Web Scraper Project

Contents

Understanding the problem statement

Part 1: Scrape the <body> content of the URLs and extract the text

Part 2: Get the 10 most used keywords (descending order) and 10 fewest used keywords (ascending order) in all URLs and print them on the console.

Approaches

- **For Data Scraping from web pages**

BeautifulSoup (without multithreading)

- The idea is to implement the scrapper to get the data from different URLs.
- This can be done by iterating through the list of URLs sequentially and extracting the data.
- This seems feasible if we have a number countable on fingers.
- What if this number increases because the get request will take the required time.
- We can solve this with the help of multithreading discussed in the next point.

BeautifulSoup (with multithreading)

- The python library **concurrent.futures** helps to launch the tasks in parallel.

Scrapy

- Scrapy is an open-source collaborative framework for extracting the data from the websites that we need.
- Its performance is ridiculously fast and it is one of the most powerful libraries available out there.

- **For keywords extraction**

To get the keywords from the extracted text, we are using the following steps:

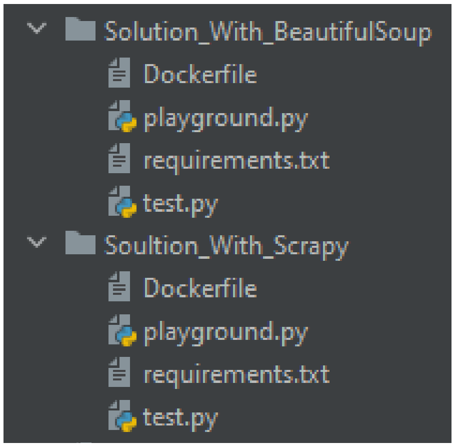
1. Get the extracted text
2. Form the list of words from the extracted text
3. Remove the punctuation marks from the words
4. Remove the stopwords like is, the, a, an, etc.
5. Get the frequency of words from this cleaned list of words and store it in the dictionary(data type in python)
6. Sort the dictionary and get the most used keywords and least used keywords.

Python Modules

Module name	Purpose	Reference link
beautifulsoup4	Python library to scrape data from web pages	beautifulsoup4
scrapy	Framework for extracting data from web pages	scrapy

nltk	Python package for natural language processing	nltk
requests	HTTP library for Python	requests
logging	Flexible event logging system	logging
unittest	To perform unit testing	unittest

Project Folder Structure



```
Solution_With_BeautifulSoup
    Dockerfile
    playground.py
    requirements.txt
    test.py
Soultion_With_Scrapy
    Dockerfile
    playground.py
    requirements.txt
    test.py
```

Fig. 1. Project Folder Structure

BeautifulSoup Vs Scrapy

- BeautifulSoup is a python library whereas scrapy is the application framework.
- BeautifulSoup parses HTML/XML documents into a tree structure to find and extract data.
- With scrapy, we can download, clean, and save the data from the web.
- Scrapy comes with the interactive shell console as well.
- When it comes to performance, scrapy is faster than BeautifulSoup. However, the performance of BeautifulSoup can be improved with multithreading.

How to reproduce the project

Step 1: Using the Git CLI, copy the following command:
git clone https://github.com/itsvaishnavi/web_scrapper.git

To download git, visit https://git-scm.com/downloads

Step 2:

(A) To run the scraper using BeautifulSoup, Navigate to the **Solution_With_BeautifulSoup** folder.

Step 2.1: With Docker

[Docker installation reference link: https://docs.docker.com/get-docker/]

Step 2.1.1: Create a docker image using Dockerfile using the command below:

```
docker build -t beautifulsoup_scraper_image .
```

Step 2.1.2: To check if the image is created, run the following command:

```
docker images
```

Step 2.1.3: Run the image using the following command:

```
docker run beautifulsoup_scraper_image
```

Step 2.2: Without Docker

Step 2.2.1: Create the virtual environment using the following command:

```
python3 -m venv /opt/venv
```

Step 2.2.2: Activate the virtual environment using the following command:

```
/opt/venv/bin/activate
```

Step 2.2.3: Install all the dependencies using the following command:

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

This downloads all the essential python modules required for the project.

Step 2.2.4: Run the scrapper script using the following command:

```
python3 playground.py
```

Step 2.2.5: You can run the unit tests using test.py. To run test.py, use the following command:

```
python -m unittest
```

(B) To run the scraper using Scrapy, Navigate to the **Soultion_With_Scrapy** folder.

Step 2.1: With Docker

[Docker installation reference link: https://docs.docker.com/get-docker/]

Step 2.1.1: Create a docker image using Dockerfile using the command below:

```
docker build -t scrapy_scraper_image .
```

Step 2.1.2: To check if the image is created, run the following command:

```
docker images
```

Step 2.1.3: Run the image using the following command:

```
docker run scrapy_scraper_image
```

Step 2.2: Without Docker

Step 2.2.1: Create the virtual environment using the following command:

```
python3 -m venv /opt/venv
```

Step 2.2.2: Activate the virtual environment using the following command:

```
/opt/venv/bin/activate
```

Step 2.2.3: Install all the dependencies using the following command:

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

This downloads all the essential python modules required for the project.

Step 2.2.4: Run the scrapper script using the following command:

```
python3 playground.py
```

Step 2.2.5: You can run the unit tests using test.py. To run test.py, use the following command:

```
python -m unittest
```

Further ideas/improvements

1. We can improve the keywords extraction by implementing NLP algorithms.
2. Performance of scraping using Scrapy vs BeautifulSoup

Scrapy vs BeautifulSoup	On local (processing time in seconds)	On cloud(google colab) (processing time in seconds)	Running Docker image (processing time in seconds)
Scrapy	542	40	60
BeautifulSoup	120	480	130

Author

Vaishnavi Piyush Kand