****

**J I A N G S U U N I V E R S I T Y**

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

[1. 设计目标 1](#_Toc22568)

[2. 关键问题 1](#_Toc19895)

[3. 整体构思和设计 1](#_Toc21382)

[4. 模块设计 3](#_Toc14358)

[4.1 GUI设计 3](#_Toc27679)

[4.2 爬取文章模块设计 4](#_Toc8138)

[4.3 自动摘要模块设计 6](#_Toc30374)

[4.3 文本分析和可视化模块设计 7](#_Toc14572)

[4.3.1 分词 8](#_Toc2105)

[4.3.2 词频分析 8](#_Toc22501)

[4.3.3 词云图 9](#_Toc28762)

[4.3.4 LDA主题建模 9](#_Toc4095)

[4.3.5 情感分析 10](#_Toc14258)

[4.4 保存文件模块设计 11](#_Toc10147)

[4.4.1 保存至Sql 11](#_Toc29156)

[4.4.2 保存至本地 12](#_Toc5979)

[5. 运行结果和调试 13](#_Toc25334)

[6. 心得体会 14](#_Toc12818)

[work.py源代码 15](#_Toc18475)

### 

### 基于Python的爬虫项目设计

### 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：

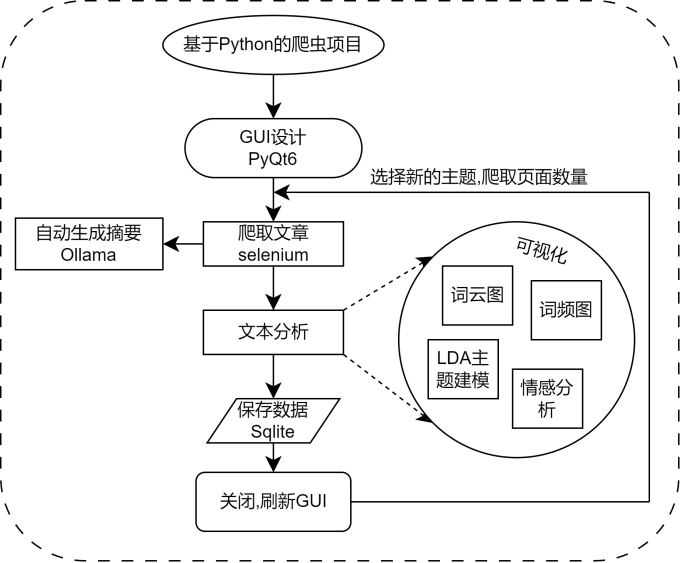


图 3.1 流程图

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：

1. import jieba # 中文分词库
2. import matplotlib.pyplot as plt # 绘图库
3. from collections import Counter # 用于统计词频
4. from wordcloud import WordCloud # 生成词云图
5. import pandas as pd
6. import requests
7. import re
8. from selenium import webdriver # 自动化测试工具，用于模拟浏览器行为
9. from selenium.webdriver.edge.service import Service
10. from selenium.webdriver.edge.options import Options
11. from bs4 import BeautifulSoup as bs # HTML解析库
12. import time
13. from PyQt6 import QtCore, QtGui, QtWidgets # GUI开发库
14. from PyQt6.QtGui import QPixmap
15. from matplotlib.backends.backend\_qtagg import FigureCanvasQTAgg as FigureCanvas
16. from snownlp import SnowNLP # 中文情感分析库
17. import numpy as np
18. import sqlite3
19. from sklearn.decomposition import LatentDirichletAllocation # LDA主题建模
20. from sklearn.feature\_extraction.text import CountVectorizer # 文本向量化
21. from ollama import chat,ChatResponse # 调用本地部署的大模型
22. import socket # 网络通信库
23. import urllib3.util.connection as urllib3\_cn

### 4. 模块设计

#### 4.1 GUI设计

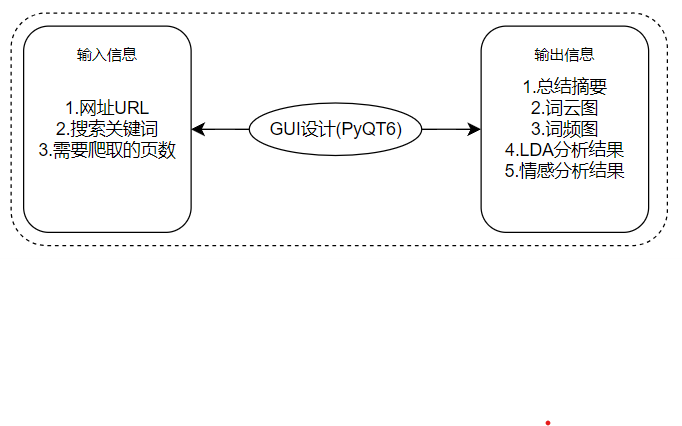


图 1 GUI模块流程图

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：

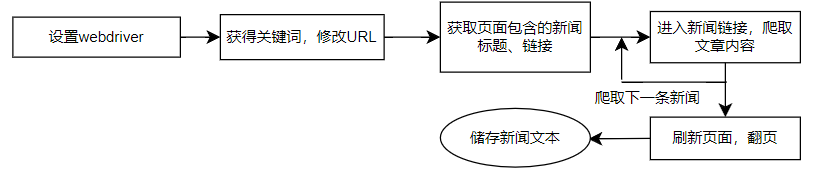


图 2 模块流程

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
2. edge\_options.add\_argument("--headless")
3. edge\_options.add\_argument("--blink-settings=imagesEnabled=false")
4. edge\_options.add\_argument('--ignore-certificate-errors')
5. service=Service(executable\_path=r"C:\Program Files (x86)\Microsoft\Edge\Application\msedgedriver.exe")
6. 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：

