Web Search Engine

Lokesh Roopkumar
Department of Computer Science
University of Illinois, Chicago
Chicago, IL, USA
lroopk2@uic.edu

***Abstract*—This document is my final project report for the course CS 582: Information Retrieval at the University of Illinois, Chicago during the Fall 2022 term. The goal of this project is to design and implement a search engine on the UIC domain that includes components for Web crawling, webpage processing, indexing, an intelligent aspect to the search engine, and a friendly user interface.**

I. Description

This project involves a search engine on the uic.edu domain that returns a set of relevant urls for a given search query. The search engine first crawls through a lot of webpages and indexes them. After the web crawling process ends, the heaps and heaps of unstructured data obtained are cleaned and structured to the best possible extent, after which the intelligent part of the IR system takes over. This module processes all the structured data and retrieves the ones that are most similar to the search query.

In essence, there are 4 main components to this search engine. They are as follows: Web Crawler, Data Processor, IR system, User Interface. Data flows through these components sequentially. We will explore them in further detail next.

A. Web Crawler

The web crawler as the name implies crawls 4000 pages on the web in the uic.edu domain and indexes 3807 pages. About 193 pages were not indexed due to some reasons like: could not establish a connection with the server, host failed to respond after multiple retries, the webpage contained less than 10 words. It took 1 hour and 36 minutes to crawl 4000 pages.

The crawler starts at https://cs.uic.edu, the official website of the Computer Science department at UIC. It extracts all the links on this page through the anchor tags present on this page. These links are unstructured, some of them are relative urls, some use http, some use https, some of them are mailto references, not all of them are links to html pages, some have query parameters and pound signs that point to the same page, etc.

Therefore, these links were first cleaned to remove question marks and hash marks. They were then normalized by converting http to https and ensuring all links always end with a forward slash at the end as that is the standard norm on the web. Next up all the relative urls are expanded.

Finally the links are checked to make sure they are proper links to webpages by confirming that they contain https:// and uic.edu as the crawler is restricted to just this domain. They are also checked if they point only to other webpages that are primarily html based by ignoring all of these extensions: .css, .js, .aspx, .pdf, .doc, .docx, .ppt, .pptx, .xls, .xlsx, .tar, .gz, .tgz, .zip, .png, .jpg, .jpeg, .gif, .svg, .ico, .mp4, .avi, .ics, /googlepublish, /ical, .xml. This is because the data processor can only process text in html tags.

The links are added to a queue and are traversed in a BFS fashion. The count of pages visited is tracked and the traversal stops either if the queue is empty or if it reaches the crawl limit which in this case is set to 4000 pages. The links are popped from the queue in accordance with FIFO, and a get request is made using the requests libary in Python to the url and the page is downloaded only if the get request is successsful.

The data from all the common tags that contain data like div, span, p etc. are extracted while ignoring script, meta, head, style and [document] tags. Any errors that are encountered while crawling are saved to the error logs text file in order to obtain better context on the pages that could not be visited/indexed. Upon manual inspection, the links in the error logs file were found to be unimportant and uncommon.

B. Data Processor

a) Data Cleaning: The html page content is downloaded for every url and then parsed using the Beautiful Soup library in Python. Out of the extracted text, all the standard cleaning and preprocessing steps are carried out in order. These involve, whitespace removal, punctuation removal, lowercase conversion, stemming, stopwords removal both before and after stemming. Words that have 1 character or lesser are removed. Words with 2 characters and more are retained as many keywords like AI, ML, CS are all 2 characters in length.

The tokens for each url is obtained and if a url has less than 10 tokens, it is removed from the index as a webpage that has less than 10 words is not significant. These tokens are then stored in a dictionary with a mapping between a url and its corresponding tokens. This dictionary is stashed away in a json file as it took a long time to crawl this data. This is because it doesn't make sense to re-crawl every time unless it is absolutely necessary. Json file was chosen over pickle so that the contents of the file can be viewed offline. Pickle file encodes it in binary, so the data cannot be evaluated to make sure things are the way they are supposed to be.

Exceptions are handled gracefully to ensure that any errors do not stop the crawler from crawling at any point. The input

query is also cleaned following the same steps mentioned above.

b) Data Structuring: After the cleaning is complete, the data is then structured using the Vector Space Model. The key idea behind this model is everything (documents, queries, terms) is a vector in a high-dimensional space. The geometry of space induces a similarity measure between documents. The documents are then ranked based on their similarity with the query.

