江苏大学

JIANGSU UNIVERSITY

计算思维与 *Python* 课程设计

学院名称：计算机科学与通信工程学院

专业班级：物联网工程 2303

学生姓名：邱佳亮

学生学号：3230611072

指导教师姓名：尹星

2025 年 1 月

目录

基于 Python 的爬虫项目设计

1. 设计目标

本次课程设计的目标是利用模块化的程序设计思想和面向对象的 Python 语言，设计并实现一个功能完善的网络爬虫软件。该软件能够从指定科技网站爬取新闻内容，支持动态加载页面的爬取，并绕过常见的反爬虫机制，确保数据的完整性和准确性。通过对爬取的新闻文本进行分词、词频统计和停用词过滤，结合自然语言处理技术提取高频词汇和关键主题，并实现情感分析功能，判断新闻的情感倾向（正面或负面）。项目还通过词云图、柱状图和 LDA 主题图等多种可视化手段，直观展示文本分析结果，并提供友好的图形用户界面（GUI），使用户能够实时查看分析结果。最后，爬取的新闻内容、分析结果和可视化图表将被保存到本地文件或数据库中，支持多种文件格式（如文本文件、图像文件）和结构化存储，确保数据的持久化和可访问性。

2. 关键问题

1.爬取与分析：针对某个科技主题，选择一个或多个科技网站，爬取相关的新闻网页并进行分析。这个模块需要考虑网络请求、反爬虫技术的绕过、HTML 解析等技术点。

2.词汇分析：对爬取到的页面进行文本分析，识别并统计高频词汇，利用词云等方法展示分析结果。

3.数据存储：将分析的结果保存到本地磁盘，支持多种文件格式，如数据库、文本文件、Excel 等，确保数据存储的高效性和可访问性。

4.数据可视化：利用图形化库将分析结果呈现出来，展示包括词云图、柱状图、饼状图等数据可视化效果，以便用户更直观地理解分析结果。

5.图形用户界面 (GUI)：设计并实现一个图形化界面，允许用户输入目标网站地址，点击按钮启动爬取过程，并实时展示分析结果。GUI 应该友好、易用。

6.创新模块：在基础模块的实现上，增加一个具有创新性的功能模块，体现软件特色，可以是对数据的更深层次分析、新的可视化形式，或其他增强功能。

3. 整体构思和设计

整体设计的流程图如下：

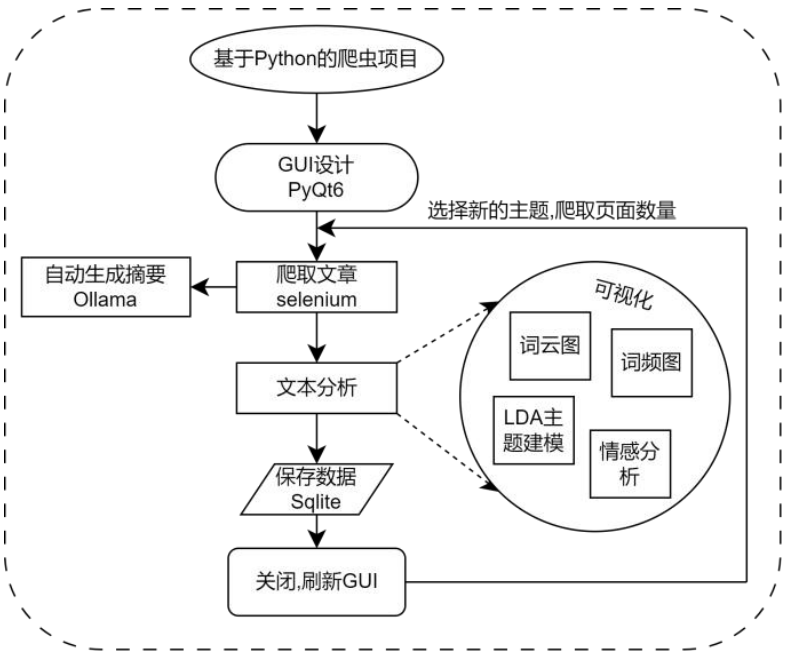

图 3.1 流程图

1.考虑到目标网站使用 JavaScript 进行动态加载，request 等静态爬取技术无法获取网站的搜索结果，故使用 selenium 库模拟浏览器行为进行爬取，利用 BeautifulSoup 库解析 html，对于具体的新闻页面，使用 request 库以快速获得新闻内容。

2.文本分析部分，使用 jieba 库进行分词、使用 collection 库进行词频分析、使用 SnowNLP 库进行情感分析、使用 WordCloud 库绘制词云图、使用 sklearn 库对文章进行 LDA 建模分类。

3.使用 Sqlite3 建立数据库存储爬取的文章文本、标题和链接。将绘制的词云图和词频图保存在本地。

4.图形用户界面使用 PyQt6、Qt-Designer 设计。

5.项目利用 Ollama 在本地部署了 Qwen2.5:3b 大模型，并利用 Ollama 库调用生成每篇新闻的摘要。

综上，项目导入的软件包如下：

```
import jieba # 中文分词库
import matplotlib.pyplot as plt # 绘图库
from collections import Counter # 用于统计词频
from wordcloud import WordCloud # 生成词云图
import pandas as pd
import requests
import re
from selenium import webdriver # 自动化测试工具，用于模拟浏览器行为
from selenium.webdriver.edge.service import Service
```

```
from selenium.webdriver.edge.options import Options
from bs4 import BeautifulSoup as bs  # HTML 解析库
import time
from PyQt6 import QtCore, QtGui, QtWidgets  # GUI 开发库
from PyQt6.QtGui import QPixmap
from matplotlib.backends.backend_qtagg import FigureCanvasQTAgg as FigureCanvas
from snownlp import SnowNLP # 中文情感分析库
import numpy as np
import sqlite3
from sklearn.decomposition import LatentDirichletAllocation # LDA 主题建模
from sklearn.feature_extraction.text import CountVectorizer # 文本向量化
from ollama import chat,ChatResponse # 调用本地部署的大模型
import socket # 网络通信库
import urllib3.util.connection as urllib3_cn
```

4. 模块设计

4.1 GUI 设计

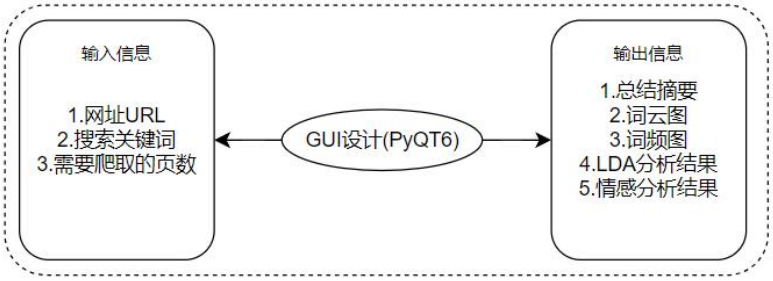

图 1　GUI 模块流程图

在 GUI 设计部分，项目采用了 PyQt6 和 Qt-Designer 进行界面开发，整体设计思路围绕简洁、易用和功能集成展开。首先，通过 Qt-Designer 工具快速搭建了主界面的布局，确保各个功能模块的合理分布。主界面分为多个区域，包括词云图、词频图、LDA 主题建模结果、情感分析结果以及输入信息区域，用户可以通过输入目标网址、关键词和爬取页数来启动爬虫和分析过程。

