UIC Web Search Engine

CS 582 Final Project

Aayush Harish Makharia
University of Illinois at Chicago
amakha3@uic.edu
676412286

ABSTRACT

This document is a report for the final project of CS 582 Information Retrieval and Web Search at the University of Illinois at Chicago. As part of the project a web search engine was developed on the UIC domain. The software includes various components such as a Web Crawler, an IR System which takes care of preprocessing and indexing, a Page Rank module and a Query module which is a Command Line Interface that lets a user query the search engine and display results. A search engine is used to carry web based searched. The search engine will then carry a search on the web in a way such that the information in the user's query is found. The results are generally presented one after the other as per the user's query. The user's query is scanned and indexed to get the relevant output. Web engines index several millions of web pages using a variety of complex techniques and respond to billions of queries every day. Therefore, it contains a detailed description of the search engine built to work on several domains, a description of the components, discussions on challenges faced, about the weighing scheme and the similarity measure used, analysis of some sample results and the future implementations.

1 Description

The software project was built using Python3, HTML, CSS, and Flask. We used the software to build a web search engine for the UIC domain from scratch. The application comprises of a – a web crawler that goes through and collect data, the IR system preprocesses the data and creates Inverted Index file and a page Ranks file of the JSON format which is then used to query the user search to process and provide the desired.

1.1 Web Crawler

The module is responsible for crawling the web pages to extract all the links and the page content that will be used to return results for the search engine. The crawler starts with the base URL of the UIC Computer Science Department page (https://cs.uic.edu) and the traversal is restricted to be only within the UIC domain. The web traversal follows a Breadth First Search (BFS) strategy with the base URL as the only element. In this project I use a crawler.py file which uses function of Stop word processing, A Preprocessor function, A Page Rank function, A Web Parser Function, A Function to create JSON file which will then be used to run the Application.py function and a Web crawling Function to crawl through 3000 web page files. Listing the steps as to how the web crawler works:

- We pass the website domain https://www.cs.uic.edu/
- Which is then crawled for 300 web pages
- We call a PageRank function to generate PageRank JSON file
- To generate PageRank file, we use stop words and preprocessing to clean the data and get only processed data
- We even used a TF-IDF function to generate scores
- Also, for PageRank file creation we need to have to clean the HTML code and parse through it
- Web crawler program provides us with an Inverted Index file which is used to have all the webpages in the domain.

1.2 IR System

The IR system comprises of important modules which are Inverted Index and matching it to the users input query and to generate a comparison. We use 2 main operations that are listed below

1.2.1 Text Processing

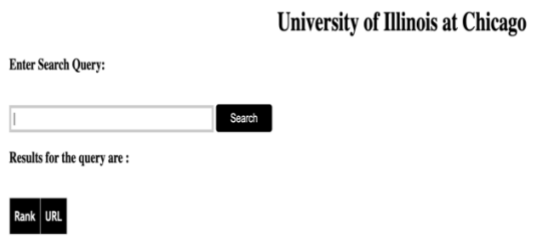
Processing is traditionally an important step for natural language processing and information retrieval tasks. It transforms text into a more digestible form so that machine learning algorithms can perform better. The text scraped from the documents are preprocessed before the building the vector space model. The preprocessing pipeline consists of removing punctuations, converting the texts to lower case, removing HTML tags, tokenization, stemming on the individual tokens, removing stop words. PorterStemmer is applied to stem all the words and removal of all the stop words. These processed words are stored in the form of a dictionary and similar processing is applied on the user query.

1.2.2 Vector Space Model

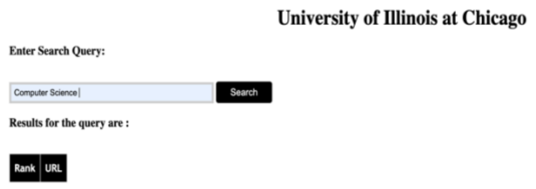
Building the inverted index table and computing web page lengths will take a considerable amount especially when we are doing this for 3000 web pages. To avoid any delay while running the query file, we build the inverted index table beforehand. The processed web page contents with respective URLs as keys in a page index table is then used for the construction of the Inverted Index Table. We initialize an inverted index table as an empty dictionary and iterate through all words in the page index table. For every word we encounter we update the counter of that word in its respective URL. In this way we can get the document frequency of every word in the vocabulary. Inverse Document frequency can be retrieved simply by getting the length of the document frequency dictionary corresponding to the word. The document lengths are also calculated beforehand as there is no query dependency. The inverted index table and the web page lengths are dumped into files using pickle library, so that they can loaded later during the query search

1.3 Query Search

The query module needed to be entered by the User in the text box when the webpage is opened. The query must be with use of common words such as UIC, Chicago, dean, department, professor, students etc. . We use InvertedIndex and PageRank JSON file to create and apply Cosine Similarity and based on that the web pages are indexed and returned with the best matched URL.


