**Web Search Engine**

Search Engine using Page Rank and Query Expansion

Prajwal Kishor Kammardi  
 Computer Science  
 University of Illinois at Chicago  
 Chicago, IL, United States of America  
 pkamma2@uic.edu

**ABSTRACT**

This document is a report on the Search Engine final project which was done as a part of the CS-582, Information retrieval course at the University of Illinois at Chicago. The software consists of three phases. First phase is a crawler which crawls the pages in the UIC domain. The crawler handles the indexing and persisting the pages in the secondary storage. There is a preprocessing pipeline to clean the data and analyzer for the retrieval logic. The smart components used are page rank and query expansion based on pseudo-relevance feedback. A front-end interface is also implemented to view and interact with the application.

**KEYWORDS**

Search Engine, Page Rank, Cosine Similarity, Pseudo Relevance Feedback, Query Expansion.

**1 SOFTWARE DESCRIPTION**

**1.1 Introduction**

The application is divided into 4 components. The Crawler, the pre-processor, the analyzer and the front-end. The application is coded in Python3 in object-oriented programming model considering modularity and extensibility. The front-end uses Django framework considering its robust Python integration and scalability. The dataset contains around 7000 pages crawled, belonging to uic.edu domain. They are persisted in the secondary storage as normal text files. A network graph is generated, while crawling which aids in the process of finding page rank values. The data preprocessing stage cleans the stored webpage data, is vectorized (TF-IDF). This app is in the repository, https://github.com/prajwalkk/SearchEngineFS

**1.2 Crawling**

The web crawling uses a Breadth First Search strategy to get the list of webpages. The crawling’s starting path is the website, cs.uic.edu. Many assumptions are made to make the strategy as error free as possible. Some assumptions might feel very strict, and they can be relaxed for a greater precision. They are listed as follows:

1. Two data structures are maintained. One is a **set** to **keep track of visited sites** called as crawled. Other one is to **keep** **track of sites to be visited**, a **queue**.
2. A link is **valid** only if it belongs to the ‘**uic.edu**’ domain. Any redirects of a URL which do not belong to the required domain are discarded. Many faculty URLs redirected to external sites, and they are rightfully discarded.
3. If a URL **redirects to an already visited site**, it is **not** **discarded**.
4. The URLs are **canonicalized**. That is, **trailing ‘/’ in every URL is stripped** . **Trailing ‘#’** were **removed**.
5. Any **http** URL becomes **https**.
6. The blacklisted URLs belong to one of these categories. If they have an invalid extension **('.gif', '.jpeg', '.jpg', '.ps', '.ppt', '.mp4', '.mp3', '.svg', 'mailto:', 'favicon', '.ico','.css', '.apk', '.js', '.png', '.gif', '.pdf', '.doc', '@', 'tel')**. Also, if the response content is anything other than **‘text/html’**.
7. The valid status codes are 200 and 301 (redirects within the uic.edu domain was allowed)
8. If a site with no content (some JavaScript heavy sites) is downloaded, it is discarded as it makes no sense to search with no data.
9. Needless to mention, 4xx, 5xx response URLs are discarded.

**1.3 Processing of Crawled Pages**

Beautiful Soup 4 is used for processing the crawled pages. All the <a> tags with ‘href’ attribute present in the body of that page are added to queue. The title of the page was saved in the document. All the text present in all the tags were crawled and saved. The text which is part of comments is ignored. The text in script tag and other irrelevant tags are removed. The data is written to file, each having the ID of the crawled link as the filename. Some tags which contained IE specific code was considered as text. Those were removed.

A graph object is generated while crawling the URLs. The current page is taken as a node. Any outgoing link from the current page becomes the outlink from the node. After the crawling of all the pages are finished, the nodes which are not a part of the final set of crawled pages are discarded. Therefore, the number of vertices becomes 7000.

