

```
html = etree.HTML(response.text)
links = html.xpath("//div[@class='right fn']/h4/a/@href")
```

使用 XPath 从 HTML 中提取每条新闻的相对路径 URL。
返回格式如：/politics/2024_05_20_739857.shtml

```
full_links = ['https://www.guancha.cn' + link for link in links]
```

最后构成完整的URL，便于后续请求。

3.解析文章详情页内容

字段名	字段说明	提取方法与选择器	清洗处理说明
网址	文章的完整URL	由主函数传入，每篇文章唯一标识	无需处理
标题	新闻标题	`soup.select_one('div.article-content > h1')``soup.find('h3')` 等备选方案	多策略选择，`.get_text(strip=True)` 去除空格
发布时间	发布的时间	`soup.select_one('span.time1')` 或 `div.time span`	`.get_text(strip=True)` 提取文本内容
内容	正文文本	`soup.select('div.article-txt-content p')` 或 `div.content p`	合并多段落文本；用正则去除广告内容、空白符号等
来源	来源平台名	默认为 `'观察者网'`	设为固定值
浏览量	阅读次数	API: `https://user.guancha.cn/news-api/view-count?postId={docid}`	JSON字段 `"view_count_text"`，如无返回设为 "0"
评论数	评论条数	API: `https://user.guancha.cn/comment/hot-comment-list-after-article?...`	正则提取 `"all_count":123` 等；如无返回设为 "0"
docid	文章唯一编号	正则从 HTML 或 URL 提取：`/(\d+)_\d+\.shtml`，或 `_DOC_ID = "..."` 等	仅用于 API 调用，不入库

关于浏览量和评论数的提取：

同样的，观察者新闻的浏览量与评论数同样是JS渲染，通过新闻HTML文本中无法提取。通过观察可以发现，在该新闻的移动端网址中可以发现浏览量和评论数都在如下网址中保存：

其中每篇新闻都有对应的唯一 docid，该数据可以从新闻的 HTML 正文中提取出来，修改上述链接并访问即可爬取相关数据。

2.2数据存储：

将爬取的新闻数据按照数据的类型存储到 **MySQL 数据库**中。

1. MySQL 介绍

MySQL 是一种开源的关系型数据库管理系统（RDBMS），广泛用于网站和应用开发中，以其高效稳定、跨平台和支持复杂查询而著称。MySQL 以结构化方式存储数据，每条记录必须遵循表结构中定义的字段类型和约束，因此在数据一致性和完整性方面具备良好保障。它支持 SQL（结构化查询语言）进行数据的增删改查和复杂事务处理。

2. 使用原理

（1）**面向表结构（Table-Oriented）：**

在 MySQL 中，所有数据都存储在一个或多个表中，每张表由若干列（字段）组成，表结构需在建表时预先定义，包括字段名称、数据类型、主键等。每条记录作为一行插入到表中。对于新闻数据，可以按需设计多张表，如 sina_news（新闻基本信息）、tifidf_vocab（关键词信息）等，以实现数据结构化存储和多表关联。

（2）**结构化与规范化（Structured & Normalized）：**

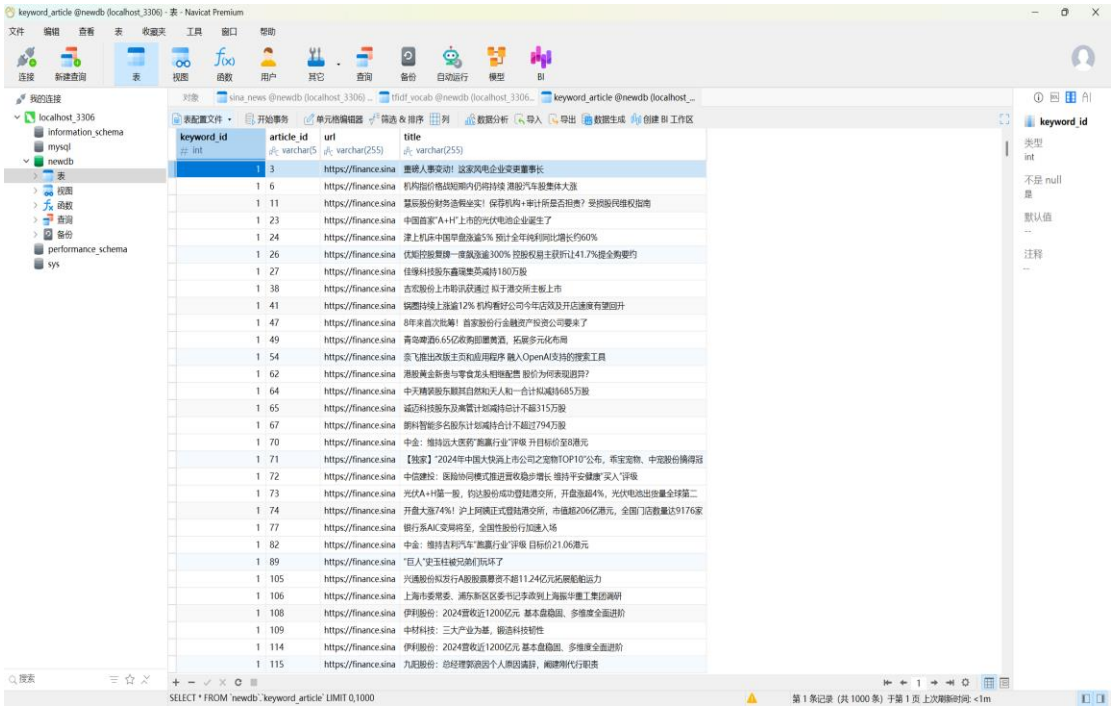
MySQL 强调数据结构的一致性和完整性，存储数据前必须定义表的字段结构。通过主外键关联、索引优化及规范化设计，可以避免冗余并提升查询效率。

（3）**SQL 查询与索引机制：**

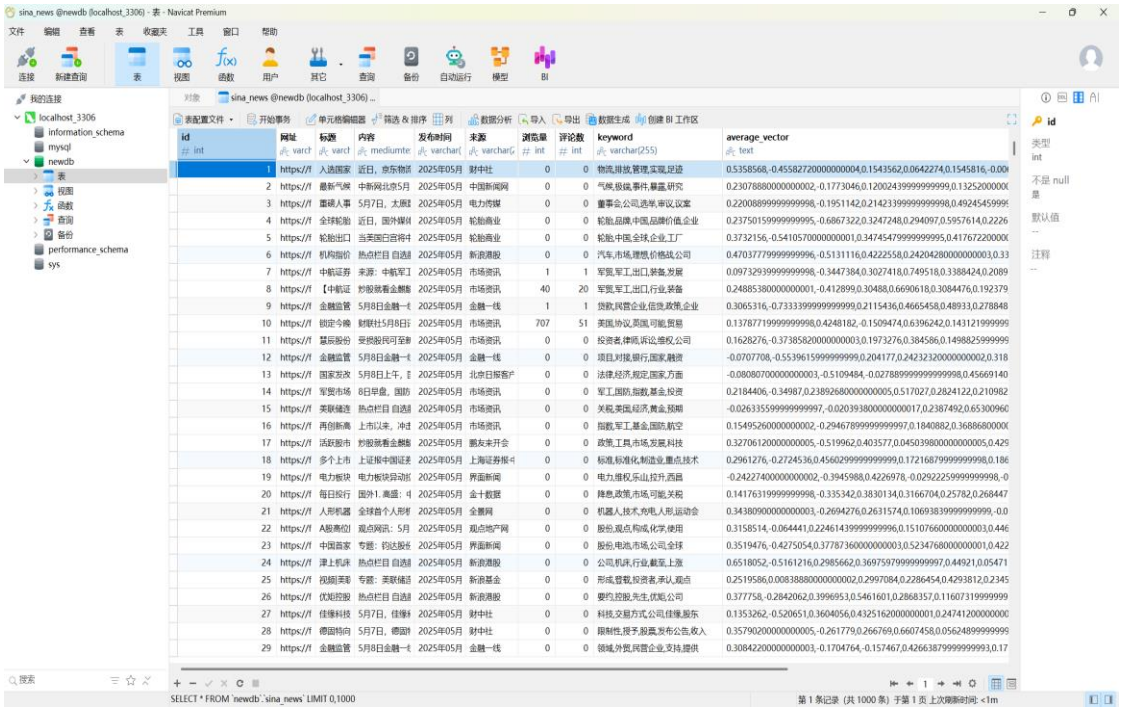
MySQL 支持结构化查询语言（SQL）对数据进行精确操作，包括多表联结（JOIN）、条件筛选（WHERE）、分组聚合（GROUP BY）等。同时支持对关键字段建立索引，显著提升大数据量条件查询的效率，尤其适合对新闻标题、关键词、来源等字段的快速检索操作。