图 4.1 GUI 界面

用户输入信息后，点击“开始”按钮，系统会实时展示爬取的新闻摘要、词云图、词频图和 LDA 主题建模结果，并通过情感分析模块提供新闻的整体情感倾向（正面或负面）。此外，界面还支持将分析结果保存到本地，包括文本文件、图像文件以及 SQLite 数据库。

4.2 爬取文章模块设计

爬取文章模块是整个项目的核心部分，负责从目标网站抓取新闻内容，并进行初步的处理和过滤。该模块的设计思路主要分为以下几个步骤：

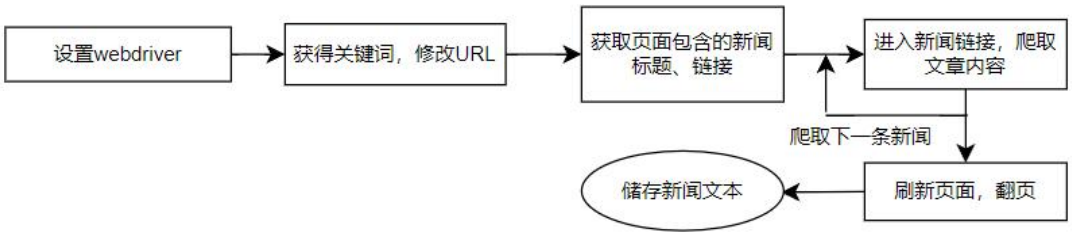

图 2　模块流程

由于目标网站的搜索结果页面是通过 JavaScript 动态加载的，传统的静态爬取技术（如 requests 库）无法获取完整的页面内容。因此，项目采用了 Selenium 库来模拟浏览器的访问行为。通过 Selenium，可以加载完整的网页内容，并获取动态生成的新闻链接和标题。

使用 Edge 浏览器作为驱动，配置了无头模式（headless），以提高爬取效率。禁用图片加载，减少不必要的资源消耗。忽略证书错误，确保爬取过程的顺利进行：

```
1.  edge_options=Options() # 配置 Selenium 的 Edge 浏览器选项
```

```
edge_options.add_argument("--headless")
edge_options.add_argument("--blink-settings=imagesEnabled=false")
edge_options.add_argument('--ignore-certificate-errors')
service=Service(executable_path=r"C:\Program Files (x86)\Microsoft\Edge\Application\msedgedriver.exe")
driver=webdriver.Edge(service=service, options=edge_options)
```

利用 driver 访问 url 后，刷新网页并等待 1s 使内容加载。获取页面的 html 并使用 beautifulsoup 解析：

```
url=f'https://so.news.cn/#search/1/{key}/{index}/0'
driver.get(url)
driver.refresh() # 刷新页面
time.sleep(0.5)
html=driver.page_source # 获取页面 HTML 源码
soup=bs(html,'lxml') # 使用 BeautifulSoup 解析 HTML
```

通过 Selenium 访问目标网站的搜索结果页面，使用 BeautifulSoup 解析 HTML 内容，提取新闻的标题和链接。为了确保数据的唯一性，使用集合（set）来过滤重复的链接和标题，若标题不重复，将该新闻的标题和链接存入字典：

```
item_divs = soup. Select('div.items > div.item') # 定位新闻条目
for item in item_divs:
    a_tag = item.select_one('div.title > a')
    if a_tag:
       href = a_tag['href']  # 获取新闻链接
       text = a_tag.get_text(strip=True) # 获取新闻标题
       if href not in seen_links and text not in seen_titles: # 去重
           seen_links.add(href)
           seen_titles.add(text)
            items.append(item)
           link_text_dict[href] = text # 存储链接和标题
```

在获取到新闻链接后，使用 requests 库对每个新闻链接进行请求，获取新闻的正文内容。为了提高爬取速度，使用 Session 对象来保持连接，并通过正则表达式对正文内容进行清理，去除多余的标点符号和括号内容，便于后续处理：

```
with requests.Session() as session:
for href, text in link_text_dict.items():
     url=href
     #print(url)
     response=session. Get(url,allow_redirects=False,headers=headers, timeout=5)
     html=response. Text
     soup=bs(html,'lxml') # 获取新闻正文
     word=soup.select_one('#detailContent').get_text(strip=True)
     if word=="":
      continue
     raw.append(word)
```

```
word=re.sub(r'（.*?）|\(.*?\)', '', word) # 去除括号内容
word=re.sub(r'[^\w\s]', '', word) # 去除标点符号
Word.append(word) # 存储处理后的新闻内容
titles.append(text)
links.Append(href)
```

