Search Engine for UIC Domain

Rahul Romil Keswani
University of Illinois at Chicago
Chicago, Illinois, US
rromil2@uic.edu

ABSTRACT

This document is a project report for "Information Retrieval - CS581" course. This contains a description of the Search Engine developed, discussions about the challenges faced, the algorithms considered and the result analysis.

KEYWORDS

search engine, web scraping, page rank , cosine similarity, thesaurus based query expansion

1 INTRODUCTION

The project developed is a Search Engine for the UIC domain. It comprises of two modules, the uic spidey and the search engine itself. uic spidey is the crawler which begins crawling the domain in a Breadth First Search manner using cs.uic.edu/ as the starting link. The scraped pages are stored in json format locally. The search engine module consists the functions to preprocess the data, calculate the page rank scores for the documents and build a vector space model using TF-IDF. The query is promptednd the cosine similarity between the query and the documents are measured, and are ranked to display the results.

2 MAJOR COMPONENTS

This section talks about the major modules and the workflow of the search process.

2.1 Crawler and Scraper

UIC Spidey is a web crawler built using the SCRAPY library. It is a Breadth First Search crawler which starts at the cs.uic.edu site and crawls upto a depth of 5. It is restricted to the UIC domain [uic.edu/] and has a limit set to 4000 documents after which the crawler stops. The limit can be modified in the settings.py file of the project. The crawler scrapes the title, url, paragraph tags and the outlink of each url and saves it as a JSON file locally. The files are numbered numerically in their order of parsing. Below is the summary of the key parameters for the `uic spidey` module:

- `start link` : https://cs.uic.edu/
- `crawl domain` : uic.edu
- `crawl strategy` : Breadth First Search
- `depth` : 5
- `limit` : 5000

2.2 PageRank

PageRank is an algorithm used by Google Search to rank web pages in their search engine results. PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

A directed graph is built with each of the document link as a node in the graph. A directed edge is added from document A to document B if document A contains a link to document B. The Networkx library was used to build graph and insert edges. Below is the summary of the key parameters for the `PageRank` algorithm:

- `dampening factor` : 0.85
- `number of iterations` : 50

The score for a page can be calculated as,

$$\textbf{PR(a)} = \sum_{b \in B_a} \frac{PR(b)}{L(b)}$$

where,
'a' is the current document
'b' are the documents adjacent to 'a'

2.3 Preprocessor

The text scraped from the documents are preprocessed before the vector model is built. The preprocessing pipeline is as follows :

The text from the JSON files are read and then,

- `Remove Punctuations` : All punctuations from the text are removed
- `Tokenize` : Every sentence is the document is tokenized into a list of tokens.
- `Stemmer` : The individual tokens are stemmed and transformed into their root words using the Porter-Stemmer library.
- `Remove Stopwords` : The stopwords from the list of tokens are removed.
- `Discard Small Words` : Tokens of size lesser than 2 (if any), are removed

2.4 Vector Space Model

The preprocessed text is now used to build the vector space model. The weight model used here is the Term Frequency - Inverse Document Frequency (TF-IDF). This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. It is a weight measure of the product of the two terms mentioned below :

- `Term Frequency` : Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization: **TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).**

$$\mathrm{tf}_t = \frac{f_t}{\max(\mathrm{f}_t)}.$$

- `Inverse Document Frequency` : Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following: **IDF(t) = loge(Total number of documents / Number of documents with term t in it)**.

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}.$$

2.5 Query Processing

The query is prompted from the user and undergo the same preprocessing steps as the documents.

2.6 Thesaurus based Query Expansion

Query expansion is the process of reformulating a given query to improve retrieval performance in information retrieval operations, particularly in the context of query understanding. In the context of search engines, query expansion involves evaluating a user's input (what words were typed into the search query area, and sometimes other types of data) and expanding the search query to match additional documents.
The expansion technique applied here is to add the synonyms of the query terms. The synonyms are looked up on a thesaurus. NTLK's sysnet library was used to find the synonyms of query terms. These new terms also are simililary preprocessed and included to find the most relevant document.