利用MySQL存储数据，共4张表keyword_article、sina_news、source_counts和tfidf_vocab。

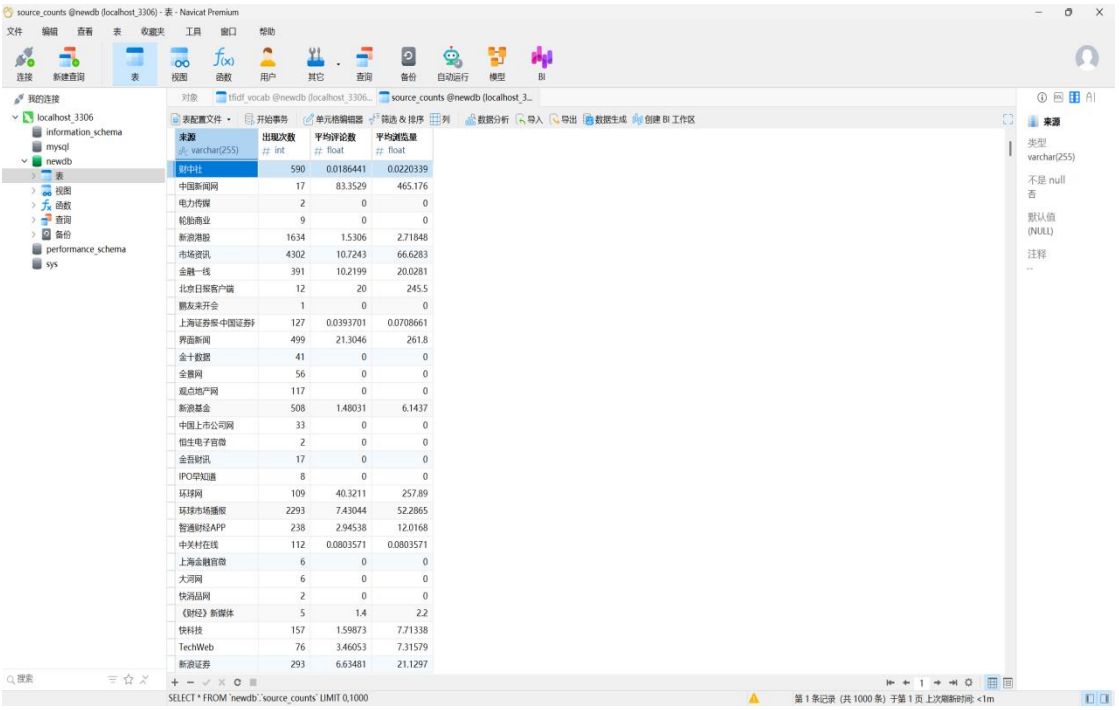
keyword_article有列keyword_id、article_id、url和title



sina_news有列id,网址、标题、内容、发布时间、来源、浏览量、评论、keyword和average_vector

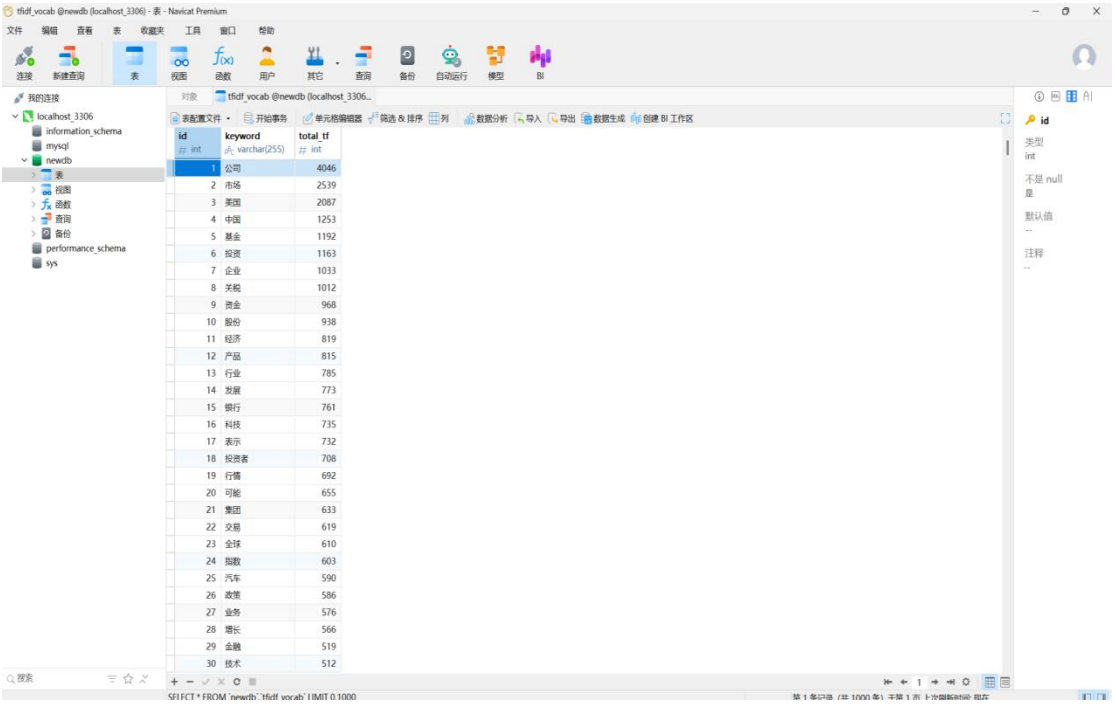


source_counts有列来源、出现次数、平均评论数和平均浏览量



来源	出现次数	平均评论数	平均浏览量
	varchar(255)	int	float	float
财中社	590	0.0186441	0.0220339
中国新闻网	17	83.3529	465.176
电力传媒	2	0	0
轮胎商业	9	0	0
新浪港股	1634	1.5306	2.71848
市场资讯	4302	10.7243	66.6283
金融一线	391	10.2199	20.0281
北京日报客户端	12	20	245.5
朋友来开会	1	0	0
上海证券报·中国证券网	127	0.0393701	0.0708661
界面新闻	499	21.3046	261.8
金十数据	41	0	0
全景网	56	0	0
观点地产网	117	0	0
新浪基金	508	1.48031	6.1437
中国上市公司网	33	0	0
恒生电子官微	2	0	0
金吾财讯	17	0	0
IPO早知道	8	0	0
环球网	109	40.3211	257.89
环球市场播报	2293	7.43044	52.2865
智通财经APP	238	2.94538	12.0168
中关村在线	112	0.0803571	0.0803571
上海金融官微	6	0	0
大河网	6	0	0
快消品网	2	0	0
《财经》新媒体	5	1.4	2.2
快科技	157	1.59873	7.71338
TechWeb	76	3.46053	7.31579
新浪证券	293	6.63481	21.1297

tfidf_vocab有列id、keyword和total_tf



id	keyword	total_tf
	int	varchar(255)	int
1	公司	4046
2	市场	2539
3	美国	2087
4	中国	1253
5	基金	1192
6	投资	1163
7	企业	1033
8	关税	1012
9	资金	968
10	股份	938
11	经济	819
12	产品	815
13	行业	785
14	发展	773
15	银行	761
16	科技	735
17	表示	732
18	投资者	708
19	行情	692
20	可能	655
21	集团	633
22	交易	619
23	全球	610
24	指数	603
25	汽车	590
26	政策	586
27	业务	576
28	增长	566
29	金融	519
30	技术	512

2.3数据分析与展示

2.3.1数据分析：

1. 关键词提取

- 关键词提取之前需要对文本进行分词处理，本项目中采取 jieba 分词去除停用词、标点、无意义词等。
- 构建图
  - 将分词得到的每一个词作为一个节点
  - 两个词如果在一定的窗口大小之内（例如5）同时出现，则两个词代表的节点之间连边
  - 边带权重，权重表示共现次数
- 计算 TextRank 分数
  - 利用 PageRank 公式 进行迭代求解：

$$S(V_i) = (1-d) + d \times \sum_{V_j \in In(V_i)} \frac{w_{ji}}{\sum_{V_k \in Out(V_j)}} S(V_j)$$

  - 其中：
    - $S(V_i)$为节点 V_i 的得分
    - d为阻尼系数
    - w_{ji}为从节点 j 到节点 i 的边权
    - $In(V_i)$为指向节点 V_i 的节点集合
    - $Out(V_j)$为节点 V_j 指出的节点集合
- 最后迭代至收敛后，选择前五个 TextRank 得分最高的词作为关键词

```
# 使用 jieba TextRank 提取关键词，若为空则用jieba分词代替
    try:
        query_keywords =
            jieba.analyse.textrank( query_text, topK=5,
            withWeight=False,
            allowPOS=('ns', 'n', 'vn', 'v', 'nr', 'a')  # 扩展词性，减少空结果
        )
        query_keywords = [kw.strip() for kw in query_keywords if kw.strip()]
        if len(query_keywords) == 0:
            print("TextRank提取关键词为空，使用jieba分词结果作为关键词")
            query_keywords = list(set(jieba.cut(query_text)))
    except Exception as e:
        print(f"提取查询关键词时出错：  {str(e)}")
        query_keywords = []
```

2. 文本向量化

我们采取 TF-IDF 方法对词以及文档进行预处理，便于构建向量空间。

Step1. 构建词表：

- 将语料库中所有文档的关键词合并并去重
- 每个词分配唯一索引

```
vocab = list(dict.fromkeys(kw for doc in doc_keywords_list for kw in doc))
```

Step2. 关于 TF-IDF:

- Term Frequency（词频、TF）

  表示词 t 在文档 d 中出现的频率:

$$TF(t,d) = \frac{\text{词}t\text{在文档}d\text{中出现的次数}}{\text{文档}d\text{的总词数}}$$

- Inverse Document Frequency（逆文档频率、IDF）

  表示词 t 在语料库中出现的“罕见程度”:

$$IDF(t) = log(\frac{N+1}{df(t)+1}) + 1$$

  其中:

