Sentiment Analysing Web Application Project

Data Investigation Report

github.com/Macsok/Sentiment-Analysis-Application

Konrad Bachór, Kuba Herka, Maciej Sokołowski

December 2024

Contents

1 Introduction

This project is a comprehensive web application designed to analyze sentiment from various online sources. It leverages advanced natural language processing (NLP) techniques to determine the sentiment of text data, categorizing it as positive, negative, or neutral. The application can scrape data from multiple platforms, including Amazon for product reviews, Twitter (X) for tweets, and YouTube for video comments, using provided credentials.

2 Project Structure

The project is organized into several directories and files, each serving a specific purpose. Below is an overview of the project structure:

```
.gitignore
credentials/
    AMAZON
    X
    YT_API_KEY
documentation.txt
LICENSE
myenv/
    Include/
        site/
    Lib/
        site-packages/
    pyvenv.cfg
    Scripts/
        activate
        activate.bat
        Activate.ps1
        chardetect.exe
        deactivate.bat
        f2py.exe
        flask.exe
README.md
requirements.txt
run.py
scripts/
    __pycache__/
    analyses.py
    scraperAmazon.py
    scraperX.py
    scraperYT.py
    sseStream.py
testing/
    scrapped_examples/
    script_testing.ipynb
    yielding.ipynb
web/
    __pycache__/
    comm/
    main.py
    static/
    templates/
```

3 Prerequisites

To run this project, you need the following software and libraries installed on your system:

- Python 3.10.0 or higher
- Virtualenv
- Playwright
- YouTube Data API Key

4 Installation

4.1 Clone the Repository

First, clone the repository from GitHub and navigate to the project directory:

```
git clone https://github.com/Macsok/Sentiment-Analysis-Application
cd Sentiment-Analysis-Application
```

4.2 Using Virtualenv (on Windows)

Use virtualenv to isolate project dependencies, ensuring no conflicts between different projects.

First, install Virtualenv if you haven't already:

```
pip install virtualenv
```

Navigate to the main project directory and create a new virtual environment:

```
python -m virtualenv myvenv
```

Activate the Virtual Environment:

```
python -m source myenv/Scripts/activate
```

When activated, your shell will show the virtual environment's name in the prompt.

Now that the environment is active, you can install packages:

```
pip install -r requirements.txt
```

Once you have finished working on your project, it's a good habit to deactivate its venv. By deactivating, you leave the virtual environment.

```
deactivate
```

Delete the Virtual Environment (if your virtual environment is in a directory called 'venv'):

```
rm -r myvenv
```

5 Browser Installation

To use Playwright for web scraping, you need to install a browser. The following command installs Firefox:

```
python -m playwright install firefox
```

Alternatively, you can use npm to install the browser:

```
npx playwright install firefox
```

6 Adding Credentials

To use online scrapers, you need to provide credentials to your platforms. Create a new directory in the main project folder (Sentiment-Analysis-Application) called 'credentials'. Then add 3 files to it: AMAZON, X, YT$_APIK_EY.Eachfileshouldconsistof$:

6.1 AMAZON

- email
- password

6.2 X

- username
- password
- email

6.3 YT_API_KEY

- API key (more at: https://developers.google.com/youtube/v3)

7 Running the Application

To run the application, execute the following command:

```
python run.py
```

8 Amazon Review Scraper Documentation

8.1 Prerequisites

- Python 3.10.0 or higher
- Playwright

8.2 How It Works

8.2.1 Amazon Login

The scraper logs into Amazon using the provided credentials.

8.2.2 Review Extraction

The scraper extracts reviews from the specified Amazon product page.

8.2.3 Data Cleaning

The extracted data is cleaned to remove any unnecessary information.

8.2.4 CSV Export

The cleaned data is exported to a CSV file.

8.3 Code Structure

The Amazon scraper is implemented in the `scraperAmazon.py` file. The main functions are:

- `get_reviews(url: str) -> None`
- `login_to_amazon(page, username: str, password: str) -> None`
- `extract_data(amazon_reviews_ratings: list, page) -> list`
- `run(playwright, url: str) -> None`
- `clean_data(data: str) -> str | None`
- `save_data_to_csv(reviews_data: list, filename: str) -> None`

8.4 Summary of Code Flow

The scraper logs into Amazon, extracts reviews, cleans the data, and exports it to a CSV file.

8.5 Example Usage

```
from scraperAmazon import get_reviews

url = "https://www.amazon.com/product-reviews/B08N5WRWNW"
get_reviews(url)
```

8.6 Important Notes

Make sure to provide valid Amazon credentials in the `credentials/AMAZON` file.

9 X.com Reply Scraper Documentation

9.1 Prerequisites

- Python 3.10.0 or higher
- Playwright

9.2 How It Works

9.2.1 Login to X.com

The scraper logs into X.com using the provided credentials.

9.2.2 Reply Extraction

The scraper extracts replies from the specified X.com post.

9.2.3 Data Cleaning

The extracted data is cleaned to remove any unnecessary information.

