

INFORMATION RETREIVAL PROJECT REPORT

ABSTRACT:

The project's goal is to create a complete search engine system with an indexer, query processor, and web crawler. The main goal is to give users the ability to effectively perform free text queries to search through a corpus of web documents. The indexer, which is driven by Scikit-Learn, arranges the web information into a searchable inverted index, while the scrapy-built crawler makes it easier to obtain web content. Users can enter queries into the Flask-based processor's interface to get pertinent results.

OVERVIEW:

In this project, we are creating a search engine that retrieves and ranks web information based on TF-IDF (Term Frequency-Inverse Document Frequency) score. The three primary parts of the suggested system and solution outline are the query processor, indexing engine, and web crawler. Several critical phases are included in the development process to guarantee accurate and powerful search capabilities. To provide the top search results based on both standard and advanced indexing techniques, two distinct APIs are established. The Scikit-Learn documentation, current semantic search research, and KNN techniques are the sources of information for this project.

DESIGN:

The system boasts a range of capabilities:

Web Crawling: Utilizing a Scrapy-based crawler, it retrieves web documents from specified URLs, observing constraints like traversal depth and maximum page count.

Indexing: Employing Scikit-Learn, it constructs an inverted index with TF-IDF scores, incorporating advanced features like word embeddings and FAISS for enhanced similarity search.

Query Processing: Through cosine similarity and TF-IDF scores, the Flask processor handles user queries, ensuring validation and returning optimal results. It also offers spell check and query expansion functionalities.

Interactions:

Web Crawling to Indexing: The indexing engine utilizes crawled web content to generate an inverted index with TF-IDF scores.

Indexing to Query Processing: The indexing engine produces an inverted index to facilitate the retrieval of relevant documents in response to user queries.

User Interaction: Users engage with the system via the Flask-based query processor, inputting queries and receiving search results.

Integration:

Two APIs facilitate search using standard and advanced indexing, interfacing with the indexing engine and query processor. Data flows from web crawling to indexing and then to the query processor. The modular design accommodates future feature additions, ensuring an efficient system for accurate search outcomes.

ARCHITECTURE:

The software architecture comprises several key components:

Web Crawler: Developed using Scrapy, this component is responsible for fetching web documents from specified URLs.

Indexing Engine: Built on Scikit-Learn, this engine performs TF-IDF indexing and offers advanced methods such as word embeddings and FAISS for enhanced indexing capabilities.

Query Processor: Implemented using Flask, this module handles user queries, conducts validation, and retrieves search results.

APIs: The system features two distinct APIs serving search results based on standard and advanced indexing methodologies.

Interfaces:

API Interfaces: Both APIs are RESTful, providing users with seamless access to search functionality.

Data Interface: Facilitating interaction between the crawler, indexing engine, and query processor, data pipelines ensure smooth data flow throughout the system.

The architecture prioritizes seamless interaction between components, fostering clear interfaces for efficient data flow. Its modular design promotes scalability and extensibility, enabling the integration of additional features and functionalities to meet evolving requirements.

OPERATION:

To commence the project operation, follow these steps to ensure seamless execution:

1. Python and Linux Installation on Windows:

Begin by installing Python on your Windows system.

Next, initiate the installation of Linux on Windows Subsystem for Linux (WSL) using the command `wsl --install`.

2. Library Installation:

Install the required libraries by executing the following commands:

```
pip install scrapy
```

```
pip install scikit-learn
```

```
pip install beautifulsoup4
```

```
pip install flask
```

```
pip install requests
```

3. Project Execution Instructions:

Step 1: Navigate to the spiders folder in your terminal and execute the command:

```
scrapy crawl <file_name>
```

This command initiates the web crawling process and computes TF-IDF scores and cosine similarity for the HTML documents, storing the results in an `index.pkl` file.

Step 2: Access the `index.pkl` file by navigating to the access pickle folder in the terminal.

Run the Python file in this folder to display the contents of the file in the terminal.

