UIC Web Search Engine

CS 494 Final Project

Sai Anish Garapati
Computer Science Department
University of Illinois at Chicago
Chicago, Illinois
sgarap2@uic.edu

ABSTRACT

This document is a report for the final project of CS 494 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 include 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.

1. SOFTWARE DESCRIPTION

This project is developed in Python3 and all the functionalities are divided into modules to ease in integrating any future enhancements or scaling. Object Oriented programming is also used to improve the readability of the code. The Web crawler crawls the web pages and uses the IR system to preprocess the text and build an Inverted Index table from the documents. The Page Rank module computes the page rank scores of the web pages. The computed Inverted Index table and Page Rank score are stored in a pickle file to reduce the query processing time. The implementation of each module is described below.

1.1 Web Crawler

This 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. Double ended Queue(deque) from the collections library is used to store the URLs.

In each iteration of the traversal process we pop the leftmost element from the deque in FIFO manner and do the following checks to see if the web page pointed to by the URL can be accessed.

- Check if the URL belongs to the UIC domain.
- Check if the web page pointed to by the URL is a HTML page.
- Check if the URL is not already processed in the index table.
- Check the robots.txt file of the URL to see if the agent is allowed to access the page

Following all the above checks, we request for the page content using the requests library and format the text using BeautifulSoup library. Then all the URLs pointed to by the current page are extracted and the following operations are applied.

- Check if the URL is not a navigation link (#)
- Remove query parameters followed by ‘?’ to remove redundant URLs
- Remove ‘/’ at the end of the URL
- If the URL is relative, then append it to the base URL
- Change http links to https links to avoid any possible duplicates.

The web graph is also formed during the traversal using an adjacency list, which is used to compute page rank scores. The URLs are then appended at the end of the deque.

The content under the tags <script>, <meta>, <style>, <a> tags is removed so that only the main body of the page will be left, which is of actual interest to the user.The resultant page content is then preprocessed and stored in a dictionary index_table with its respective URL as the key.

This process of web traversal keeps going until the URL queue becomes empty or the number of processed pages in the page index table exceeds a certain limit which is 7500 web pages in this case.

1.2 IR System

This module takes care of the text preprocessing, building the inverted index table and making the actual comparison between the user's input query and all the web pages. The two main operations of this module are described below.

1.2.1 Text Preprocessing

Most of the preprocessing is already done before reaching this point. The content of the webpage is preprocessed before storing it in the page index table. Any SGML tags, numbers, punctuations and stop words are removed from the text and all text is converted to lowercase. PorterStemmer is applied on all the words to stem them and resultant stop words from stemming are removed again. Words with less than three characters are not removed in this case to avoid deletion of import abbreviations like 'CS' or 'MS' or any other course abbreviations which will be given as input by the user.

This processed list of words or tokens is then stored in a dictionary with its respective URL as the key. Similar text preprocessing is also applied on the input query during the search process.

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 7500 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

This module consists of a Command line interface for the user to give a query and retrieve a set of URLs related to the given query. We load the pickle files consisting of the inverted index table, web page lengths and pagerank scores and apply Cosine similarity on the query and all the web pages that are indexed to return the best matching URLs and display them to the user. The User can also give an option in the query to order the retrieved results using the page rank scores.

2. Challenges

- I have never done web crawling previously, so it took quite some time to understand the basics of it. It took quite a bit of time compared to the rest of the project, as we had to crawl 7500 pages and there were quite a few odd URLs which when encountered, had to be handled to prevent any edge cases.
- With par experience in Python, I found it challenging to design the software and split it into different modules according to their functionality and also incorporate Object Oriented programming to enhance readability of the code.
- When I first implemented the engine with basic crawler, I ran into a lot of issues like having a lot of javascript code terms in the inverted index table. Then I modified the crawler to remove the code under certain tags to retrieve only the body content of a website.
- Implementing the pagerank algorithm needed some thinking as we are applying the algorithm on a subset of the web pages as opposed to the entire web. So I had to make choices as to what pages I want to track the pagerank scores.
- Since I am preprocessing everything to avoid delays, I had to rerun the crawling part multiple times in order to try with other similarities. I could have designed the project to avoid this.

