

Jacob Baskiewicz

CS429

12/01/25

Final Project Report

Abstract

The objective of this project is to create a Flask-based search application for allowing text queries to return a list of scraped web pages and their correlating cosine similarity for the query provided. The list of the top 5 pages is sorted in descending order of their cosine similarity to provide the user with the webpage with the most relevant results. This project may be useful for individuals searching for several subpages or to have a local search engine environment. Next steps would be combining the system into one usable application with UI, such as providing a space for seed URLs, and entering in free-text queries and getting a result immediately. Another addition would be semantic searching and query expansion.

Overview

To accomplish the application, the project is broken down into 3 major parts. Scrapy is used to build the html crawler, which can use any seed URL to return the first 20 pages with a max depth of 2. After scraping the page, Scikit-learn is used to build an index of all terms found. TF-IDF scores are calculated to find term relevance, and cosine similarity is found to compare with a provided query. Lastly, Flask is used as the user application. A user can provide a CSV file of queries, which will be searched in the inverted index to find the pages with the most relevance, and saved to a text document. The top-k results is set to be 5. In cases of spelling errors, NLTK is used to find edit distance between the term in the query, and the next closest in the inverted index. The suggested term is then shown to the user for the provided term.

Design

The capabilities of the system are limited to scraping, building an inverted index, and providing a CSV file with queries to do a search. The interactions between the systems are as follows.

Firstly, the Scrapy crawler will crawl and save the webpages, which the Scikit-learn indexer will use to build a TF-IDF inverted index using a helper python function to run. The flask application will read the resulting JSON file or run the indexer if no JSON is found. The flask application then reads the resulting JSON file and will return a text file output based on the search.

Architecture

- Scrapy crawler

The scrapy crawler named “html_crawler”, takes a seed URL, and will crawl for 20 max pages with a max depth of 2. The default seed URL is a commonly used seed for testing Scrapy crawlers known as “quotes.toscrape.com”. The crawler will initialize and perform its crawl, saving all webpages in the project folder as “downloaded_page_x.html”.

- Scikit-learn Indexer

The Scikit-learn indexer uses Beautiful Soup 4 to parse the html pages and save all unique terms separated by spaces. It builds the TF-IDF matrix using the default list of English stop words, also making sure not to include duplicates within the index. The resulting TF-IDF matrix is organized as a JSONfile, with each term having resulted "doc_id": "downloaded_page_x.html" and "tfidf": 0.x. The resulting index is saved as a JSON in the project folder with the name “inverted_index.json”. To avoid needing console input, a python file called “run_indexer.py” was created to run the indexer remotely.

- Flask Application

The Flask Application has two python files to run. The file “app.py” has the basic functionality of the application. The flask application first will check and see if an inverted index exists, and either use one if it exists or create one if not. The query processor function will check if a CSV file with queries exists and check to make sure that a column for such queries exists. If found, the application will build the output, first checking to see if the query has a suggestion if not found in the inverted index. If a word is found in the inverted index, the top 5 documents will be returned to the user. The output is formatted to read better for an average user. The entire application has many error checks responding with a status 400 if anything prompts an error when reading the inverted index.

Operation

The following libraries are needed to operate the system. Scrapy, Flask, NLTK, Scikit-learn, Beautiful Soup 4.

When scraping, the following needs to be done to operate. Input a seed url into the HTML crawler python class, and in a terminal within the directory of the crawler, run the command “scrapy crawl html_crawler” which will crawl the page based on the seed url and attributes described in the crawler initialization. All of the pages will then be saved in the project folder for use by the indexer.

When indexing, all that needs to be done is to run the python file “run_indexer.py” which will run the python file “indexer.py”. This will save the resulting inverted index in the project folder.

When running the flask application, first, the “app.py” python file needs to be run from the terminal using “python app.py”. The user may input the queries they want to search in the “queries.csv” file in the project. Then, the user may run the queries by running the “test_flask_queries.py” file in a second terminal. From there, the results are saved as a text file and can be viewed.