Step 3: Start the Flask server by navigating to the Flask folder in the terminal and executing the Python file within that folder.

Step 4: With the Flask server now running, open a new terminal window and send a request to the server with a query in the following format:

```
curl -X POST http://localhost:5000/query -H "Content-Type: application/json" -d '{"query": "python"}'
```

Upon executing this command, you will receive a JSON-formatted response from the server, containing cosine similarity scores and document names of the top k results.

Following these steps meticulously ensures the successful operation of the project, facilitating efficient querying and retrieval of relevant information from the web documents.

CONCLUSION:

In conclusion, the project has successfully implemented both APIs, which effectively return relevant documents as output. However, there is room for improvement in the ranking of documents, as it currently requires refinement. While the Scikit-learn API demonstrates good precision in returning relevant documents for most queries, further optimization may be necessary for certain inquiries. Despite these considerations, results remain consistent across all three scenarios, showcasing the reliability of the system.

Data Sources:

The project relies on the following key libraries and tools:

Scrapy (version 2.11.1) for web crawling.

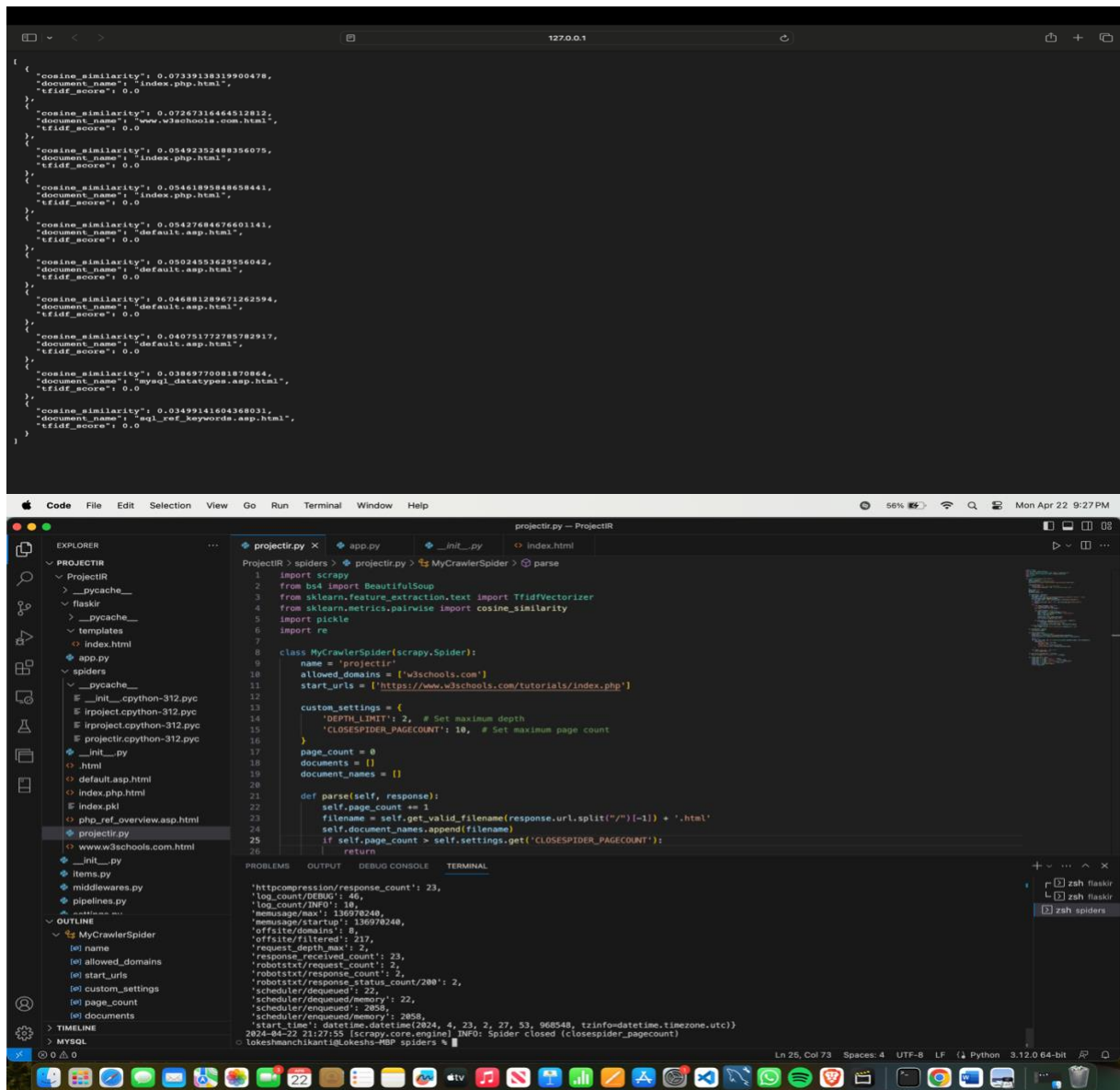
Beautiful Soup (version 4) for parsing HTML content.