4.3 自动摘要模块设计

自动摘要模块是项目的创新部分，旨在通过大模型生成新闻的简洁摘要，帮助用户快速理解新闻的核心内容。该模块的设计思路主要分为以下几个步骤：

1.模型选择与部署

项目选择了 Qwen2.5:3b 大模型作为摘要生成的核心工具。该模型具有较强的自然语言处理能力，能够生成简洁且准确的摘要。

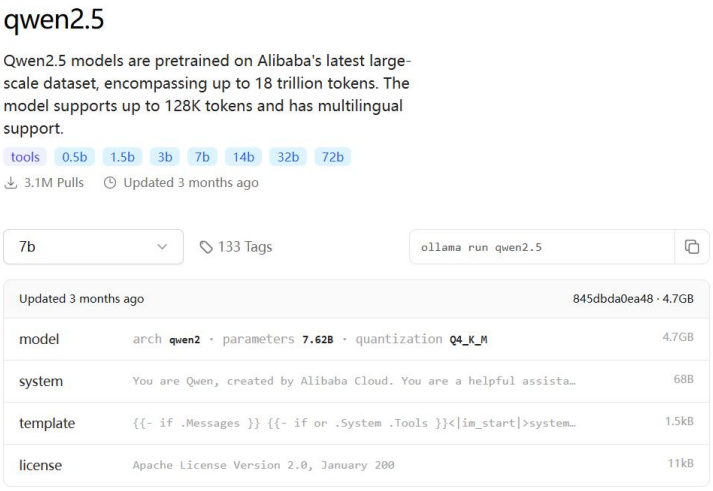

图 3 Ollama 官网上的模型界面

2.在命令行界面输入：ollama run qwen2.5，将模型下载至本地

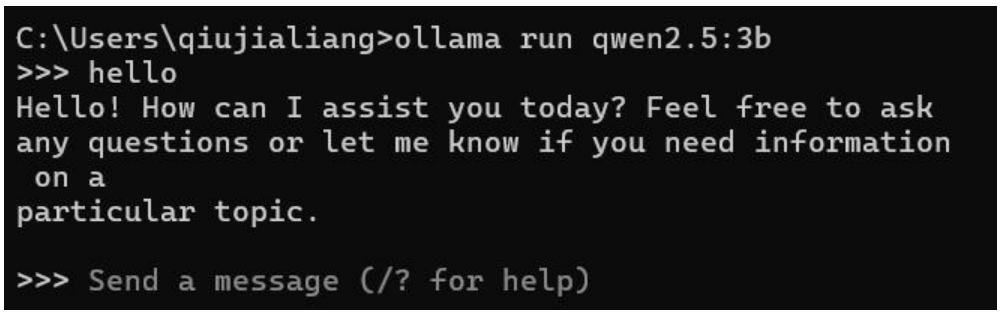

图 4.3.1 模型已部署在本地

如上图，通过 cmd 访问部署的模型，向其发送了“Hello”的消息并收到回应。

3.为了确保模型的快速响应和本地化处理，项目使用 Ollama 库在本地部署了该模型。通过 Ollama 提供的 API 接口与模型进行交互，生成新闻摘要。

在爬取到新闻正文后，自动摘要模块会调用本地部署的 Qwen2.5:3b 模型，生成每篇新闻的摘要，生成的摘要会实时显示在 GUI 界面中，用户可以通过界面查看每篇新闻的标题、链接和摘要内容。摘要展示部分通过 PyQt6 的 QLabel 组件实现，确保界面简洁且易于阅读。：

```
response: ChatResponse = chat(model="qwen2.5:3b", messages=[
        { # 调用大模型生成新闻摘要
            'role': 'system',
            'content': '''请提供一个简洁的摘要,概括这篇新闻的核
                    心内容和主要观点.
                        必须基于原文,尽量简洁,不多于 50 字.不可
                    分段.直接输出结果即可.''',
        },
        {
            'role': 'user',
            'content': word,
    }
])
```

进一步，可以使用大模型生成英文摘要，并在 GUI 中显示：

图 4　摘要生成样例

```
response_EN: ChatResponse = chat(model="qwen2.5:3b", messages=[
        { # 调用大模型生成英文新闻摘要
            'role': 'system',
            'content': '''将给出的摘要翻译为英文.''',
        },
        {
            'role': 'user',
            'content': response['message']['content'],
        },
])
```

4.3 文本分析和可视化模块设计

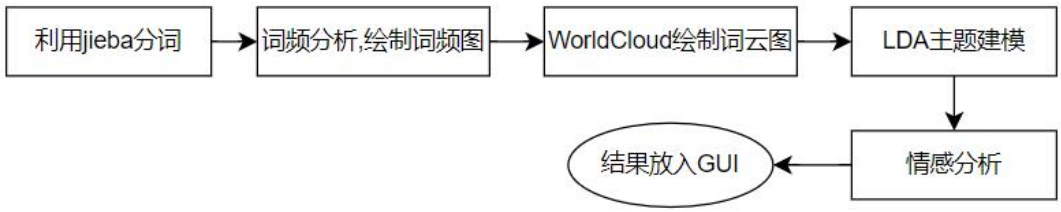

图 5　模块流程

4.3.1 分词

首先，使用 jieba 库对新闻文本进行分词，将文本分割成单独的词语。并根据一些常见停用词（如“的”、“是”、“在”等）对结果进行了过滤：

```
words=jieba.cut(text)
word_list=list(words)
stopwords = { # 定义停用词列表
        '的', '是', '在', '和', '了', '有', '与', '对', '为', '也', '说', '根据', '报道',
        '专家', '称', '表示', '发言人', '记者', '提出', '指出', '现', '时', '将', '当',
        '来', '到', '关于', '同', '这些', '今年', '月', '日', '去年', '本', '后', '那',
        '此', '以后', '以后', '前', '之后', '方面', '开始', '结束', '有', '看到', '通过',
        '认为', '此', '情况', '必须', '进行', '实施', '消息', '表示', '听到', '现已', '已经'
        ,'新华社记者'
    }
filtered_words = [word for word in word_list if word not in stopwords and len(word.strip()) > 1]  # 过滤停用词
word_list=list(filtered_words)
```

4.3.2 词频分析

通过 collections.Counter 统计每个词语的出现频率，生成词频数据。将分词转化为 DataFrame 后进行排序。绘制前十词频的柱状图并绘制在 UI 上：

```
df=pd.DataFrame(word_freq.items(),columns=["Word","Freq"]).sort_values(by="Freq",ascending=False) # 绘制词频柱状图
fig2,ax2=plt.subplots(figsize=(10,10))
plt.rcParams['font.sans-serif'] = ['SimHei'] # 设置中文字体
```

```
df.head(10).plot(kind='bar',x='Word',y='Freq',legend=False,color='skyblue',ax=ax2)
ax2.set_xlabel('词语', fontsize=10, fontproperties='SimHei')
ax2.set_ylabel('频率', fontsize=10)
ax2.set_xticklabels(df.head(10)['Word'], rotation=45, ha='right', fontsize=8)
ax2.tick_params(axis='x', labelsize=8)
layout = QtWidgets.QVBoxLayout(self.pinshu)  # 将词频图嵌入到 GUI 界面中
canvas = FigureCanvas(fig2)
self.fig_pinshu=fig2
layout.addWidget(canvas)
self.pinshu.setLayout(layout)
canvas.draw()
```

4.3.3 词云图

使用 WordCloud 绘制词云图，词云图能够直观地展示高频词汇，词语的大小与其出现频率成正比。并嵌入 UI：

```
wordcloud=WordCloud(font_path='C:/Windows/Fonts/simfang.ttf',background_color='white',width=800,height=600).generate_from_frequencies(word_freq)
fig=plt.figure(figsize=(10,10))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.tight_layout()
canvas = FigureCanvas(fig) # 将词云图嵌入到 GUI 界面中
self.fig_wordcloud=fig
layout = QtWidgets.QVBoxLayout(self.wordcloud)
layout.addWidget(canvas)
self.wordcloud.setLayout(layout)
canvas.draw()
```

4.3.4 LDA 主题建模

LDA(Latent Dirichlet Allocation)由 Blei, David M.、Ng, Andrew Y.、Jordan 于 2003 年提出，是一种主题模型，它可以将每篇文档的主题以概率分布的形式给出，从而通过分析一些文档抽取出它们的主题（分布）出来后，便可以根据主题（分布）进行主题聚类或文本分类。

