A Web Search Engine

Srikanth Maganti
University of Illinois at Chicago
smagan20@uic.edu

ABSTRACT

Search engine basically index hundreds of millions of web pages with comparable number of distinct terms. These search engines mainly work on the principle of creating indexes, or large databases of web sites mostly based on keywords, key-phrases, titles, and the text in the pages. In this project, I have built a web search engine for UIC domain from scratch. This project mainly consists of web crawling, preprocessing of crawled pages, adding the intelligent component , and finally adding a Graphical User Interface.I have implemented cosine similarity on top of TF-IDF method to find the relevant pages. The results are very accurate and it provided the scope for future work

Keywords

Information Retrieval, Search Engine

1. INTRODUCTION

A search engine is built from scratch for UIC domain. Web scraping of all the web pages is done by using BFS algorithm. All the web pages that are scraped are stored in a local directory. The scraped web pages are preprocessed by removing all the metadata or unnecessary content. The threshold value for number of web pages to be scraped is 3000 If we opt to increase this value we can increase. After scrapping all the web pages, each page is considered as a document and in each document for each word TF-IDF value is calculated. All these TF-IDF values are stored for further use. The main component of my project is to calculate cosine similarity. In order to compute cosine similarity I will be using 2 dictionaries in the form of pickles in main file. Details about these methods will be discussed in further sections.

2. CRAWLER

The script that does the crawling is "web-crawler.py". It works on the concept of Breadth First Search. Initially it takes root node as UIC-CS domain i.e "www.cs.uic.edu" as it keeps iterating through all the web pages. If there are any web pages which are already visited those will be marked and won't be visited again and adds all the unvisited links to the queue. The HTML content of each link is extracted through "Beautiful Soup" library, further links which are residing in the content of the base link will be added to the queue after it met necessary conditions.

Few links with extensions ('.avi','.ppt','.gz','.zip','.tar', '.tgz','.docx','.ico','.css', '.js','.jpg','jpeg','.png','.gif','.pdf', '.doc','JPG','.mp4','.svg') are not considered because it takes more time to extract content.

The threshold value for maximum number of pages to be crawled is set to be 3000. If the count of total number of pages is 3000 or queue gets empty the web crawler program gets terminated.

During the crawling, all the links were stored in a dictionary as values and its corresponding count of page number as key. This dictionary is dumped using pickle for future use

3. PREPROCESSING

The script is implemented for preprocessing and named it as "preprocessing.py". Firstly it extracts all the content of the <body> and <title> tags of each downloaded page which is stored in the local directory. In the second method all the necessary preprocessing steps like stemming using porter stemmer, stop word removal by predefining stop words, and removing numbers, special characters, and duplicates will be done. During the process of extraction, raw term frequency dictionary is created and to keep note of all the tokens in a particular document a dictionary is created with tokens as values and document numbers as keys. These dictionaries will be stored as pickle's for further use.

4. TFIDF INDEXING

I implemented an inverted index from the above mentioned dictionaries. They are document number and term frequency dictionary which were mentioned in the previous section. In each document for each word TF-IDF is calculated by implementing different code for term frequency, inverse document frequency calculation. After calculating term frequency it is normalized with total number of words in a document to eliminate bias for shorter and longer documents. Finally multiplying both to get TF-IDF value. This TF-IDF vector model is stored as another pickle.

5. GRAPHICAL USER INTERFACE

For user interface I have used python library EASYGUI. Basically this Graphical User Interface includes necessary basic functionalities for displaying results.

Three steps were implemented in this total functionality. First stage of this function questions us to chose one among the choice that is either search query or exit search engine. All the options mentioned in this stage can be seen in figure 1

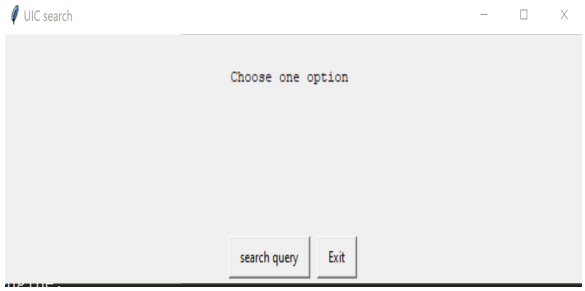

Figure 1: Stage 1 of GUI

In the second step an option to choose whether to confirm with the query or recheck the query. Choices can be seen in figure 2