**1.4 Processing of Data for Retrieval**

The data retrieved from the crawled pages are now used to build an inverted index. The punctuations are removed, the stop words are removed, the words are tokenized and lemmatized. These are done by using the spaCy python library. The words smaller than 3 letters are not considered. The feature size is limited to 80000. The files are saved in the disk as pickle files.

**2 INTELLIGENT COMPONENTS**

**2.1 PageRank**

The page rank algorithm is used as the first intelligent component. During the crawling of pages, a directed graph is created based on the link stricture of the uic.edu domain. It contained 7000 nodes, the sites which were successfully crawled. If a link contained any other page which was not crawled, it was discarded. Thereby, eliminating dangling edges. If any unconnected node is present, it is discarded. PageRank computes a ranking of the nodes in the graph G based on the structure of the incoming links. The NetworkX implementation of PageRank was used. The eigenvector calculation is done by the power iteration method and has no guarantee of convergence. The iteration will stop after max iterations or an error tolerance limit specified has been reached. The maximum iterations specified was 10. Damping parameter was 0.85.

|  |  |
| --- | --- |
| **PageRank** | **Value** |
| https://uic.edu | 0.0307 |
| https://disabilityresources.uic.edu | 0.0131 |
| https://library.uic.edu | 0.0131 |
| https://maps.uic.edu | 0.0118 |
| https://emergency.uic.edu | 0.0116 |
| https://catalog.uic.edu/ucat/academic-calendar | 0.0113 |
| https://uic.edu/apps/departments-az/search | 0.0108 |
| https://today.uic.edu | 0.0104 |
| https://uic.edu/about/job-opportunities | 0.0103 |
| https://today.uic.edu/events | 0.0083 |

Table 1: PageRank value of the top-10 pages in the uic.edu domain

The table shows that the UIC homepage has the greatest number of links connecting to it. More than 10 iterations were tried, but the scores more or less converged after 10 iterations.

**2.2 Query Expansion and Feedback**

Rocchio’s Algorithm was implemented for the relevance feedback. The idea of the current context of the query was considered, and display of similar results to the current query looked like a good implementation. The feedback is dynamic and related to the current query of the user and the returned documents. The initial idea was to let the user select the number of relevant results returned from the searched query, but it was quite complex and time consuming as the tagging had a new overhead. Also, the main belief in user experience is, the user is always dumb. So, the results selected by the user are not always relevant. Hence, the assumption that the first 10 results returned would be considered as relevant. And the rest as non-relevant. The computation was done by using:

(1)

The values for α, β, γ were chosen 1, 0.75, 0.15, respectively. The non-relevant documents were given a lesser weightage as it felt the irrelevant documents would otherwise get more preference.

The implementation of the feedback was quite simple. The top 10 results were considered as relevant. Then the TF-IDF of these document vectors corresponding to the vectors were added to the query TF-IDF vector. Rest of the 6990 documents were considered irrelevant and subtracted from them. The resultant vector’s top 20 features were extracted. Here as unigram, bigram, trigrams were considered as features while getting the TF-IDF, there were a lot of repeated words. Therefore, the unique features were considered, there n-grams > 1 were given a higher preference. The resultant query expanded terms made quite good suggestions. At first the given query was attached to the resultant query terms and then displayed. But that idea seemed to quite skew the results in favor of the query and there by biased. Therefore, that implementation was discarded and just the query terms were returned. Some of the examples for query ‘UIC Career’ are:

|  |  |
| --- | --- |
| **Sl.No** | **Query Expanded term** |
| 1. | logo student employment |
| 2. | uic career service |
| 3. | student wage plan |
| 4. | job listing |
| 5. | job description |
| 6. | job fair |
| 7. | campus job |
| 8. | uic logo student |
| 9. | career service uic |

Table 2: Query expansion results for the search query “UIC Career”