To do this, first an inverted index is constructed for the urls and the query. A 2D dictionary is used for this purpose. The TF-IDF system is then applied. So the term frequencies are calculated for every term in every url, the document frequencies are also calculated for every word and they are all stored in the inverted index. The frequency of the most frequently occuring term is found and the TF component is normalized by dividing the term frequency by the frequency of the most occuring term. The DF component is also normalized by computing the IDF which is the log base 2 of the total number of urls divided by the DF of every word.

The weights of every term in each url is then calculated by multiplying the TF and the IDF. These are stored in a separate 2D dictionary mapped to urls for easy access. The TF-IDF weights for the query are also calculated by following the same steps.

C. IR system

The data now is well structured and ready for the IR system. All that is needed for computing the Cosine Similarity is readily available. Therefore, it is now time to compute the Cosine Similarity. Cosine similarity measures the cosine of the angle between two vectors. This is how it is represented in the form of an equation:

$$CosSim(d_j, q) = \frac{\langle d_j, q \rangle}{\|d_j\| \cdot \|q\|} = \frac{\sum_{i=1}^{t} w_{ij} w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$

As evident from the equation, the weights of the terms in the query and each url are multiplied and divided by the square root of the product of the vector lengths of the query and the url. The Cosine Similarity scores for each url are then stored in a dictionary mapped to its corresponding url. The values are indexed only if their cosine similarity score is greater than 0. This dictionary is then sorted by its values. This will rank all the urls in descending order of the Cosine Similarity scores. All the values during computation are rounded off to 8 decimal places for the highest accuracy.

D. User Interface

The UI takes as input a search query from the user. The top 10 ranked urls are returned in order as the search results. The user is then given an option as to whether they want to continue and see further search results or stop. All of the code has been written in Jupyter Notebook as it provides a nice environment to execute code and visualize output.

II. Challenges

- I faced the biggest challenge that all people face in the Data Engineering domain. Dealing with unstructured data. Unstructured data represents itself in so many forms, so it is very hard to account for all forms of unstructured data. One example of this is how I ignore extensions like .pdf, .mp4, .png as these pages need not be crawled. But it is next to impossible to account for every such extension. For example .ics which represents a calendar event, I didn't know to ignore that until I manually checked some pages and links myself in the uic.edu domain. So yeah it is not possible to do this on a larger scale manually.
- Web crawling takes a lot of time. So it is vital that the code that is in place is as efficent as possible. This involves making sure that the access time to most data structures is O(1) instead of O(n). It was a bit of a challenge to optimize code in as many areas as possible to account for this.
- The get request to many urls failed because of the lack of proper SSL certificate validation. Therefore I added verify = false as a parameter when making the get request to avoid requesting for SSL certificate verification. This made it possible to index pages like housing.uic.edu which previously failed due to SSL certificate issues.
- It was a challenge for sure to handle exceptions in such a way that it does not stop the web crawler at any point. This involved quite a bit of thought and experimentation using try and except blocks and figuring out the best way and place to put it in such a way that it always keeps the program running but just logs the errors to a file to be able to view them later.
- Parsing the html page was also a challenge as no library out there on the internet was able to do it properly. With the some libaries, it either did not obtain all the required text or obtained way too much text that included javascript code, and other unnecessary text. So I had to write this part myself and find a proper balance between getting all the required text and not getting anything that is not required by writing a filter function that filters the parsed data.
- Identifying relative urls and expanding them was also a bit of a mystery as I did it only assuming relative urls always start with a forward slash / always. I'm not quite sure if realtive urls start with other characters. I haven't had a chance to look at a huge dataset of relative urls to know for sure.
- Identifying all forms of data that need to be normalized was also a challenge. As I do not work with unstructured data frequently, I only did it to what I learned from class which is http to https and making sure all urls end with a forward slash. I'm sure that there are many others that I am not aware of.

III. Weighting Scheme and Similarity Measure

A. Weighting Scheme

The weighting scheme I have used in this project is the TF-IDF of words in webpages. Although it is a relatively simple and straightforward weighting scheme, over the years, it has proven to be one of the most effective and efficient weighting schemes. Every token is given a very fair weight that perfectly represents its presence relative to other tokens. It is also very fast, so the computation time is very less compared to any other weighting scheme out there.