  - N 代表语料库中文档总数。
  - $df(t)$ 代表含词 t 的文档数。

- $TF-IDF$ 公式

$$TF-IDF(t,d) = TF(t,d) \times IDF(t)$$

  不难看出，若一个词在当前文档中频繁出现，但在其他文档中很少出现，则其 $TF-IDF$ 值就很高。

```
# 计算 TF-IDF 值
def calculate_tfidf(df):
    doc_keywords_list = []
    for line in df['keyword']:
        if pd.isna(line):
            doc_keywords_list.append([])
        else:
            keywords = line.replace(', ', ',').split(',')
            keywords = [kw.strip() for kw in keywords if kw.strip()]
            doc_keywords_list.append(keywords)

    # 去重并保留顺序
    vocab = list(dict.fromkeys(kw for doc in doc_keywords_list for kw in
doc))
    vocab.sort()
    word_to_index = {word: idx for idx, word in enumerate(vocab)}

    #  计算TF矩阵（词频归一化）
    tf_matrix = []
    for doc_keywords in doc_keywords_list:
        tf = defaultdict(int)
        for kw in doc_keywords:
            tf[kw] += 1
        total = sum(tf.values())
        tf_row = {word: tf[word] / total if total > 0 else 0 for word in
vocab}
        tf_matrix.append(tf_row)

    # 计算IDF
    doc_count = len(tf_matrix)
    idf = {}
```

```
    for word in vocab:
        df_count = sum(1 for tf_row in tf_matrix if tf_row[word] > 0)
        idf[word] = math.log((doc_count + 1) / (df_count + 1)) + 1
```

Step3. 优化:

在上述构造的向量空间中，我们会发现语料库中的每一篇新闻都拥有成千上百个维度，但实际自身有效占有的维度仅五个（因为一篇新闻仅提取五个关键词），在与用户进行相似度计算的时候会进行大量无用的计算。因此我们可以记录每篇新闻有效的维度下标以及对应值，优化时间效率。

下面详细介绍时间效率上的优化:

记语料库共 N 篇新闻，设词表大小为 k ,那么一定存在 $1 \leq k \leq 5 \times N$

不妨记新闻 i 的向量为 $A_i =< p_1, p_2, \ldots, p_k >$，其中 $p_l, p_{l+1}, \ldots, p_r$,为有效维度。

有$p_l, p_{l+1}, \ldots, p_r \neq 0, \forall j \notin \{l, l+1, \ldots, r\}, p_j = 0$

那么对于任意查询 $B =< s_1, s_2, \ldots, s_k >$,都有计算

$$\sqrt{\sum_{j=1}^{k_i}(p_{idx_j} - s_{idx_j})^2}$$

式中 k_j代表来两向量的有效维度数量总和，显然存在 $1 \leq k_j \leq 10$ 。

对于全部语料库的相似度计算，有时间复杂度 $O(\sum_{i=1}^{N} k_j) = O(N)$

对于原查询，很容易得出最劣时间复杂度为 $O(N^2)$

```
# 构建TF的index和value字段
    tf_index_list = []
    tf_value_list = []

    for tf_row in tf_matrix:
        tfidf_indices = []
        tfidf_values = []

        tf_indices = []
        tf_values = []

        for word, tf_val in tf_row.items():
            score = tf_val * idf[word]
            idx = word_to_index[word]

            # TF-IDF 部分（只保存非0值）
            if score > 0:
                tfidf_indices.append(str(idx))
                tfidf_values.append(f"{score:.4f}")

            # TF 部分（保存所有词频，即使为0）
            tf_indices.append(str(idx))
            tf_values.append(f"{tf_val:.6f}")

        tfidf_index_list.append(' '.join(tfidf_indices))
        tfidf_value_list.append(' '.join(tfidf_values))

        tf_index_list.append(' '.join(tf_indices))
        tf_value_list.append(' '.join(tf_values))
```

```
    # 保存到DataFrame
    df['TFIDF_IndexValue'] = tfidf_index_list
    df['TFIDF_Values'] = tfidf_value_list
    df['TF_IndexValue'] = tf_index_list
    df['TF_Values'] = tf_value_list

    #  保存词汇表和词频（累加未归一化的出现次数）
    save_vocab_to_db(vocab, word_to_index, doc_keywords_list)
```

3. 词向量实现查询

对于上述根据 $TF-IDF$ 构建向量空间实现查询相似度排序，我们会发现当模式文本并不能与匹配文本进行完全匹配，即使语义十分相同，仍会出现相似度极低的情况，所以我们引入词向量。

- 词向量：一种将词语表示为连续实数向量的方法，使得语义相近的词在向量空间中也接近。
- 模型结构（简化神经网络）
  - 输入层：one-hot 向量
  - 投影层：隐藏层权重矩阵 W （即词向量矩阵）
  - 输出层：softmax分类输出（上下文词）
- 数学形式

  给定一个中心词 w_t ，上下文词集合为 w_{t+j}

  Skip_gram 目标是最大化条件概率：

$$\prod_{-c\leq j\leq c, j\neq 0} P(w_{t+j}|w_t)$$

  用 softmax表达每个词的概率：

$$P(w_O|w_I)=\frac{exp(v_{w_O}\times v_{w_I})}{\sum_{w=1}^{|V|}exp(v_w\times v_{w_I})}$$

  其中：
  - v_{w_I} 表示输入词向量
  - v_{w_O} 表示输出词向量
  - $|V|$ 表示词汇表大小

我们运用两种训练算法类型

- **Dense embeddings**：使用 Skip-Gram with Negative Sampling (SGNS)，本质上是 Word2Vec类型。
- **Sparse embeddings** ：使用 PPMI 模型，稀疏表示和传统词袋模型相似，但加入了信息熵权重。

结合多种语料来源（百度百科、中文维基、人民日报等）得到词向量集合，为后续语义匹配、文本分析、查询排序做基础。

```
# 加载词向量
word_vectors =
{} vector_dim =
0
print(f"正在加载词向量库:  baidubaike_word_vectorslbq.txt")

with open("baidubaike_word_vectorslbq.txt", 'r', encoding='utf-8') as
    f: first_line = f.readline().split()
    if len(first_line) >= 2:
        try:
            total_words = int(first_line[0])
            vector_dim = int(first_line[1])
            print(f"词向量库信息:  总词数={total_words},  维度={vector_dim}")
        except ValueError:
            print("警告: 无法解析第一行的基本信息")
            vector_dim = len(first_line) - 1
            print(f"假设向量维度为: {vector_dim}")
    else:
        print("警告: 第一行格式不正确")
        vector_dim = 300
        print(f"假设向量维度为:  {vector_dim}")

    for line in f:
        parts = line.strip().split()
        if len(parts) < 10:
            continue
        try:
            word = parts[0]
            vec = np.array([float(x) for x in parts[1:vector_dim+1]])
            word_vectors[word] = vec
        except ValueError:
            try:
                first_num_index = next(
                    i for i, x in enumerate(parts[1:], 1)
                    if x.replace('.', '', 1).isdigit() or
                       (x[0] == '-' and x[1:].replace('.', '', 1).isdigit())
                )
                word = "".join(parts[:first_num_index])
                vec = np.array([float(x) for x in
parts[first_num_index:first_num_index+vector_dim]])
                word_vectors[word] =
            vec except:
                continue

print(f"成功加载 {len(word_vectors)} 个词向量")
```

关键词查询模块

根据用户输入的查询文本（如"市场经济"），计算其平均词向量，并于所有新闻的关键词向量进行余弦相似度比较，按照相关性大小降序给出新闻。

步骤详解：

- 数据加载：从 sina_news 表中去除所有新闻记录

```
cursor.execute("SELECT * FROM sina_news")
```

- 为每条新闻计算其平均词向量

```
keywords = row[8]  # 关键词字段
vec = average_word_vector(keywords)  # 根据关键词计算平均词向量
```

  - 每篇新闻都已经预先处理提取出关键词，通过将这些词向量平均，得到该新闻的语义表示向量。
- 对查询文本进行关键词提取（基于 TextRank 算法）：

```
try:
        query_keywords =
            jieba.analyse.textrank( query_text, topK=5,
            withWeight=False,
            allowPOS=('ns', 'n', 'vn', 'v', 'nr', 'a')  # 扩展词性，减少空结果
        )
        query_keywords = [kw.strip() for kw in query_keywords if
        kw.strip()] if len(query_keywords) == 0:
            print("TextRank提取关键词为空，使用jieba分词结果作为关键词")
            query_keywords = list(set(jieba.cut(query_text)))
    except Exception as e:
        print(f"提取查询关键词时出错: {str(e)}")
        query_keywords = []
```

  - 如果提取失败（文本长度过短，无法进行上下文联系），则直接将 jieba 分词结果作为关键词。
- 计算查询平均词向量

```
query_vec = average_word_vector(query_keywords)
```

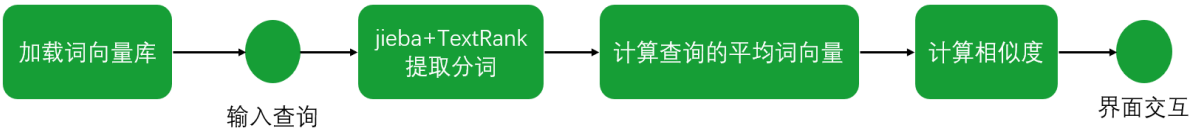
- 批量计算查询与所有文档向量的余弦相似度

```
doc_sims = cosine_similarity_batch(query_vec, doc_matrix)
```