1. url=f'https://so.news.cn/*#search/1/{key}/{index}/0'*
2. driver.get(url)
3. driver.refresh() # 刷新页面
4. time.sleep(0.5)
5. html=driver.page\_source # 获取页面HTML源码
6. 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：

1. item\_divs = soup. Select(*'div.items > div.item'*) # 定位新闻条目
2. for item in item\_divs:
3. a\_tag = item.select\_one(*'div.title > a'*)
4. if a\_tag:
5. href = a\_tag[*'href'*]*# 获取新闻链接*
6. text = a\_tag.get\_text(strip=True) # 获取新闻标题
7. if href not in seen\_links and text not in seen\_titles: # 去重
8. seen\_links.add(href)
9. seen\_titles.add(text)
10. items.append(item)
11. 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：

1. with requests.Session() as session:
2. for href, text in link\_text\_dict.items():
3. url=href
4. *#print(url)*
5. response=session. Get(url,allow\_redirects=False,headers=headers, timeout=5)
6. html=response. Text
7. soup=bs(html,'lxml') # 获取新闻正文
8. word=soup.select\_one('#detailContent').get\_text(strip=True)
9. if word=="":
10. continue
11. raw.append(word)
12. word=re.sub(r'（.\*?）|\(.\*?\)', '', word) # 去除括号内容
13. word=re.sub(r'[^\w\s]', '', word) # 去除标点符号
14. Word.append(word) # 存储处理后的新闻内容
15. titles.append(text)
16. 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。

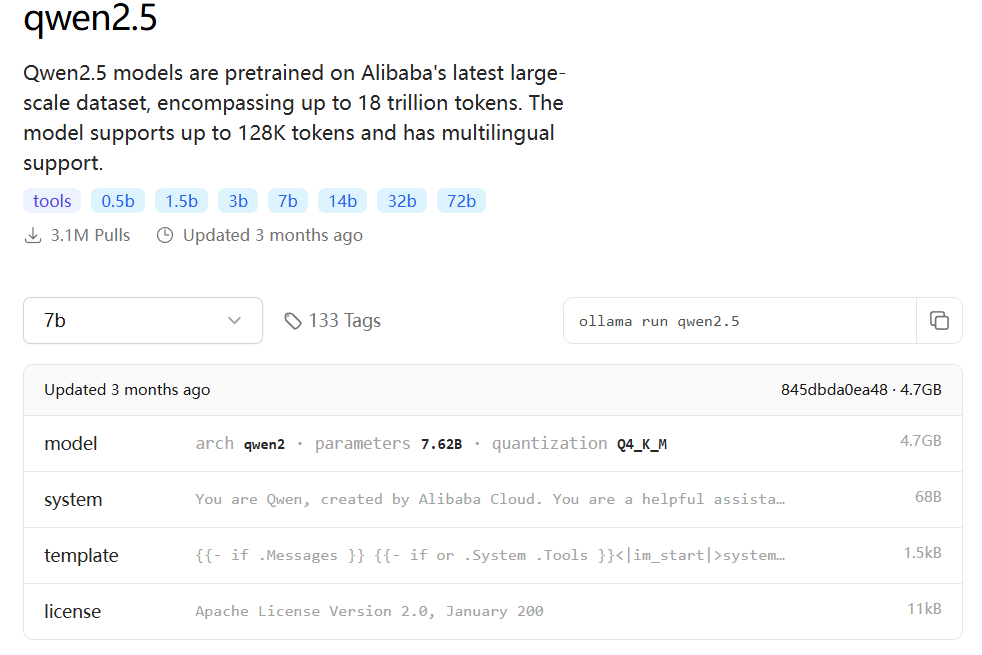


图 3 Ollama官网上的模型界面

在命令行界面输入：ollama run qwen2.5，将模型下载至本地。

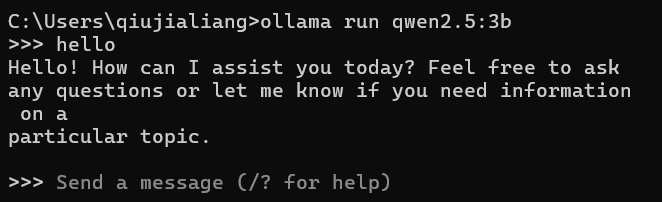


图 4.3.1 模型已部署在本地

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。

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。：

1. response: ChatResponse = chat(model="qwen2.5:3b", messages=[
2. { # 调用大模型生成新闻摘要
3. 'role': 'system',
4. '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.
5. 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.'''，
6. },
7. {
8. 'role': 'user',
9. 'content': word,
10. }
11. ])

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

1. response\_EN: ChatResponse = chat(model="qwen2.5:3b", messages=[
2. { # 调用大模型生成英文新闻摘要
3. 'role': 'system',
4. 'content': '''将给出的摘要翻译为英文.''',
5. },
6. {
7. 'role': 'user',
8. 'content': response['message']['content'],
9. },
10. ])

