Web Search Engine UIC

Vinay Kaushik Kammara
Master's in Computer Science
University of Illinois at Chicago
Chicago, Illinois
vkamma2@uic.edu

ABSTRACT

This document is a report for the final project of CS 582 Information Retrieval at University of Illinois-Chicago. The high-level idea of the project is to build a search engine for the University of Illinois-Chicago. This search engine would yield results for any query from the user with respect to the University of Illinois at Chicago. This application involves crawling of web pages, preprocessing, indexing and a GUI for entering the query and getting the results.

KEYWORDS

Search Engine, Web crawler/spider, TFIDF.

Introduction

The software is written in Jupyter Notebook so that it would be easy to display the results in a step by step approach. Since the web crawler takes a lot of time to crawl the pages and download the content into a file, all the necessary files are included in the folder itself. In this way, you can directly run the **main.ipynb** or **main.py** file and get the output immediately without any delay. After the file is run, a GUI appears where the user is asked to enter the query which would further return the result of ranked pages.

The detailed description of the modules is given below in the subsections.

1 Web Crawling

The web crawling can be done by running '**spider.ipynb**'. The crawling starts from the seed page 'https://www.cs.uic.edu' and crawls up to 4000 pages as mentioned in the code. The number of pages can also be increased in the code. The crawling uses a traditional approach that goes from one page to another using Breadth First Search approach. The spider/crawler starts from the root page '**https://www.cs.uic.edu**' and retrieves all the hyperlinks and the from the webpage which are then added to the queue. This process is repeated until the queue is empty or the maximum number of pages specified is reached. BeautifulSoup library is used in order to pull/download the data from the webpage. All the links obtained are then checked if they belong to the UIC domain and then added to the queue. Try and catch blocks are written for the webpage response time so that if it is unable to connect to the webpage, it will skip that page without causing for the entire process to be stuck. There were a few links with extensions such as

('.avi','.ppt','.zip','.gz','.tar','.png','.gif','.doc','.pdf','.mp4',
'.svg','.css') and these files are not considered since they took a lot of time to download.

2 Term Frequency Inverse Document Frequency

Term Frequency and Inverse Document Frequency values are calculated using inverted index. The main purpose of the inverted index is to allow at query time to efficiently retrieve and rank the relevant document with respect to the given query. To do that the Inverted Index consists of a dictionary with words in the collection as entry and the list of containing documents as value. This allows us to retrieve only the documents that contain at least one word of the query, reducing the need of computation in ranking the documents. All the words in the webpage are processed before adding it to the inverted index i.e. conversion of the word to lower case, removal of whitespaces, using porter stemmer to stem the words. Nltk library is used here to import the Porter Stemmer.

3 PageRank

PageRank is an algorithm which is used to rank the webpages in the search engine results.

$$PR(p_i) = \frac{1-d}{N} + \sum_{p_j \in M(p_i)} \frac{PR(p_i)}{L(p_i)}$$

The above formula handles cycles in the graph and deals with dangling nodes. Nodes which do not have outgoing links are called dangling node (or) sinks. These can cause the page rank to collapse, causing the page rank score to be concentrated at the sink. In the experiment, verification was done by adding up the page rank of all 4000 crawled and making sure this summed up to a value of approximately 1, ensuring the score is split in the correct way.

4 Main Challenges

Implementation of the spider took me a lot of time as I was not aware of the libraries used to extract the data/content from the HTML.

The first spider which I wrote took me hours to download few pages as it downloaded the pages which had extensions of ('.avi','.ppt','.zip','.gz','.tar','.png','.gif','.doc','.pdf','.mp4', '.svg','.css'). After the exclusion of all this, it took me less time downloading the pages.

Sometimes there were many broken links or unresponsive pages. So, in order to handle those websites, I checked for the status code. Therefore, if the status code is 200, only then the page is allowed to be crawled. All the other sites are handled in the exception block.

Another challenge was to learn the basics of creating a GUI in python.

One of the main challenges was to evaluate the query results. Since we did not have any relevant documents given and the relevancy of the results of query was subjective to the user, it was difficult to evaluate the results.

5 User Interface

The user interface used here is a simple GUI imported from **easygui** library which asks for a query and number of results needed to display.

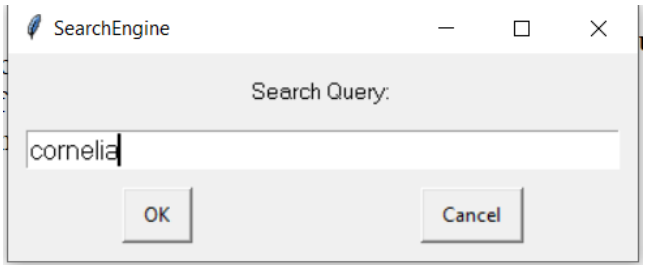

Figure 1: simple GUI asking user for input

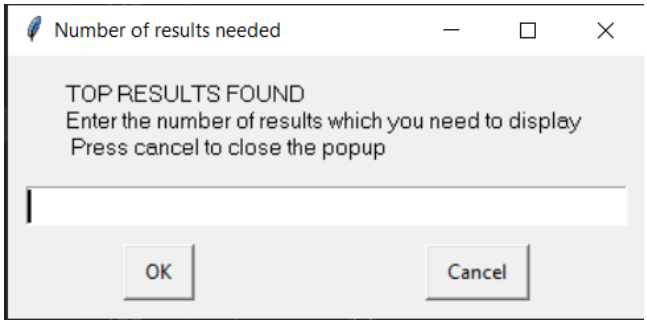

Figure 2: Number of results required

6 Experiments and Results

I have experimented on 5 random queries and printed top 10 webpages.

