A Web Search Engine

Search Engine using Page Rank and Query Expansion

Karan Venkatesh Davanam
Master's in Computer Science
University of Illinois - Chicago
Chicago, Illinois
kdavan2@uic.edu

ABSTRACT

This Report is for the final term project(CS 582) which describe the components, functionality and evaluation of a web search engine that crawls the UIC domain in breadth-first search fashion. The search engine is backed with the ability to rank the search results according to the PageRank Algorithm and provides a feature to get words to expand user queries based on the Rocchio algorithm. The search engine provides a basic GUI and command-line option for the user to search their query and see the results. The mechanism for the search engine starts from crawling, page preprocessing, PageRank score calculation for the crawled links, similarity measurements based on various schemes, query expansion, and finally the graphical user interface. The following sections are present in the report

KEYWORDS

Rocchio, PageRank , Cosine Similarity , Vector Space Model.

1 Introduction

The crawler uses the combination of requests, urlib.parse, and BeautifulSoup to crawl the 4001 web pages inside the UIC domain in the breadth-first search fashion. The seed link used to start the crawling is https://www.cs.uic.edu/. The adjacency out link neighbor table which keeps track of all the out links from each page is built using networkx. The Simple version of the PageRank algorithm is implemented using the directed graph created using the networkx. The vector space model is used to index the webpages based on the cosine similarity. The search is powered with two intelligent components, one is the PageRank algorithm, which is used to rank the search results, and second, the Pseudo relevance feedback based on the Rocchio Algorithm. The search engine displays the ranked related links for each user query based on the concept of combining multi notions of relevance in search. The query dependent notion of Cosine Similarity is combined with the query-independent notion of PageRank to give a relevant set results for each user query.

2 Components

The Web search engine has the following components working together to get the relevant results for each user query. The description is each component is explained in detail.

2.1 Web Crawler

The implementation for the crawling is written in crawler_web.py. The crawler traverses through 4001 webpages in breadth-first search fashion, it even makes sure that the same link is not visited again and even checks for self-loops. The crawler only visits HTML based pages and does not visit pages like .pdf, .docx, .png, .mp4 etc. The queueing strategy used in this search engine is FIFO which is implemented using deque this makes sure that the traversal happens in the breadth-first fashion. The page owner restrictions are obeyed by checking the robots.txt and this implementation can be found in the crawl_allowed.py. Once the agent can fetch the results passing the restriction check, the contents of the web page are parsed to extract the links from the webpage and these links are subjected to link Canonicalization. The canonicalized links are used to index the respective web page contents for further preprocessing. The links that are extracted from each anchor tag from a page is added to the back of the queue for traversal. The entire 4001 pages are crawled in less than 4 hours.

2.2 Web Page Parser

Implementation of HTML parser is present in request_parser.py. The parser gets the response text of the current URL under investigation, it parses the web contents by removing the comments and removing non-important tags like 'document', 'noscript','header','html','meta','head','input','script','style','link','svg' ,'path','nav','form' and 'img'. Finally, the remaining contents are stores in the file for further preprocessing. A dictionary keeps track of the mapping of a file name to a URL.

2.3 Pre-Processor

The pre-processor component implementation can be found inside the get_all_tokens.py. The contents of the webpage are subjected to the series of pipelined stages of processing which includes word tokenizer based on white spaces, punctuation removal, stop word removal using NLTK stopwords set, and finally stemming of the candidate words using Porter Stemmer. The query string is also passed though the same pre-processing pipeline.

2.4 Indexer

The indexer implementation can be found in space_vector_model. The input to this component is pre-processed token from the pre-processor and an inverted index is created for the retrieval model and the data structure used for the inverted index is a dictionary.

2.5 Retrieval

The implementation vector space retrieval model can be found in the space_vector_model. The TF-IDF weighting scheme is used on each candidate stemmed word. The similarity measure used to retrieve the URL links with the given user query in the ranked fashion is the cosine similarity.