2.7 Retrieval

The Tf-Idf for the query is calculated and the query here now is treated as a document. We use Cosine Similarity as a measure here to find the most relevant document to the query document. It is a metric used to determine how similar the documents are irrespective of their size.

$$\mathrm{CosSim}(\mathrm{d}_j, q) = \frac{\sum_{i=1}^{t} w_{ij}\, w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$

2.8 Ranking

A combination of the page rank score and the cosine similarity scores are used to rank the similar documents.

$$score_d = (0.25 * pagerankscore_d) + (0.75 * cosinesimilarityscore_d)$$

3 CHALLENGES

Some of the challenges faced during the development of the search engine are :

- `Urllib vs Scrapy` : Deciding between the type of crawler was the first of many challenges. Conceding to the speed of crawling, I decided to stick to scrapy because it crawled quite faster comparatively.
- `Deciding the depth of crawler` : The crawler crawled different pages for different depths. A higher depth meant having deeper links from one page itself, and a lower depth meant a widespread domain area. Owing to the limit of documents I had to parse, I decided to have a depth of 5. So that the crawler would cover a larger area than to go in detail in a smaller set.
- `Text to parse` : The html structure of the pages is not standard. After inspecting quite a few number of pages, I decided to use only the paragraph tags and Title as the text to parse.
- `Text to parse` : The html structure of the pages is not standard. After inspecting quite a few number of pages, I decided to use only the paragraph tags and Title as the text to parse.
- `Indexing the documents` : One major challenge was to decide where to index the contents of the link, and in what structure. I figured storing it locally would avoid network lags that could occur by storing it in some cloud database. And since there was different parts of the document being indexed, a dictionary data structure seemed a quite good choice, hence I decided on JSON format.
- `Extracting synonyms for Query expansion` : There are many libraries like PyDictionary, Wordnet etc available to extract synonyms of terms. Wordnet also returned many different Sysnets, and therefore

I decided to use only the word sysnet to extract the lemmas, therefore avoiding processing delays.

4 WEIGHTING SCHEMES, SIMILARITY MEASURES AND COMPARISONS

4.1 Weighting Scheme

The straight forward choice for the weighting scheme was Tf-Idf which has been discussed in Section 2.4

4.2 Similarity Measures

Jaccard Similarity and Cosine Similarity were the two measures evaluated during development. On analysis of a small subset of the corpus, Cosine Similarity fetched better results and hence it is used in this project.

4.3 Other Comparisons

Different combinations for combining the page rank and cosine similarity scores were evaluated. Eventually the equation in Section 2.7 was found to work well enough and fetch decent results.

5 EVALUATION

The queries 'computer science' and 'assistantship' are used here to compare results.

5.1 Scores combined as Geometric Mean

From Figure 1 and Figure 2, it is evident that this score calculation measure didn't give expected results.

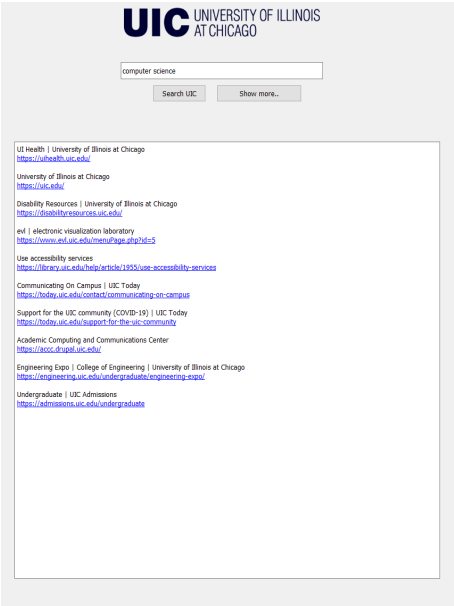

Figure 1: Query - "computer science" , Score - "Geometric Mean".

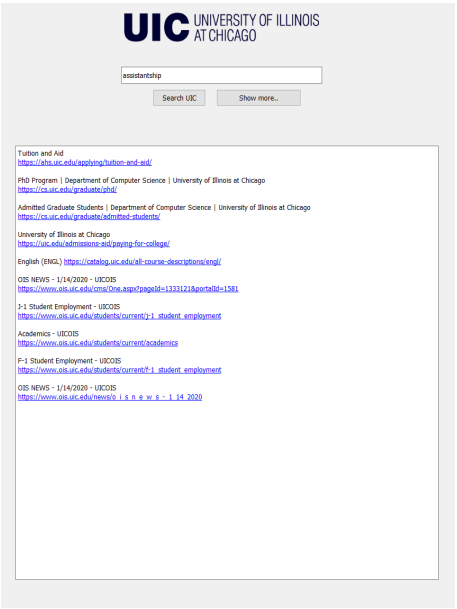

Figure 2: Query - "assistantship" , Score - "Geometric Mean".