#### 4.3 文本分析和可视化模块设计

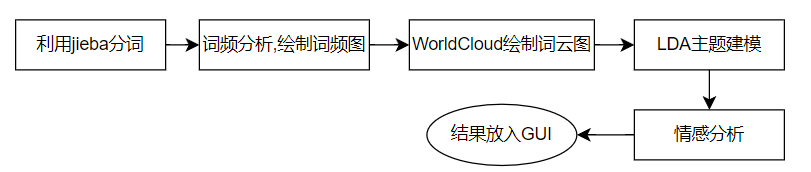


图 5 模块流程

##### 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：

1. words=jieba.cut(text)
2. word\_list=list(words)
3. stopwords = { # 定义停用词列表
4. '的', '是', '在', '和', '了', '有', '与', '对', '为', '也', '说', '根据', '报道',
5. '专家', '称', '表示', '发言人', '记者', '提出', '指出', '现', '时', '将', '当',
6. '来', '到', '关于', '同', '这些', '今年', '月', '日', '去年', '本', '后', '那',
7. '此', '以后', '以后', '前', '之后', '方面', '开始', '结束', '有', '看到', '通过',
8. '认为', '此', '情况', '必须', '进行', '实施', '消息', '表示', '听到', '现已', '已经'
9. ,'新华社记者'
10. }
11. filtered\_words = [word for word in word\_list if word not in stopwords and len(word.strip()) > 1] # 过滤停用词
12. 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：

1. df=pd.DataFrame(word\_freq.items(),columns=["Word","Freq"]).sort\_values(by="Freq",ascending=False) # 绘制词频柱状图
2. fig2,ax2=plt.subplots(figsize=(10,10))
3. plt.rcParams['font.sans-serif'] = ['SimHei'] # 设置中文字体
4. df.head(10).plot(kind='bar',x='Word',y='Freq',legend=False,color='skyblue',ax=ax2)
5. ax2.set\_xlabel('词语', fontsize=10, fontproperties='SimHei')
6. ax2.set\_ylabel('频率', fontsize=10)
7. ax2.set\_xticklabels(df.head(10)['Word'], rotation=45, ha='right', fontsize=8)
8. ax2.tick\_params(axis='x', labelsize=8)
9. layout = QtWidgets.QVBoxLayout(self.pinshu) # 将词频图嵌入到GUI界面中
10. canvas = FigureCanvas(fig2)
11. self.fig\_pinshu=fig2
12. layout.addWidget(canvas)
13. self.pinshu.setLayout(layout)
14. 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：

1. wordcloud=WordCloud(font\_path='C:/Windows/Fonts/simfang.ttf',background\_color='white',width=800,height=600).generate\_from\_frequencies(word\_freq)
2. fig=plt.figure(figsize=(10,10))
3. plt.imshow(wordcloud,interpolation='bilinear')
4. plt.axis('off')
5. plt.tight\_layout()
6. canvas = FigureCanvas(fig) # 将词云图嵌入到GUI界面中
7. self.fig\_wordcloud=fig
8. layout = QtWidgets.QVBoxLayout(self.wordcloud)
9. layout.addWidget(canvas)
10. self.wordcloud.setLayout(layout)
11. 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LatentDirichletAllocation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。

1. vectorizer = CountVectorizer(stop\_words=list(stopwords)) # 文本向量化
2. X = vectorizer.fit\_transform(word\_list) # 将文本转换为词频矩阵
3. lda=LatentDirichletAllocation(n\_components=3,random\_state=42) # 初始化LDA模型
4. lda.fit(X) # 训练LDA模型
5. colormaps = ['Blues', 'Reds', 'Greens'] # 绘制LDA主题词云图
6. def plot\_wordcloud\_for\_topic(topic\_idx, feature\_names, topic\_weights,ax, colormap):
7. top\_n = 5
8. top\_indices = topic\_weights.argsort()[-top\_n:][::-1]
9. word\_freq = {feature\_names[i]: topic\_weights[i] for i in top\_indices}
10. wordcloud = WordCloud(font\_path='C:/Windows/Fonts/simfang.ttf',width=800, height=400, background\_color="white",colormap=colormap).generate\_from\_frequencies(word\_freq)
11. ax.imshow(wordcloud, interpolation="bilinear")
12. ax.axis("off")
13. fig3, axes = plt.subplots(1, 3, figsize=(15, 5)) # 将LDA主题图嵌入到GUI界面中
14. for idx,(topic, ax, cmap) in enumerate(zip(lda.components\_, axes, colormaps)):
15. print(idx)
16. print([vectorizer.get\_feature\_names\_out()[i] for i in topic.argsort()[-5:]])
17. plot\_wordcloud\_for\_topic(idx, vectorizer.get\_feature\_names\_out(), topic, ax, cmap)
18. plt.tight\_layout()
19. canvas=FigureCanvas(fig3)
20. self.fig\_LDA=fig3
21. layout = QtWidgets.QVBoxLayout(self.LDA)
22. layout.addWidget(canvas)
23. self.LDA.setLayout(layout)
24. 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：

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

#### 4.4 保存文件模块设计

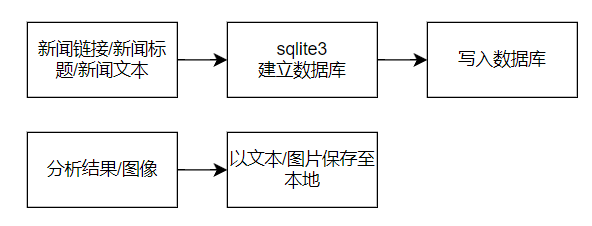


图 6 模块流程

##### 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：

1. conn=sqlite3.connect('keshe/word.db') # 将数据保存到SQLite数据库
2. cursor=conn.cursor()
3. cursor.execute(
4. '''
5. CREATE TABLE IF NOT EXISTS texts (
6. id INTEGER PRIMARY KEY,
7. text\_content TEXT,
8. type TEXT
9. )
10. '''
11. )
12. cursor.execute(f'DELETE FROM texts') # 清空表
13. conn.commit()
14. for word in Word: # 插入分词结果、原始新闻内容、标题和链接
15. cursor.execute('INSERT INTO texts (text\_content, type) VALUES (?, ?)',(word,'Word'))
16. for raw in raws:
17. cursor.execute('INSERT INTO texts (text\_content, type) VALUES (?, ?)',(raw,'Raws'))
18. for title in titles:
19. cursor.execute('INSERT INTO texts (text\_content, type) VALUES (?, ?)',(title,'titles'))
20. for link in links:
21. cursor.execute('INSERT INTO texts (text\_content, type) VALUES (?, ?)',(link,'links'))
22. conn.commit()
23. cursor.close()
24. 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：