2.6 PageRank

The web search is powered with the PageRank algorithm to rank the retrieved links from the retrieval component. In this project, the PageRank Score is added with the Cosine Similarity score by assigning certain weights to both the relevance measure. The PageRank is implemented in pageRank.py. The ranking of the links can be found inside query_result.py. The multi notion relevance is assigned to links when the user clicks on search from the GUI.

2.7 Query Expansion

The pseudo relevance feedback using the Rocchio Algorithm is implemented in rocchio_expansion.py. The top 10 links are considered as the relevant links and words are returned from the algorithm so that it can be used to enhance the existing query to get more accurate results.

2.8 GUI

The graphical user interface is built using python Tkinter and it provides a maximum of 10 results on each page and with an option to move between pages. The GUI also provides a button to get the results from the query expansion component. The results seen using the GUI are ranked using the combination of cosine similarity and PageRank. To work with different similarity measures there is an option to use the command-line interface.

2.9 Pickle Loader

To increase the retrieval speed of results some pre-computed dictionaries are stored inside pickle files. The main dictionaries that are stored are the inverted index, out link tracker for each web page, PageRank score dictionary which holds the PageRank score for each link.

3 Weighting scheme and similarity measure

The weighting scheme used in the Web Search Engine is TF-IDF. The formula for TF-IDF is given below:

$$wij = tfij\ idfi = tfij\ log2(N/dfi)$$

where wij weight of the candidate term i in web page j. tf is the term frequency. df is the document frequency of term i. log2(N/dfi) is the inverse document frequency. N is the total number of web pages that are crawled.

In this project, the cosine similarity measure is used but different vector space similarity measures like Inner Product and Dice coefficient have been tried out. It is observed that the cosine similarity provided a relevant set of results when compared to the other similarity measure schemes. In many cases, the Dice coefficient and Cosine similarity showed similar results. The observations were compared with standard UIC search results found in all UIC websites that is enhanced by Google.

Below tables show the results of different similarity measures for the query ‘Cornelia Caragea’

```
Enter search query: Cornelia Caragea
Enter similarity measure choice
 Press 1:Cosine Similarity
 Press 2:InnerProd
 Press 3:Dice Coefficient
Enter similarity choice: 1
Links ranked according to the selected Similarity Measure
https://cs.uic.edu/profiles/cornelia-caragea
https://cs.uic.edu/~cornelia
https://cs.uic.edu/news-stories/cs-welcomes-13-new-faculty-members
https://engineering.uic.edu/news-stories/cs-welcomes-13-new-faculty-members
https://go.uic.edu/cs-instructor-officehours
https://cs.uic.edu/cs-research/research-areas-2
https://cs.uic.edu/faculty-staff/faculty
```

Figure 1: Results Based on Cosine Similarity Measure

```
Enter search query: Cornelia Caragea
Enter similarity measure choice
 Press 1:Cosine Similarity
 Press 2:InnerProd
 Press 3:Dice Coefficient
Enter similarity choice: 2
Links ranked according to the selected Similarity Measure
https://cs.uic.edu/cs-research/research-areas-2
https://cs.uic.edu/profiles/cornelia-caragea
https://cs.uic.edu/news-stories/cs-welcomes-13-new-faculty-members
https://cs.uic.edu/~cornelia
https://go.uic.edu/cs-instructor-officehours
https://cs.uic.edu/faculty-staff/faculty
https://engineering.uic.edu/news-stories/cs-welcomes-13-new-faculty-members
```

Figure 2: Results Based on Inner Product Similarity Measure

```
Enter search query: Cornelia Caragea
Enter similarity measure choice
 Press 1:Cosine Similarity
 Press 2:InnerProd
 Press 3:Dice Coefficient
Enter similarity choice: 3
Links ranked according to the selected Similarity Measure
https://cs.uic.edu/profiles/cornelia-caragea
https://cs.uic.edu/~cornelia
https://cs.uic.edu/news-stories/cs-welcomes-13-new-faculty-members
https://engineering.uic.edu/news-stories/cs-welcomes-13-new-faculty-members
https://go.uic.edu/cs-instructor-officehours
https://cs.uic.edu/faculty-staff/faculty
https://cs.uic.edu/cs-research/research-areas-2
```