9.3 Code Structure

The X.com scraper is implemented in the `scraperX.py` file. The main functions are:

- `login_to_x(page, username: str, password: str) -> None`
- `extract_replies(x_replies_ratings: list, page) -> list`
- `run(playwright, url: str) -> None`
- `clean_data(data: str) -> str | None`
- `save_data_to_csv(replies_data: list, filename: str) -> None`

9.4 Summary of Code Flow

The scraper logs into X.com, extracts replies, cleans the data, and exports it to a CSV file.

9.5 Example Usage

```
from scraperX import get_replies

url = "https://www.x.com/post/1234567890"
get_replies(url)
```

9.6 Important Notes

Make sure to provide valid X.com credentials in the `credentials/X` file.

10 YouTube Comment Scraper Documentation

10.1 Prerequisites

- Python 3.10.0 or higher
- YouTube Data API Key

10.2 How It Works

10.2.1 API Key

The scraper uses the YouTube Data API key to access video comments.

10.2.2 Comment Extraction

The scraper extracts comments from the specified YouTube video.

10.2.3 Data Cleaning

The extracted data is cleaned to remove any unnecessary information.

10.3 Code Structure

The YouTube scraper is implemented in the `scraperYT.py` file. The main functions are:

- `get_comments(video_id: str) -> None`
- `clean_data(data: str) -> str | None`
- `save_data_to_csv(comments_data: list, filename: str) -> None`

10.4 Summary of Code Flow

The scraper uses the YouTube Data API key to access video comments, cleans the data, and exports it to a CSV file.

10.5 Example Usage

```
from scraperYT import get_comments

video_id = "dQw4w9WgXcQ"
get_comments(video_id)
```

10.6 Important Notes

Make sure to provide a valid YouTube Data API key in the `credentials/YT_API_KEY` file.

11 Web Interface

The web interface is built with Flask and provides an intuitive user experience for interacting with the sentiment analysis features.

11.1 Routes

The main routes in the web interface are:

- `/` - Home page
- `/about` - About page
- `/pepe` - Pepe page
- `/singlereview` - Single review analysis
- `/amazon_review` - Amazon review scraping
- `/yt_review` - YouTube review scraping
- `/start_scraping` - Start the scraping process

11.2 Example Route Implementation

The following is an example implementation of the `/singlereview` route:

```
@app.route("/singlereview", methods=["GET", "POST"])
def singlereview():
    """Handle single review analysis."""
    if request.method == "POST":
        start = time.time()
        text = request.form["textinput"]
        print(text)
        scores = analyses.default_analysis(text)
        sentiment = analyses.get_sentiment(scores[3])
        end = time.time()
        adbreak = 15
        wait_time = max(0, (adbreak - (end - start)) * 1000)
        return render_template(
            "singlereview.html",
            textinput=text,
            sentiment=sentiment,
            wait_time=wait_time
        )
    else:
        return render_template("singlereview.html")
```

12 Testing

The `testing/` directory contains various testing-related files, including:

- `scrapped_examples/` - Examples of scrapped data
- `script_testing.ipynb` - Jupyter notebook for testing scripts
- `yielding.ipynb` - Jupyter notebook for testing yielding functions

13 Conclusion

This documentation provides an overview of the Sentiment Analysis Application, including its structure, installation steps, and usage. For more detailed information, refer to the individual script files and the `README.md` and `documentation.txt` files.

14 Appendix

14.1 Detailed Code Examples

```
def analyse_text(text: str = '', translate: bool = True, skip_non_eng: bool = False):
    """
    Analyzes the sentiment of a provided text. Translates any non-English
    text by default.

    Returns:
        - 1 if language detection fails.
        - 2 if language translation fails.
        - 0 for text that was not in English and was skipped.

    Returns sentiment scores (positive, negative, neutral, compound),
    detected language, and (if applicable) translated text.
    """
    # Default return
    if text == '':
        return ''

    # Creating objects to analyze and translate
    sia = SentimentIntensityAnalyzer()
    translator = Translator()

    # Language detection
    try:
        language = translator.detect(text).lang
    except Exception:
        return 1

    # Skip non-English texts
    if skip_non_eng and language != 'en':
        return 0

    # Skipping this part if not translating -> better performance
    if translate:
        if language != 'en':
            # Translating detected language to English
            try:
                text = translator.translate(text, src=language, dest='en').text
            except Exception:
                return 2
        # Scores for translated text
        scores = sia.polarity_scores(text)
        return scores['pos'], scores['neg'], scores['neu'], scores['compound'], language, text

    # Sentiment analysis
    scores = sia.polarity_scores(text)

    # Return results
    return scores['pos'], scores['neg'], scores['neu'], scores['compound'], language, text
```

14.2 Advanced Configuration

- How to configure the application for different environments (development, testing, production)
- How to set up logging and monitoring