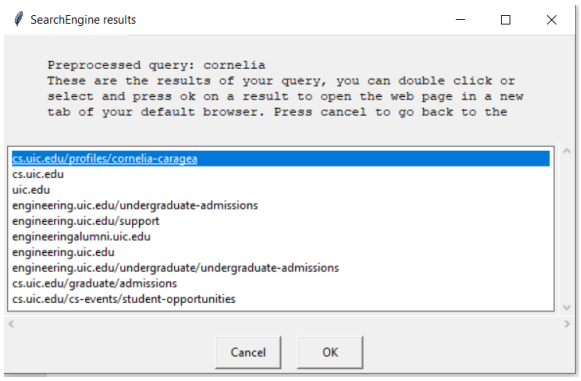

Figure 3: Displaying 10 results for 'cornelia'

In Figure 3, if you click on any of the links, it will open your default browser and redirect you to that page. Precision of 1.0 can be given since it displays the profile of 'Cornelia'

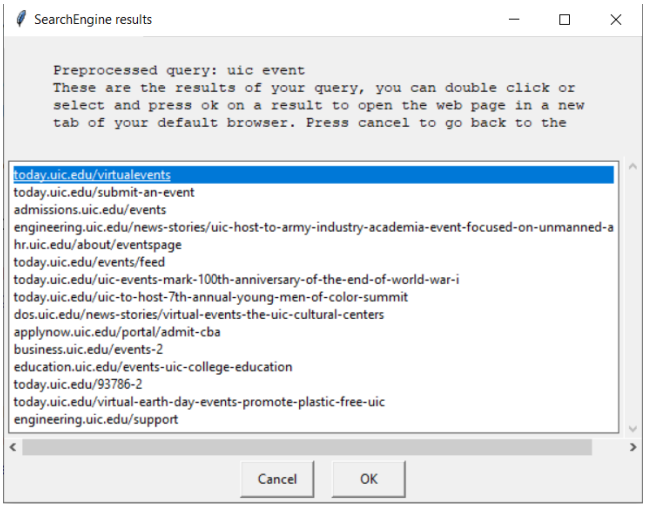

Figure 4: Displaying 15 results for UIC EVENT

In Figure 4, I have given the input as UIC EVENT in uppercase but the preprocessing is done and it gets converted into lowercase in the software and shows the results.

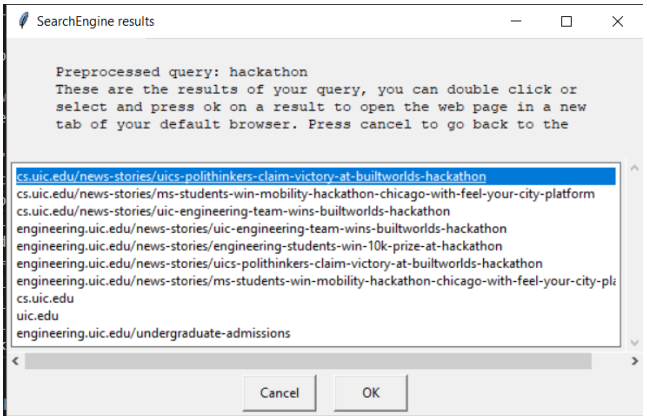

Figure 5: Displaying 10 results for hackathon

For query 'hackathon', a precision of 0.8 can be given since it does not generate the top or relevant result