University of Illinois at Chicago
Enter Search Query:
Search
Results for the query are :
Rank
URL

With Search Query,

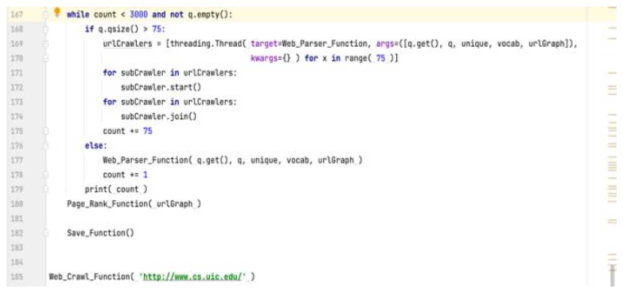

University of Illinois at Chicago
Enter Search Query:
Computer Science
Search
Results for the query are :
Rank
URL

User will enter the query to be searched for and when hitting search button will give you a search result.

2 Challenges

Some of the Challenges that was faced when I was building the UIC web search engine are described below:

- The web crawler can be more sophisticated in a way that it would work with websites of any domain, as I had to do some tweaks to make it work for UIC domain websites through trial and error.
- The crawling process took a lot of time in the project development and had to be run multiple times whenever there was a mistake or some change. To speed this up, distributed crawling could be used to give the job of requesting web pages to different threads
- A user-friendly GUI would be a nice addition to enhance search experience and better visibility of the results.
- A better scoring mechanism could be used like more combinations involving the PageRank algorithm could have been tried out
- Another challenge I faced was the technical aspect of the BFS implementation and how to resolve duplication. I used queue data structure to store the neighboring links from a given link and had a track of all the visited links to avoid duplicate visits or cyclic visits.
- The processing of 3000 webpages was taking a long time to compute and I had a hard time processing it.



```
while count < 3000 and not q.empty():
    if q.qsize() > 75:
        urlCrawlers = [threading.Thread( target=Web_Parser_Function, args=([q.get(), q, unique, vocab, urlGraph]),
                                         kwargs={} ) for x in range( 75 )]
        for subCrawler in urlCrawlers:
            subCrawler.start()
        for subCrawler in urlCrawlers:
            subCrawler.join()
        count += 75
    else:
        Web_Parser_Function( q.get(), q, unique, vocab, urlGraph )
        count += 1
    print( count )
Page_Rank_Function( urlGraph )

Save_Function()


Web_Crawl_Function( 'http://www.cs.uic.edu/' )
```

3 Weighing Scheme and Similarity Measure

3.1 Weighting Scheme

TF-IDF weighting scheme was used for the words in documents since most of the URLs in the UIC domain has lengthy documents with high use of common words such as UIC, Chicago, dean, department, professor, students etc. For that reason, it is better to use TF-IDF instead of Term Frequency alone. It is one of the most effective weighting schemes to calculate document similarity. It is effective as it considers the frequency of a term in the document as well as the impact of frequency of a term in all the documents. It is simple to calculate, and it is computationally cheap.

- TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).
- IDF(t) = log (Total no. of documents / Number of documents with term t in it).

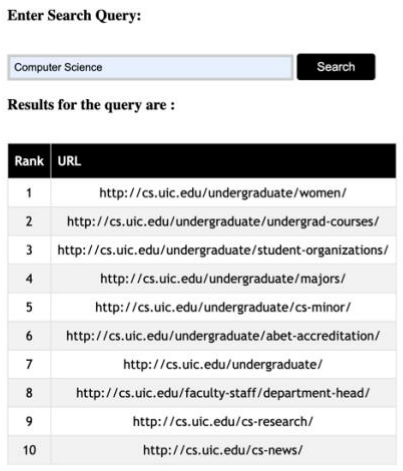
3.2 Similarity Measure

The similarity measure used for this project is the Cosine Similarity which is a popular and widely used similarity measure. It compares two documents well, even if the sizes of documents differ by a lot. Cosine similarity considers the frequency of the terms in documents and computes similarity by calculating the dot product. Another similarity measure tried as part of this project is the dice coefficient, which is not effective as it considers only the presence of terms and not their frequencies. The user enters the query term in the text box and clicks the search button. The query text is then processed, and a similarity score is generated from the cosine similarity. The similarity score is then computed between each of the corpus document. After the similarity scores are generated, the results are sorted in descending order based on the similarity scores. The links that have content that is a very close match to the given query are returned as the highest similarity score.

