CS 582 Information Retrieval

Phakphoum Suraminitkul

Master of Science in Computer Science
University of Illinois at Chicago
psuram2@uic.edu

ABSTRACT

The main focus of this project is to implement a functional web search engine of the UIC's domain. The web search engine is a software system that is designed to store information, categorized it and show the end user what happens each time we conduct a search in a query-based format.

The web search engine takes in a webpage which is the UIC domain, https://www.cs.uic.edu/, and start the crawling process. The threshold of web crawling is around 3000 pages. The results of the ranking are output as a list in the Graphical User Interface where the top of the ranking is the most relevant documents to the user input query.

One main criteria of the project are to implement an intelligent component. In this experiment, I try to optimize the search engine by implementing an expanded query search and comparing it to the normal query search. A Page Rank implementation is also integrated and the details can be found later on in this document.

The order of the repot will start with web crawling, preprocessing, indexing, ranking, query search, discussion of results, challenges encountered during the building of the search engine and future work.

Introduction

The software is written in a modularized Object-Oriented Programming style in Python 3. Preset environment is being created to run the code with several packages and libraries needed to be downloaded and imported.

To help reduce the time to execute the time-consuming web crawling and page preprocessing procedure, I have uploaded the processed dataset as well as the necessary files needed to run the program in the repository. Nonetheless, the scripts to run the crawling and preprocessing are also attached in their relevant files.

1. Crawling

The initial web crawling module can be performed by running the *crawling.py* script. The crawling begins at the UIC Computer Science domain, https://www.cs.uic.edu/.

The crawling starts at the root of the tree and the subsequent crawled pages are added to the tree in this manner. The content of each page are then parsed using the *html.parser* library which can be found in *BeautifulSoup*.

The crawled webpages are first checked if they are in the scope of the UIC domain. This is done by providing the patten that I have put in place. *avoid* is a list that I have created to avoid the formats that I deemed unnecessary, and would not provide meaningful contextual clues for this purpose. Several other formats are also manually put in place to curb with error which arises whilst crawling.

2. Preprocessing

The backslash of each URLS are first removed and the start of all the URLs are then modified to *https* with the aim to remove duplicates. Pages that have the same URLs are discarded. The URLs which have the anchor tag <a> after the href are taken into consideration. URls that meets these checks are saved into *crawled.txt* file waiting to be further processed.

The files which are in pdf and html formats are being processed through the *processor* function. For each html file, a built in API called *BeautifulSoup* is used to remove the CSS and JavaScript component by processing each page and taking only the plain text in all the tags apart from those in "scripts" and "style" tags.

The textual content of the pages is first saved onto a temporary dictionary called *web_pages* having the key as web URLs and values as content. As for the pdf files, each page is read, parsed and joined together using a PdfFileReader and saved onto *web_pages*.

Next, the documents go through the text-preprocessing phrase where the words in the *web_pages* are tokenized, stemmed and the stop words removed. The final results are then save onto a json file called *urlWord.json*.

3. Indexing

Next, the *Indexer.py* script illustrate how the corresponding *tf*, *idf*, and *document length* are built.

4. Ranking

'Background: for perfect matching scores, it will almost always fail for longer queries.'

The ranking of webpages in this software is calculated using a similar Page Rank implementation. The most difficult part of the project was to implement a page rank algorithm and its' page authority score. Part of the implementation uses the idea of HITS algorithm, a link analysis algorithm that helps rates the web pages. Even though, some of the results obtained does not provide a clear ranking due to the variation of the weighing of hub and authority score.

To begin with, each page has two scores, hub and authority, which estimates the value of the content of the page. Given that more page points to a certain page, it's value of authority increases.
Fact: In the query retrieval process, we know that the authority score is the sum of all the hub scores of pages that points to it.
Knowing this, the next step is to find a way to calculate the weight of an incoming URLs to a page from various other pages. The other would be to find the number of outgoing URLs from a page.

Using this idea, I first try to normalize both the values of the authority score and tf-tdf score during the calculations, hoping to come up with a better scoring technique where when the original authority score is 0, the equation would not return 0 according to the formula of page authority score.

Furthermore, using the padding strategy, a constant variable is put in place to avoid the page from being unfairly scored when the page starts with a zero score. The documentation of how the ranking works is shown in *ExpanderSearch.py* file.

5. Query Search

The main focus of the project is achieved here in the expanded query search. Given the keyed query from the user, the query words go through preprocessing just like the other documents mentioned earlier, and it is here where the words subsets are being created to increase the quantity of those words which could be used to process a better well-defined result.

A generated subset of the query words is first created. With different variations of subset of words, the software then searches for synonyms of those words via a standard well-known lexical database for English, wordnet and word2vec.