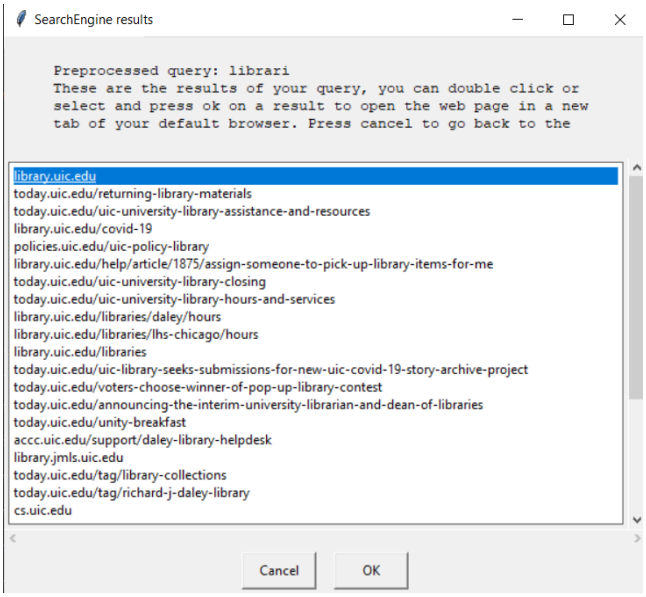

Figure 6: Displaying results for library

Precision of 1 can be given since the top result is accurate

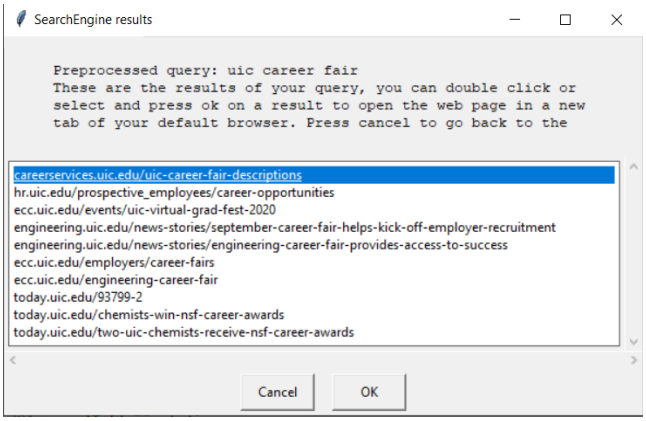

Figure 7: Displaying results for uic career fair

In Figure 7, we see that for the query 'uic career fair', we have got the correct top 10 results accurately. For 3-word sentence as a query, this can be a result with high precision of 1.0

7 Error Analysis

Since this is a basic model there are many results which are not relevant to the query entered. For example, in Figure 8 for the query of '**information retrieval**', the results do not show the CS-582 course, rather it focusses more on the keyword '**information**'. Since emergency services provide lot of information, it is displayed on the top which is not appropriate as per user's input.

If you give **1000** as **number of results needed,** then you will get the relevant result in the rank of 300-400.

For the query '**information retrieval**', a precision of 0.3-0.4 can be given as it mostly focusses on the keyword 'information' and ranks the course between 300-400.

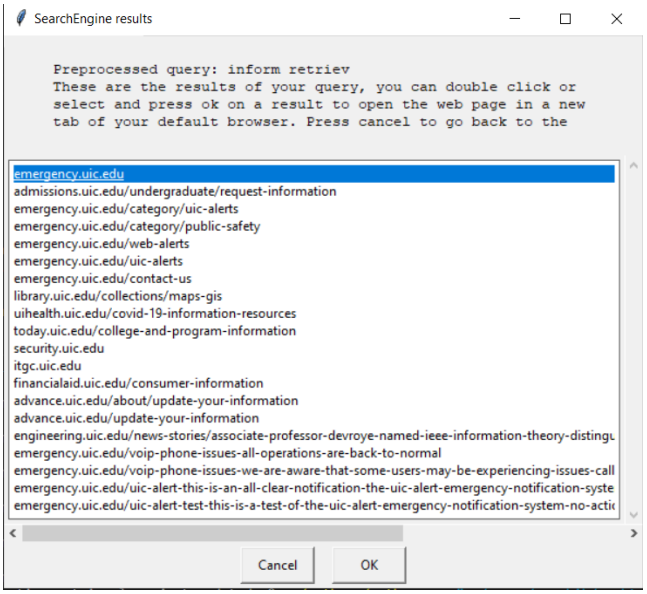

Figure 8: Results for information retrieval

Also, there are a few repetitions for the same **emergency** domains with different subdomains that can be avoided.

8 Conclusion

The overall performance of the search engine is decent enough for providing relevant results. Although few queries were dependent mostly only

the first keyword, rest of the queries generated results in a ranked order basis.

9 Relevant work

There were a few articles which focused more on web crawlers on extracting the content in an efficient manner. Also, there are many projects which were developed in the similar way such as

- https://academic.microsoft.com/home
- http://education.iseek.com/iseek/home.page

I have also tried to use the additional features which can be extracted from the webpage title, body or URL. These features are chosen from Microsoft's Learning to Rank dataset [1] which holds many features to find relevance of a page with respect to others. However, they are not released to the public and was not able to implement properly in this project. Some of the features include stream length, sum of TF, mean of TF, number of slashes, inlink count, outlink count etc.

10 Future Work

The additional features from the webpage can be implemented which can yield more proper results for the query entered since they are both independent and dependent on the query. Integration of this extracted features along with neural networks can be considered as my future work since it would yield better results.

11 References

[1] T. Qin, T.-Y. Liu, J. Xu and H. Li, "LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval," Information Retrieval, vol. 13, no. 4, pp. 346 - 374, August 2010.

[2] Introduction to Information Retrieval by Christopher D. Manning, Prabhakar Raghavan and Hinrich Schutze.

[3] https://arxiv.org/ftp/arxiv/papers/1403/1403.1939.pdf

Apart from this, there are also few websites which helped me in developing this project. They are listed below.

- https://www.geeksforgeeks.org/page-rank-algorithm-implementation
- https://medium.freecodecamp.org/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3
- https://realpython.com/beautiful-soup-web-scraper-python/