1. with open("keshe/save\_Word.txt",'w',encoding='utf-8') as f:
2. for word in Word: # 将分词结果保存到本地文件
3. f.write(word+'\n')
4. print("分词写入完毕")
5. with open("keshe/save\_raw.txt",'w',encoding='utf-8') as f:
6. for raw in raws: # 将原始新闻内容保存到本地文件
7. f.write(raw+'\n')
8. print("原文本写入完毕")
9. self.fig\_pinshu.savefig('keshe/pinshu.jpg') # 保存词频图、词云图和LDA主题图
10. self.fig\_wordcloud.savefig('keshe/wordcloud.jpg')
11. self.fig\_LDA.savefig('keshe/LDA.jpg')
12. print("图像写入完毕")

### 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：

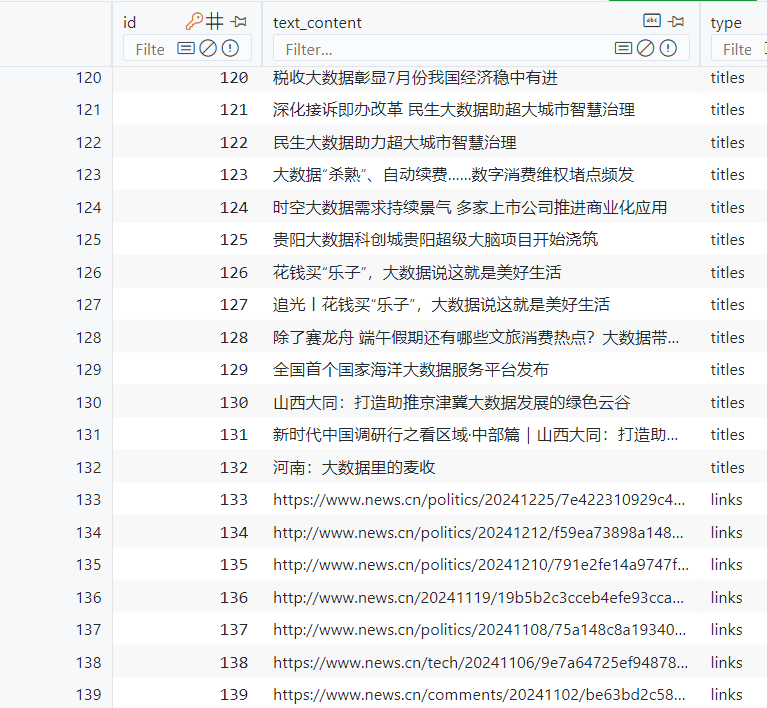


图 5.2 数据库中的文本

图片、文本也以图像和文本文件形式保存在了本地：

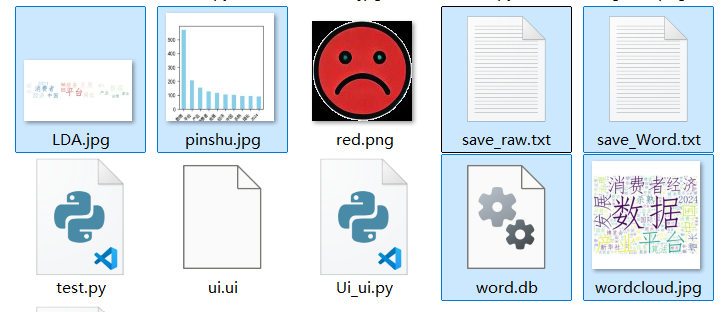


图 5.3 本地保存的文件

### 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

1. import jieba
2. import matplotlib.pyplot as plt
3. from collections import Counter
4. from wordcloud import WordCloud
5. import pandas as pd
6. import requests
7. import re
8. from selenium import webdriver
9. from selenium.webdriver.edge.service import Service
10. from selenium.webdriver.edge.options import Options
11. from bs4 import BeautifulSoup as bs
12. import time
13. from PyQt6 import QtCore, QtGui, QtWidgets
14. from PyQt6.QtGui import QPixmap
15. from matplotlib.backends.backend\_qtagg import FigureCanvasQTAgg as FigureCanvas
16. from snownlp import SnowNLP
17. import numpy as np
18. import sqlite3
19. from sklearn.decomposition import LatentDirichletAllocation
20. from sklearn.feature\_extraction.text import CountVectorizer
21. from ollama import chat,ChatResponse
22. import socket
23. import urllib3.util.connection as urllib3\_cn
24. def allowed\_gai\_family():
25. return socket.AF\_INET
26. urllib3\_cn.allowed\_gai\_family = allowed\_gai\_family
27. class Ui\_MainWindow(object):
28. def setupUi(self, MainWindow):
29. MainWindow.setObjectName("MainWindow")
30. MainWindow.resize(1366, 803)
31. self.centralwidget = QtWidgets.QWidget(parent=MainWindow)
32. self.centralwidget.setObjectName("centralwidget")
33. self.groupBox = QtWidgets.QGroupBox(parent=self.centralwidget)
34. self.groupBox.setGeometry(QtCore.QRect(20, 20, 1321, 751))
35. font = QtGui.QFont()
36. font.setFamily("Arial")
37. font.setPointSize(20)
38. font.setBold(True)
39. font.setItalic(False)
40. font.setUnderline(False)
41. font.setWeight(75)
42. self.groupBox.setFont(font)
43. self.groupBox.setCheckable(False)
44. self.groupBox.setObjectName("groupBox")
45. self.label\_7 = QtWidgets.QLabel(parent=self.groupBox)
46. self.label\_7.setGeometry(QtCore.QRect(20, 40, 71, 21))
47. font = QtGui.QFont()
48. font.setPointSize(12)
49. self.label\_7.setFont(font)
50. self.label\_7.setObjectName("label\_7")
51. self.label\_2 = QtWidgets.QLabel(parent=self.groupBox)
52. self.label\_2.setGeometry(QtCore.QRect(10, 350, 91, 31))
53. font = QtGui.QFont()
54. font.setPointSize(12)
55. self.label\_2.setFont(font)
56. self.label\_2.setObjectName("label\_2")
57. self.pinshu = QtWidgets.QWidget(parent=self.groupBox)
58. self.pinshu.setGeometry(QtCore.QRect(90, 340, 300, 300))
59. self.pinshu.setObjectName("pinshu")
60. self.wordcloud = QtWidgets.QWidget(parent=self.groupBox)
61. self.wordcloud.setGeometry(QtCore.QRect(90, 20, 300, 300))
62. self.wordcloud.setObjectName("wordcloud")
63. self.label\_6 = QtWidgets.QLabel(parent=self.groupBox)
64. self.label\_6.setGeometry(QtCore.QRect(400, 40, 121, 31))
65. font = QtGui.QFont()
66. font.setPointSize(12)
67. self.label\_6.setFont(font)
68. self.label\_6.setObjectName("label\_6")
69. self.label\_8 = QtWidgets.QLabel(parent=self.groupBox)
70. self.label\_8.setGeometry(QtCore.QRect(410, 380, 91, 31))
71. font = QtGui.QFont()
72. font.setPointSize(12)
73. self.label\_8.setFont(font)
74. self.label\_8.setObjectName("label\_8")
75. self.LDA = QtWidgets.QWidget(parent=self.groupBox)
76. self.LDA.setGeometry(QtCore.QRect(400, 420, 521, 300))
77. self.LDA.setObjectName("LDA")
78. self.label\_s = QtWidgets.QLabel(parent=self.groupBox)