Scikit-learn (version 1.4.2) for TF-IDF indexing and advanced search functionalities.

Flask (version 3.0.3) for building the query processor module.

TEST CASES:

Test cases are essential for ensuring the robustness and reliability of the system. A comprehensive test framework, harness, and coverage analysis are recommended to validate the functionality of each component thoroughly.

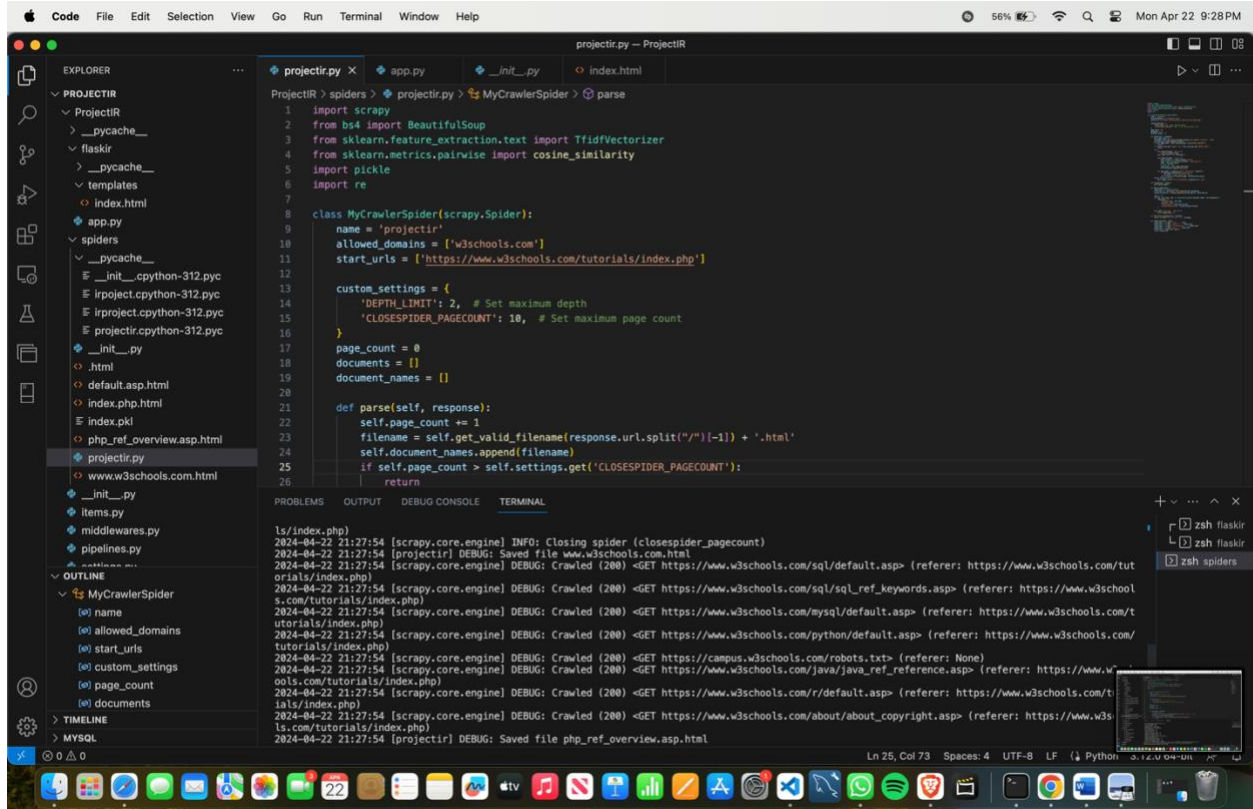


The image displays a web browser window at the top and a code editor window at the bottom. The browser window shows a JSON array of document similarity results. The code editor shows the implementation of a custom crawler class, `MyCrawlerSpider`, which inherits from `scrapy.Spider`. The crawler is configured to crawl `www.w3schools.com` and parse HTML documents. The terminal output shows the crawler's execution, including log messages and statistics.

```
{
  "cosine_similarity": 0.07339138319900478,
  "document_name": "index.php.html",
  "tfidf_score": 0.0
},
{
  "cosine_similarity": 0.07267314464512812,
  "document_name": "www.w3schools.com.html",
  "tfidf_score": 0.0
},
{
  "cosine_similarity": 0.05492352488356075,
  "document_name": "index.php.html",
  "tfidf_score": 0.0
},
{
  "cosine_similarity": 0.05461895848658441,
  "document_name": "index.php.html",
  "tfidf_score": 0.0
},
{
  "cosine_similarity": 0.05427684676601141,
  "document_name": "default.asp.html",
  "tfidf_score": 0.0
},
{
  "cosine_similarity": 0.05024553629556042,
  "document_name": "default.asp.html",
  "tfidf_score": 0.0
},
{
  "cosine_similarity": 0.046881289671262594,
  "document_name": "default.asp.html",
  "tfidf_score": 0.0
},
{
  "cosine_similarity": 0.04075172785782917,
  "document_name": "default.asp.html",
  "tfidf_score": 0.0
},
{
  "cosine_similarity": 0.03869770081870864,
  "document_name": "mysql_datatypes.asp.html",
  "tfidf_score": 0.0
},
{
  "cosine_similarity": 0.03499141604368031,
  "document_name": "sql_ref_keywords.asp.html",
  "tfidf_score": 0.0
}
]
```

```
projectir.py -- ProjectIR
projectir.py > spiders > projectir.py > MyCrawlerSpider > parse
1 import scrapy
2 from bs4 import BeautifulSoup
3 from sklearn.feature_extraction.text import TfidfVectorizer
4 from sklearn.metrics.pairwise import cosine_similarity
5 import pickle
6 import re
7