Figure 2: Stage 2 of GUI

In the next step you can see the results and when you want to check for more results there you can see 20 top results of that particular query if you click on "ok". Details can be seen in figure 3

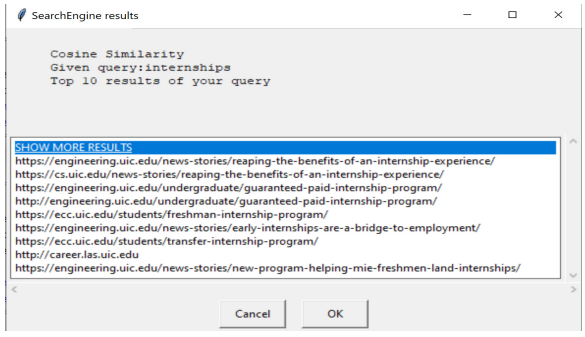

Figure 3: Stage 3 of GUI

6. WEIGHTING AND SIMILARITY

6.1 Weighting Scheme

The weighting scheme that I have used is TF-IDF[**?**] of words. During the initial stage of the project, I have planned to implement just term frequency and as the term frequency alone is not a good method because there will be high usage of few words like UIC, students, department etc. so I have used TF-IDF as a weighting scheme.TF-IDF is calculated by using below formula

$$w_{ij} = tf_{ij} \times \log_2 \frac{N}{n}, \text{ where}$$

w_{ij} = weight of term T_j in document D_i
tf_{ij} = frequency of term T_j in document D_i
N = number of documents in collection
n = number of documents where T_j occurs at least once

6.2 Similarity measure

Cosine Similarity is used for similarity measure between each document and query.During the initial phase I have implemented inner product between each document and query. In later stages, I have calculated document length and query length to calculate cosine similarity value for that word in that particular document. After calculating all the similarity measures between each document and query. They are sorted according to ranking and have shown top 10 results. If user wants to see more results first 20 results were shown. Cosine similarity is calculated by using below formula

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

7. ALTERNATIVES

Instead of using cosine similarity, we can actually take inner product into consideration but this approach doesn't take length of query and document into consideration which leads to bad results.

My assumption is that integrating TF-IDF scores with pagerank which can lead to better results. For each of the word in each document TF-IDF is calculated by making use of these values the scores that are calculated by in-links and out-links of that page. The page with more in-links will be getting higher probability when all these scores are integrated with TF-IDF values and giving priority for query word can be increased in accuracy.

8. EVALUATION AND RESULTS

The cosine similarity method outputs the list of pages based on similarity score between the page and the query. The below shown figures outputs the top 10 results for the corresponding query

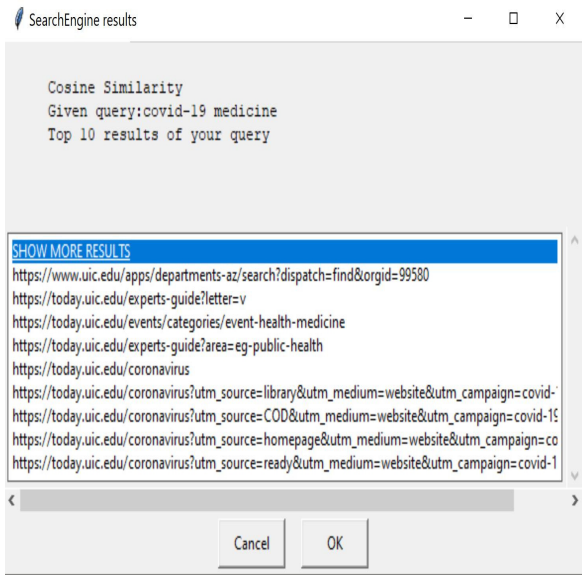

Figure 4: TF-IDF search on "covid-19 medicine"

In the above result it clearly shows that many of the links are very closely related to covid-19 medicine. only few links like tends to go for uic departments and uic guide experts.