Figure 3: Result Based on Dice Coefficient Measure

Based on the above figures Cosine and Dice Coefficient almost have a similar set of results except for the last two links. The link ‘https://cs.uic.edu/cs-research/research-areas-2’ is more relevant than ‘https://cs.uic.edu/faculty-staff/faculty’. So, according to cosine similarity, the link ‘https://cs.uic.edu/cs-research/research-areas-2’ is ranked higher so the **recall@6** according to Cosine similarity is higher than the Dice Coefficient. Hence, **the Cosine Similarity measure is used in this project**.

4 Evaluation Using Cosine Similarity

A Sample of 5 queries is being used to analyze the average precision and recall at Rank 10. The relevant links are obtained from the top 10 common search results of the UIC search enhanced by google and results obtained from Cosine Similarity.

The below table shows the queries used for the evaluation.

Number	Query	Relevant docs
1	uic computer science	2
2	accc	3
3	uic research	1
4	cornelia caragea	2
5	uic jobs	1

Table 1: Queries used for evaluation

The following results shows the precision@10 using only cosine similarity:

Query Number	Precision@10
1	0.2
2	0.3
3	0.1
4	0.2
5	0.1
Average	0.18

Table 2: Precision for each query and overall average

```
Enter search query: uic jobs
Enter similarity measure choice
 Press 1:Cosine Similarity
 Press 2:InnerProd
 Press 3:Dice Coefficient
Enter similarity choice: 1
Links ranked according to the selected Similarity Measure
https://jobs.uic.edu/Chicago/default.cfm?&start=1&per=5000
https://jobs.uic.edu/Rockford/default.cfm?&start=1&per=5000
https://jobs.uic.edu/job-search/default.cfm?&start=1&per=5000
https://jobs.uic.edu/Peoria/default.cfm?&start=1&per=5000
https://accc.uic.edu/about/jobs
https://jobs.uic.edu/JMLS/default.cfm?&start=1&per=5000
http://studentemployment.uic.edu
https://ece.uic.edu/undergraduate/internships-and-jobs
http://ecc.uic.edu/career-toolbox/go-search
https://cs.uic.edu/undergraduate/internships-and-jobs
```

Figure 4: Results for Query 5 using cosine similarity

```
Enter search query: accc
Enter similarity measure choice
 Press 1:Cosine Similarity
 Press 2:InnerProd
 Press 3:Dice Coefficient
Enter similarity choice: 1
Links ranked according to the selected Similarity Measure
https://accc.uic.edu/about
https://accc.drupal.uic.edu
https://accc.uic.edu/support/academic-continuity
https://accc.uic.edu/about/jobs
https://accc.uic.edu/about/reports
https://accc.uic.edu/support/academic-continuity/accc-operations
http://engineering.uic.edu/about/faculty/teaching-resources
https://accc.uic.edu/service-status/service-notices
https://accc.uic.edu/support/lts-support-office
https://engineeringalumni.uic.edu/profiles/herrera-lindstrom-cynthia
```

Figure 5: Results for Query 2 using cosine similarity

```
Enter search query: uic computer science
Enter similarity measure choice
 Press 1:Cosine Similarity
 Press 2:InnerProd
 Press 3:Dice Coefficient
Enter similarity choice: 1
Links ranked according to the selected Similarity Measure
https://cs.uic.edu/faculty-staff/faculty
https://cs.uic.edu/undergraduate/cs-major
https://cs.uic.edu/undergraduate
https://cs.uic.edu/faculty-staff/staff
https://catalog.uic.edu/ucat/colleges-depts/engineering/cs/minor-cs
https://cs.uic.edu/cs-events/calendar
https://catalog.uic.edu/ucat/colleges-depts/engineering/cs/minor-it
https://cs.uic.edu/undergraduate/cs-minor
https://catalog.uic.edu/ucat/colleges-depts/engineering/cs
https://cs.uic.edu/cs-research
```