5.2 High weightage to CosineSimarity score

From Figure 3 and Figure 4, we can see that these fetch results close enough to our interest.

5.3 Actual run for different queries

Figures 5 - 9 show the final results for some search queries. As we can see, the search does retrieve atleast a few appropriate results in addition to the extra ones.

6 RELATED WORK

The search engine was an extension of the assignments done over the course. Alternative approaches to the components used here could be using Rocchio algorithm for Relevance Feedback, Pseudo Relevance Feedback or query expansion using word embedding. Hyperlink Induced Topic Search (HITS) which makes use of Hubs and Authorities could be tried in place of PageRank.

7 LIMITATIONS ANND FUTURE WORK

Following are some of the limitations and possible improvements that could be implemented :

(1) Though thesaurus based query expansion using synonyms is implemented, it doesn't translate the semantic meaning of the query
(2) Typos in the query term may lead to no results being fetched. So an autocorrect feature could be handy.
(3) Based on the retrieved documents, suggestions of the top keywords and phrases could be made.

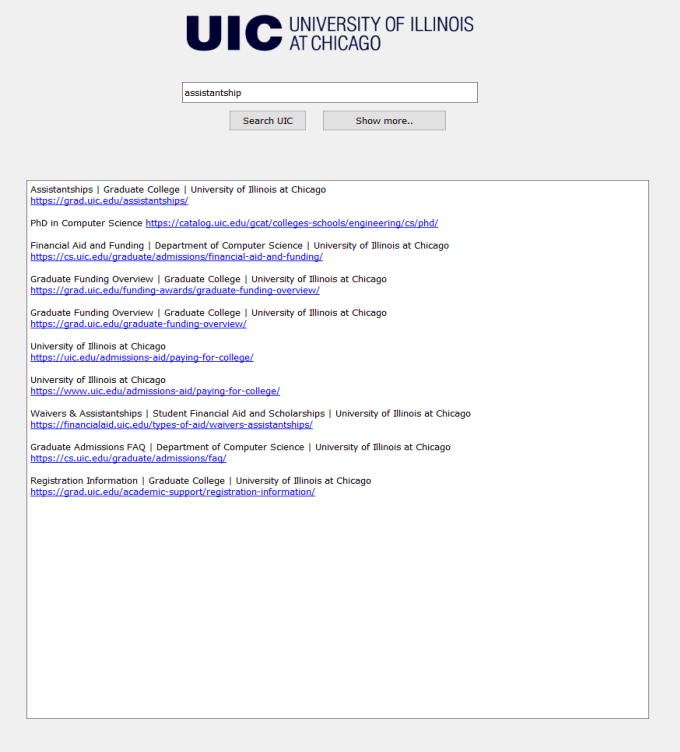

Figure 4: Query - "assistantship" , Score - "High CosSim Weightage Mean".

