UIC Web Search Engine

Cosine Similarity approach

Anusha Voloju
Computer Science
University of Illinois at Chicago
Chicago, Illinois
avoloj2@uic.edu

ABSTRACT

This document is a report for the final project of CS 582 Information Retrieval at the University of Illinois at Chicago. The project included building a Web Search Engine for the UIC domain. The application is built including the components web crawling, webpage preprocessing, indexing and an interactive user interface.

1 Introduction

The user can select his preference to directly use the stored Inverted Index built from the crawled pages to do the search, instead of performing the crawling, preprocessing and indexing of the pages.

If the user wishes to use the text files created from the crawled pages to build the Inverted Index and then perform the search, by skipping the crawling, the user can specify his preference.

The user can also prefer to crawl the pages, perform preprocessing and the build the Inverted Index from the crawled pages to perform the search.

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

2 Web Crawling

The web crawling can be done by running Crawler.py file. The crawling starts from the seed page https://www.cs.uic.edu/ and crawls 3000 pages, which is specified in the code. All pages visited during crawling are listed to a queue by the crawler functionality. It uses a breadth-first search strategy where every page is picked from the queue, downloaded and parsed using BeautifulSoup library.

All the in-links are extracted and checked if they belong to the UIC domain, the relative URLs are completed using current page URL and checked if they are not already visited or already added to the queue and then are added to the queue if they do not belong to some excluded formats like .pdf, .jpg, .jpeg, .doc, .docx, etc. Try and Except blocks are used in the code, which helps to skip a page if there is any connection issue.

While crawling each page only https pages are extracted and HTTP pages are excluded. The slash at the end of each page is removed. For each URL being processed from the queue, the URL and the text content of the page are extracted and a text document is created for each corresponding page using BeautifulSoup.

Crawled HTML pages were uploaded into a dropbox folder and can be accessed using:

https://www.dropbox.com/sh/5fcb8l3ua42g42q/AACIB64ILpOcllQmuCpYEBhBa?dl=0

3 Text Preprocessing

The preprocessing of the crawled pages is done by InvertedIndex.py file. During the preprocessing, the text content of each page stored as text documents is processed to preprocess the text of each page. The text details are stored in a dictionary with URL of the page as key and the text content of the page as the value.

The text retrieved is preprocessed by removing punctuations, tokenizing the text using enumerate(text) to get words list, cleaning the words by removing the words whose length is less than three and by removing the stop words, stemming the words using PorterStemmer and normalizing the words by removing numbers, punctuations and by converting the words to lowercase.

In the preprocessing Inverted Index is also built. For building the inverted index, the words list formed by preprocessing of the text is used to forms lists of indexes of each word and map the indexes with the page URL respectively.

After building the inverted index, it is scanned to calculate the tf-idf of each term and uses these tf-idf values to calculate the document length of each text document corresponding to a page.

4 Retrieval

The retrieval of the pages for a given query is done by CosineSimilarity.py file. When the user requests result for a query, the query text is preprocessed using the same steps followed for processing each web page text content.

Once the query words are retrieved after preprocessing, the text documents containing the query words are retrieved from the inverted index and the cosine similarity between each retrieved document and the query is calculated.

Then the pages of the corresponding text documents are ranked in the decreasing order of the cosine similarity and the top-ranked pages are presented to the user.

5 User Interface

The user interface is the command prompt which is used to run the Python program. When the user runs the Main.py file, the user will be provided with three options to go forward.

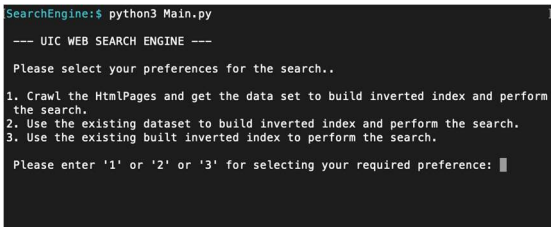

Figure 1: User interface when the program initates

The first option is to crawl the pages. It will create text document for each crawled page to build the dataset, perform text preprocessing and build the Inverted Index from the text files to perform the search operation.

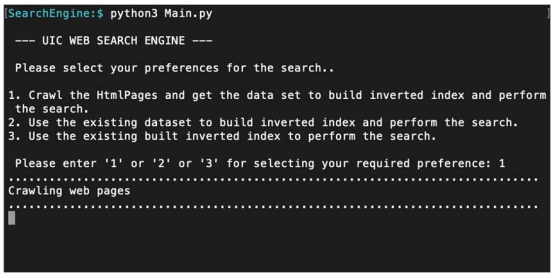

Figure 2: Crawling web pages

The second option is to build inverted index and perform the search operation. It will skip the web crawling process and use the existing dataset of crawled pages, perform text preprocessing and build the Inverted Index to perform the search operation.

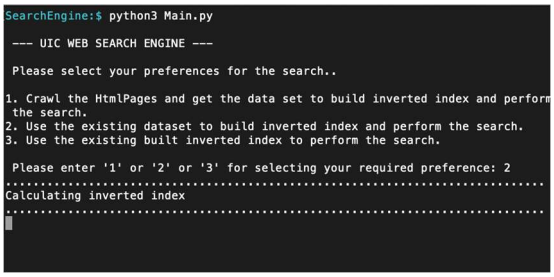

Figure 3: Calculating inverted index

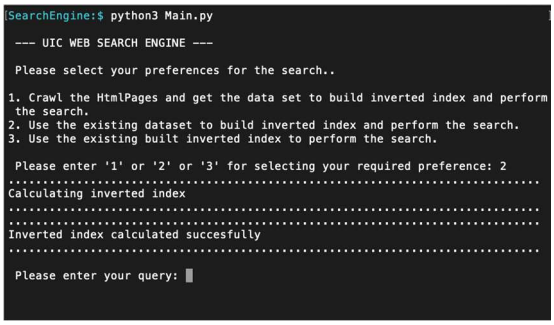

Figure 4: Program prompting user to enter a search string after calculating the inverted index

The third option is to use the Inverted Index and perform the search operation. It will skip the web crawling process, text preprocessing and inverted index calculation. It retrieves the pre-evaluated inverted index stored a CSV file and performs the search operation.