  - 向量越接近，相似度值越接近于1。
- 获取前30篇最相关新闻，并生成关键词词频统计：

```
top10_results = [datalist[idx] for idx, sim in sorted_results[:30]]
```

2.3.2信息检索：



1. 加载词向量库。
2. 输入查询，并对查询进行jieba、TextRank分词
3. 根据分词结果进行查询的平均词向量计算
4. 将查询的平均词向量和数据库中所有的平均词向量进行余弦相似度计算，排序得到相似度排行前10的新闻，并将结果交给前端。

（1） 加载词向量库

```
# 加载词向量
word_vectors = {}
vector_dim = 0
print(f"正在加载词向量库: baidubaike_word_vectorslbq.txt")

with open("baidubaike_word_vectorslbq.txt", 'r', encoding='utf-8') as
f:
    first_line = f.readline().split()
    if len(first_line) >= 2:
        try:
            total_words = int(first_line[0])
            vector_dim = int(first_line[1])
            print(f"词向量库信息: 总词数={total_words}, 维度
={vector_dim}")
        except ValueError:
            print("警告: 无法解析第一行的基本信息")
            vector_dim = len(first_line) - 1
            print(f"假设向量维度为: {vector_dim}")
    else:
        print("警告: 第一行格式不正确")
        vector_dim = 300
        print(f"假设向量维度为: {vector_dim}")

    for line in f:
        parts = line.strip().split()
        if len(parts) < 10:
            continue
        try:
            word = parts[0]
            vec = np.array([float(x) for x in parts[1:vector_dim+1]])
            word_vectors[word] = vec
        except ValueError:
            try:
```

```
                first_num_index = next(
                    i for i, x in enumerate(parts[1:], 1)
                    if x.replace('.', '', 1).isdigit() or
                       (x[0] == '-' and x[1:].replace('.', '',
1).isdigit())
                )
                word = "".join(parts[:first_num_index])
                vec = np.array([float(x) for x in
parts[first_num_index:first_num_index+vector_dim]])
                word_vectors[word] = vec
            except:
                continue

print(f"成功加载 {len(word_vectors)} 个词向量")
```

（2） 连接数据库，并获取所有新闻的平均词向量列表

```
    datalist = []
    cnn = pymysql.connect(host='localhost', user='root',
password='20051004', port=3306, database='newdb',
                          charset='utf8')
    cursor = cnn.cursor()
    cursor.execute("SELECT * FROM sina_news")
    for item in cursor.fetchall():
        datalist.append(item)

    doc_vectors = []
    for row in datalist:
        keyword_str = row[8]
        if keyword_str:
            keywords = [kw.strip() for kw in keyword_str.replace(', ',
',').split(',') if kw.strip()]
            vec = average_word_vector(keywords)
            doc_vectors.append(vec)
        else:
            doc_vectors.append(np.zeros(vector_dim))
```

（3） 对查进行使用 jieba TextRank 提取关键词，若为空则用jieba分词代替

```
try:
        query_keywords = jieba.analyse.textrank(
            query_text, topK=5, withWeight=False,
            allowPOS=('ns', 'n', 'vn', 'v', 'nr', 'a')  # 扩展词性，减少
空结果
        )
        query_keywords = [kw.strip() for kw in query_keywords if
kw.strip()]
```

```
        if len(query_keywords) == 0:
            print("TextRank 提取关键词为空，使用 jieba 分词结果作为关键词")
            query_keywords = list(set(jieba.cut(query_text)))
    except Exception as e:
        print(f"提取查询关键词时出错: {str(e)}")
        query_keywords = []
    print("\n 查询关键词 (最终使用):", query_keywords)
```

（4） 计算查询的平均词向量

```
# 计算查询的平均词向量
    query_vec = average_word_vector(query_keywords) if query_keywords
else np.zeros(vector_dim)

    # 转换为矩阵形式
    doc_matrix = np.vstack(doc_vectors)  # (N, vector_dim)
    query_vec = query_vec.reshape(1, -1)  # (1, vector_dim)
```

（5） 计算相似度

```
# 批量计算余弦相似度函数
def cosine_similarity_batch(query_vec, doc_matrix):
    dot_products = np.dot(doc_matrix, query_vec.T).flatten()  # (N,)
    doc_norms = np.linalg.norm(doc_matrix, axis=1)
    query_norm = np.linalg.norm(query_vec)
    denom = doc_norms * query_norm
    denom[denom == 0] = 1e-10  # 防止除零
    return dot_products / denom


# 计算相似度
    doc_sims = cosine_similarity_batch(query_vec, doc_matrix)
```

（6） 输出前十相关新闻

```
    # 排序输出
    sorted_results = sorted(doc_similarities, key=lambda x: x[1],
reverse=True)
    # 返回相似度最高的前 10 条 datalist
    top10_results = [datalist[idx] for idx, sim in sorted_results[:30]]
    word_freq = {}
    for doc in top10_results:
        keywords = doc[8].split(',')
        for kw in keywords:
            kw = kw.strip()
            if kw not in word_freq:
                word_freq[kw] = 0
            word_freq[kw] += 1  # 统计关键词的出现次数
```

2.3.3可视化界面：

使用HTML，CSS和JS等前端技术和FLASK框架结合

1.HTML（超文本标记语言）

HTML（HyperText Markup Language）是构建网页结构的标准标记语言。它通过一系列标签定义网页的基本元素，如文本、图像、表格和链接等，是网页内容的“骨架”。

2.CSS（层叠样式表）

CSS 用于描述 HTML 元素的表现形式，控制页面的颜色、字体、布局、动画等视觉效果。通过分离结构与样式，CSS 使网页更美观、响应更灵活，也便于维护和重用。

2. JavaScript（JS）

JavaScript 是一种轻量级、解释型或即时编译型的脚本语言，广泛用于实现网页的动态行为和交互逻辑。它支持原型继承和多种编程范式（包括面向对象、命令式、声明式和函数式编程），不仅在浏览器中广泛应用，也被用于服务器端和嵌入式系统中。

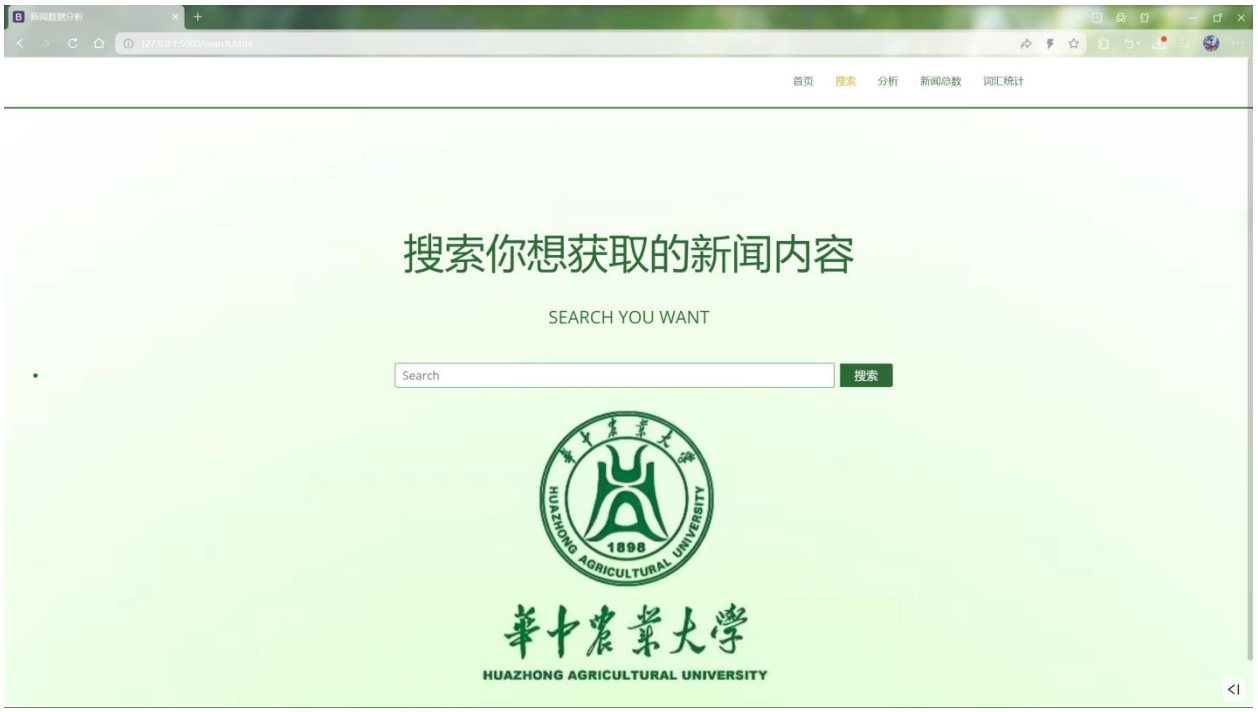
3. Flask（Python Web 框架）

Flask 是一个用 Python 编写的 Web 应用框架，基于 Werkzeug（WSGI 工具包）和 Jinja2（模板引擎）。它属于“微框架”（microframework），核心简洁、灵活，不强制集成数据库或表单验证工具。开发者可以根据需求通过扩展插件添加功能，使其适应从简单网站到复杂系统的不同应用场景。

可视化界面展示：

1. 首页主界面

2. 搜索界面



3. 输入检索信息，得到该查询相关的热度分析，词云和前十相关的新闻（可以选择排序方式）

4. 分析界面，可以进行类型和时间选择

词云：


首页
搜索
分析
新闻总数
词汇统计
词频统计
数据时间范围：2025-05-08T17:29 到 2025-05-15T17:29
根据 4122 条新闻的描述，提取 300 个关键词生成词云树，可以让我们更加清晰的感受到新闻热点
关于新闻
不知道你从这些热点词中悟到了什么呢?