3. Weighting Scheme and Similarity measure

3.1 Weighting Scheme

The weighting scheme used in this case is the Term frequency-Inverse document frequency (TF-IDF). It is one of the most effective weighting schemes to calculate document similarity. It is effective as it takes into account the frequency of a term in the document as well as the impact of frequency of a term in all the documents.

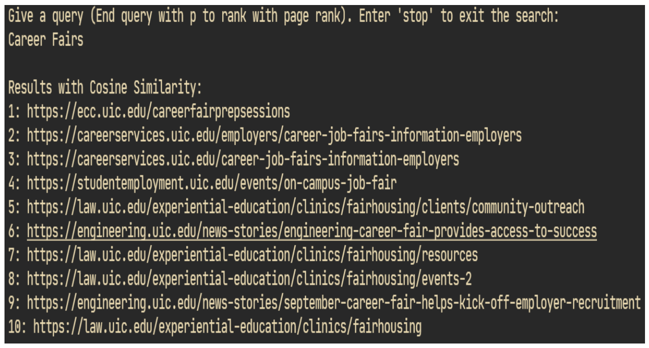
3.2 Similarity measure

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 takes into account 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 takes into account only the presence of terms and not their frequencies.

4. Manual Evaluation

Below are the results of a manual evaluation done on five custom queries using the search engine based on cosine similarity.

1. Query: Career Fairs

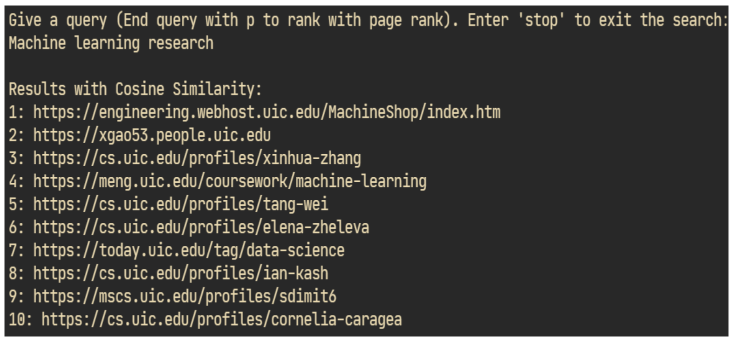


```
Give a query (End query with p to rank with page rank). Enter 'stop' to exit the search:
Career Fairs

Results with Cosine Similarity:
1: https://ecc.uic.edu/careerfairprepsessions
2: https://careerservices.uic.edu/employers/career-job-fairs-information-employers
3: https://careerservices.uic.edu/career-job-fairs-information-employers
4: https://studentemployment.uic.edu/events/on-campus-job-fair
5: https://law.uic.edu/experiential-education/clinics/fairhousing/clients/community-outreach
6: https://engineering.uic.edu/news-stories/engineering-career-fair-provides-access-to-success
7: https://law.uic.edu/experiential-education/clinics/fairhousing/resources
8: https://law.uic.edu/experiential-education/clinics/fairhousing/events-2
9: https://engineering.uic.edu/news-stories/september-career-fair-helps-kick-off-employer-recruitment
10: https://law.uic.edu/experiential-education/clinics/fairhousing
```

Quite a few retrieved queries are about some fair housing due to the use of the word Fairs which is not desirable for us. The precision for this query is 0.6