79. self.label\_s.setGeometry(QtCore.QRect(1060, 400, 151, 31))
80. font = QtGui.QFont()
81. font.setPointSize(12)
82. self.label\_s.setFont(font)
83. self.label\_s.setText("")
84. self.label\_s.setObjectName("label\_s")
85. self.label\_5 = QtWidgets.QLabel(parent=self.groupBox)
86. self.label\_5.setGeometry(QtCore.QRect(960, 400, 91, 31))
87. font = QtGui.QFont()
88. font.setPointSize(12)
89. self.label\_5.setFont(font)
90. self.label\_5.setObjectName("label\_5")
91. self.groupBox\_2 = QtWidgets.QGroupBox(parent=self.groupBox)
92. self.groupBox\_2.setGeometry(QtCore.QRect(950, 450, 341, 261))
93. font = QtGui.QFont()
94. font.setPointSize(20)
95. self.groupBox\_2.setFont(font)
96. self.groupBox\_2.setObjectName("groupBox\_2")
97. self.label\_3 = QtWidgets.QLabel(parent=self.groupBox\_2)
98. self.label\_3.setGeometry(QtCore.QRect(10, 40, 61, 31))
99. font = QtGui.QFont()
100. font.setPointSize(12)
101. self.label\_3.setFont(font)
102. self.label\_3.setObjectName("label\_3")
103. self.netname = QtWidgets.QLineEdit(parent=self.groupBox\_2)
104. self.netname.setGeometry(QtCore.QRect(90, 40, 241, 41))
105. font = QtGui.QFont()
106. font.setPointSize(12)
107. self.netname.setFont(font)
108. self.netname.setText("")
109. self.netname.setObjectName("netname")
110. self.key = QtWidgets.QLineEdit(parent=self.groupBox\_2)
111. self.key.setGeometry(QtCore.QRect(90, 90, 241, 41))
112. font = QtGui.QFont()
113. font.setPointSize(12)
114. self.key.setFont(font)
115. self.key.setObjectName("key")
116. self.label\_4 = QtWidgets.QLabel(parent=self.groupBox\_2)
117. self.label\_4.setGeometry(QtCore.QRect(0, 90, 71, 31))
118. font = QtGui.QFont()
119. font.setPointSize(12)
120. self.label\_4.setFont(font)
121. self.label\_4.setObjectName("label\_4")
122. self.start = QtWidgets.QPushButton(parent=self.groupBox\_2)
123. self.start.setGeometry(QtCore.QRect(0, 210, 93, 28))
124. font = QtGui.QFont()
125. font.setPointSize(12)
126. self.start.setFont(font)
127. self.start.setObjectName("start")
128. self.save = QtWidgets.QPushButton(parent=self.groupBox\_2)
129. self.save.setGeometry(QtCore.QRect(110, 210, 93, 28))
130. font = QtGui.QFont()
131. font.setPointSize(12)
132. self.save.setFont(font)
133. self.save.setObjectName("save")
134. self.close = QtWidgets.QPushButton(parent=self.groupBox\_2)
135. self.close.setGeometry(QtCore.QRect(220, 210, 93, 28))
136. font = QtGui.QFont()
137. font.setPointSize(12)
138. self.close.setFont(font)
139. self.close.setObjectName("close")
140. self.label\_9 = QtWidgets.QLabel(parent=self.groupBox\_2)
141. self.label\_9.setGeometry(QtCore.QRect(10, 140, 71, 31))
142. font = QtGui.QFont()
143. font.setPointSize(12)
144. self.label\_9.setFont(font)
145. self.label\_9.setObjectName("label\_9")
146. self.page = QtWidgets.QLineEdit(parent=self.groupBox\_2)
147. self.page.setGeometry(QtCore.QRect(90, 140, 241, 41))
148. font = QtGui.QFont()
149. font.setPointSize(12)
150. self.page.setFont(font)
151. self.page.setObjectName("page")
152. self.scrollArea = QtWidgets.QScrollArea(parent=self.groupBox)
153. self.scrollArea.setGeometry(QtCore.QRect(420, 80, 851, 291))
154. self.scrollArea.setWidgetResizable(True)
155. self.scrollArea.setObjectName("scrollArea")
156. self.scrollAreaWidgetContents = QtWidgets.QWidget()
157. self.scrollAreaWidgetContents.setGeometry(QtCore.QRect(0, 0, 849, 289))
158. self.scrollAreaWidgetContents.setObjectName("scrollAreaWidgetContents")
159. self.title = QtWidgets.QLabel(parent=self.scrollAreaWidgetContents)
160. self.title.setGeometry(QtCore.QRect(0, 0, 821, 281))
161. font = QtGui.QFont()
162. font.setPointSize(12)
163. self.title.setFont(font)
164. self.title.setText("")
165. self.title.setAlignment(QtCore.Qt.AlignmentFlag.AlignLeading|QtCore.Qt.AlignmentFlag.AlignLeft|QtCore.Qt.AlignmentFlag.AlignTop)
166. self.title.setWordWrap(True)
167. self.title.setObjectName("title")
168. self.scrollArea.setWidget(self.title)