可视化：


首页
搜索
分析
新闻总数
词汇统计
新闻内容关键词频率分布
数据时间范围：2025-05-10T17:31 到 2025-05-13T17:31
250
200
150
100
50
0
公司
市场
美国
关税
中国
投资
经济
行业
企业
基金
资金
银行
股份
政策
产品
预期
增长
价格
科技
汽车

热度分析：

5. 新闻来源界面

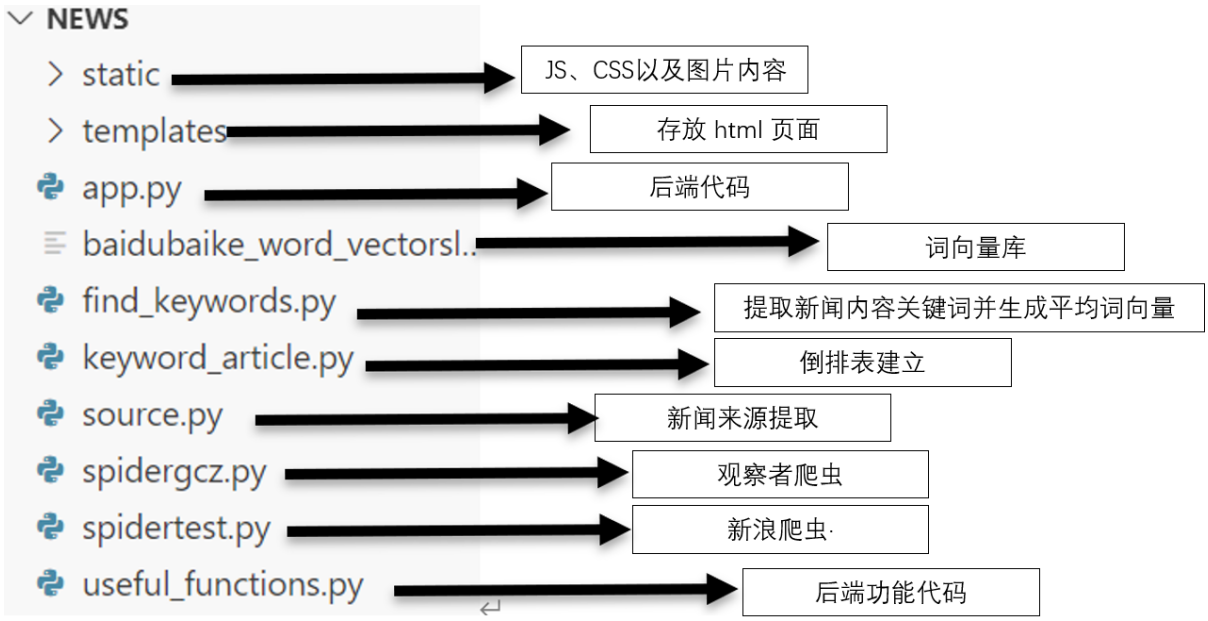
6. 媒体来源的热度对比

7. **新闻总数**，可进行时间选择点击标题可跳转至源网址。

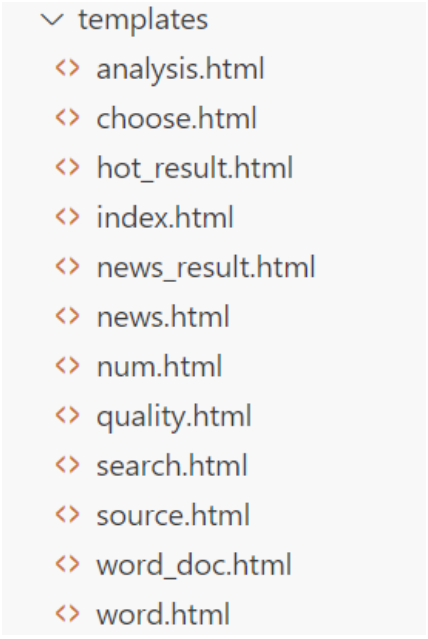
8. **词汇统计界面**，显示所有提取的关键词，鼠标移动可显示出现频率，点击即可看到相关文章。

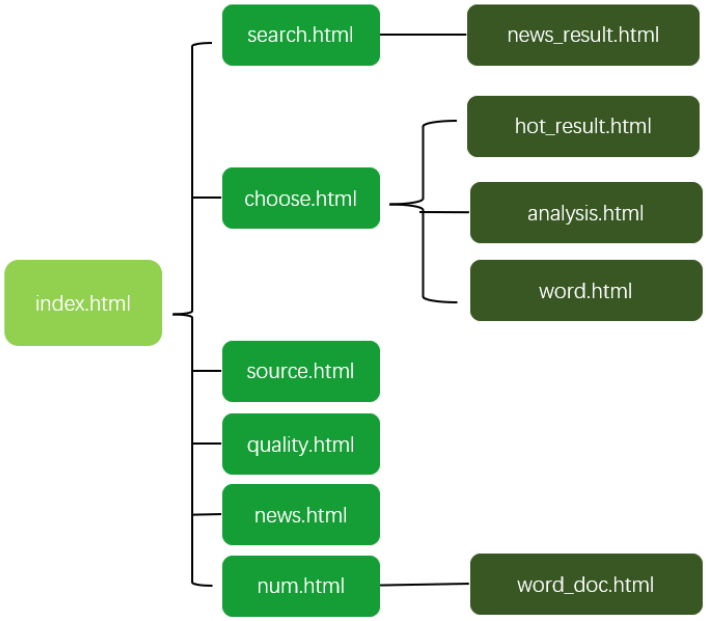
2.4代码展示

使用了Flask框架，前后端交互，以下是我们的代码结构：



前端部分的结构：





*index.html*是首页界面，*search.html*是搜索界面，*choose.html*分析界，*source.html*是媒体来源的界面，*quality.html*是媒体来源的热度对比界面，*news.html*是全部新闻界，*num.html*是词汇统计界面，*news_result.html*是搜索结果页面界面，*hot_result.html*是主题热度分析结果界面，*analysis.html*是新闻内容关键词频率分布的界面，*word.html*是词云界面，*word_doc.html*是词汇相关文档的界面。

app.py使前端能和数据库联通，并实现相应功能，“…”省略了部分代码，其中主要功能都是调用useful_functions.py中的代码来实现的

```
app = Flask(__name__)
app.config["SECRET_KEY"] = "12345678"

class SearchForm(FlaskForm):
    search_creater = StringField('creater',
validators=[DataRequired()])
    submit = SubmitField('搜索')

@app.route('/')
def index():
    form = SearchForm()
    return render_template("index.html", form=form)

@app.route('/search.html')
def search():
    return render_template('search.html')

@app.route('/news.html', methods=['GET'])
def news_page():
    …
    return render_template("news.html", news=news_page, page=page,
pages=pages)

@app.route('/word.html')
def word_page():
    …
    return render_template("word.html", img_base64=img_base64,
news_count=news_count,
keyword_count=keyword_count,start_date=start_date,end_date=end_date)

@app.route('/analysis.html')
def analysis_page():
    …
    return render_template("analysis.html",words = words, weights
=weights,start_date=start_date,end_date=end_date)


@app.route('/news_result.html', methods=['POST', 'GET'])
def newsResult_page():
    …
    return render_template("news_result.html",
form=form,wordcloud_data=wordcloud_data,news=search_list,titles=titles,
pub_times=pub_times,views=views,comments=comments)

@app.route('/hot_result.html', methods=['GET', 'POST'])
def hot_hot_newsResult_page():
    …
    return render_template
```

```
("hot_result.html",news=datalist_sorted,titles=titles,pub_times=pub_tim
es,views=views,comments=comments,start_date=start_date,end_date=end_dat
e)

@app.route('/num.html')
def keywords():
    …
    return render_template('num.html', word_data=word_data)

@app.route('/word_doc.html')
def word_doc():
    …
    return render_template('word_doc.html', detail=detail)

@app.route('/index.html')
def get_index():
    …
    return render_template("index.html", news_count=news_count,
keyword_count=keyword_count)

@app.route('/source.html')
def get_source():
   …
   return render_template("source.html",words = words, weights
=weights)

@app.route('/quality.html')
def quality_result():
    …
    return render_template('quality.html',data = result)

@app.route('/choose.html')
def choose():
    return render_template('choose.html')

if __name__ == '__main__':
    app.run()
```