2. Query: Machine Learning research

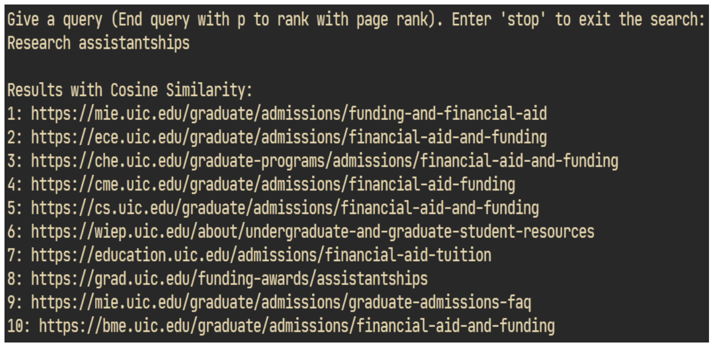


```
Give a query (End query with p to rank with page rank). Enter 'stop' to exit the search:
Machine learning research

Results with Cosine Similarity:
1: https://engineering.webhost.uic.edu/MachineShop/index.htm
2: https://xgao53.people.uic.edu
3: https://cs.uic.edu/profiles/xinhua-zhang
4: https://meng.uic.edu/coursework/machine-learning
5: https://cs.uic.edu/profiles/tang-wei
6: https://cs.uic.edu/profiles/elena-zheleva
7: https://today.uic.edu/tag/data-science
8: https://cs.uic.edu/profiles/ian-kash
9: https://mscs.uic.edu/profiles/sdimit6
10: https://cs.uic.edu/profiles/cornelia-caragea
```

Except for one query, remaining results are about the research done by professors in Machine learning. The precision for this query is 0.9

3. Query: Research assistantships

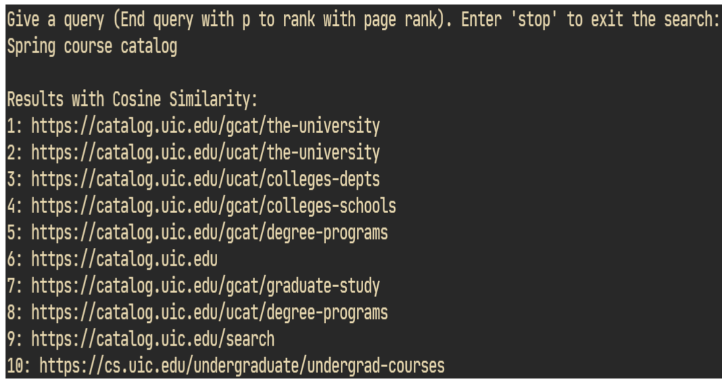


```
Give a query (End query with p to rank with page rank). Enter 'stop' to exit the search:
Research assistantships

Results with Cosine Similarity:
1: https://mie.uic.edu/graduate/admissions/funding-and-financial-aid
2: https://ece.uic.edu/graduate/admissions/financial-aid-and-funding
3: https://che.uic.edu/graduate-programs/admissions/financial-aid-and-funding
4: https://cme.uic.edu/graduate/admissions/financial-aid-funding
5: https://cs.uic.edu/graduate/admissions/financial-aid-and-funding
6: https://wiep.uic.edu/about/undergraduate-and-graduate-student-resources
7: https://education.uic.edu/admissions/financial-aid-tuition
8: https://grad.uic.edu/funding-awards/assistantships
9: https://mie.uic.edu/graduate/admissions/graduate-admissions-faq
10: https://bme.uic.edu/graduate/admissions/financial-aid-and-funding
```

All results are related to some form of assistantships or financial aid. The precision for this query is 1.0