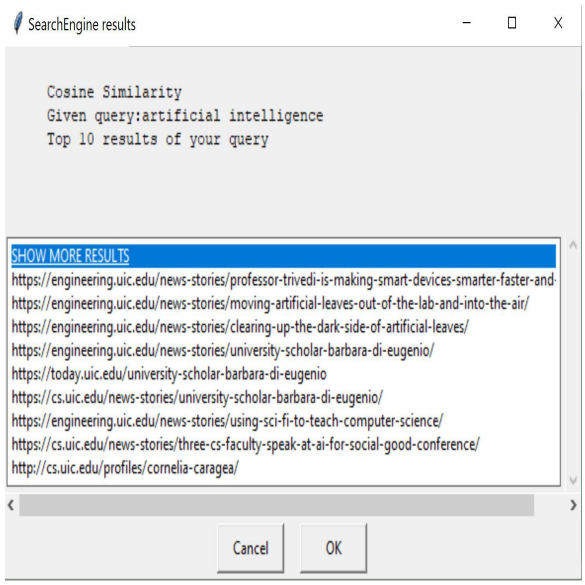

Figure 5: TF-IDF search on "Artificial Intelligence"

In the above result almost all of the results are very relevant for artificial intelligence only 2 results were out of context. It takes artificial as one key word and checks the results for artificial instead of cumulative query artificial intelligence

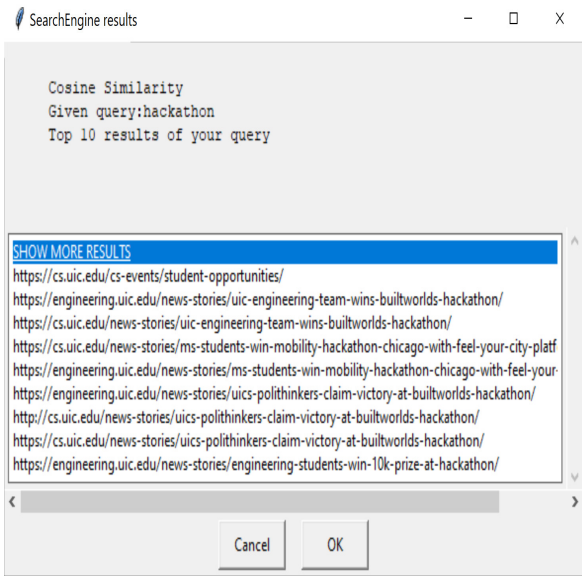

Figure 6: TF-IDF search on "hackathon"

In the above figure all the links were very much related to the hackathon. In the first link when I have checked the website https://cs.uic.edu/cs-events/student-opportunities/ I found that hackathons place a major role in that page and it stands on top when compared with remaining pages during manual evaluation

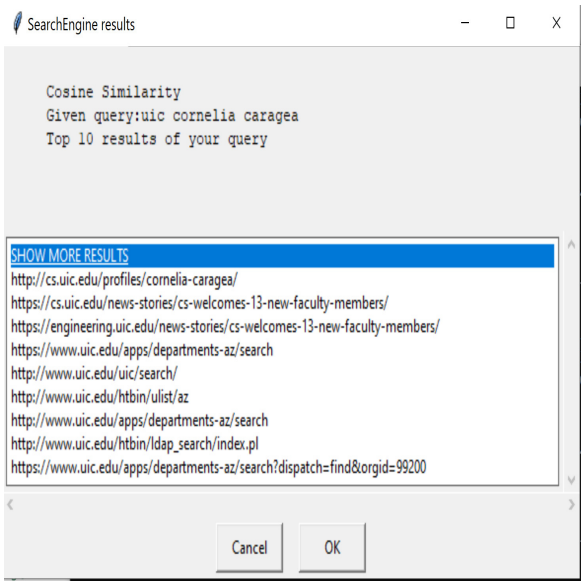

Figure 7: TF-IDF search on "UIC cornelia caragea"

In the above figure first 3 results are very accurate and most of the links after that are related to just uic word not for the whole query. Even in google search engine when I have given query UIC cornelia caragea. It shows the first 3 results same as what I have retrieved from my search engine

In the above figure the first link https://oef.uic.edu directs

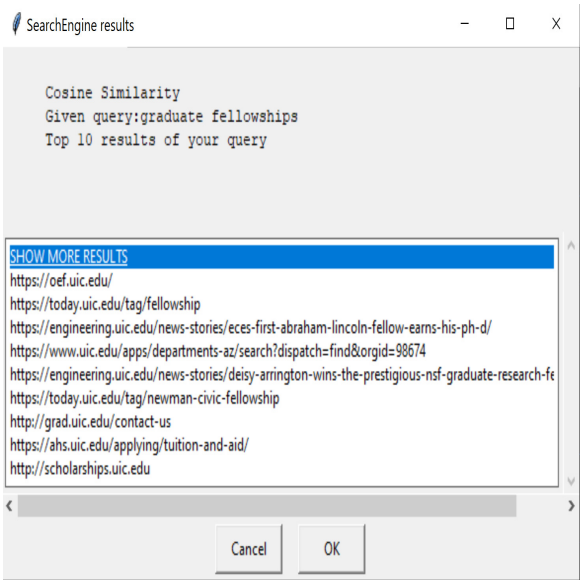

Figure 8: TF-IDF search on "graduate fellowships"

to office of external fellowships for uic which clearly shows that search engine works very well and I have observed in few links it considers just "fellowships" as entire query and gives the results