useful_functions.py 实现相应功能

举例

1. 实现搜索结果的功能函数

```
def get_anwers(query_text):
    datalist = []
    cnn = pymysql.connect(host='localhost', user='root',
password='20051004', port=3306, database='newdb',
                          charset='utf8')
```

```
    cursor = cnn.cursor()
    cursor.execute("SELECT * FROM sina_news")
    for item in cursor.fetchall():
        datalist.append(item)

    doc_vectors = []
    for row in datalist:
        keyword_str = row[8]
        if keyword_str:
            keywords = [kw.strip() for kw in keyword_str.replace(', ',
',').split(',') if kw.strip()]
            vec = average_word_vector(keywords)
            doc_vectors.append(vec)
        else:
            doc_vectors.append(np.zeros(vector_dim))


    # 使用 jieba TextRank 提取关键词，若为空则用 jieba 分词代替
    try:
        query_keywords = jieba.analyse.textrank(
            query_text, topK=5, withWeight=False,
            allowPOS=('ns', 'n', 'vn', 'v', 'nr', 'a')  # 扩展词性，减少
空结果
        )
        query_keywords = [kw.strip() for kw in query_keywords if
kw.strip()]
        if len(query_keywords) == 0:
            print("TextRank 提取关键词为空，使用 jieba 分词结果作为关键词")
            query_keywords = list(set(jieba.cut(query_text)))
    except Exception as e:
        print(f"提取查询关键词时出错: {str(e)}")
        query_keywords = []
    print("\n 查询关键词 (最终使用):", query_keywords)
    # 计算查询的平均词向量
    query_vec = average_word_vector(query_keywords) if query_keywords
else np.zeros(vector_dim)

    # 转换为矩阵形式
    doc_matrix = np.vstack(doc_vectors)  # (N, vector_dim)
    query_vec = query_vec.reshape(1, -1)  # (1, vector_dim)

    # 计算相似度
    doc_sims = cosine_similarity_batch(query_vec, doc_matrix)

    # 组合索引与相似度
    doc_similarities = list(enumerate(doc_sims))

    # 排序输出
```

```
    sorted_results = sorted(doc_similarities, key=lambda x: x[1],
reverse=True)
    # 返回相似度最高的前 10 条 datalist
    top10_results = [datalist[idx] for idx, sim in sorted_results[:30]]
    word_freq = {}
    for doc in top10_results:
        keywords = doc[8].split(',')  # 用逗号拆分关键词
        for kw in keywords:
            kw = kw.strip()
            if kw not in word_freq:
                word_freq[kw] = 0
            word_freq[kw] += 1  # 统计关键词的出现次数

    # 取前 100 词（根据出现次数排序）
    top_n = 100
    top_words = sorted(word_freq.items(), key=lambda x: x[1],
reverse=True)[:top_n]

    wordcloud_data = [{"name": k, "value": v} for k, v in top_words]

    return wordcloud_data, top10_results
```

2. 实现词到文档功能的函数

```
def get_word_detail_by_id(id, page=1, per_page=10):

    cnn = pymysql.connect(host='localhost', user='root',
password='20051004', port=3306, database='newdb',
                          charset='utf8',
cursorclass=pymysql.cursors.DictCursor)
    cursor = cnn.cursor()

    # 查询关键词基本信息
    cursor.execute("SELECT keyword, total_tf FROM tfidf_vocab WHERE id
= %s", (id,))
    keyword_info = cursor.fetchone()

    if not keyword_info:
        cnn.close()
        return None

    # 查询该关键词对应文章总数，用于分页
    cursor.execute("SELECT COUNT(*) AS total_count FROM keyword_article
WHERE keyword_id = %s", (id,))
    total_count = cursor.fetchone()['total_count']

    # 计算分页起始位置
    offset = (page - 1) * per_page
```

```
    # 查询关键词对应文章列表，分页查询
    cursor.execute("""
        SELECT article_id, url, title
        FROM keyword_article
        WHERE keyword_id = %s
        ORDER BY CAST(article_id AS UNSIGNED) ASC
        LIMIT %s OFFSET %s
    """, (id, per_page, offset))
    articles = cursor.fetchall()

    cnn.close()

    # 计算总页数
    total_pages = (total_count + per_page - 1) // per_page

    return {
        'id': id,
        'keyword': keyword_info['keyword'],
        'total_tf': keyword_info['total_tf'],
        'articles': articles,
        'total_count': total_count,
        'pages': total_pages,
        'current_page': page,
        'page': page,
        'per_page': per_page
    }
```

仅举两例，因为所有函数的逻辑都是相同的。即连接数据库，通过查询语句获得相关信息，再通过代码对相关信息进行处理，然后将处理后的数据传递给前端。

搜索相关代码：

```
<section id="banner">
    <h2>搜索你想获取的新闻内容</h2>
    <p>Search You Want</p>
    <ul class="actions">
        <li>
            <form action="news_result.html" method="GET" name="search"
role="search">
                <input id="query" name="query" type="text" value=""
placeholder="Search"/>
                <button type="submit">搜索</button>
            </form>
        </li>
    </ul>
  </section>
```

实现词到文档关键代码：

```
<div class="keyword-cloud">
          {% for initial, words in word_data.items() %}
          <h3 class="letter-title" id="letter-
{{ initial }}">{{ initial }}</h3>
          <div class="keyword-group">
            {% for item in words %}
              <a
                class="keyword-item"
                href="word_doc.html?id={{ item.id }}"
                title="出现频率：{{ item.weight }}"
                target="_blank"
              >
              {{ item.word }}
              </a>
            {% endfor %}
          </div>
         {% endfor %}
      </div>
```

日期选择器部分前端代码：

```
        <div class="counts-section-title" style="text-align:center;
margin-bottom: 20px;"data-aos="fade-up" data-aos-delay="200">
          <h4>选择日期和时间范围</h4>
        </div>
        <form id="date-form">
          <div class="row justify-content-center">
            <div class="col-md-4 col-sm-6"data-aos="fade-up" data-aos-
delay="250">
              <div class="form-group">
                <label for="start-date">开始日期和时间</label>
                <input
                  type="datetime-local"
                  class="form-control"
                  id="start-date"
                  name="start-date"
                />
              </div>
            </div>
            <div class="col-md-4 col-sm-6"data-aos="fade-up" data-aos-
delay="300">
              <div class="form-group">
                <label for="end-date">结束日期和时间</label>
                <input
                  type="datetime-local"
                  class="form-control"
                  id="end-date"
                  name="end-date"
```

```
                />
              </div>
            </div>
          </div>
          <div class="row justify-content-center mt-3">
            <div class="col-md-3 col-sm-6"data-aos="fade-up" data-aos-
delay="400">
              <button
                type="button"
                class="btn btn-primary w-100"
                onclick="submitForm()"
              >
                确定
              </button>
            </div>
          </div>
        </form>
```

```
// 点击确定后提交数据，根据类型跳转页面并带时间参数
    function submitForm() {
      const startDate = document.getElementById('start-date').value;
      const endDate = document.getElementById('end-date').value;

      if (!selectedOption) {
        alert('请选择一个类型！');
        return;
      }
      if (!startDate || !endDate) {
        alert('请选择开始和结束日期！');
        return;
      }
      if (startDate > endDate) {
        alert('开始日期不能晚于结束日期！');
        return;
      }

      let targetPage = '';
      switch (selectedOption) {
        case 'word':
          targetPage = 'word.html';
          break;
        case 'analysis':
          targetPage = 'analysis.html';
          break;
        case 'hot_result':
          targetPage = 'hot_result.html';
          break;
        default:
          alert('未知的类型！');
```

```
            return;
        }

        const url = `${targetPage}?start-
date=${encodeURIComponent(startDate)}&end-
date=${encodeURIComponent(endDate)}`;

        window.location.href = url;
    }
```

柱状图部分前端代码：

```
<script type="text/javascript">

var dom = document.getElementById("main");
var myChart = echarts.init(dom);
var app = {};
option = null;
option = {

    color:['#28a745'],
    tooltip: {
        trigger: 'axis',
        axisPointer: {
            type: 'shadow'
        }
    },
    grid: {
        left: '3%',
        right:'4%',
        bottom:'3%',
        containLabel:true
    },
    xAxis: {
        type: 'category',
        data: {{ words|tojson }}
    },
    yAxis: {
        type: 'value'
    },
    series: [{
        data: {{ weights }},
        barWidth:'60%',
        type:'bar'
    }]
};
if (option && typeof option === "object") {
    myChart.setOption(option, true);
}
```

```
</script>
```

折线图部分代码：

```
<script type="text/javascript">
    var dom = document.getElementById("main");
    var myChart = echarts.init(dom);
    var option = {
      color: ['#007bff', '#dc3545'],

      tooltip: {
        trigger: 'axis',
        formatter: function (params) {
          let index = params[0].dataIndex;
          let titles = {{ titles | tojson }};
          let pub_times = {{ pub_times | tojson }};
          let views = params[0].value;
          let comments = params[1].value;

          return `
            <strong>标题：</strong>${titles[index]}<br/>
            <strong>时间：</strong>${pub_times[index]}<br/>
            <strong>浏览量：</strong>${views}<br/>
            <strong>评论数：</strong>${comments}
          `;
        }
      },
      legend: {
        data: ['浏览量', '评论数'],
        top: '10%'
      },
      grid: {
        left: '3%',
        right: '4%',
        bottom: '8%',
        containLabel: true
      },
      xAxis: {
        type: 'category',
        boundaryGap: false,
        data: {{ pub_times | tojson }}
      },
      yAxis: [
        {
          type: 'value',
          name: '浏览量',
          position: 'left',
          axisLine: {
```

```
            lineStyle: {
              color: '#007bff'
            }
          },
          axisLabel: {
            formatter: '{value}'
          }
        },
        {
          type: 'value',
          name: '评论数',
          position: 'right',
          axisLine: {
            lineStyle: {
              color: '#dc3545'
            }
          },
          axisLabel: {
            formatter: '{value}'
          }
        }
      ],
      series: [
        {
          name: '浏览量',
          type: 'line',
          data: {{ views | tojson }},
          smooth: true,
          yAxisIndex: 0,
          lineStyle: {
            color: '#007bff'
          }
        },
        {
          name: '评论数',
          type: 'line',
          data: {{ comments | tojson }},
          smooth: true,
          yAxisIndex: 1,
          lineStyle: {
            color: 'rgba(220, 53, 69, 0.7)',  // 半透明红色
          }
        }
      ]
    };
    myChart.setOption(option);
  </script>
```