4. Query: Spring course catalog

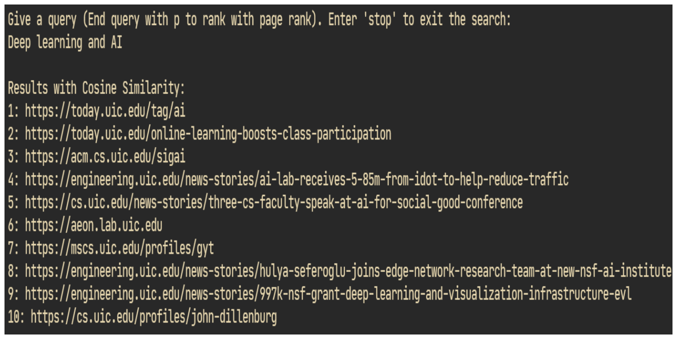


```
Give a query (End query with p to rank with page rank). Enter 'stop' to exit the search:
Spring course catalog

Results with Cosine Similarity:
1: https://catalog.uic.edu/gcat/the-university
2: https://catalog.uic.edu/ucat/the-university
3: https://catalog.uic.edu/ucat/colleges-depts
4: https://catalog.uic.edu/gcat/colleges-schools
5: https://catalog.uic.edu/gcat/degree-programs
6: https://catalog.uic.edu
7: https://catalog.uic.edu/gcat/graduate-study
8: https://catalog.uic.edu/ucat/degree-programs
9: https://catalog.uic.edu/search
10: https://cs.uic.edu/undergraduate/undergrad-courses
```

All the retrieved results are about the course offerings available. The precision for this query is 1.0

5. Query: Deep learning and AI



```
Give a query (End query with p to rank with page rank). Enter 'stop' to exit the search:
Deep learning and AI

Results with Cosine Similarity:
1: https://today.uic.edu/tag/ai
2: https://today.uic.edu/online-learning-boosts-class-participation
3: https://acm.cs.uic.edu/sigai
4: https://engineering.uic.edu/news-stories/ai-lab-receives-5-85m-from-idot-to-help-reduce-traffic
5: https://cs.uic.edu/news-stories/three-cs-faculty-speak-at-ai-for-social-good-conference
6: https://aeon.lab.uic.edu
7: https://mscs.uic.edu/profiles/gyt
8: https://engineering.uic.edu/news-stories/hulya-seferoglu-joins-edge-network-research-team-at-new-nsf-ai-institute
9: https://engineering.uic.edu/news-stories/997k-nsf-grant-deep-learning-and-visualization-infrastructure-evl
10: https://cs.uic.edu/profiles/john-dillenburg
```

All the retrieved results talk about some form of deep learning and AI research and professors except one result. The precision for this query is 0.9.

5. Error analysis

- The search engine did a decent job on the custom queries. Most of the results are actually the ones that we look for most of the time.
- From all the above queries, one thing I observed is that as we used TF-IDF and cosine similarity, more weight is being given to the term frequencies which could impact the results of queries containing common terms like machine, learning, research, etc.
- From query 5, we can see that query with abbreviation also returned good results related to AI because of the decision made not to drop words with less than two characters.
- Sometimes, though the results are related to the query, there is some dependency among the results, like both the web page and its domain page are both retrieved. This should be taken care of to avoid possible duplicate information.

6. Intelligent Component

6.1.1 PageRank Algorithm

I have also tried implementing the PageRank algorithm to see how it fares against the usual cosine similarity. During the web crawling itself, I build the web graph. I used adjacency list instead of adjacency matrix to build the web graph due to memory constraints. After building the web graph, I computed the PageRank scores for all web pages that are present in the page index table.

I first tried out a combination of cosine similarity and pagerank (0.5 * cosine + 0.5 * pagerank), but the pagerank scores are not in the same range as similarities. Even after normalizing, I didn't get decent results. Then I extracted results based on cosine similarity and then ordered them based on their pagerank scores which gave some decent results for certain queries which did not involve the exact wording as in the documents.

7. Related Work

In order to build the web crawler, I looked up online about various third party crawlers and finally wrote one from scratch using the requests library and referring to its documentation. To build the IR-system and the pagerank scores, I made use of the lectures and previous assignments. After having the idea to combine cosine similarity with pagerank, I checked on the internet and referred to a few questions and articles found in stackoverflow and other sources.

8. Considerations for Future work

- 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.
- To make a more sophisticated search engine, deep learning can be tried especially Long Term Short Memory (LSTMs) with attention in order to retrieve results for a query not just by word matching but also context matching.