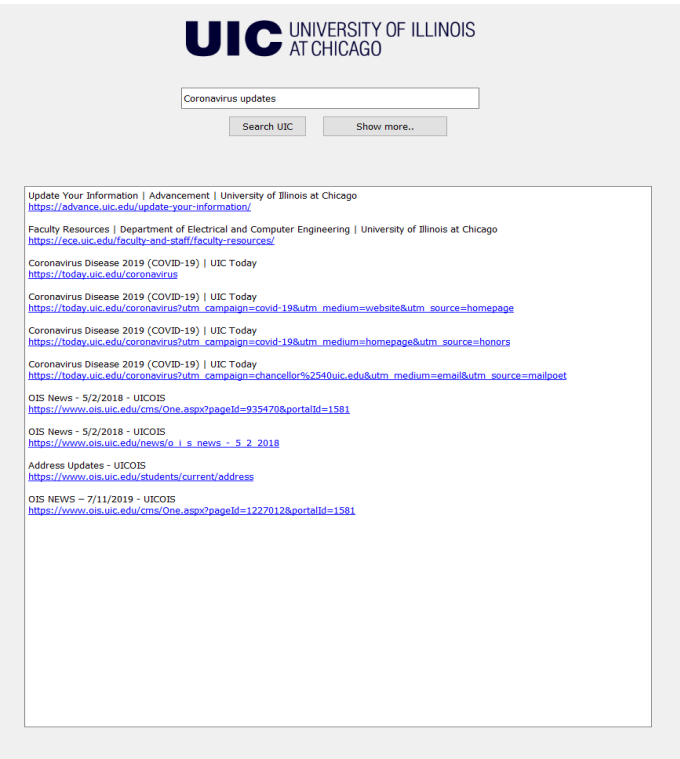

Figure 5: Query - "coronavirus updates"

Figure 6: Query - "dining services"

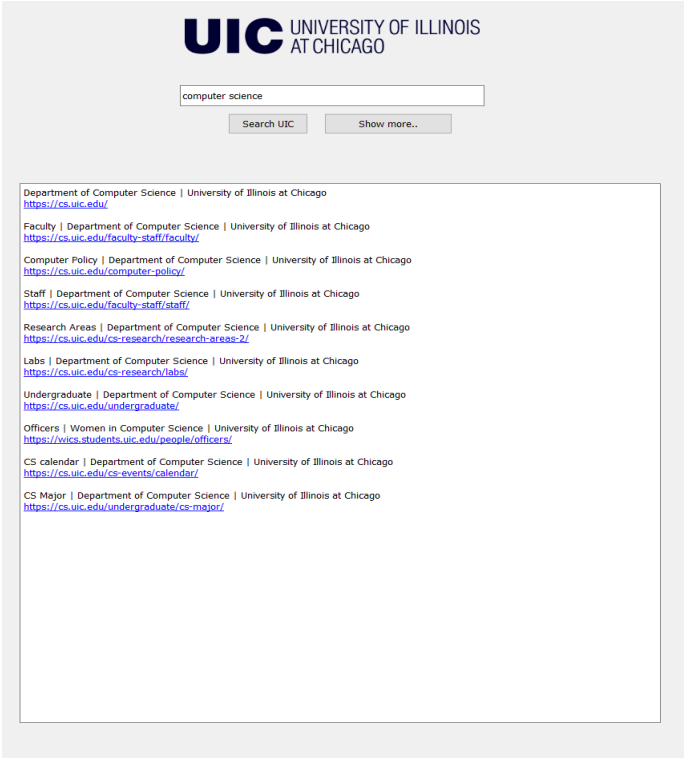

Figure 3: Query - "computer science" , Score - "High CosSim Weightage Mean".

Search Engine for UIC Domain

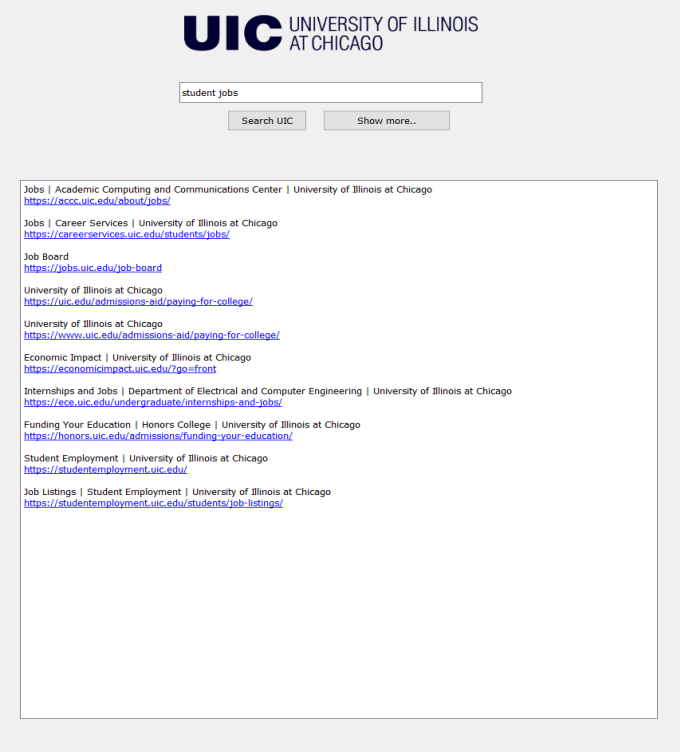

Figure 8: Query - "student jobs"