使用 sklearn 库中的 LatentDirichletAllocation（LDA）模型对新闻文本进行主题建模，使用 CountVectorizer 将分词后的文本转换为词频矩阵。使用 LatentDiri

chletAllocation 进行主题建模，生成不同主题的词云图，展示每个主题的关键词。

例如，通过 LDA 模型，新闻可以被分类为“消费者”、“经济”、“平台”等主题，每个主题的关键词通过不同颜色的词云图展示，直观且易于理解。

```
vectorizer = CountVectorizer(stop_words=list(stopwords))  # 文本向量化
X = vectorizer.fit_transform(word_list) # 将文本转换为词频矩阵
lda=LatentDirichletAllocation(n_components=3,random_state=42) # 初始化 LDA 模型
lda.fit(X) # 训练 LDA 模型
colormaps = ['Blues', 'Reds', 'Greens'] # 绘制 LDA 主题词云图
def plot_wordcloud_for_topic(topic_idx, feature_names, topic_weights,ax, colormap):
        top_n = 5
        top_indices = topic_weights.argsort()[-top_n:][::-1]
        word_freq = {feature_names[i]: topic_weights[i] for i in top_indices}
    wordcloud = WordCloud(font_path='C:/Windows/Fonts/simfang.ttf',width=800, height=400, background_color="white",colormap=colormap).generate_from_frequencies(word_freq)
        ax.imshow(wordcloud, interpolation="bilinear")
    ax.axis("off")
fig3, axes = plt.subplots(1, 3, figsize=(15, 5)) # 将 LDA 主题图嵌入到 GUI 界面中
for idx,(topic, ax, cmap) in enumerate(zip(lda.components_, axes, colormaps)):
        print(idx)
        print([vectorizer.get_feature_names_out()[i] for i in topic.argsort()[-5:]])
        plot_wordcloud_for_topic(idx, vectorizer.get_feature_names_out(), topic, ax, cmap)
plt.tight_layout()
canvas=FigureCanvas(fig3)
self.fig_LDA=fig3
layout = QtWidgets.QVBoxLayout(self.LDA)
layout.addWidget(canvas)
self.LDA.setLayout(layout)
canvas.draw()
```

4.3.5 情感分析

情感分析（Sentiment Analysis），也称为意见挖掘（Opinion Mining），是自然语言处理（NLP）领域中的一个重要任务，广泛应用于舆情监控、市场调研、

客户反馈分析等领域。它的目标是通过分析文本数据来识别和提取其中的情感信息。

使用 SnowNLP 库对每篇新闻进行情感分析，计算新闻的情感值（0 到 1 之间），并根据情感值判断新闻的情感倾向（正面或负面）。若结果正向，在 UI 中打印绿色笑脸图像，图像由 GPT 生成，已保存在本地：

```
raws=self.raw# 将 LDA 主题图嵌入到 GUI 界面中
res=[]
for raw in raws:
    s=SnowNLP(raw)  # 使用 SnowNLP 进行情感分析
    res.append(s.sentiments)  # 获取情感值（0 到 1 之间）
ave=np.mean(res)  # 计算平均情感值
print(ave)
  if ave>0.5:  # 根据情感值显示不同的结果
      self.label_s.setStyleSheet("QLabel { color: green; }")
  self.label_s.setText(f"{ave:.5f} 正面")
      pixmap=QPixmap(r"keshe\green.png") # 加载绿色笑脸图标
    self.label.setPixmap(pixmap)
    self.label.setScaledContents(True)
else:
    self.label_s.setStyleSheet("QLabel { color: red; ")
    self.label_s.setText(f"{ave:.5f} 负面")
    pixmap=QPixmap(r"keshe\red.png")
    self.label.setPixmap(pixmap)
    self.label.setScaledContents(True)
```

4.4 保存文件模块设计

图 6　模块流程

4.4.1 保存至 Sql

为了确保数据的结构化存储和高效查询，项目使用 SQLite 数据库保存爬取的新闻标题、链接、正文内容以及分析结果。通过 sqlite3 库，将数据存储到本地数据库中：

```
conn=sqlite3.connect('keshe/word.db') # 将数据保存到 SQLite 数据库
cursor=conn.cursor()
cursor.execute(
      '''
        CREATE TABLE IF NOT EXISTS texts (
        id INTEGER PRIMARY KEY,
        text_content TEXT,
        type TEXT
        )
       '''
)
cursor.execute(f'DELETE FROM texts') # 清空表
conn.commit()
for word in Word:  # 插入分词结果、原始新闻内容、标题和链接
    cursor.execute('INSERT INTO texts (text_content, type) VALUES (?, ?)',(word,'Word'))
for raw in raws:
    cursor.execute('INSERT INTO texts (text_content, type) VALUES (?, ?)',(raw,'Raws'))
for title in titles:
    cursor.execute('INSERT INTO texts (text_content, type) VALUES (?, ?)',(title,'titles'))
for link in links:
    cursor.execute('INSERT INTO texts (text_content, type) VALUES (?, ?)',(link,'links'))
conn.commit()
cursor.close()
conn.close()
```

4.4.2 保存至本地

项目支持将爬取的新闻内容、分词结果、词频分析结果以及生成的图像文件（如词云图、词频图、LDA 主题图）保存到本地磁盘。文件保存格式包括文本文件（.txt）和图像文件（.jpg）。使用 with open 语句确保文件操作的安全性：

```
with open("keshe/save_Word.txt",'w',encoding='utf-8') as f:
for word in Word: # 将分词结果保存到本地文件
      f.write(word+'\n')
      print("分词写入完毕")
with open("keshe/save_raw.txt",'w',encoding='utf-8') as f:
```

```
        for raw in raws: # 将原始新闻内容保存到本地文件
            f.write(raw+'\n')
            print("原文本写入完毕")
    self.fig_pinshu.savefig('keshe/pinshu.jpg')  # 保存词频图、词云图和 LDA 主题图
    self.fig_wordcloud.savefig('keshe/wordcloud.jpg')
    self.fig_LDA.savefig('keshe/LDA.jpg')
    print("图像写入完毕")
```

5. 运行结果和调试

以“大数据”为关键词进行爬取，爬取页数设置为 10，结果如下：

图 5.1 运行结果

从运行结果来看，项目成功爬取了 44 篇与“大数据”相关的新闻，发现“数据”、“消费者”、“平台”等词出现的频率较高。LDA 主题建模将新闻分为三类，关键词分别为“消费者”、“经济”、“平台”、“服务”和“数据”、“产业”，帮助用户快速理解新闻的核心话题。情感分析结果显示，所有新闻的平均情感值为 0.88，表明整体情感倾向为正面。此外，每篇新闻的摘要通过大模型生成，简洁地概括了新闻的核心内容。