**3 CHALLENGES FACED**

1. The crawler had major issues with the design. The pages were not getting saved in the required format as the text in the internet was not consistent. Some failed to save in the UTF-8 format. So, a hacky solution was created, and the errors were made to ignore. The smallest of quotes which were required for the beautifying the appearance of the website, used to crash the application which had taken almost 30 minutes to get to the point.
2. The pages had to be crawled more than once. Lots of 404 errors broke the application. Sometimes there were redirects to external sites when the final links were checked. All these errors had to be handled.
3. The preprocessing of the pages was especially challenging as the pages had JavaScript codes written inside valid tags and the parser used to consider them as text. Lots of lines of code were wasted for this venture. The IE compatibility code especially made the work take longer.
4. Another challenging aspect was lemmatizing of the words. Initially NLTK library was used. It needs POS tags for efficient lemmatization which were not specified. Therefore, by default all the words were considered as nouns. It rather had weird outputs where CS became C. Later, spaCy was used. Which eliminated the hassle.
5. Deciding the intelligent component was quite challenging. The project was new and there was a lot of learning to be covered before trying to implement the component and each component seemed more intimidating than the other.
6. The hyperparameter tuning for vectorization, PageRank, Query Expansion took a lot of time. The Query relevance is subjective. The weighting factor to be given for the relevant and non-relevant documents was hard to come up. Later, the values in the textbook actually worked well. There was also a dilemma whether to consider unigrams, bigrams, or trigrams for the Query expansion. Trigrams gave good looking results.
7. Lot of focus and time was spent on creating the front-end and the integration with the backend was quite the task. A lot of time went into fixing errors, learning about the libraries which broke because of the framework and the workarounds needed to make the app running again.

**4 WEIGHTING AND MEASURES**

**4.1 Weighting Scheme**

The weighting scheme used is **TF-IDF**. TF-IDF was used instead of other schemes like BoW is because, the BoW is going to be very sparse on this dataset. TF-IDF is preferred to plain TF as TF-IDF scales down the impact of the frequency of occurrence of a particular word. The TF-IDF Vectorizer of sci-kit learn library was used. The hyperparameter tuning was done based on trial and error. **Sublinear TF** was applied as multiple occurrences of a word does not necessarily mean it is very significant. And the logarithmic scale dampens this effect. The IDF was smoothened to eliminate 0 division errors. N-gram size varying from 1 to 3 is taken which would later be use in Query expansion. The custom tokenizer stripped accents, lemmatized, removed the stop words. Feature size chosen was 80000. The accuracy did not increase for queries if more than 80000 vocabulary was taken for the limited set of queries executed.

**4.2 Similarity Measure**

The similarity measuring scheme used here is the **cosine similarity.** This measure made sense as vector comparison is involved, and it gives a better similarity score of two similar but extremely distant documents in a corpus by looking at their orientation and not distances with taking their lengths into consideration. Effort was made to extend the cosine similarity by using soft-cosine similarity i.e., by taking synonyms. But the complexity increased greatly.

**4.3 Feature Weight Assignment**

The cosine similarity and PageRank are some of the main features used to rank the search results. In this project, a trial and error method were used to determine the importance of each of the feature. The PageRank is query independent and the cosine similarity is query dependent.

The weight for PageRank was given a higher preference as general pages would be returned for generalized queries.

**4.4 Comparison with possible alternatives**

The possible alternative that one could think of would be google and the UIC’s search features. The comparison google is rather unfair, but they can be done for a good retrospection.

Based on the queries, the intelligent component gives good results. Google which calculates the PageRank score on a broader corpus sometimes tends to miss some of the relevant pages in the UIC domain in the top-10 results. As the given application is more specific to the uic.edu domain, it fares better in the specificity department. An example for this would be, if the query term was “career services” the google result would be of a broader and different corpus altogether rather than the UIC one.