B. Similarity Measure

The similarity measure I have used in this project to rank relevant webpages is Cosine Similarity. Cosine similarity takes into consideration the length of both the document and the query. Moreover, the tokens that are not there in either the document or the query do not affect the cosine similarity in any way. Usually, the query is relatively short, and therefore its vector is extremely sparse, which means cosine similarity scores can be calculated relatively quickly. The cosine similarity is quite beneficial because even if the two similar data objects are far apart by the Euclidean distance because of the size, they could still have a smaller angle between them. When plotted on a multi-dimensional space, the cosine similarity captures the orientation (the angle) of the data objects and not the magnitude.

C. Alternative Weighting Schemes

The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. The vectors are chosen carefully such that they capture the semantic and syntactic qualities of words. GloVe, coined from Global Vectors, is a model for distributed word representation. The model is an unsupervised learning algorithm for obtaining vector representations for words. This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity. Logarithmic Term Frequency scales the Term Frequency weightings logarithmically. This normalizes Term Frequency. Frequent words will still score a higher TF weighting, but the gap between the most frequent terms and less frequent ones will be narrowed. This can provide more accurate classification results when term frequency is not as important. Class Frequency alters the standard TF-IDF weightings by adding another variable to the equation. It modifies the weighting mechanism from TF-IDF to TF-IDF-CF. Class Frequency (CF) refers to the number of times a term (or feature) appears within a class of documents. Class Frequency can be a particularly effective way to mitigate outlier terms that only appear on a single trained example.

D. Alternative Similarity Measures

Manhattan distance is a metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates. It is the total sum of the difference between the x-coordinates and y-coordinates. The Euclidean distance between two points in either the plane or 3-dimensional space measures the length of a segment connecting the two points. The Pythagorean Theorem can be used to calculate the distance between two points. Minkowski distance is a generalisation of the Euclidean and Manhattan distances. The Jaccard index, also known as Intersection over Union and the Jaccard similarity coefficient is a statistic used for gauging the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets.

IV. Manual Evaluation

The search engine was evaluated for 5 sample queries and the top 10 ranked webpages retrieved for each of those queries are as follows:

1) Query: events

```
Enter a search query: events

Search Results:
1 https://ecc.uic.edu/find-local-events/
2 https://ecc.uic.edu/events-2/find-local-events/
3 https://cs.uic.edu/cs-events/
4 https://bme.uic.edu/bme-events/calendar/
5 https://today.uic.edu/events/categories/event-health-medicine/
6 https://today.uic.edu/events/categories/event-featured/
7 https://ecc.uic.edu/events-2/
8 https://bme.uic.edu/bme-events/upcoming-events/
9 https://bme.uic.edu/bme-events/
10 https://uofi.uic.edu/calendar/list/5492/
Do you want to continue (y/n): n
```

All webpages retrieved are directly related to the query. Precision = 1.0

2) Query: alumni

```
Enter a search query: alumni

Search Results:
1 https://advance.uic.edu/alumni-association/
2 https://engineering.uic.edu/about/alumni/
3 https://advance.uic.edu/alumni-association/uic-alumni-board/
4 https://advance.uic.edu/alumni-association/alumniexchange/
5 https://advance.uic.edu/alumni-association/alumni-career-services/
6 https://www.uic.edu/alumni/
7 https://uic.edu/alumni/
8 https://advance.uic.edu/alumni-association/get-involved/
9 https://advance.uic.edu/alumni-association/2021-uicaa-year-in-review/
10 https://ecc.uic.edu/students/alumni/
Do you want to continue (y/n): n
```

All webpages retrieved are directly related to the query. Precision = 1.0

3) Query: employment

```
Enter a search query: employment

Search Results:
1 https://ecc.uic.edu/employers/graduation-survey-results/
2 https://ecc.uic.edu/employers/
3 https://ecc.uic.edu/events-2/engineering-career-fair/
4 https://ecc.uic.edu/students/graduation-survey-results/
5 https://ecc.uic.edu/career-tools/managing-offers/
6 https://ecc.uic.edu/employers/services-to-employers/
7 https://careerservices.uic.edu/students/career-programs-events/
8 https://ecc.uic.edu/events-2/
9 https://www.uic.edu/depts/st_empl/
10 https://studentemployment.uic.edu/
Do you want to continue (y/n): n
```

All webpages retrieved are directly related to the query. Precision = 1.0

4) Query: admissions

```
Enter a search query: admissions

Search Results:
1 https://admissions.uic.edu/contact-admissions/
2 https://admissions.uic.edu/contact/
3 https://law.uic.edu/admission/staff/
4 https://admissions.uic.edu/undergraduate/request-information/
5 https://grad.uic.edu/admissions/deadlines/
6 https://admissions.uic.edu/graduate-professional/contact-graduate-and-professional-admissions/
7 https://admissions.uic.edu/
8 https://grad.uic.edu/admissions/
9 https://admissions.uic.edu/undergraduate/contact-undergraduate-admissions/
10 https://law.uic.edu/admission/
Do you want to continue (y/n): n
```