14.3 Troubleshooting

- Common issues and their solutions
- How to debug the application

14.4 Future Work

- Planned features and improvements
- How to contribute to the project

15 Detailed Code Examples

15.1 Amazon Review Scraper

The Amazon Review Scraper is designed to log into Amazon, extract reviews from a specified product page, clean the data, and export it to a CSV file. Below is a detailed example of how to use the scraper:

```
from scraperAmazon import get_reviews

# URL of the Amazon product page
url = "https://www.amazon.com/product-reviews/B08N5WRWNW"

# Call the function to get reviews
get_reviews(url)
```

15.2 X.com Reply Scraper

The X.com Reply Scraper logs into X.com, extracts replies from a specified post, cleans the data, and exports it to a CSV file. Below is a detailed example of how to use the scraper:

```
from scraperX import get_replies

# URL of the X.com post
url = "https://www.x.com/post/1234567890"

# Call the function to get replies
get_replies(url)
```

15.3 YouTube Comment Scraper

The YouTube Comment Scraper uses the YouTube Data API key to access video comments, cleans the data, and exports it to a CSV file. Below is a detailed example of how to use the scraper:

```
from scraperYT import get_comments

# YouTube video ID
video_id = "dQw4w9WgXcQ"

# Call the function to get comments
get_comments(video_id)
```

15.4 Web Interface Routes

The web interface is built with Flask and provides various routes for interacting with the sentiment analysis features. Below is a detailed example of the `/singlereview` route implementation:

```
@app.route("/singlereview", methods=["GET", "POST"])
def singlereview():
    """Handle single review analysis."""
    if request.method == "POST":
        start = time.time()
        text = request.form["textinput"]
        print(text)
        scores = analyses.default_analysis(text)
        sentiment = analyses.get_sentiment(scores[3])
        end = time.time()
        adbreak = 15
        wait_time = max(0, (adbreak - (end - start)) * 1000)
        return render_template(
            "singlereview.html",
            textinput=text,
            sentiment=sentiment,
            positive=scores[0] * 100,
            negative=scores[1] * 100,
            neutral=scores[2] * 100,
            language=scores[4].upper(),
            text=scores[5],
            wait_time=wait_time
        )
    else:
        return render_template("singlereview.html", wait_time=0)
```

16 Advanced Configuration

16.1 Configuring for Different Environments

To configure the application for different environments (development, testing, production), you can use environment variables and configuration files. Below is an example of how to set up different configurations:

```
import os

class Config:
    DEBUG = False
    TESTING = False
    DATABASE_URI = 'sqlite://:memory:'

class DevelopmentConfig(Config):
    DEBUG = True
    DATABASE_URI = 'sqlite:///dev.db'

class TestingConfig(Config):
    TESTING = True
    DATABASE_URI = 'sqlite:///test.db'

class ProductionConfig(Config):
    DATABASE_URI = 'sqlite:///prod.db'

def create_app(config_class=Config):
    app = Flask(__name__)
    app.config.from_object(config_class)
    return app

app = create_app(os.getenv('FLASK_CONFIG') or 'config.DevelopmentConfig')
```

16.2 Setting Up Logging and Monitoring

To set up logging and monitoring for the application, you can use the built-in logging module and third-party services like Sentry. Below is an example of how to set up logging:

```
import logging
from logging.handlers import RotatingFileHandler

# Set up logging
handler = RotatingFileHandler('app.log', maxBytes=10000, backupCount=1)
handler.setLevel(logging.INFO)
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
handler.setFormatter(formatter)
app.logger.addHandler(handler)

# Example usage
@app.route('/example')
def example():
    app.logger.info('Example route accessed')
    return 'Example route'
```

17 Troubleshooting

17.1 Common Issues and Solutions

Here are some common issues you might encounter while using the application and their solutions:

- **Issue:** Virtual environment activation fails.
- **Solution:** Ensure you are using the correct command for your operating system. On Windows, use `myenv\Scripts\activate`. On macOS/Linux, use `source myenv/bin/activate`.
- **Issue:** Browser installation fails.
- **Solution:** Ensure you have the necessary permissions to install software on your system. Try running the command with elevated privileges (e.g., using `sudo` on macOS/Linux).
- **Issue:** Credentials not found.
- **Solution:** Ensure you have created the `credentials` directory and added the necessary files (AMAZON, X, YT_API_KEY) with the correct format.

17.2 Debugging the Application

To debug the application, you can use the built-in debugging tools provided by Flask and Python. Below are some tips for debugging:

- Use the `debug` mode in Flask to get detailed error messages and stack traces.
- Use breakpoints and the `pdb` module to step through your code and inspect variables.
- Check the application logs for any error messages or warnings.

18 Future Work

18.1 Planned Features and Improvements

Here are some planned features and improvements for the Sentiment Analysis Application:

- Add support for more platforms (e.g., Facebook, Instagram).
- Improve the accuracy of sentiment analysis by using more advanced NLP models.

- Add a user authentication system to restrict access to certain features.
- Implement a dashboard for visualizing sentiment analysis results.

18.2 How to Contribute to the Project

If you would like to contribute to the project, please follow these steps:

- Fork the repository on GitHub.
- Create a new branch for your feature or bug fix.
- Make your changes and commit them with descriptive commit messages.
- Push your changes to your forked repository.
- Create a pull request to merge your changes into the main repository.