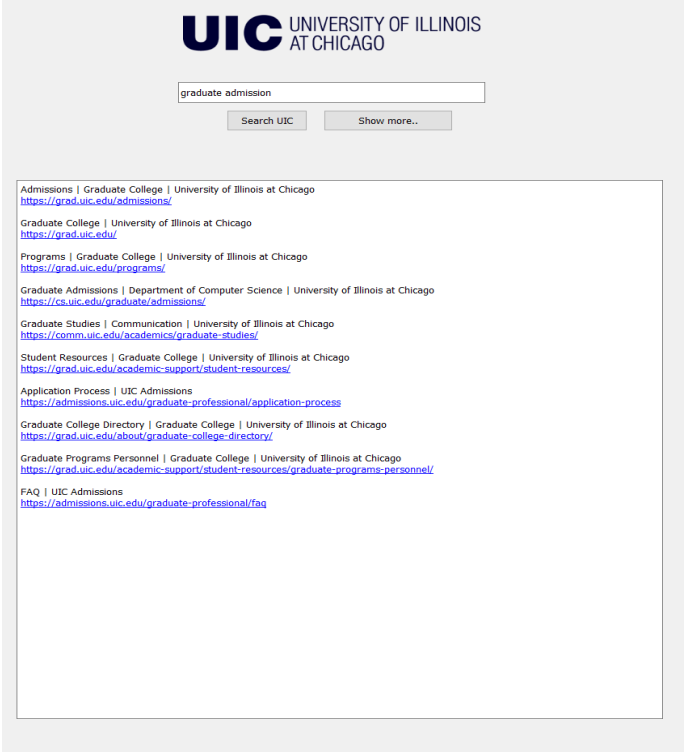

Figure 7: Query - "graduate admission"

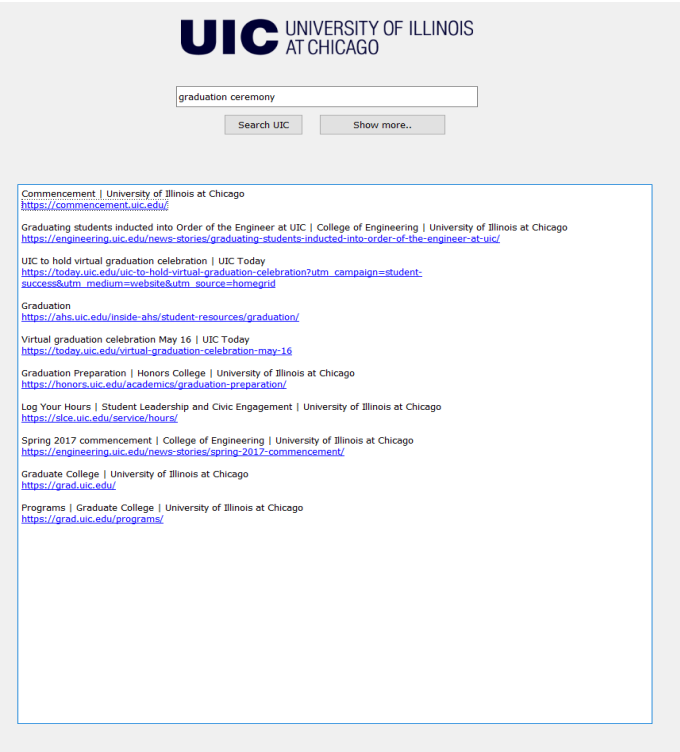

Figure 9: Query - "graduation ceremony"

REFERENCES

[1] TFIDF *http://www.tfidf.com/.*

[2] Wikipedia
en.wikipedia.org/wiki/PageRank

[3] Stanford -
nlp.stanford.edu/IR-book/pdf/06vect.pdf

[4] Cornelia Caragea
UIC Spring'20 Slides