8 class MyCrawlerSpider(scrapy.Spider):
9     name = 'projectir'
10    allowed_domains = ['w3schools.com']
11    start_urls = ['https://www.w3schools.com/tutorials/index.php']
12
13    custom_settings = {
14        'DEPTH_LIMIT': 2, # Set maximum depth
15        'CLOSESPIDER_PAGECOUNT': 10, # Set maximum page count
16    }
17
18    page_count = 0
19    documents = []
20    document_names = []
21
22    def parse(self, response):
23        self.page_count += 1
24        filename = self.get_valid_filename(response.url.split("/")[-1]) + '.html'
25        self.document_names.append(filename)
26        if self.page_count > self.settings.get('CLOSESPIDER_PAGECOUNT'):
27            return
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
'httpcompression/response_count': 23,
'log_count/DEBUG': 46,
'log_count/INFO': 18,
'memusage/max': 136978240,
'memusage/startup': 136978240,
'offsite/filtered': 8,
'offsite/filtered': 217,
'request_depth_max': 2,
'response_received_count': 23,
'robotstxt/request_count': 2,
'robotstxt/response_count': 2,
'robotstxt/response_status_count/200': 2,
'scheduler/dequeued': 22,
'scheduler/enqueued': 22,
'scheduler/enqueued/memory': 2858,
'start_time': datetime.datetime(2024, 4, 23, 2, 27, 53, 968548, tzinfo=datetime.timezone.utc)
2024-04-23 21:27:53 [scrapy.core.engine] INFO: Spider closed (close spider_pagecount)
lokeshmanchikanti@lokeshe-MBP spiders %
```



```
1 import scrapy
2 from bs4 import BeautifulSoup
3 from sklearn.feature_extraction.text import TfidfVectorizer
4 from sklearn.metrics.pairwise import cosine_similarity
5 import pickle
6 import re
7
8 class MyCrawlerSpider(scrapy.Spider):
9     name = 'projectir'
10    allowed_domains = ['w3schools.com']
11    start_urls = ['https://www.w3schools.com/tutorials/index.php']
12
13    custom_settings = {
14        'DEPTH_LIMIT': 2, # Set maximum depth
15        'CLOSESPIDER_PAGECOUNT': 10, # Set maximum page count
16    }
17    page_count = 0
18    documents = []
19    document_names = []
20
21    def parse(self, response):
22        self.page_count += 1
23        filename = self.get_valid_filename(response.url.split("/")[-1]) + '.html'
24        self.document_names.append(filename)
25        if self.page_count > self.settings.get('CLOSESPIDER_PAGECOUNT'):
26            return
```