Conclusion

Concluding the development of the process, there were a number of successes and failures during the development of the project. Setting up the Scrapy crawler became the first big roadblock, with a number of issues arising in its development such as infinite crawling and not stopping after reaching max depth, to not saving the pages correctly. These were overcome by changing a few of the project settings and debugging using Scrapy documentation and LLMs explaining the reason for the issue. The indexer was a large success, working as intended quite early in the process; however, it is only limited to the requirements of the project. The flask application is also a success, allowing free-text queries in a CSV format to search. It does, however, have the caveat of only running locally and requiring two terminals to operate. It also is less user friendly and it requires queries in a CSV format instead of console or some other type of input. It is also lacking spelling correction and query expansion.

A caution when developing a project like this is to plan the architecture of the project in advance, since managing several files and folders without a plan can lead to issues or inconveniences during development.

Data Sources

[1] <https://quotes.toscrape.com>

Test Cases

When testing during the development of the project, I first tested each individual section before testing the system as a whole. For the Scrapy crawler, I tested quotes.toscrape.com with different max pages and page depth until an efficient and effective parameter was found, that being 20 pages and 2 max depth. For testing the indexer, I ran it with groups of pages from 10-50, to see if the program correctly made an index for each group. During the development I also included a number of print statements to make sure TFIDF, Cosine similarity, and documents were calculated and read correctly. Saving the inverted index as a JSON file added clear visibility and organization to the parsed HTML pages. When testing the Flask application, I had to ensure that the queries the user passed existed, were correct, and had a correlating term in the inverted index. I also had to ensure the inverted index existed in the first place. If any issues arose, error status 400 was thrown from the flask application. When checking specific queries, I had to make sure that I included some that were misspelled to display suggested terms if one was not found within the inverted index. I also had to check for words that may show up frequently in a document, and

infrequently in many documents to see if the top 5 returned documents sorted correctly. Below is a test case example:

Query: quotes

Results:

downloaded_page_12.html -> 0.29662660165228943
downloaded_page_7.html -> 0.2546243690774928
downloaded_page_8.html -> 0.2515624369648701
downloaded_page_5.html -> 0.2510613501789357
downloaded_page_2.html -> 0.2443986794990816

Suggestions: None

Query: stupid

Results:

downloaded_page_12.html -> 0.20750141045778497
downloaded_page_10.html -> 0.15090770532077524
downloaded_page_11.html -> 0.10049849694766169
downloaded_page_1.html -> 0.05762524820913743
downloaded_page_17.html -> 0.050740502076156345

Suggestions: None

Query: stuplld

Results:

downloaded_page_1.html -> 0.0
downloaded_page_10.html -> 0.0
downloaded_page_11.html -> 0.0
downloaded_page_12.html -> 0.0
downloaded_page_13.html -> 0.0

Suggestions: stuplld -> stupid

Source Code:

[1] Scrapy 2.13 documentation — Scrapy 2.1.3 documentation. (n.d.). Docs.scrapy.org.

<https://docs.scrapy.org/en/latest/>

[2] Examples. (2025). Scikit-Learn. https://scikit-learn.org/stable/auto_examples/

[3] Flask. (2010). Welcome to Flask — Flask Documentation (3.0.x). Palletsprojects.com.

<https://flask.palletsprojects.com/en/stable/>

[4] NLTK :: nltk. (n.d.). Wwww.nltk.org. https://www.nltk.org/_modules/nltk.html

[5] Richardson, L. (2019). Beautiful Soup Documentation — Beautiful Soup 4.4.0

documentation. Crummy.com. <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

Bibliography

- [1] Scrapy Tutorial — Scrapy 2.3.0 documentation. (n.d.). Docs.scrapy.org.
<https://docs.scrapy.org/en/latest/intro/tutorial.html>
- [2] OpenAI. (2025, September 12). ChatGPT. Chatgpt.com; OpenAI. <https://chatgpt.com>
- [3] Working With Text Data. (2024). Scikit-Learn. https://scikit-learn.org/1.4/tutorial/text_analytics/working_with_text_data.html
- [4] GeeksforGeeks. (2017, October 23). Flask (Creating first simple application).
GeeksforGeeks. <https://www.geeksforgeeks.org/python/flask-creating-first-simple-application/>