The plain Cosine Similarity results and the PageRank scores did not make a huge impact on the top-10 results as the PageRank score was given a preferably higher weightage. But after top-20 is considered, more websites with lesser PageRank given a lower rank. Therefore, a hub is rightly given a higher score as many people do have the need to search for generalized pages rather than specific pages. The query expansion then helps them navigate to the more specific page if needed.

Comparing to the UIC’s inbuilt search, the application performs well. The query expansion is not implanted in the UIC’s search and hence, no grounds for comparison. But the results are rather similar.

**5 EVALUATION**

1. Query: “*jobs*”, the first link was *economicimpact.uic.edu* and the third link was uic.edu homepage. Rest of the links were jobs related links, but the second link should not be there for this context and the first link should not be that high in the rank.
2. Query: “*Cornelia*, the first two links were exact homepage and required results. The next four were linked to the result. The rest of the results were not relevant, and the pages with the highest PageRank populated the spots. The unintelligent component fared better in this.
3. Query: “*SRF address”* This result matched the UIC’s results. Even though an acronym was given, it fared well as SRF was quite common in the documents. The page rank and Cosine similarity had similar results.
4. Query: “*Information retrieval”* It was a very generalized query. The results returned were quite accurate. The results in both intelligent and unintelligent were similar
5. Query: “*career services”.* This query was run to check if jobs and career services would have the same results. But it was quite different. The results for intelligent and unintelligent differed vastly. The cosine similarity results had results of some of the people who were a part of the department. One observation is seen that the intelligent component fares very well on non-specific queries. The unintelligent one fares well on the specific requests.

|  |  |  |
| --- | --- | --- |
| Query | PageRank Score | Cosine Similarity Score |
| jobs | 0.8 | 0.8 |
| Cornelia | 0.6 | 0.7 |
| SRF address | 0.8 | 0.8 |
| Information retrieval | 0.8 | 0.8 |
| career services | 0.9 | 0.9 |

Table 3: Precision at 10 for the search queries.

**6 RESULTS**

The results were quite positive. The application returned relevant documents and the Intelligent components did really add to the specificity of the results.

Somethings that did not work:

1. The crawling component could not be multithreaded as the software made went too complex.
2. Feature weight assessment needed more tries. The number of queries used to change the weights were simply too less and more tries would have been needed for better testing.
3. The Query Expansion sometimes is a miss and returns whole website link as opposed to words. This is because the site link is also used as a feature
4. The website takes a lot of time to get results. This can be easily tuned.

**7 RELATED WORK**

Many projects were referenced to help build this search engine. A similar project in GitHub ‘ozancagalayan/SearchEngine’ which indexes daily stories of AP news wires in the year 1989. There was also a user ‘mohit155/SearchEngine’ who indexed Wikipedia for the pages containing “cricket”. His project has a great reference for Pseudo-relevance feedback which I took help from.

**8 FUTURE WORK**

A better hyper parameter tuning must be added. One idea was to implement the synonym search for the common nouns used and synonym tagging for some of the terms in the corpus, like the faculty names, the acronyms of the given branches and courses. This was the initial idea of my expansion. But it would be implemented in the future. Also, some more features like the URL path length, etc., could also be used as feature to rank the links. Document clustering would also be great feature to implement.

**ACKNOWLEDGMENTS**

This project was a great learning experience. I would like to thank Prof. Cornelia Caragea for sharing her wisdom which helped greatly during this project. I would also like to thank my peers in the UIC who helped me in polishing my project.

**REFERENCES**

1. A. Langville and C. Meyer, “A survey of eigenvector methods of web information retrieval.” <http://citeseer.ist.psu.edu/713792.html>
2. Page, Lawrence; Brin, Sergey; Motwani, Rajeev and Winograd, Terry, The PageRank citation ranking: Bringing order to the Web. 1999 <http://dbpubs.stanford.edu:8090/pub/showDoc.Fulltext?lang=en&doc=1999-66&format=pdf>
3. Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, “Exploring network structure, dynamics, and function using NetworkX”, in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008
4. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.