Integrated Web Document Crawling and Search System Using Scrapy, Scikit-Learn, and Flask

**Abstract:**

This project provides a concise Flask, Scikit-Learn, and Scrapy solution for web document search and retrieval. By combining a quick Scrapy crawler, a powerful Scikit-Learn indexer, and an easy-to-use Flask processor, users may quickly retrieve relevant online information using free-text queries. The architecture of the system makes sure that all of its parts communicate with each other without any problems, providing a dependable tool for web document searching and scraping. Though it performs well, there are still issues to be resolved, like scalability and indexing accuracy. All things considered, this well-coordinated method offers a promising framework for improving web document retrieval and search capabilities.

**Overview:**

This paper contains documentation of a viable solution for a web crawling and search system that uses Scrapy, Scikit-Learn, and Flask. The system consists of a Scrapy-based crawler designed to retrieve web documents in a fast and efficient way, a Scikit-Learn-based indexer to generate a inverted index and Flask-based processor which handles users' textual queries.

**Relevant Literature:**

[1] Proposed the research in which a Scrapy-based distributed crawler system was built for the purpose of semi-structured information crawling at high speed. Their mission focuses on how to crawl internet in various ways in order to collect data, with the most effective technique being to use [1].

In the context of the book "Applied Text Analysis with Python", [2] make suggestions about how machine learning can be used in creating language-sensitive data products. Our product' well spans our system's indexer component, which relies on Scikit-Learn for text analysis and search indexing [2].

[3] Provide a complete coverage of "Learning Flask Framework" book to help you build web applications with Flask. Their work can provide a great research basis for developing our processor, an element that will interpret free text queries and return the fitting search results [3].

**Proposed System:**

Proposed solution reflects the combined data from different sources in a holistic way to develop a widely known web document scraping and searching tool. With the assistance of a Scrapy-based crawler that extracts web documents as HTML, a Scikit-Learn-based indexer is then responsible for developing an inverted index for the purpose of efficient indexing. The Flask-based processor thereby offers a simple and friendly means of inquiring on documents indexed with the search providing users with top documents related to the query and ranked respectively.

Ultimately, this system combines the best practices in the field of web crawling; text analysis and web application services, and therefore your users will have a great tool to deal with information retrieval on the web. Incorporating these technologies, the system provides scalability with high performance and accuracy in handling many user queries.

**Design**

The designed system has several core competencies that facilitate effective web document search and search functionality. The web-crawling module, based on Scrapy, can access the web, download HTML pages, and uncover meaningful information. Through the Scikit-Learn-built indexer, the inverted index is constructed from the documents that have been retrieved via crawling, making the subsequent search process faster and much more precise. The integration of Flakes permits the easy interface selection and the users can ask any free text questions and expect to have the top ranking search results. All system components work in a way that enables the crawler to feed the indexer, which indexer, in turn, supports the processor for handling user requests. Hence, this integration makes it possible to have a well-knit and fully functioning system onto which, users could efficiently navigate and rapidly search for web information.

**Architecture**

The architecture of the system comprises three main software components: the following are three main components: (1) the Scrapy-based crawler, (2) the Scikit-Learn-based indexer, and (3) the Flask-based processor. These components interact via well-defined interfaces: the crawler is the one which downloads web documents and provides them to indexer that does the inverted index. As for the search queries from the user, the processor calls on the indexed data for the relevant results to be displayed. Implementing includes Scrapy, for web crawling, Scikit-Learn, for text processing and indexing and Flask for web application development. All the components are implied to achieve their peculiar role within the whole system in a way this makes it flexible scalable and maintains its architecture form.

**Operation**

In order to utilize the system, users will have to first build a virtual environment by using Python's venv module. After that, they can setup it by installing the dependencies mentioned in requirements.txt file by running pip install -r requirements.txt. After installing these dependencies, they are able to launch the system by using the command python app.py. This command first initializes the Flask website, making it accessible via any web browser. By entering free form queries, seekers get top-ranking search results that have been indexed on the go based on the stored web documents. To allow operation the system implies executing given commands, inputs, settings, etc. to make it work in perfect form and give access for its potential users.