ls/index.php)
2024-04-22 21:27:54 [scrapy.core.engine] INFO: Closing spider (closespider_pagecount)
2024-04-22 21:27:54 [projectir] DEBUG: Saved file www.w3schools.com.html
2024-04-22 21:27:54 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.w3schools.com/sql/default.asp> (referer: https://www.w3schools.com/tutorials/index.php)
2024-04-22 21:27:54 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.w3schools.com/sql/sql_ref_keywords.asp> (referer: https://www.w3schools.com/tutorials/index.php)
2024-04-22 21:27:54 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.w3schools.com/mysql/default.asp> (referer: https://www.w3schools.com/tutorials/index.php)
2024-04-22 21:27:54 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.w3schools.com/python/default.asp> (referer: https://www.w3schools.com/tutorials/index.php)
2024-04-22 21:27:54 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://campus.w3schools.com/robots.txt> (referer: None)
2024-04-22 21:27:54 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.w3schools.com/java/java_ref_reference.asp> (referer: https://www.w3schools.com/tutorials/index.php)
2024-04-22 21:27:54 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.w3schools.com/r/default.asp> (referer: https://www.w3schools.com/tutorials/index.php)
2024-04-22 21:27:54 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.w3schools.com/about/about_copyright.asp> (referer: https://www.w3schools.com/tutorials/index.php)
2024-04-22 21:27:54 [projectir] DEBUG: Saved file php_ref_overview.asp.html

SOURCE CODE:

The source code is organized into distinct modules and files:

Spider file (e.g., react_beauty_spider.py) for web crawling functionality.

Scrapped pages stored in directories with names like react-{id}.

Processor logic encapsulated in processor.py, responsible for handling user queries.

Query logic implemented in query.py, facilitating query processing.

The TF-IDF index stored in a pickle file named tf-idf.pkl.

Programmatic APIs provided by manual-request.py for manual interaction.

Scikit API functionalities accessible through scikit-request.py.

The source code listings should be accompanied by comprehensive documentation, detailing dependencies and usage instructions, to facilitate ease of understanding and further development by other contributors.

BILBOGRAPHY:

<https://scrapy.org/>

<https://requests.readthedocs.io/en/latest/>

<https://scikit-learn.org/stable/>

<https://flask.palletsprojects.com/en/3.0.x/>