点击保存按钮，新闻文本、标题和链接保存至了数据库中：

	id	text_content	type
120	120	税收大数据彰显7月份我国经济稳中有进	titles
121	121	深化接诉即办改革 民生大数据助超大城市智慧治理	titles
122	122	民生大数据助力超大城市智慧治理	titles
123	123	大数据"杀熟"、自动续费……数字消费维权堵点频发	titles
124	124	时空大数据需求持续景气 多家上市公司推进商业化应用	titles
125	125	贵阳大数据科创城贵阳超级大脑项目开始浇筑	titles
126	126	花钱买"乐子"，大数据说这就是美好生活	titles
127	127	追光丨花钱买"乐子"，大数据说这就是美好生活	titles
128	128	除了赛龙舟 端午假期还有哪些文旅消费热点？大数据带...	titles
129	129	全国首个国家海洋大数据服务平台发布	titles
130	130	山西大同：打造助推京津冀大数据发展的绿色云谷	titles
131	131	新时代中国调研行之看区域·中部篇｜山西大同：打造助...	titles
132	132	河南：大数据里的麦收	titles
133	133	https://www.news.cn/politics/20241225/7e422310929c4...	links
134	134	http://www.news.cn/politics/20241212/f59ea73898a148...	links
135	135	http://www.news.cn/politics/20241210/791e2fe14a9747f...	links
136	136	http://www.news.cn/20241119/19b5b2c3cceb4efe93cca...	links
137	137	http://www.news.cn/politics/20241108/75a148c8a19340...	links
138	138	https://www.news.cn/tech/20241106/9e7a64725ef94878...	links
139	139	https://www.news.cn/comments/20241102/be63bd2c58...	links

图 5.2 数据库中的文本

图片、文本也以图像和文本文件形式保存在了本地：

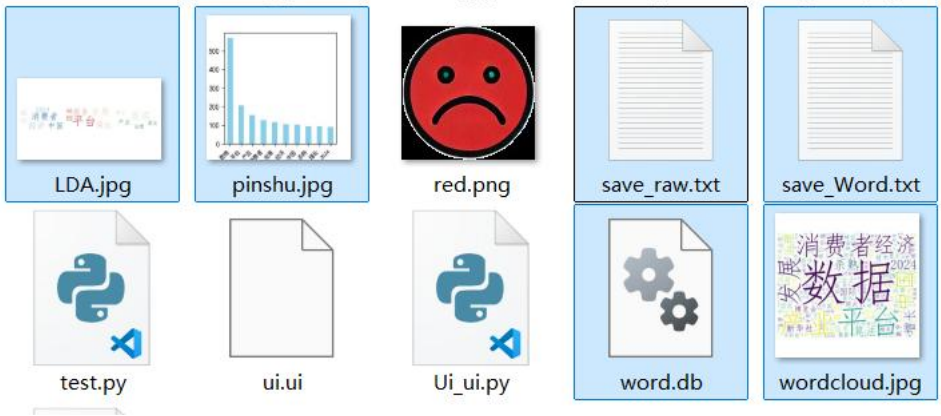

图 5.3 本地保存的文件

6. 心得体会

在这次计算思维与 Python 课程设计中，我深刻体会到了模块化程序设计和面向对象编程的重要性。通过使用 Python 的各种库和工具，我成功地设计并实现了一个基于 PyQt 和 selenium 的网络爬虫软件。这个项目不仅让我掌握了网络爬虫、文本分析、数据存储和数据可视化的技术，还让我在 GUI 设计和创新功能实现方面有了更多的实践机会。

在项目的设计过程中，我首先考虑到了目标网站的动态加载机制，选择了 selenium 库来模拟浏览器行为，从而成功绕过了反爬虫技术。此外，通过使用 BeautifulSoup 库解析 HTML，我能够高效地提取网页中的有用信息。

在文本分析部分，我学习并应用了 jieba 库进行分词，collection 库进行词频分析，以及 SnowNLP 库进行情感分析。这些技术让我能够从爬取的新闻文本中提取出高频词汇，并通过词云图和柱状图等形式直观地展示分析结果。同时，我还在项目中引入了 Ollama 库来调用 Qwen2.5:3b 大模型，生成每篇新闻的摘要，这使得项目更具创新性和实用性。

此外，项目的 GUI 设计让我在用户体验方面有了更深的理解。通过 PyQt6 和 Qt-Designer，我设计了一个简洁易用的界面，使得用户可以方便地输入目标网站地址、关键词和爬取页数，并实时查看分析结果。

最后，通过 Sqlite3 库，我将爬取的新闻文本、标题和链接存储在数据库中，确保了数据的持久性和可访问性。同时，通过 matplotlib 和 WordCloud 库，我将分析结果以图形化的方式呈现出来，使得用户可以更直观地理解数据。

总之，这次课程设计让我在多个方面都得到了锻炼和提高，不仅巩固了我 Python 编程的能力，还让我在爬虫技术、文本分析、数据可视化和 GUI 设计等方面有了更深入的理解。我相信，这次项目的经验将对我未来的学习和工作产生积极的影响。