169. self.res = QtWidgets.QLabel(parent=self.groupBox)
170. self.res.setGeometry(QtCore.QRect(1080, 40, 191, 31))
171. font = QtGui.QFont()
172. font.setPointSize(12)
173. self.res.setFont(font)
174. self.res.setText("")
175. self.res.setObjectName("res")
176. self.label = QtWidgets.QLabel(parent=self.groupBox)
177. self.label.setGeometry(QtCore.QRect(1240, 390, 60, 60))
178. font = QtGui.QFont()
179. font.setPointSize(12)
180. self.label.setFont(font)
181. self.label.setText("")
182. self.label.setObjectName("label")
183. MainWindow.setCentralWidget(self.centralwidget)
184. self.menubar = QtWidgets.QMenuBar(parent=MainWindow)
185. self.menubar.setGeometry(QtCore.QRect(0, 0, 1366, 26))
186. self.menubar.setObjectName("menubar")
187. MainWindow.setMenuBar(self.menubar)
188. self.statusbar = QtWidgets.QStatusBar(parent=MainWindow)
189. self.statusbar.setObjectName("statusbar")
190. MainWindow.setStatusBar(self.statusbar)
191. self.retranslateUi(MainWindow)
192. QtCore.QMetaObject.connectSlotsByName(MainWindow)
193. def retranslateUi(self, MainWindow):
194. \_translate = QtCore.QCoreApplication.translate
195. MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))
196. self.groupBox.setTitle(\_translate("MainWindow", "分析"))
197. self.label\_7.setText(\_translate("MainWindow", "词云图"))
198. self.label\_2.setText(\_translate("MainWindow", "词频图"))
199. self.label\_6.setText(\_translate("MainWindow", "题目及摘要:"))
200. self.label\_8.setText(\_translate("MainWindow", "LDA"))
201. self.label\_5.setText(\_translate("MainWindow", "情感分析:"))
202. self.groupBox\_2.setTitle(\_translate("MainWindow", "输入信息"))
203. self.label\_3.setText(\_translate("MainWindow", "网址:"))
204. self.label\_4.setText(\_translate("MainWindow", "关键词:"))
205. self.start.setText(\_translate("MainWindow", "开始"))
206. self.save.setText(\_translate("MainWindow", "保存"))
207. self.close.setText(\_translate("MainWindow", "关闭"))
208. self.label\_9.setText(\_translate("MainWindow", "页数:"))
209. self.netname.setText("https://so.news.cn/")
210. self.netname.textChanged.connect(self.showText)
211. self.key.textChanged.connect(self.showText)
212. self.start.clicked.connect(self.startFun)
213. self.close.clicked.connect(self.endFun)
214. self.save.clicked.connect(self.saveFun)
215. def showText(self):
216. self.netname.setText(self.netname.text())
217. self.key.setText(self.key.text())
218. *#print(self.netname.text())*
220. def getWord(self):
221. key=self.key.text()
222. page=self.page.text()
223. if page=="":
224. page=2
225. if key=="":
226. key='人工智能'
227. print(key)
228. print(page)
229. edge\_options=Options()
230. edge\_options.add\_argument("--headless")
231. edge\_options.add\_argument("--blink-settings=imagesEnabled=false")
232. edge\_options.add\_argument('--ignore-certificate-errors')
233. service=Service(executable\_path=r"C:\Program Files (x86)\Microsoft\Edge\Application\msedgedriver.exe")
234. driver=webdriver.Edge(service=service,options=edge\_options)
235. seen\_links = set()
236. seen\_titles=set()
237. items = []
238. link\_text\_dict = {}
239. for index in range(1,int(page)+1):
240. url=f'https://so.news.cn/#search/1/{key}/{index}/0'
241. driver.get(url)
242. driver.refresh()
243. time.sleep(0.5)
244. html=driver.page\_source
245. soup=bs(html,'lxml')
246. item\_divs = soup.select('div.items > div.item')
247. for item in item\_divs:
248. a\_tag = item.select\_one('div.title > a')
249. if a\_tag:
250. href = a\_tag['href']
251. text = a\_tag.get\_text(strip=True)
252. if href not in seen\_links and text not in seen\_titles:
253. seen\_links.add(href)
254. seen\_titles.add(text)
255. items.append(item)
256. link\_text\_dict[href] = text
257. driver.quit()
258. Word=[]
259. raw=[]
260. titles=[]
261. links=[]
262. i=1
263. headers={
264. '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'
265. }
266. with requests.Session() as session:
267. for href, text in link\_text\_dict.items():