Figure 6: Results for Query 1 using cosine similarity

The low score observed in Table 2 is due to the usage of only cosine similarity in finding out relevant links and the standard used to compare the search engine result is the UIC search enhanced by Google

5 Intelligent Component

In order to retrieve relevant links the web search is combined with two intelligent components :

- PageRank Algorithm
- Rocchio Algorithm with Pseudo Relevance Feedback

5.1 PageRank Component

The PageRank component is a query independent relevance measure. It assigns scores to each webpage based on how many authoritative pages point to it. In this project, once the crawling of 4001 pages is finished the PageRank component is triggered to calculate the PageRank score, this component runs until the page rank scores of the web pages converge.

$$S(A) = \left[(1-\epsilon)\sum_{B \to A}\frac{S(B)}{out(B)}\right] + \frac{\epsilon}{n}$$

Initially, the score of all the pages is 1/n where n is the number of pages crawled. The value of epsilon is 0.15. In this project, the self-loops to a page are ignored. Out() is the outdegree of each node that is the number of valid anchor tags present in a page. The directed graph constructed using networkx is used to calculate the PageRank score for each page using the above formula.

```
Links ranked according to the Page Rank Score
https://disabilityresources.uic.edu
https://today.uic.edu/resources/current-students
https://today.uic.edu/resources/faculty-staff
https://today.uic.edu/frequently-asked-questions
https://today.uic.edu/teaching-learning-and-working-resources
https://today.uic.edu/contact/social-media-directory
http://accc.uic.edu/service/identity-and-access-management
https://accc.uic.edu
https://accc.uic.edu/service/g-suite
https://news.uic.edu/social-media-directory
```

Figure 7: Results for Query 2 using only PageRank score.

The notion of combining multiple relevance for query search is realized using the below formula:

score(page) = w1*Cosine Similarity + w2 * PageRank Score

In this project the w1 score is 0.65 and w2 score is 0.35. Higher weight is given to Cosine similarity because the results that were obtained from Cosine Similarity is more relevant to the Google Enhanced UIC search.

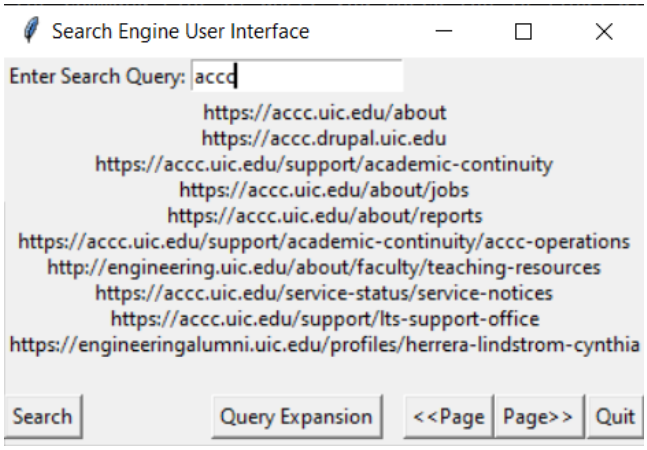

Figure 8: Results for Query 2 using Cosine Similarity and PageRank score.

5.2 Rocchio Query Expansion with Pseudo Relevance Feedback

Feedback in the Vector Space Model is realized using Pseudo Relevance by considering the top 10 search results as the relevant links for the search result. Using the Rocchio algorithm top 10 words are sent back to the user so that the user can use these words in the query in order to increase the accuracy of the results. This component is provided to guide the user in enhancing the query.

This component is activated when the user clicks on search and after getting the results clicks on query expansion that is shown according to Figure 8. The user is provided with a set of stemmed words which can be used to enhance the query and get more accurate results.