Next, *SpaCy*, a python pre-trained word model is used to find the similarity between generated words subsets of the query and the synonyms of the words from in the wordnet. To determine the level of similarities of the meaning of the words, a certain threshold is also being put in-placed to allows only query words which have a certain allocated (high) level of cosine similarity to be returned from the function.

**The weighing scheme uses the standard tf-idf to calculate the cosine similarity which puts the relevance of each document in the correct order for the search engine. Simply put, it is one of the most efficient means to find the importance of words in each document. **

6. Results Analysis

6.1. For an input query, "Richard Daley Library".

The results from the top ranking did not appear to have anything related to Richard Daley Library, yet the results from the department of bioengineering homepage was returned. After having gone through the pages, it becomes apparent that the search only managed to return part of the query, in this instance, Richard. Where Richard is the name of the Professor and former Bioengineering department head, Richard Magin.

Presumably, the crawlers only managed to crawl through these pages but not of the actually library website starting from the computer science domain. With the expanded query search, the results also point similarly to those of the normal search but with a difference in a page which points toward the computing and communication center where Richard Daley library '*help-desk'* was mentioned.

6.2 For an input query, "Admitted Student".

The results return a generic relevant URLs with numerous contents pointing to '*admission'*, '*admitted-student'* from several departments such as the Biology engineering department and the Office of International Student.

With the expanded query search, the results return relevant pages from the OIS, the '*hours'* and '*location'* where a lot of links are pointed to it. However, the top few pages returned less relevance information, where it only shows the webpages for the '*Student Veterans'*. The reason for these outlier results are from the fact that the frequency of the word '*student'* has skewed the calculation of the cosine similarity.

6.3 For an input query, "Computer Algorithm".

As anticipated, relevant webpages '*department of computer science'* where the course is mentioned are returned. However, the '*image analysis algorithms*' in visualization laboratory pages are also mentioned and are ranked under the former webpages.

With the expanded query search, the top 5 returned webpages are the same as the former search but the results differ. This could be due to the subset of words generated during the expanded search are different but close enough to the original word thus the ranking score of the remaining pages differs. Within the returned results, the words "*compute, computation, computational*" are mentioned in the pages as well.

6.4 For an input query, "Student scholarships".

The results return all the relevant generic pages such as '*financial-aid-scholarships'*, and '*honors.uic.edu/scholarships'* from the department of liberal arts.

With the expanded query search, it returns a more specific pages from the '*OIS/support_the_office_of_international_services*' and '*uic.edu/admission-aid/paying-for-college'*. The results obtained are more directed towards pages to apply for various scholarships with all the links and heading from the OIS. From this query, the assumption that the value of penalty during the calculation could drastically affect the final page rank score is confirmed. With the top 5 returned pages being of less relevance to the query.

6.5 For an input query, "Course catalog".

The search returns all the relevant results such as the UIC '*course-descriptions*', UIC '*degree programs*' as such.

With the expanded query search, the result returns more or less the same webpages, but the difference is that of the final rankings. With further analysis, I have found out that most of the webpages actually points to the same catalog webpage which is the '*uic.edu/cat*'. Similar to those from the former search.

7. Difficulty faced

At the beginning to the project, one of the key components is to crawl the webpages, the main hurdle to overcome was the use of different parsing techniques to successfully obtain the textual contents in the pages. Having spent hours going through and searching for certain files format to opted out of, those which does not provide much information or to individually remove the files using trial and error.

Moreover, the crawling process consume a huge amount of time to crawl the pages. I believe the work could be improved by utilizing a multi-threading to run the process of crawling. Another hurdle which was difficult to cross was the amount of incentive and penalty to give to the extended query search algorithm. With countless tries, though the software functions well but on several occasion, it returns less relevant pages based on the query.

It is also rather difficult to verify if the accuracy score of the results as the golden standard or a bench mark does not exist to compare the results to.

According to the end user, even the best scored and ranked pages are may not be as objective. Taken into consideration that some of the end users may not have the background on how to construct a better query or the word expression are not correct. Perhaps, testing out different ranking techniques could help curb this challenge.

8. Future work

Tuning of the hyperparameters is considered to be one of the essential parts of the implementation. With the knowledge that some of the pages may work better than the others with respect to the penalty given during the scoring of pages.

Another point worth mentioning is having a gold standard to compare the results of the search engine to could prove be worth looking into, but this manual process and implementation could be very time consuming and expensive in the real world. One a side note, to ensure that the relevance of the pages returned, one could do so by crawling more webpages.

Building a more specific search engine instead of a robust search engine which aims to crawl one specific domain could improve the relevance of the query and retrieved documents.