268. url=href
269. *#print(url)*
270. response=session.get(url,allow\_redirects=False,headers=headers,timeout=5)
271. html=response.text
272. soup=bs(html,'lxml')
273. word=soup.select\_one('#detailContent').get\_text(strip=True)
274. if word=="":
275. continue
276. raw.append(word)
277. word=re.sub(r'（.\*?）|\(.\*?\)', '', word)
278. word=re.sub(r'[^\w\s]', '', word)
279. Word.append(word)
280. titles.append(text)
281. links.append(href)
282. response: ChatResponse = chat(model="qwen2.5:3b", messages=[
283. {
284. 'role': 'system',
285. '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.
286. 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.''',
287. },
288. {
289. 'role': 'user',
290. 'content': word,
291. },
292. ])
293. *#print(response.message.content)*
294. self.title.setText(self.title.text()+'\n'+f"{i}.{text}:{href}"+'\n'+f"摘要:{response['message']['content']}")
295. i+=1
296. self.res.setStyleSheet("QLabel { color: red; }")
297. self.res.setText(f"获取新闻{i-1}篇")
298. *#print(Word)*
299. return Word,raw,titles,links
300. def plot\_pinsu\_word(self):
301. Word=self.Word
302. raw=self.raw
303. text=""
304. for word in Word:
305. text=text+word
306. words=jieba.cut(text)
307. word\_list=list(words)
308. stopwords = {
309. '的', '是', '在', '和', '了', '有', '与', '对', '为', '也', '说', '根据', '报道',
310. '专家', '称', '表示', '发言人', '记者', '提出', '指出', '现', '时', '将', '当',
311. '来', '到', '关于', '同', '这些', '今年', '月', '日', '去年', '本', '后', '那',
312. '此', '以后', '以后', '前', '之后', '方面', '开始', '结束', '有', '看到', '通过',
313. '认为', '此', '情况', '必须', '进行', '实施', '消息', '表示', '听到', '现已', '已经'
314. ,'新华社记者'
315. }
316. filtered\_words = [word for word in word\_list if word not in stopwords and len(word.strip()) > 1]
317. word\_list=list(filtered\_words)
318. self.res.setText(self.res.text()+f"总词数{len(word\_list)}个")
319. for stopword in stopwords:
320. for word in raw:
321. word = word.replace(stopword, '')
322. word\_freq=Counter(filtered\_words)
323. df=pd.DataFrame(word\_freq.items(),columns=["Word","Freq"]).sort\_values(by="Freq",ascending=False)
324. fig2,ax2=plt.subplots(figsize=(10,10))
325. plt.rcParams['font.sans-serif'] = ['SimHei']
326. df.head(10).plot(kind='bar',x='Word',y='Freq',legend=False,color='skyblue',ax=ax2)
327. ax2.set\_xlabel('词语', fontsize=10, fontproperties='SimHei')
328. ax2.set\_ylabel('频率', fontsize=10)
329. ax2.set\_xticklabels(df.head(10)['Word'], rotation=45, ha='right', fontsize=8)
330. ax2.tick\_params(axis='x', labelsize=8)
331. layout = QtWidgets.QVBoxLayout(self.pinshu)
332. canvas = FigureCanvas(fig2)
333. self.fig\_pinshu=fig2
334. layout.addWidget(canvas)
335. self.pinshu.setLayout(layout)
336. canvas.draw()
338. wordcloud=WordCloud(font\_path='C:/Windows/Fonts/simfang.ttf',background\_color='white',width=800,height=600).generate\_from\_frequencies(word\_freq)
339. fig=plt.figure(figsize=(10,10))
340. plt.imshow(wordcloud,interpolation='bilinear')
341. plt.axis('off')
342. plt.tight\_layout()
343. canvas = FigureCanvas(fig)
344. self.fig\_wordcloud=fig
345. layout = QtWidgets.QVBoxLayout(self.wordcloud)
346. layout.addWidget(canvas)
347. self.wordcloud.setLayout(layout)
348. canvas.draw()
350. vectorizer = CountVectorizer(stop\_words=list(stopwords))
351. X = vectorizer.fit\_transform(word\_list)
352. lda=LatentDirichletAllocation(n\_components=3,random\_state=42)
353. lda.fit(X)
354. colormaps = ['Blues', 'Reds', 'Greens']
355. def plot\_wordcloud\_for\_topic(topic\_idx, feature\_names, topic\_weights,ax, colormap):
356. top\_n = 5
357. top\_indices = topic\_weights.argsort()[-top\_n:][::-1]
358. word\_freq = {feature\_names[i]: topic\_weights[i] for i in top\_indices}
360. wordcloud = WordCloud(font\_path='C:/Windows/Fonts/simfang.ttf',width=800, height=400
361. , background\_color="white",colormap=colormap).generate\_from\_frequencies(word\_freq)
362. ax.imshow(wordcloud, interpolation="bilinear")
363. ax.axis("off")