4 Evaluation

Some Examples are:

Computer Science



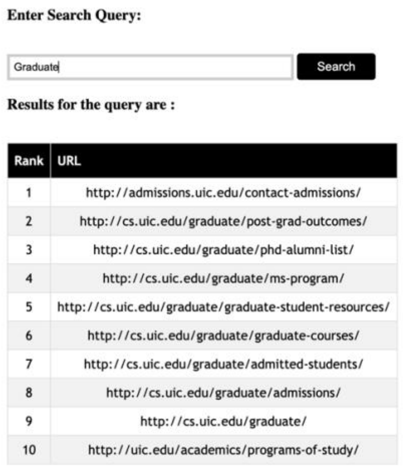
Enter Search Query:

Computer Science

Results for the query are :

Rank	URL
1	http://cs.uic.edu/undergraduate/women/
2	http://cs.uic.edu/undergraduate/undergrad-courses/
3	http://cs.uic.edu/undergraduate/student-organizations/
4	http://cs.uic.edu/undergraduate/majors/
5	http://cs.uic.edu/undergraduate/cs-minor/
6	http://cs.uic.edu/undergraduate/abet-accreditation/
7	http://cs.uic.edu/undergraduate/
8	http://cs.uic.edu/faculty-staff/department-head/
9	http://cs.uic.edu/cs-research/
10	http://cs.uic.edu/cs-news/

Graduate



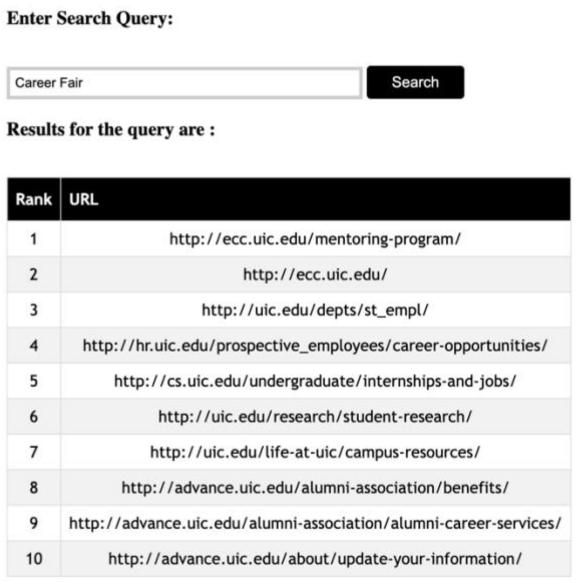
Enter Search Query:

Graduate

Results for the query are :

Rank	URL
1	http://admissions.uic.edu/contact-admissions/
2	http://cs.uic.edu/graduate/post-grad-outcomes/
3	http://cs.uic.edu/graduate/phd-alumni-list/
4	http://cs.uic.edu/graduate/ms-program/
5	http://cs.uic.edu/graduate/graduate-student-resources/
6	http://cs.uic.edu/graduate/graduate-courses/
7	http://cs.uic.edu/graduate/admitted-students/
8	http://cs.uic.edu/graduate/admissions/
9	http://cs.uic.edu/graduate/
10	http://uic.edu/academics/programs-of-study/

Career Fair



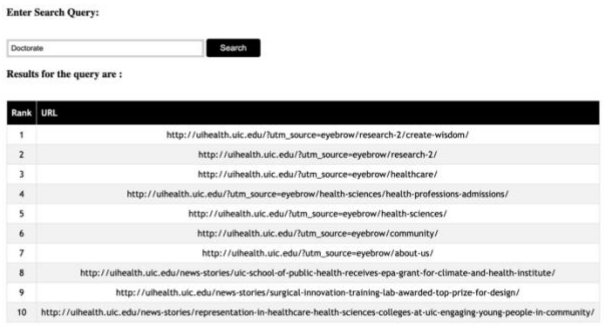
Enter Search Query:

Career Fair

Results for the query are :

Rank	URL
1	http://ecc.uic.edu/mentoring-program/
2	http://ecc.uic.edu/
3	http://uic.edu/depts/st_empl/
4	http://hr.uic.edu/prospective_employees/career-opportunities/
5	http://cs.uic.edu/undergraduate/internships-and-jobs/
6	http://uic.edu/research/student-research/
7	http://uic.edu/life-at-uic/campus-resources/
8	http://advance.uic.edu/alumni-association/benefits/
9	http://advance.uic.edu/alumni-association/alumni-career-services/
10	http://advance.uic.edu/about/update-your-information/