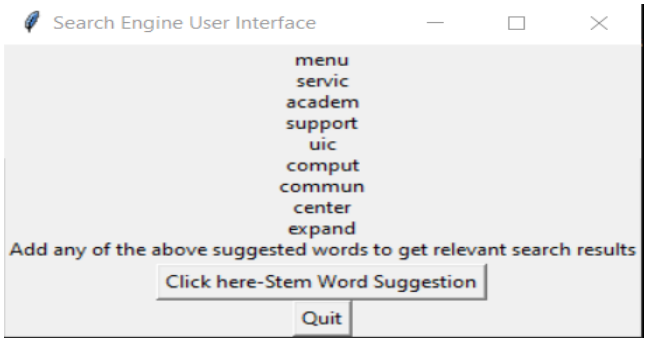

Figure 9: Results from query expansion which the user can add to existing query from Figure 8.

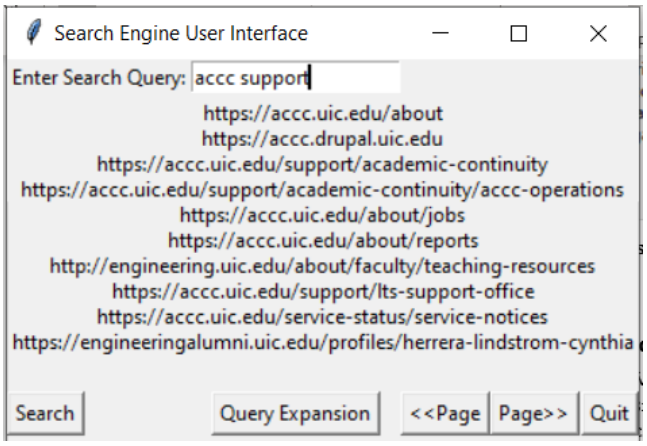

Figure 9: Results obtained for query 'accc' by adding word 'support'.

By using the correct set of words in the search query, the accuracy is increased. For the above query the number of relevant links with the Google Enhanced UIC search is 4.

6 Main Challenges

The links present in the anchor tags are not in a consistent format hence to get the absolute link was done using many if-else statements of different format present in the <a> tags.

Many of the pages that are inactivate were being pointed out from active pages hence an explicit condition is set up to check if the out-link page from the current page exists or not. (404 error check)

Cleaning the webpages that contained comments had to be handled with a special condition using beautiful soup.

While reading the contents of the webpage it contained some special characters that were breaking the whole parsing process this was handled using the ASCII encoding scheme.

While reading the robots.txt some of the webpages did not have robots.txt hence a special try-catch block was added to ignore such pages.

It was difficult to clean the menu contents of each webpage hence it affects the weights in the inverted index.

Some web links were redirecting the results to links with new names, so capturing these changes were difficult.

7 Error Analysis

What worked :

The notion of combining multiple relevance with stemmed words gave better results.

Pickling the state of inverted index into a pickle file helped improve the performance of the search engine.

Traversing the link structure using FIFO data structure helped the crawler not getting stuck in infinite loop

Stemming the words reduced the index size.

Limitations:

Crawling 4001 web pages is taking a long time due to the use of a single thread.

Since a small number of webpages have been crawled the page rank score cannot be only used to rank the web pages.

Sending stemmed words back to the user from query expansion is not an appropriate approach, using lemmatization is more appropriate.

The current web crawler is not scalable to crawl a larger set of web pages.

Getting the proper weight for features in multiple relevances can be done in a more empirical manner.

Single gram processing is not giving a good set of results.

9 Future Work

Instead of stemming the data want to pre-process the data using lemmatization.

Implement the web crawler using scrapy python framework to increase the speed of the crawler.

Improve the User interface using Django framework.

8 References

[1]http://www.michaelnielsen.org/ddi/how-to-combine-multiple-notions-of-relevance-in-search/

[2]https://nlp.stanford.edu/IR-book/pdf/07system.pdf