9. DISCUSSION OF RESULTS

All the results are evaluated based on precision value. Formula for precision is mentioned below

$$\text{Precision} = \frac{\text{Number of relevant item retrieved}}{\text{Total number of items retrieved}} \times 100$$

Table 1: Precision for different queries.

No	Query	Precision
1	covid-19 medicine	0.8
2	artificial intelligence	0.9
3	hackathon	1.0
4	uic cornelia caragea	0.3
5	graduate fellowships	0.8

All the results in the above table clearly shows that how good the model is and almost all the precision values in the table shows the high value except one value and by seeing the result it is clearly evident that term frequency plays a major role

10. ERROR ANALYSIS

In the results, I found out a small redundancy between few links repeating starting with http,https and if there is a case such that only few results like less than 10 are related in top 10 links but search engine shows 10 results. so, what exactly happening in the above mentioned case is elaborated in the below screenshots

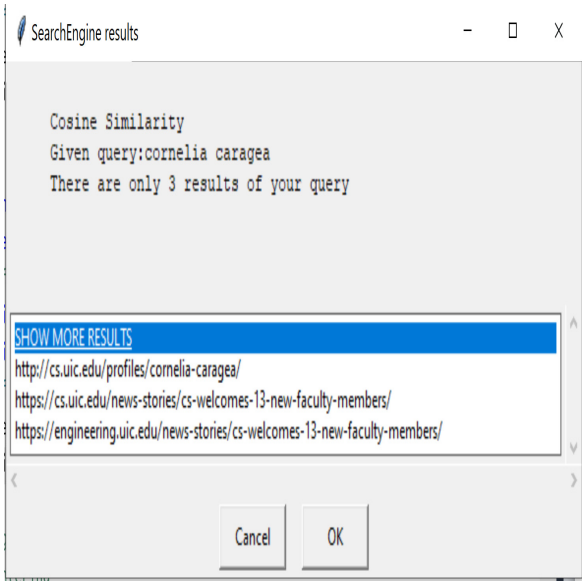

Figure 9: TF-IDF search on "Cornelia Caragea"

when you compare figure 9 and figure 7, figure 9 shows only 3 results which are highly related whereas when we have taken figure 9 into account it shows 10 results all the 7 results apart from first 3 are related to "uic" not for "cornelia caragea"

11. CHALLENGES

The main challenges that I have experienced during the implementation of this project are: As this is the first time working on web crawler, it went very easy for extracting all the HTML content but the toughest part is web scraped pages had lot of black listed format such as ('.avi','.ppt','.gz' ,'.zip','.tar','.tgz','.docx','.ico','.css', '.js','.jpg','jpeg','.png', '.gif','.pdf','.doc','JPG','.mp4','.svg') in order to figure out these formats.it took lot of time because crawler takes lot of time to crawl all the web pages.

During the implementation of GUI, I struggled a lot with setting up "show more results" as the default options for easyGUI is either "OK" or "Cancel".In order to integrate this feature into GUI. it took lot of time to make it work.

I am in the final stages of implementing query dependent page rank algorithm. It is explained in the below paragraphs

Through Page Rank: The Random Surfer and HITS there is problem with topic drift. pages which do have most inlinks tend to dominate than compared with other links.In order to overcome this problem, I prefer choosing query tokens as key ingredients for the pagerank

It simultaneously checks content of the pages query tokens for which the surfer is looking for. The probability distribution over pages is similar to Page Rank: The Random Surfer just with few minor changes below is the formula

used to compute Query Dependent Page Rank

$$P_q(j) = (1-\beta)P'_q(j) + \beta \sum_{i \in B_j} P_q(i)P_q(i \to j)$$

when we elaborate the above terms

$$P'_q(j) = \frac{R_q(j)}{\sum_{k \in W} R_q(k)} \qquad P_q(i \to j) = \frac{R_q(j)}{\sum_{k \in F_i} R_q(k)}$$

Both of the above terms are dependent on R_q(j) by calculating how relevant page j to the query we can assign the value. The first term is calculated by current document tfidf value normalizing with sum of all the tfidf values of that term in inlinks. Later term is calculated by multiplying probability of the current document for corresponding term in outlinks with probability of the inlink document value

I have implemented almost 90 percent of coding part and the results can be seen in the below screenshots.