All webpages retrieved are directly related to the query. Precision = 1.0

5) Query: covid

```
Enter a search query: covid

Search Results:
1 https://today.uic.edu/coronavirus/
2 https://ehso.uic.edu/covid19/
3 https://today.uic.edu/uic-fall-covid-19-guidance/
4 https://today.uic.edu/covid-19-vaccination-guidelines/
5 https://law.uic.edu/coronavirus/
6 https://dos.uic.edu/community-standards/uic-covid-19-guidance/
7 https://today.uic.edu/frequently-asked-questions/
8 https://research.uic.edu/covid-19ovcr/
9 https://today.uic.edu/updates-to-covid-19-university-sponsored-travel-approvals/
10 https://vcha.uic.edu/about/vcha-initiatives/covid-19-university-travel-request-authorization-form/
Do you want to continue (y/n): n
```

All webpages retrieved are directly related to the query. Precision = 1.0

V. Results

As evident from the results of the sample queries from the manual evaluation section, the precision for the queries is quite good. Given that this search engine was built by one person completely from scratch in a relatively short period of time, I think this is a very good start/attempt at a realistic search engine.

There are many for which it did not produce the best results. For example, if I search for 'faculty', pages that have 'professors' do not show up. This can be improved by using query expansion.

Another issue to note was if I search for 'covid testing', pages that contain information about tests/exams that students take in computer science courses are also retrieved. These are irrelevant to the meaning conveyed by the search query. But since the word 'testing' gets stemmed to 'test' and the word 'test' appears in course webpages as well, the system retrieves it. However, these results appear towards the end of the search results.

I also noticed cases where if I say search for 'graduation', I'm expecting pages related to steps required to graduate or graduation ceremony details etc. But most pages contained details about grad degree programs, grad faculty, grad courses etc. This is because the term 'graduation' got stemmed to grad and it returned all results related to grad and couldn't differentiate between grad and graduation.

It looks like stemming does affect the quality and precision sometimes. Not very sure if there is a workaround for this that is a better alternative. Some words in the English language are ambiguous, they can be interpreted differently, like orange the fruit vs orange the color, apple the company vs apple the fruit, information retrieval the course vs information retrieval meaning retrieve any information. So it still remains an open challenge to tackle all these issues. Active NLP and ML research efforts are being put in to improve these further.

VI. Related Work

I researched several articles, videos online in order to understand web crawling and its nuances. I spent a lot of time trying to understand why the get request fails for some urls and the reason behind it. I also tried to see if there is anything I can do to increase the success rate of these get requests,

I went through several resources online to know what all alternatives are there to the weighting scheme and similarity scheme used in this project. These results have been summarized in the section Weighting Scheme and Similarity Measure.

I did research about how to handle exceptions without causing the code to crash. I feel like this is a very important skill and it helped quite a lot in this project. Even the error logs file provided a lot of information about urls that could not be indexed which opened up avenues for further improvements.

All the lectures and lecture slides were also definitely very helpful. I referred to everything about TF-IDF, Cosine Similarity, Web Crawling and many other concepts that were used in this project from the lecture slides.

VII. Future Work

There is a lot of future work that can be done to imporve this search engine. This is the section that I was most excited to write.

- Query expansion can be done to add synonyms of words also to the query. This can have a big impact on the search results. But on the other hand this will make the query very big as there are many synonyms for every word in the English language. There is a trade-off between what synonyms to choose without affecting the meaning and context of the word in play.
- Page ranking can be implemented on the uic.edu domain for all the urls and the top say 8000 urls can then be indexed. This will significantly improve the results. This is because right now I am crawling 4000 pages at random, most of these pages may not be significant at all compared to a lot others.
- The meaning and context of the query terms can be better accounted for by using more advanced approaches to return only the relevant links. Deep Learning using Long Term Short Memory (LSTMs) can be tried in order to achieve better results by context matching.
- Multithreading or distributed cloud computing approaches can be used to crawl much faster parallelly. However, this will make the code much more complicated and may even make it more harder to debug. But this is definitely the way forward.

- Running this crawler on other domains and seeing the results it produces will also help understand other areas of improvement for the future.