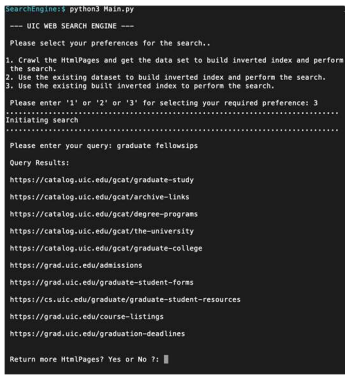

Figure 5: Search results based on query string input provided by the user

After selecting one of the options, the system asks the user to input a query string. After entering the query string and pressing enter, the user is provided with the top 10 results for the query and is asked if he wants more results.

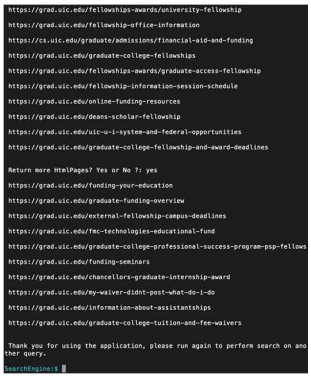

Figure 6: Additional search results are displayed based on user response

When the user enters Yes, 10 more results are displayed. Otherwise the program exits.

6 Challenges

The main challenges that I have experienced during the implementation of this project are:

1. I did not work on web crawling earlier, so I have spent a lot of time in understanding and implementing the crawling of web pages and extracting the text from HTML pages.
2. I had to crawl all the pages multiple times since I faced many errors and issues by crawling pages in different format like pdf files initially, then I have added a condition to exclude all the pages with different formats like .pdf, .doc, .jpg, .jpeg, etc.
3. The main challenge is to evaluate the query results. Since we do not have any gold standards or relevant documents given, and since the relevancy of the results to the query is subjective to the user, it was difficult to evaluate the results.

7 Measures Used

7.1 Weighing Scheme

The weighting scheme used in this project is TF-IDF weight of the terms in the text files since it takes into account the importance of the word in a particular document and also in the entire collection. Though TF might give better results in some collections, since TF-IDF is known to be one of the best weighing schemes for search engines, I have used the TF-IDF weighting scheme.

7.2 Similarity Measure

The similarity measure used in this project to measure the similarity between the query and the text files corresponding to the web pages is Cosine Similarity. Cosine Similarity is one of the best similarity measures used since it considers the document length and query length in addition to the matching terms in query and document. Though PageRank can be used to rank the pages, it only considers the authority of the page but not the content. However, PageRank can be integrated with Cosine Similarity to rank the pages

8 Evaluation

Below is the evaluation I have done for some sample queries by considering the precision at 10. (since I was retrieving top 10 results). Since we do not know the exact number of relevant results, I did not consider a recall.

Query: graduate fellowships

The first result was https://grad.uic.edu/fellowships-awards/university-fellowship and I guess all the results are related to fellowships. I would give a precision of 1.0 to this query.

Query: library policies

The first result was https://library.uic.edu/about/policies and I think all the results are related to the library, though not completely to its policies. I would still give a precision of 1.0 to this query.

Query: news today

The first result was https://today.uic.edu/contact/communicating-on-campus and since there is a page for UIC today, and all the results are from UIC today, I would give a precision of 1.0 to this query

Query: cs research and thesis

The first result was https://cs.uic.edu/graduate/ms-program/thesis and though most of the results are relevant to the query, a couple of results are the thesis courses in bio, so i would give a precision of 0.8 to this query.

Query: job opportunities

The first result was https://uic.edu/about/job-opportunities and most of the results are related to the query, but there are a couple of results one about UIC contacts and other admissions aid, so I would give a precision of 0.8 to this query.

Query: emergency

The first result was https://emergency.uic.edu and through all the results are related to the query, there are a couple of results for courses, so I would give a precision of 0.8 to this query.

Query: campuscare

The first result was https://campuscare.uic.edu/faqs and most of the results are related to the query, but there are a couple of results one for tuition fee and other for academic calendar, I would give a precision of 0.8 to this query.

Query: graduate tuition fee

The first result was https://admissions.uic.edu/graduate-professional/tuition-fees and all the results are related to the query, but there are results for registration policies, assistantships and sitemap, I would give a precision of 0.7 to this query.

To summarize, though there are a few irrelevant results for some queries, almost every query returned at least one relevant result

9 Discussion of results (Error analysis)

As per the evaluation, the search engine produced good results.

Using Cosine Similarity worked for the queries for which the data set contains relevant pages, based on the user's query the search engine could get the topic and retrieve the results, for example for the queries "library policies" and "news today", it retrieved library and UIC today pages respectively.

For the queries for which the data set does not contain many relevant pages, the search engine retrieved the course pages having the query words, for example for the query "health benefits" it retrieved public health course page. And in some cases, the pages in which the query word is present in header or footer are retrieved in the results

10 Future work

The existing search engine can be integrated with PageRank and also with relevance feedback to get more accurate results. By using PageRank, best authoritative pages can be retrieved even in the scenario where the data set does not contain many relevant pages for the query words.

REFERENCES

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