364. fig3, axes = plt.subplots(1, 3, figsize=(15, 5))
365. for idx,(topic, ax, cmap) in enumerate(zip(lda.components\_, axes, colormaps)):
366. print(idx)
367. print([vectorizer.get\_feature\_names\_out()[i] for i in topic.argsort()[-5:]])
368. plot\_wordcloud\_for\_topic(idx, vectorizer.get\_feature\_names\_out(), topic, ax, cmap)
369. plt.tight\_layout()
370. canvas=FigureCanvas(fig3)
371. self.fig\_LDA=fig3
372. layout = QtWidgets.QVBoxLayout(self.LDA)
373. layout.addWidget(canvas)
374. self.LDA.setLayout(layout)
375. canvas.draw()
377. def get\_s(self):
378. raws=self.raw
379. res=[]
380. for raw in raws:
381. s=SnowNLP(raw)
382. res.append(s.sentiments)
383. ave=np.mean(res)
384. print(ave)
385. if ave>0.5:
386. self.label\_s.setStyleSheet("QLabel { color: green; }")
387. self.label\_s.setText(f"{ave:.5f} 正面")
388. pixmap=QPixmap(r"keshe\green.png")
389. self.label.setPixmap(pixmap)
390. self.label.setScaledContents(True)
391. else:
392. self.label\_s.setStyleSheet("QLabel { color: red; }")
393. self.label\_s.setText(f"{ave:.5f} 负面")
394. pixmap=QPixmap(r"keshe\red.png")
395. self.label.setPixmap(pixmap)
396. self.label.setScaledContents(True)
397. def saveFun(self):
398. Word=self.Word
399. raws=self.raw
400. titles=self.titles
401. links=self.links
402. with open("keshe/save\_Word.txt",'w',encoding='utf-8') as f:
403. for word in Word:
404. f.write(word+'\n')
405. print("分词写入完毕")
406. with open("keshe/save\_raw.txt",'w',encoding='utf-8') as f:
407. for raw in raws:
408. f.write(raw+'\n')
409. print("原文本写入完毕")
410. self.fig\_pinshu.savefig('keshe/pinshu.jpg')
411. self.fig\_wordcloud.savefig('keshe/wordcloud.jpg')
412. self.fig\_LDA.savefig('keshe/LDA.jpg')
413. print("图像写入完毕")
414. conn=sqlite3.connect('keshe/word.db')
415. cursor=conn.cursor()
416. cursor.execute(
417. '''
418. CREATE TABLE IF NOT EXISTS texts (
419. id INTEGER PRIMARY KEY,
420. text\_content TEXT,
421. type TEXT
422. )
423. '''
424. )
425. cursor.execute(f'DELETE FROM texts')
426. conn.commit()
427. for word in Word:
428. cursor.execute('INSERT INTO texts (text\_content, type) VALUES (?, ?)',(word,'Word'))
429. for raw in raws:
430. cursor.execute('INSERT INTO texts (text\_content, type) VALUES (?, ?)',(raw,'Raws'))
431. for title in titles:
432. cursor.execute('INSERT INTO texts (text\_content, type) VALUES (?, ?)',(title,'titles'))
433. for link in links:
434. cursor.execute('INSERT INTO texts (text\_content, type) VALUES (?, ?)',(link,'links'))
435. conn.commit()
436. cursor.close()
437. conn.close()
438. def startFun(self):
439. *#key=input()*
440. def del\_layout(self,layout):
441. if layout is not None:
442. for i in reversed(range(layout.count())):
443. widget = layout.itemAt(i).widget()
444. if widget is not None:
445. widget.deleteLater()  *# 删除控件*
446. layout.deleteLater()
447. del\_layout(self,self.LDA.layout())
448. del\_layout(self,self.wordcloud.layout())
449. del\_layout(self,self.pinshu.layout())
450. self.title.clear()
451. self.label\_s.clear()
452. self.res.clear()
454. Word,raw,titles,links=self.getWord()
455. self.Word=Word
456. self.raw=raw
457. self.titles=titles
458. self.links=links
459. self.plot\_pinsu\_word()
460. self.get\_s()
461. *#self.get\_title()*
462. def endFun(self):
463. def del\_layout(self,layout):
464. if layout is not None:
465. for i in reversed(range(layout.count())):
466. widget = layout.itemAt(i).widget()
467. if widget is not None:
468. widget.deleteLater()  *# 删除控件*
469. layout.deleteLater()
470. del\_layout(self,self.LDA.layout())
471. del\_layout(self,self.wordcloud.layout())
472. del\_layout(self,self.pinshu.layout())
473. self.label.clear()
474. self.title.clear()
475. self.label\_s.clear()
476. self.res.clear()
478. if \_\_name\_\_=="\_\_main\_\_":
479. import sys
480. app=QtWidgets.QApplication(sys.argv)
481. Server=QtWidgets.QMainWindow()
482. ui=Ui\_MainWindow()
483. ui.setupUi(Server)
484. Server.show()
485. sys.exit(app.exec())