附录：代码

work.py 源代码

```
import jieba
import matplotlib.pyplot as plt
from collections import Counter
from wordcloud import WordCloud
import pandas as pd
import requests
import re
from selenium import webdriver
from selenium.webdriver.edge.service import Service
from selenium.webdriver.edge.options import Options
from bs4 import BeautifulSoup as bs
import time
from PyQt6 import QtCore, QtGui, QtWidgets
from PyQt6.QtGui import QPixmap
from matplotlib.backends.backend_qtagg import FigureCanvasQTAgg as FigureCanvas
from snownlp import SnowNLP
import numpy as np
import sqlite3
```

```
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer
from ollama import chat,ChatResponse
import socket
import urllib3.util.connection as urllib3_cn
def allowed_gai_family():
    return socket.AF_INET
urllib3_cn.allowed_gai_family = allowed_gai_family

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1366, 803)
        self.centralwidget = QtWidgets.QWidget(parent=MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.groupBox = QtWidgets.QGroupBox(parent=self.centralwidget)
        self.groupBox.setGeometry(QtCore.QRect(20, 20, 1321, 751))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(20)
        font.setBold(True)
        font.setItalic(False)
        font.setUnderline(False)
        font.setWeight(75)
        self.groupBox.setFont(font)
        self.groupBox.setCheckable(False)
        self.groupBox.setObjectName("groupBox")
        self.label_7 = QtWidgets.QLabel(parent=self.groupBox)
        self.label_7.setGeometry(QtCore.QRect(20, 40, 71, 21))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.label_7.setFont(font)
        self.label_7.setObjectName("label_7")
        self.label_2 = QtWidgets.QLabel(parent=self.groupBox)
        self.label_2.setGeometry(QtCore.QRect(10, 350, 91, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.label_2.setFont(font)
        self.label_2.setObjectName("label_2")
        self.pinshu = QtWidgets.QWidget(parent=self.groupBox)
        self.pinshu.setGeometry(QtCore.QRect(90, 340, 300, 300))
        self.pinshu.setObjectName("pinshu")
        self.wordcloud = QtWidgets.QWidget(parent=self.groupBox)
        self.wordcloud.setGeometry(QtCore.QRect(90, 20, 300, 300))
```

```
63.         self.wordcloud.setObjectName("wordcloud")
64.         self.label_6 = QtWidgets.QLabel(parent=self.groupBox)
65.         self.label_6.setGeometry(QtCore.QRect(400, 40, 121, 31))
66.         font = QtGui.QFont()
67.         font.setPointSize(12)
68.         self.label_6.setFont(font)
69.         self.label_6.setObjectName("label_6")
70.         self.label_8 = QtWidgets.QLabel(parent=self.groupBox)
71.         self.label_8.setGeometry(QtCore.QRect(410, 380, 91, 31))
72.         font = QtGui.QFont()
73.         font.setPointSize(12)
74.         self.label_8.setFont(font)
75.         self.label_8.setObjectName("label_8")
76.         self.LDA = QtWidgets.QWidget(parent=self.groupBox)
77.         self.LDA.setGeometry(QtCore.QRect(400, 420, 521, 300))
78.         self.LDA.setObjectName("LDA")
79.         self.label_s = QtWidgets.QLabel(parent=self.groupBox)
80.         self.label_s.setGeometry(QtCore.QRect(1060, 400, 151, 31))
81.         font = QtGui.QFont()
82.         font.setPointSize(12)
83.         self.label_s.setFont(font)
84.         self.label_s.setText("")
85.         self.label_s.setObjectName("label_s")
86.         self.label_5 = QtWidgets.QLabel(parent=self.groupBox)
87.         self.label_5.setGeometry(QtCore.QRect(960, 400, 91, 31))
88.         font = QtGui.QFont()
89.         font.setPointSize(12)
90.         self.label_5.setFont(font)
91.         self.label_5.setObjectName("label_5")
92.         self.groupBox_2 = QtWidgets.QGroupBox(parent=self.groupBox)
93.         self.groupBox_2.setGeometry(QtCore.QRect(950, 450, 341, 261))
94.         font = QtGui.QFont()
95.         font.setPointSize(20)
96.         self.groupBox_2.setFont(font)
97.         self.groupBox_2.setObjectName("groupBox_2")
98.         self.label_3 = QtWidgets.QLabel(parent=self.groupBox_2)
99.         self.label_3.setGeometry(QtCore.QRect(10, 40, 61, 31))
100.         font = QtGui.QFont()
101.         font.setPointSize(12)
102.         self.label_3.setFont(font)
103.         self.label_3.setObjectName("label_3")
104.         self.netname = QtWidgets.QLineEdit(parent=self.groupBox_2)
105.         self.netname.setGeometry(QtCore.QRect(90, 40, 241, 41))
```

```
106.         font = QtGui.QFont()
107.         font.setPointSize(12)
108.         self.netname.setFont(font)
109.         self.netname.setText("")
110.         self.netname.setObjectName("netname")
111.         self.key = QtWidgets.QLineEdit(parent=self.groupBox_2)
112.         self.key.setGeometry(QtCore.QRect(90, 90, 241, 41))
113.         font = QtGui.QFont()
114.         font.setPointSize(12)
115.         self.key.setFont(font)
116.         self.key.setObjectName("key")
117.         self.label_4 = QtWidgets.QLabel(parent=self.groupBox_2)
118.         self.label_4.setGeometry(QtCore.QRect(0, 90, 71, 31))
119.         font = QtGui.QFont()
120.         font.setPointSize(12)
121.         self.label_4.setFont(font)
122.         self.label_4.setObjectName("label_4")
123.         self.start = QtWidgets.QPushButton(parent=self.groupBox_2)
124.         self.start.setGeometry(QtCore.QRect(0, 210, 93, 28))
125.         font = QtGui.QFont()
126.         font.setPointSize(12)
127.         self.start.setFont(font)
128.         self.start.setObjectName("start")
129.         self.save = QtWidgets.QPushButton(parent=self.groupBox_2)
130.         self.save.setGeometry(QtCore.QRect(110, 210, 93, 28))
131.         font = QtGui.QFont()
132.         font.setPointSize(12)
133.         self.save.setFont(font)
134.         self.save.setObjectName("save")
135.         self.close = QtWidgets.QPushButton(parent=self.groupBox_2)
136.         self.close.setGeometry(QtCore.QRect(220, 210, 93, 28))
137.         font = QtGui.QFont()
138.         font.setPointSize(12)
139.         self.close.setFont(font)
140.         self.close.setObjectName("close")
141.         self.label_9 = QtWidgets.QLabel(parent=self.groupBox_2)
142.         self.label_9.setGeometry(QtCore.QRect(10, 140, 71, 31))
143.         font = QtGui.QFont()
144.         font.setPointSize(12)
145.         self.label_9.setFont(font)
146.         self.label_9.setObjectName("label_9")
147.         self.page = QtWidgets.QLineEdit(parent=self.groupBox_2)
148.         self.page.setGeometry(QtCore.QRect(90, 140, 241, 41))
149.         font = QtGui.QFont()
```

```
150.         font.setPointSize(12)
151.         self.page.setFont(font)
152.         self.page.setObjectName("page")
153.         self.scrollArea = QtWidgets.QScrollArea(parent=self.groupBox)
154.         self.scrollArea.setGeometry(QtCore.QRect(420, 80, 851, 291))
155.         self.scrollArea.setWidgetResizable(True)
156.         self.scrollArea.setObjectName("scrollArea")
157.         self.scrollAreaWidgetContents = QtWidgets.QWidget()
158.         self.scrollAreaWidgetContents.setGeometry(QtCore.QRect(0, 0, 849, 28
9))
159.         self.scrollAreaWidgetContents.setObjectName("scrollAreaWidgetContents
")
160.         self.title = QtWidgets.QLabel(parent=self.scrollAreaWidgetContents)
161.         self.title.setGeometry(QtCore.QRect(0, 0, 821, 281))