动态词云部分代码：

```
<script>
  var wordCloudDom = document.getElementById('wordcloud');
  var wordCloudChart = echarts.init(wordCloudDom);

  var option = {
    tooltip: {
      show: true
    },
    series: [{
      type: 'wordCloud',
      shape: 'square',  // 词云的形状
      left: 'center',
      top: 'center',
      width: '100%',
      height: '100%',
      sizeRange: [14, 50],  // 词的大小范围
      rotationRange: [-45, 90],  // 旋转角度范围
      rotationStep: 45,  // 旋转步长
      gridSize: 8,  // 网格大小
      drawOutOfBound: false,  // 防止超出边界
      textStyle: {
        color: 'rgb(0, 255, 0) !important'  // 统一绿色
      },
      emphasis: {
        focus: 'self',
        textStyle: {
          shadowBlur: 10,
          shadowColor: '#333'
        }
      },
      data: {{ wordcloud_data | tojson }}
    }]
  };

  wordCloudChart.setOption(option);
</script>
```

散点图部分代码：

```
<script type="text/javascript">
    var dom = document.getElementById("main");
    var myChart = echarts.init(dom);
    var rawData = {{ data | tojson }};
    // 防止平均浏览量或评论数为 0，log 轴无效，统一加偏移量 1
    var offset = 1;
    // 转换为散点图数据格式：  [ [平均浏览量+offset, 平均评论数+offset, 来源
```

```
名称] ]
    var scatterData = rawData.map(function (item) {
      return [
        item["平均浏览量"] + offset,
        item["平均评论数"] + offset,
        item["来源"]
      ];
    });
    // 计算加偏移后的平均浏览量和平均评论数
    var avg浏览量 = rawData.reduce(function (sum, item) {
      return sum + (item["平均浏览量"] + offset);
    }, 0) / rawData.length;
    var avg评论数 = rawData.reduce(function (sum, item) {
      return sum + (item["平均评论数"] + offset);
    }, 0) / rawData.length;
    var option = {
      title: {
        text: "媒体来源与热度对比（对数坐标轴）",
        left: "center",
      },
      tooltip: {
        trigger: "item",
        formatter: function (params) {
          return (
            "来源: " +
            params.value[2] +
            "<br/>平均浏览量: " +
            (params.value[0] - offset).toFixed(2) +
            "<br/>平均评论数: " +
            (params.value[1] - offset).toFixed(2)
          );
        },
      },
      xAxis: {
        name: "平均浏览量的对数刻度",
        type: "log",
        axisLabel: {
          formatter: function (value) {
            return (value - offset).toFixed(0);
          }
        },
        splitLine: {
          show: true
        }
      },
      yAxis: {
        name: "平均评论数的对数刻度",
        type: "log",
```

```
        axisLabel: {
          formatter: function (value) {
            return (value - offset).toFixed(0);
          }
        },
        splitLine: {
          show: true
        }
      },
      series: [
        {
          name: "媒体来源",
          type: "scatter",
          symbolSize: 12,
          data: scatterData,
          itemStyle: {
            color: "#28a745",
          },
          markLine: {
            silent: true,
            lineStyle: {
              type: "dashed",
              color: "#FF4500",
            },
            label: {
              show: true,
              formatter: function (param) {
                if (param.data.xAxis !== undefined) return "平均浏览量";
                if (param.data.yAxis !== undefined) return "平均评论数";
                return "";
              },
              position: "end",
              color: "#FF4500",
              fontWeight: "bold",
            },
            data: [
              { xAxis: avg 浏览量 },
              { yAxis: avg 评论数 },
            ],
          },
          emphasis: {
            label: {
              show: true,
              formatter: function (param) {
                return param.data[2]; // 显示来源名称
              },
              position: "top",
            },
```

```
          },
        },
      ],
      grid: {
        left: "10%",
        right: "10%",
        bottom: "15%",
        containLabel: true,
      },
    };

    myChart.setOption(option);
  </script>
```

饼状图代码：

```
<script type="text/javascript">
    var dom = document.getElementById("main");
    var myChart = echarts.init(dom);
    var app = {};

    // 将关键词与权重合并为饼图需要的格式
    var words = {{ words|tojson }};
    var weights = {{ weights }};
    var pieData = [];
    for (var i = 0; i < words.length; i++) {
      pieData.push({ value: weights[i], name: words[i] });
    }

    var option = {
        color: [
    '#ff7f50',
    '#87cefa',
    '#da70d6',
    '#32cd32',
    '#ffd700',
    '#00ced1',
    '#ff69b4',
    '#ba55d3',
    '#ffa500',
    '#40e0d0'
  ],
      tooltip: {
        trigger: 'item'
      },
      series: [
        {
          type: 'pie',
```

```
          radius: '100%',
          center: ['50%', '50%'],
          data: pieData,
          labelLine: {show: false},
          label: { show: false },
          emphasis: {
            labelLine: { show: false },
            itemStyle: {
              shadowBlur: 0,
              shadowOffsetX: 0,
              shadowColor: 'transparent' // 透明阴影
            }
          }
        }
      ]
    };
    if (option && typeof option === "object") {
      myChart.setOption(option, true);
    }
  </script>
```

3 实训总结

3.1个人总结

3.1.1王滢

在本次实训项目中，我们围绕新闻数据的采集、分析与可视化构建了一个完整的数据分析系统。从最初对任务的不确定，到逐步理清流程并搭建起系统框架，我收获了很多实用技术与项目实践经验。

在数据采集阶段，主要挑战是如何从 动态网页 中提取新闻主体内容。这些信息往往通过 JavaScript 动态渲染，不直接显示在 HTML 源码中。为了解决这一问题，我通过浏览器开发者工具分析网络请求，结合 requests 和 selenium 等工具，模拟浏览器行为并抓取所需数据。期间还学习了如何识别网页中的通用 JS 数据结构与接口，提取数据字段并构建爬虫模板。这一过程大大加深了我对前端结构的理解和数据抓取策略的掌握。

在后端开发方面，我们使用了 Flask 框架 搭建项目的后端逻辑。通过 Flask 路由系统将前端页面与后端数据处理模块连接起来，实现了新闻数据的查询、关键词提取与热度分析功能。后端中还整合了文本预处理、关键词提取（如 TextRank 算法）词向量，用于生成词云图和热度趋势图。这个过程让我熟悉了 Flask 的基本使用方式，也让我意识到接口设计和模块解耦在项目中的重要性。

在系统前端展示方面，采用了 HTML+CSS+JS 的 Web 前端形式，将后端分析结果以图表的方式动态呈现在网页中。界面设计上通过引入可视化库 ECharts 提升了展示效果，增强了用户的交互体验。这一过程也让我体会到前

后端联调中的细节处理，比如数据格式的转换、接口响应的异常处理等问题。

总体来说，本次项目涵盖了从数据采集、后端逻辑到前端可视化的完整开发流程，不仅提升了我在 Flask 后端开发和数据分析方面的技能，也让我更清晰地理解了一个完整系统的搭建过程。感谢团队的协作与支持，也希望今后能继续提升自己的工程实践能力，更高效地完成类似项目的开发与优化。

3.1.2雷博琦

在本次数据采集与分析实践中，我系统地掌握了从网络新闻源自动化采集信息、处理文本数据、提取有价值内容并进行多层次分析的完整流程，提升了我对数据驱动型项目的理解与实操能力。

在数据采集阶段，我选取了新浪新闻与观察者网作为主要数据来源，分别编写了多线程新闻爬虫，自动化获取新闻标题、时间、来源、链接、浏览量、评论数和正文内容等关键信息。针对新浪新闻，我利用其滚动新闻接口配合 JSON 数据解析，结合线程锁机制实现高效多页采集，并通过 BeautifulSoup 提取正文与元数据。尤其是针对浏览量与评论数的 JS 渲染问题，我研究其评论API和 HTML 中的 meta 标 签，通过提取 channel 与 newsid 动态拼接评论请求地址，最终实现了数据的完整采集。观察者网的数据采集同样克服了反爬限制，利用随机 User-Agent 和请求重试机制有效提升了爬取稳定性，并通过 XPath 和正则表达式对详情页内容进行精准解析。

在数据分析部分，我以新闻文本为对象，开展了从关键词提取到向量建模再到语义查询的多个步骤。首先，我基于 jieba 分词与 TextRank 算法，构建了关键词图模型，借助共现关系与边权重进行迭代评分，最终提取每篇新闻的核心关键词。接着，通过 TF-IDF 算法实现了稀疏向量表示，利用词频与逆文档频率权重组合，刻画词在文档中的重要程度。为了优化计算效率，我对每篇文档仅记录有效的向量维度及其值，降低了整体时间复杂度至 O(N)。考虑到 TF-IDF 无法表达词义相近但词面不同的情况，我进一步引入了 Word2Vec 词向量模型。通过加载百度百科等大型语料训练的词向量，对每篇新闻关键词进行向量平均，构建了具有语义表征能力的文档向量库。在查询时，将用户输入的文本进行关键词提取与向量平均，再与新闻语义向量进行余弦相似度匹配，排序后返回最相关的结果，大大提升了检索的智能性与相关性。