Doctorate



Enter Search Query:

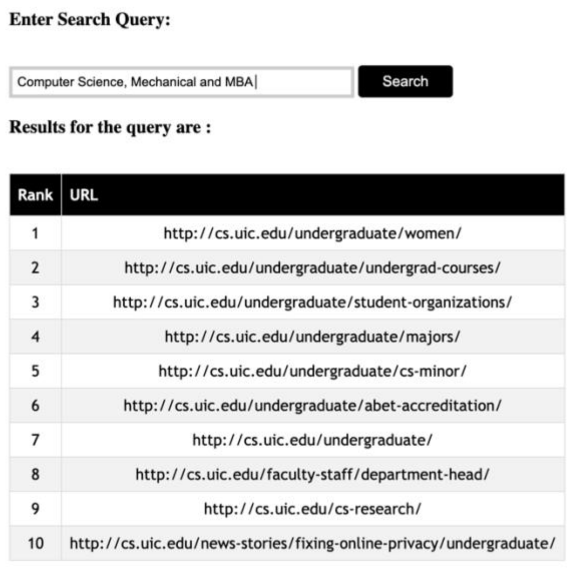
Doctorate

Results for the query are :

Rank	URL
1	http://uihealth.uic.edu/?utm_source=eyebrow/research-2/create-wisdom/
2	http://uihealth.uic.edu/?utm_source=eyebrow/research-2/
3	http://uihealth.uic.edu/?utm_source=eyebrow/healthcare/
4	http://uihealth.uic.edu/?utm_source=eyebrow/health-sciences/health-professions-admissions/
5	http://uihealth.uic.edu/?utm_source=eyebrow/health-sciences/
6	http://uihealth.uic.edu/?utm_source=eyebrow/community/
7	http://uihealth.uic.edu/?utm_source=eyebrow/about-us/
8	http://uihealth.uic.edu/news-stories/uic-school-of-public-health-receives-epa-grant-for-climate-and-health-institute/
9	http://uihealth.uic.edu/news-stories/surgical-innovation-training-lab-awarded-top-prize-for-design/
10	http://uihealth.uic.edu/news-stories/representation-in-healthcare-health-sciences-colleges-at-uic-engaging-young-people-in-community/

5 Error Analysis

The error is very less when we see the results for the user query, but the error is present in the software that has been developed. An example for this would be where there is a possibility of Duplication in the information, there is a chance that the retrieval has overlap coz of the similarity in the webpage and its domain name are retrieved. Also, small query size will result in better results as compared to a larger or a longer query size. Longer query size will give ambiguity as each word in the query can indicate to different aspects of things and therefore result in error. Exam if I search a query Computer Science, Mechanical and MBA, it will give me a result which will classify cosine similarity on each word of the query and give a result based on each word rather than the whole query.


Enter Search Query:

Computer Science, Mechanical and MBA

Results for the query are :

Rank	URL
1	http://cs.uic.edu/undergraduate/women/
2	http://cs.uic.edu/undergraduate/undergrad-courses/
3	http://cs.uic.edu/undergraduate/student-organizations/
4	http://cs.uic.edu/undergraduate/majors/
5	http://cs.uic.edu/undergraduate/cs-minor/
6	http://cs.uic.edu/undergraduate/abet-accreditation/
7	http://cs.uic.edu/undergraduate/
8	http://cs.uic.edu/faculty-staff/department-head/
9	http://cs.uic.edu/cs-research/
10	http://cs.uic.edu/news-stories/fixing-online-privacy/undergraduate/

We can see the result is based mostly on Computer Science and nothing on Mechanical and MBA

6 Related Work

The thing that helped me the most was the previous assignments which have most of the things implemented, there are multiple projects online which can be used to refer. Those projects will give you an idea on how to do it and that was what I did to implement this project. To build the IR-system and the PageRank Scores takes time depending on the system and how the code has been written. There are other implementations as well like Delta TF-IDF, Word2Vec using gensim, bag of words, BERT to implement the Weighting Scheme and each of them can be used to implement this project.

7 Future Work

The web crawler can be more sophisticated and work better in a way that it gives more accurate results and have a better implementation for query that are longer. A better implementation would be to have a bigger data set as 3000 files took a lot of time but still is a small set to implement, also improvement of the processing time will help in better execution of the whole system. I would also love to improve the GUI in general as I would love to implement a database to show previous searches and improve overall user experience, one more amazing implementation would be to have multiple threads working to make the whole experience much faster.