162.         font = QtGui.QFont()
163.         font.setPointSize(12)
164.         self.title.setFont(font)
165.         self.title.setText("")
166.         self.title.setAlignment(QtCore.Qt.AlignmentFlag.AlignLeading|QtCore.Qt.
AlignmentFlag.AlignLeft|QtCore.Qt.AlignmentFlag.AlignTop)
167.         self.title.setWordWrap(True)
168.         self.title.setObjectName("title")
169.         self.scrollArea.setWidget(self.title)
170.         self.res = QtWidgets.QLabel(parent=self.groupBox)
171.         self.res.setGeometry(QtCore.QRect(1080, 40, 191, 31))
172.         font = QtGui.QFont()
173.         font.setPointSize(12)
174.         self.res.setFont(font)
175.         self.res.setText("")
176.         self.res.setObjectName("res")
177.         self.label = QtWidgets.QLabel(parent=self.groupBox)
178.         self.label.setGeometry(QtCore.QRect(1240, 390, 60, 60))
179.         font = QtGui.QFont()
180.         font.setPointSize(12)
181.         self.label.setFont(font)
182.         self.label.setText("")
183.         self.label.setObjectName("label")
184.         MainWindow.setCentralWidget(self.centralwidget)
185.         self.menubar = QtWidgets.QMenuBar(parent=MainWindow)
186.         self.menubar.setGeometry(QtCore.QRect(0, 0, 1366, 26))
187.         self.menubar.setObjectName("menubar")
188.         MainWindow.setMenuBar(self.menubar)
189.         self.statusbar = QtWidgets.QStatusBar(parent=MainWindow)
```

```
190.         self.statusbar.setObjectName("statusbar")
191.         MainWindow.setStatusBar(self.statusbar)
192.
193.         self.retranslateUi(MainWindow)
194.         QtCore.QMetaObject.connectSlotsByName(MainWindow)
195.
196.     def retranslateUi(self, MainWindow):
197.         _translate = QtCore.QCoreApplication.translate
198.         MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
199.         self.groupBox.setTitle(_translate("MainWindow", "分析"))
200.         self.label_7.setText(_translate("MainWindow", "词云图"))
201.         self.label_2.setText(_translate("MainWindow", "词频图"))
202.         self.label_6.setText(_translate("MainWindow", "题目及摘要:"))
203.         self.label_8.setText(_translate("MainWindow", "LDA"))
204.         self.label_5.setText(_translate("MainWindow", "情感分析:"))
205.         self.groupBox_2.setTitle(_translate("MainWindow", "输入信息"))
206.         self.label_3.setText(_translate("MainWindow", "网址:"))
207.         self.label_4.setText(_translate("MainWindow", "关键词:"))
208.         self.start.setText(_translate("MainWindow", "开始"))
209.         self.save.setText(_translate("MainWindow", "保存"))
210.         self.close.setText(_translate("MainWindow", "关闭"))
211.         self.label_9.setText(_translate("MainWindow", "页数:"))
212.         self.netname.setText("https://so.news.cn/")
213.         self.netname.textChanged.connect(self.showText)
214.         self.key.textChanged.connect(self.showText)
215.         self.start.clicked.connect(self.startFun)
216.         self.close.clicked.connect(self.endFun)
217.         self.save.clicked.connect(self.saveFun)
218.     def showText(self):
219.         self.netname.setText(self.netname.text())
220.         self.key.setText(self.key.text())
221.         #print(self.netname.text())
222.
223.     def getWord(self):
224.         key=self.key.text()
225.         page=self.page.text()
226.         if page=="":
227.             page=2
228.         if key=="":
229.             key='人工智能'
230.         print(key)
231.         print(page)
232.         edge_options=Options()
233.         edge_options.add_argument("--headless")
```

```
        edge_options.add_argument("--blink-settings=imagesEnabled=false")
        edge_options.add_argument('--ignore-certificate-errors')
        service=Service(executable_path=r"C:\Program Files (x86)\Microsoft\Edge\Application\msedgedriver.exe")
        driver=webdriver.Edge(service=service,options=edge_options)
        seen_links = set()
        seen_titles=set()
        items = []
        link_text_dict = {}
        for index in range(1,int(page)+1):
            url=f'https://so.news.cn/#search/1/{key}/{index}/0'
            driver.get(url)
            driver.refresh()
            time.sleep(0.5)
            html=driver.page_source
            soup=bs(html,'lxml')
            item_divs = soup.select('div.items > div.item')
            for item in item_divs:
                a_tag = item.select_one('div.title > a')
                if a_tag:
                    href = a_tag['href']
                    text = a_tag.get_text(strip=True)
                    if href not in seen_links and text not in seen_titles:
                        seen_links.add(href)
                        seen_titles.add(text)
                        items.append(item)
                        link_text_dict[href] = text
        driver.quit()
        Word=[]
        raw=[]
        titles=[]
        links=[]
        i=1
        headers={
            'user-agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36 Edg/131.0.0.0'
        }
        with requests.Session() as session:
            for href, text in link_text_dict.items():
                url=href
                #print(url)
                response=session.get(url,allow_redirects=False,headers=headers,timeout=5)
```

```
                html=response.text
                soup=bs(html,'lxml')
                word=soup.select_one('#detailContent').get_text(strip=True)
                if word=="":
                   continue
                raw.append(word)
                word=re.sub(r'（.*?）|\(.*?\)', '', word)
                word=re.sub(r'[^\w\s]', '', word)
                Word.append(word)
                titles.append(text)
                links.append(href)
                response: ChatResponse = chat(model="qwen2.5:3b", messages=[
                    {
                        'role': 'system',
                        'content': '''请提供一个简洁的摘要,概括这篇新闻的核心内容和主要观点.
                                      必须基于原文,尽量简洁,不多于 50 字.不可分段.直接输出结果即可.''',
                    },
                      {
                          'role': 'user',
                          'content': word,
                     },
                ])
                #print(response.message.content)
                self.title.setText(self.title.text()+'\n'+f"{i}.{text}:{href}"+'\n'+f"摘要:{response['message']['content']}")
                i+=1
        self.res.setStyleSheet("QLabel { color: red; }")
        self.res.setText(f"获取新闻{i-1}篇")
        #print(Word)
        return Word,raw,titles,links
    def plot_pinsu_word(self):
        Word=self.Word
        raw=self.raw
        text=""
        for word in Word:
            text=text+word
        words=jieba.cut(text)
        word_list=list(words)
        stopwords = {
            '的', '是', '在', '和', '了', '有', '与', '对', '为', '也', '说', '根据', '报道',
```

```
313.             '专家', '称', '表示', '发言人', '记者', '提出', '指出', '现', '时', '将', '当',
314.             '来', '到', '关于', '同', '这些', '今年', '月', '日', '去年', '本', '后', '那',
315.             '此', '以后', '以后', '前', '之后', '方面', '开始', '结束', '有', '看到', '通过',
316.             '认为', '此', '情况', '必须', '进行', '实施', '消息', '表示', '听到', '现已', '已经'
317.             ,'新华社记者'
318.         }
319.         filtered_words = [word for word in word_list if word not in stopwords and len(word.strip()) > 1]
320.         word_list=list(filtered_words)