总的来说，这次项目不仅让我掌握了从零构建数据采集和分析系统的能力，也让我意识到文本处理中的挑战，如反爬机制、JS 渲染、语义歧义、维度稀疏性等问题的应对方式。通过实际编码与优化过程，我对数据采集的工程实现与数据分析的理论方法有了更深入的理解，为今后从事数据类项目打下了坚实基础。

3.1.3易安麒

在过去的几周里，我全身心投入到了实时新闻分析系统的开发工作中，从零开始构建了一个能够自动采集、分析和可视化新闻数据的完整系统。这个项目不仅让我掌握了多项实用技术，更重要的是培养了我解决复杂问题的能力，让我对数据处理全流程有了更深刻的理解。

项目初期，我们首先攻克了网络爬虫技术这一难关。通过学BeautifulSoup库，我们成功实现了对多个新闻网站的数据抓取。在这个过程中，我遇到了各

种反爬虫机制的阻碍。通过研究解决方案，我们学会了使用User-Agent轮换、IP代理池等技术手段，并设计了合理的请求频率控制策略，最终构建了一个稳定运行的爬虫系统。之后为了提升爬取信息的速度，我们还学习应用了多线程技术来提升爬虫速度。

数据获取之后，面临的挑战是如何处理和分析这些非结构化的新闻文本。我们研究了自然语言处理技术，使用jieba分词工具对中文文本进行预处理，并加载了百度中文词向量库。这些技术帮助我实现了新闻关键词提取、文本相似度匹配等功能。

为了让分析结果更直观易懂，我重点研究了数据可视化技术。ECharts库的强大功能让我印象深刻，经过反复尝试和调试，我实现了多种动态图表，包括新闻热度趋势图、关键词词云等。

在实现过程中，这个项目开发过程中遇到了不少困难。记得在初期，不同新闻网站的页面结构差异导致数据提取非常困难，经常出现提取不全或提取错误的情况。经过反复试验，我们设计了一套解析规则，结合正则表达式，准确提取出了数据。

回顾整个项目，虽然已经实现了基本功能，但我深知还有很多可以改进的地方。未来我计划引入更先进的深度学习模型来提升文本分析的准确率，增加个性化推荐功能让系统更智能。同时，异常处理机制的完善也是下一步的重点工作。

这次项目经历让我深刻认识到，真正的学习发生在解决问题的过程中。从最开始的毫无头绪，到现在的游刃有余，这个转变过程让我倍感欣慰。实时新闻分析系统的开发不仅丰富了我的技术栈，更让我对数据处理的全流程有了系统性的认识。我相信这些经验和能力将为我未来的技术之路打下坚实基础，也让我更有信心迎接更大的挑战。技术的价值在于解决实际问题，而我会继续在这条路上探索前行，用代码创造更多的可能。这个实时新闻分析系统的开发过程让我深刻体会到理论知识与实践结合的重要性。面对不断出现的技术难题，我学会了如何有效查找资料、分析问题并找到解决方案。未来我将继续完善这个系统，并探索更多数据分析与可视化的可能性。这次项目经历为我后续的技术学习和职业发展打下了坚实基础。

3.2实训日志

日　期：　2025.5.8

序号	学生姓名	今日进展情况	存在问题	5.9日计划
1	王滢	学习爬取新浪新闻和如何把信息存入数据库	动态界面爬取困难	学习如何爬取动态界面信息

2	雷博琦	学习爬取新浪新闻和如何把信息存入数据库	动态界面爬取困难	学习如何爬取动态界面信息
3	易安麒	学习爬取新浪新闻和如何把信息存入数据库	动态界面爬取困难	学习如何爬取动态界面信息

日　期：2025.5.9

序号	学生姓名	今日进展情况	存在问题	5.17日计划
1	王滢	完成爬虫新浪新闻并存入数据库	无	爬取观察者网新闻标题、时间和内容
2	雷博琦	完成爬取新浪新闻并存入数据库	无	爬取观察者网新闻点击量和评论数
3	易安麒	完成爬取新浪新闻并存入数据库	无	学习多线程

日　期：2025.5.17

序号	学生姓名	今日进展情况	存在问题	5.18日计划
1	王滢	尝试爬取观察者网新闻标题、时间和内容	观察者网反爬取很厉害，爬取不到信息	学习爬取观察者网新闻标题、时间和内容
2	雷博琦	尝试爬取观察者网新闻点击量和评论数	观察者网反爬取很厉害，爬取不到信息	学习爬取观察者网新闻点击量和评论数
3	易安麒	将新浪爬虫改为多线程	无	学习如何生成词云

日　期：2025.5.18

序号	学生姓名	今日进展情况	存在问题	5.22日计划
1	王滢	爬取观察者网新闻标题、时间和内容	无	学习建立前端搜索页面和前后端连接
2	雷博琦	爬取观察者网新闻点击量和评论数，初步完成相关度查询输出	无	学习建立前端搜索页面和前后端连接
3	易安麒	完成词云代码	无	学习将爬取正文及相关内容的代码和爬取评论数和阅读量的代码合并。

日　期：2025.5.22

序号	学生姓名	今日进展情况	存在问题	次日计划

1	王滢	建立前端搜索页面和前后端连接	搜索功能连接数据库受阻	完善html的设计和前后端连接
2	雷博琦	建立前端搜索页面和前后端连接	搜索功能连接数据库受阻	完善html的设计和前后端连接
3	易安麒	将爬取观察者网正文及相关内容的代码和爬取评论数和阅读量的代码合并。	无	将爬取观察者网新闻的代码改为多线程

日　期：2025.5.23

序号	学生姓名	今日进展情况	存在问题	次日计划
1	王滢	完善html的设计和前后端连接	词云、词频分析和热度分析是针对数据库全部新闻的，没有针对性	更改前后端代码，使词云词频分析和热度分析可以进行时间选择和搜索选择
2	雷博琦	完善html的设计和前后端连接	查询精确度不够高	改进查询算法
3	易安麒	将爬取观察者网新闻的代码改为多线程	无	完成词到文档的代码实现和界面实现

日　期：2025.5.29

序号	学生姓名	今日进展情况	存在问题	次日计划
1	王滢	更改前后端代码，使词云词频分析和热度分析可以进行时间选择和搜索选择	无	继续更改前后端代码，使词云词频分析和热度分析可以进行时间选择和搜索选择
2	雷博琦	利用词向量语库改进查询算法	无	继续改进查询算法
3	易安麒	完成词到文档的代码实现和界面实现	无	继续完成词到文档的代码实现和界面实现

日　期：2025.5.30

序号	学生姓名	今日进展情况	存在问题	次日计划
1	王滢	更改前后端代码，使词云词频分析和热度分析可以进行时间选择和搜索选择	无	制作答辩PPT
2	雷博琦	利用词向量语库改进查询算法	无	制作答辩PPT
3	易安麒	完成词到文档的代码实现和界面实现	无	制作答辩PPT

3.3项目总结与未来展望

3.3.1项目总结

1.项目概述

我们团队开发的"实时新闻主题热度分析系统"是一个集数据采集、处理、分析和可视化于一体的综合性平台。该系统能够从新浪新闻和观察者网等主流新闻网站实时抓取新闻数据，通过自然语言处理技术提取关键词，计算新闻热度，并以丰富的可视化形式展示分析结果。项目涵盖了从前端界面到后端处理的全栈开发流程，实现了新闻数据的全生命周期管理。

2.技术实现亮点

（1）高效的数据采集系统

采用多线程爬虫技术，结合随机User-Agent和IP代理池，有效应对反爬机制

针对动态渲染内容(如浏览量、评论数)设计了专门的解析方案

构建了从数据采集到展示的完整处理流水线实现了增量爬取策略，优化了资源利用率

（2）先进的文本分析能力

结合jieba分词和TextRank算法提取新闻关键词

加载百度百科预训练词向量，实现语义级别的新闻相似度计算

通过TF-IDF和词向量平均方法构建新闻特征表示
（3）丰富的可视化展示
使用ECharts实现了词云、热度趋势图、来源分布图等多种可视化形式
开发了交互式前端界面，支持时间范围选择和关键词搜索
设计了响应式布局，适配不同终端设备
（4）完整的系统架构
采用Flask框架搭建后端服务，实现前后端分离
使用MySQL数据库结构化存储新闻数据

3.3.2未来展望

1.算法升级：用BERT等模型提升语义分析能力，优化热度计算（加入时间衰减因子）
2.功能增强：
自动事件归类：开发简单易用的自动分类功能，将新闻自动归类为"政治"、"经济"、"社会"等常见类型，方便用户快速筛选。比如点击"科技"标签就能看到所有相关新闻。
热点预警通知：设置个性化提醒功能，当某个话题的热度突然上升时（比如上涨超过50%），系统会自动发送邮件或APP推送通知。
简易趋势预测：基于过去一周的数据，用简单的线性预测模型展示未来3天可能的热度变化趋势，用"上升箭头"、"持平"或"下降"等直观方式呈现。
3.系统稳定：部署分布式爬虫和自动恢复机制，提升抗风险能力