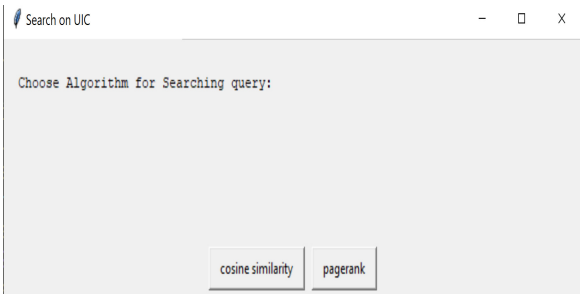

Figure 10: implementation of GUI for pagerank

11.1 Sample screenshots of pagerank

As due to some kind of bug in the code. The results are little oscillating than what I have expected and due to the current deadline I am not integrating this code with the original code.

Through by making use of TF-IDF indexing which I have stored in the form of pickle is used for calculating query dependent page rank values. All these values will be stored in dictionary format as pickles.

During the crawling process all out-links will be appended for the dictionary of that particular document where as in-links will be calculated by making use of current documents and out-links.

In figure 11, all of the results were getting redirected to uic health because many of the links contribute towards ”medicine” word in ”covid-19 medicine” whereas in figure 12, all of the results are very related except first few results and I ran the algorithm for the remaining queries as well which are used for cosine similarity evaluation and I get to know that if there are any common words like UIC,department,students etc all these words tend to dominate other important words in query.

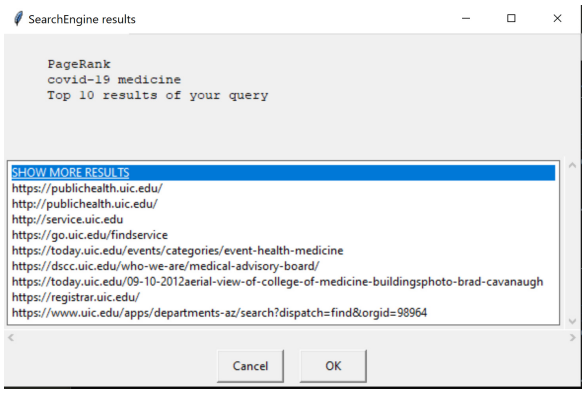

Figure 11: pagerank search on ”covid-19 medicine”

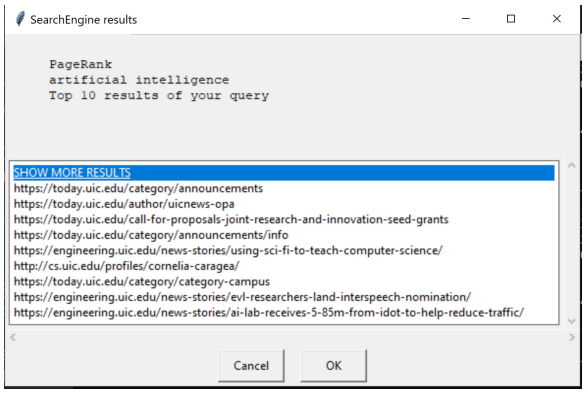

Figure 12: pagerank search on ”Artificial Intelligence”

12. RELATED WORK

Many articles on web crawling, how to do efficient web crawling and also many papers on page rank algorithm were read. So, first I have decided to work on cosine similarity and then decided to work on query dependent page rank algorithm which basically integrates page rank with similarity measure. pro-page rank algorithm is similar to query dependent page rank which takes difference of weights into consideration

13. FUTURE WORK

The search engine works very well but want to work on the existing code to improve the efficiency of page rank algorithm and planning to integrate different way of assigning weight for each word of query. Instead of just working with unigrams want to work on bi-grams and trigrams as well which can improve the performance of the search engine. As crawling takes lot of time. So, I am planning to write more efficient code to make computation time minimal.

14. REFERENCES

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