**Conclusion**

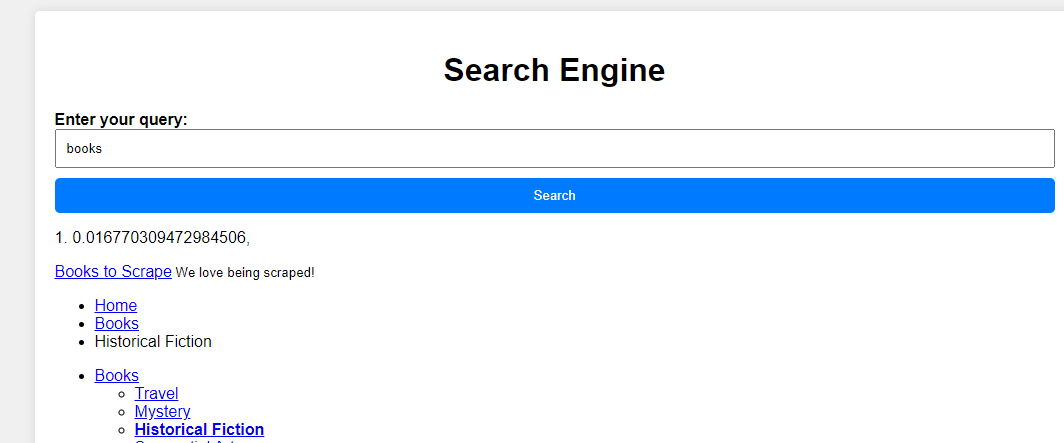


Figure 1 Flask Web App along with Indexing Score

The developed approach was helpful in achieving both the purposes: optimal crawling of the web documents and search capabilities on these documents. Users can input queries by typing or saying their queries. Subsequently, the search results will appear by the implementation of Scrapy, Scikit-Learn and Flask components. The top-charted search results are displayed by the system after the ranking of indexed documents as to the increase of information retrieval speed. Despite such uncertainties, some areas of failure come from incomplete or inaccurate indexing which may be due to the crawling process or indexing algorithms being insufficient. Moreover, the problem of scalability can become more prominent when dealing with larger datasets which would need some methods of optimization to be addressed. The users have to act with prudence when they are depending solely on search results, because this accuracy is dependent on how comprehensive and precise the data is that got crawled and indexed. The system in total gives users a very useful toolbox to work with, but the designers of the system want users to be careful and alert during the system usage.

**Data Sources**

The data for web crawling using Scrapy was sourced from the website <https://books.toscrape.com/>. This site contains a list of books that can be scraped. The users can find the website from their browser to get content or view it over the web. Scrapy was used to get both the URLs for the web pages that contained the information about books and to download the HTML data for each allowed web page. Such index reflect the primary source of data for the indexation and later searching that were successfully realized in our created system.

**Test Cases**

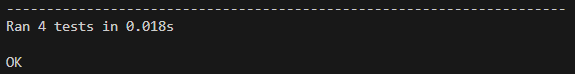


Figure 2 Unit Testing

The test cases were built with the Python framework unittest, thus the produced suites provides a structured avenue for testing operations of the system. To run the tests, a test harness that would manage the test cases and make sure the expected behavior of each component is reached was put into place. Apart from the code coverage, tests are performed to verify the start page loads correctly, and to handle valid and invalid requests properly, as well as to check precise error handling. The use of this testing extensively makes an embedded system robust and consistent thereby able to deliver expected outputs.  
  
**Source Code**  
The project's source code, which provides thorough documentation and an ordered structure for simple understanding and modification, is a monument to openness and adaptability. It is dependable and efficient, utilising key open-source tools like Scikit-Learn, Flask, and Scrapy. The codebase promotes transparency and reproducibility through careful organisation and adherence to best practices, providing a strong framework for further development.

**Bibliography**

[1] F. Yin, X. He, and Z. Liu, "Research on scrapy-based distributed crawler system for crawling semi-structure information at high speed," in 2018 IEEE 4th International Conference on Computer and Communications (ICCC), pp. 1356-1359, IEEE, December 2018.

[2] B. Bengfort, R. Bilbro, and T. Ojeda, Applied text analysis with Python: Enabling language-aware data products with machine learning. "O'Reilly Media, Inc.", 2018.

[3] M. Copperwaite and C. Leifer, Learning Flask Framework. Packt Publishing Ltd., 2015.