321.         self.res.setText(self.res.text()+f"总词数{len(word_list)}个")
322.         for stopword in stopwords:
323.             for word in raw:
324.                 word = word.replace(stopword, '')
325.                 word_freq=Counter(filtered_words)
326.         df=pd.DataFrame(word_freq.items(),columns=["Word","Freq"]).sort_values(by="Freq",ascending=False)
327.         fig2,ax2=plt.subplots(figsize=(10,10))
328.         plt.rcParams['font.sans-serif'] = ['SimHei']
329.         df.head(10).plot(kind='bar',x='Word',y='Freq',legend=False,color='skyblue',ax=ax2)
330.         ax2.set_xlabel('词语', fontsize=10, fontproperties='SimHei')
331.         ax2.set_ylabel('频率', fontsize=10)
332.         ax2.set_xticklabels(df.head(10)['Word'], rotation=45, ha='right', fontsize=8)
333.         ax2.tick_params(axis='x', labelsize=8)
334.         layout = QtWidgets.QVBoxLayout(self.pinshu)
335.         canvas = FigureCanvas(fig2)
336.         self.fig_pinshu=fig2
337.         layout.addWidget(canvas)
338.         self.pinshu.setLayout(layout)
339.         canvas.draw()
340.
341.         wordcloud=WordCloud(font_path='C:/Windows/Fonts/simfang.ttf',background_color='white',width=800,height=600).generate_from_frequencies(word_freq)
342.         fig=plt.figure(figsize=(10,10))
343.         plt.imshow(wordcloud,interpolation='bilinear')
344.         plt.axis('off')
345.         plt.tight_layout()
346.         canvas = FigureCanvas(fig)
```

```
347.         self.fig_wordcloud=fig
348.         layout = QtWidgets.QVBoxLayout(self.wordcloud)
349.         layout.addWidget(canvas)
350.         self.wordcloud.setLayout(layout)
351.         canvas.draw()
352.
353.         vectorizer = CountVectorizer(stop_words=list(stopwords))
354.         X = vectorizer.fit_transform(word_list)
355.         lda=LatentDirichletAllocation(n_components=3,random_state=42)
356.         lda.fit(X)
357.         colormaps = ['Blues', 'Reds', 'Greens']
358.         def plot_wordcloud_for_topic(topic_idx, feature_names, topic_weights,a
x, colormap):
359.             top_n = 5
360.             top_indices = topic_weights.argsort()[-top_n:][::-1]
361.             word_freq = {feature_names[i]: topic_weights[i] for i in top_indic
es}
362.
363.             wordcloud = WordCloud(font_path='C:/Windows/Fonts/simfang.ttf',wid
th=800, height=400
364.                                   , background_color="white",colormap=colormap).
generate_from_frequencies(word_freq)
365.             ax.imshow(wordcloud, interpolation="bilinear")
366.             ax.axis("off")
367.         fig3, axes = plt.subplots(1, 3, figsize=(15, 5))
368.         for idx,(topic, ax, cmap) in enumerate(zip(lda.components_, axes, colo
rmaps)):
369.             print(idx)
370.             print([vectorizer.get_feature_names_out()[i] for i in topic.argsor
t()[-5:]])
371.             plot_wordcloud_for_topic(idx, vectorizer.get_feature_names_out
(), topic, ax, cmap)
372.         plt.tight_layout()
373.         canvas=FigureCanvas(fig3)
374.         self.fig_LDA=fig3
375.         layout = QtWidgets.QVBoxLayout(self.LDA)
376.         layout.addWidget(canvas)
377.         self.LDA.setLayout(layout)
378.         canvas.draw()
379.
380.     def get_s(self):
381.         raws=self.raw
382.         res=[]
383.         for raw in raws:
```

```
            s=SnowNLP(raw)
            res.append(s.sentiments)
        ave=np.mean(res)
        print(ave)
        if ave>0.5:
            self.label_s.setStyleSheet("QLabel { color: green; }")
            self.label_s.setText(f"{ave:.5f} 正面")
            pixmap=QPixmap(r"keshe\green.png")
            self.label.setPixmap(pixmap)
            self.label.setScaledContents(True)
        else:
            self.label_s.setStyleSheet("QLabel { color: red; }")
            self.label_s.setText(f"{ave:.5f} 负面")
            pixmap=QPixmap(r"keshe\red.png")
            self.label.setPixmap(pixmap)
            self.label.setScaledContents(True)
    def saveFun(self):
        Word=self.Word
        raws=self.raw
        titles=self.titles
        links=self.links
        with open("keshe/save_Word.txt",'w',encoding='utf-8') as f:
            for word in Word:
                f.write(word+'\n')
            print("分词写入完毕")
        with open("keshe/save_raw.txt",'w',encoding='utf-8') as f:
            for raw in raws:
                f.write(raw+'\n')
            print("原文本写入完毕")
        self.fig_pinshu.savefig('keshe/pinshu.jpg')
        self.fig_wordcloud.savefig('keshe/wordcloud.jpg')
        self.fig_LDA.savefig('keshe/LDA.jpg')
        print("图像写入完毕")
        conn=sqlite3.connect('keshe/word.db')
        cursor=conn.cursor()
        cursor.execute(
            '''
            CREATE TABLE IF NOT EXISTS texts (
            id INTEGER PRIMARY KEY,
            text_content TEXT,
            type TEXT
            )
            '''
```

```
427.         )
428.         cursor.execute(f'DELETE FROM texts')
429.         conn.commit()
430.         for word in Word:
431.             cursor.execute('INSERT INTO texts (text_content, type) VALUE
S (?, ?)',(word,'Word'))
432.         for raw in raws:
433.             cursor.execute('INSERT INTO texts (text_content, type) VALUE
S (?, ?)',(raw,'Raws'))
434.         for title in titles:
435.             cursor.execute('INSERT INTO texts (text_content, type) VALUE
S (?, ?)',(title,'titles'))
436.         for link in links:
437.             cursor.execute('INSERT INTO texts (text_content, type) VALUE
S (?, ?)',(link,'links'))
438.         conn.commit()
439.         cursor.close()
440.         conn.close()
441.     def startFun(self):
442.         #key=input()
443.         def del_layout(self,layout):
444.             if layout is not None:
445.                 for i in reversed(range(layout.count())):
446.                     widget = layout.itemAt(i).widget()
447.                     if widget is not None:
448.                         widget.deleteLater()  # 删除控件
449.                 layout.deleteLater()
450.         del_layout(self,self.LDA.layout())
451.         del_layout(self,self.wordcloud.layout())
452.         del_layout(self,self.pinshu.layout())
453.         self.title.clear()
454.         self.label_s.clear()
455.         self.res.clear()
456.
457.         Word,raw,titles,links=self.getWord()
458.         self.Word=Word
459.         self.raw=raw
460.         self.titles=titles
461.         self.links=links
462.         self.plot_pinsu_word()
463.         self.get_s()
464.         #self.get_title()
465.     def endFun(self):
466.         def del_layout(self,layout):
```

```
            if layout is not None:
                for i in reversed(range(layout.count())):
                    widget = layout.itemAt(i).widget()
                    if widget is not None:
                        widget.deleteLater()  # 删除控件
                layout.deleteLater()
        del_layout(self,self.LDA.layout())
        del_layout(self,self.wordcloud.layout())
        del_layout(self,self.pinshu.layout())
        self.label.clear()
        self.title.clear()
        self.label_s.clear()
        self.res.clear()

if __name__=="__main__":
    import sys
    app=QtWidgets.QApplication(sys.argv)
    Server=QtWidgets.QMainWindow()
    ui=Ui_MainWindow()
    ui.setupUi(Server)
    Server.show()
    sys.exit(app.exec())
```