A Web Search Engine

Search Engine Using Learning to Rank approach

Manoj Prabhakar Nallabothula
Master's in Computer Science
University of Illinois - Chicago
Chicago, Illinois
mnalla2@uic.edu

ABSTRACT

This is a Report for the final project of CS 582 Information Retrieval at University of Illinois at Chicago. The project consisted in building a web search engine for the UIC domain from scratch. The software was built modularly, starting from web crawling, going through pages preprocessing, indexing and finally adding a basic Graphical User Interface.

General Terms

Algorithms, Measurement, Performance, Design, Reliability

Keywords

Search Engines, Similarity, Page Rank, Learning To Rank.

1 Introduction

The software is written in Python3 in an Object-Oriented programming fashion to make it easily extensible for the future. In order to make it easy to download and test the code without having to perform an extensive and time-consuming crawling and page preprocessing, the dataset containing 6000 pages crawled from the UIC domain: https://www.uic.edu/, and the preprocessed pages and needed files generated (Index, Page Ranks, Document lengths, documents' tokens, URLs) are included in my github page (https://github.com/man93oj/A-Web-Search-Engine). In this way by cloning the repository the ***SearchEngine.py*** can be run and in less than a second the search engine is ready to get queries in input. However, the script ***main.py*** needs to run for crawling, preprocessing, Indexing and calculating Page ranks are also included and explained in the following subsections with all the other components.

2 Crawling

The web crawling logic is at ***WebCrawler.py*** script, which uses a traditional approach that goes from one page to another in a Breadth First Search (BFS) fashion. The crawling starts from the UIC CS subdomain: https://www.cs.uic.edu/ which is the root node. From here, it adds all other hyperlinks in this webpage to a queue and the webpage is removed. The next page in the queue is chosen and the process is repeated until the queue is empty or a threshold is reached. Obviously, considering the vastness of the World Wide Web (WWW), the process was terminated with the help of a threshold/page limit in this experiment. Every page is dequeued, downloaded and parsed using the *HTMLParser* library, its links are extracted and checked to belong to the UIC domain, then added to the FIFO queue if it is in an appropriate format. The pages were downloaded initially and then preprocessed, so that the process is faster. The inlinks and outlinks from each page were also stored, so that the PageRank of each page can be determined. Pages which took too long to respond or were not available were not added to the inverted index. Each page stored in the inverted index is referred to using a number rather than the page URL (for more efficiency and ease of coding). A blacklist of all bad formats was derived by me while seeing the results that I was getting in the crawling and it consists in this 18 formats: ".docx", ".doc", ".avi", ".mp4", ".jpg", ".jpeg", ".png", ".gif", ".pdf", ".gz", ".rar", ".tar", ".tgz", ".zip", ".exe", ".js", ".css", ".ppt". A timeout of 4 seconds is specified to continue for the next page if that obtained page is not valid. Roughly, around 6000 web pages within the UIC domain were traversed and stored in an inverted index. Several other features were also stored from each page which are explained in the subsequent sections.

3 Term-frequency and Inverse document frequency

TF and IDF values are calculated in the traditional way i.e., using an inverted index. From a programming perspective, two dictionaries were used to hold both features. The term frequency is held by a 2-D dictionary (one dimension for document number and second dimension for words) and inverse document frequency is a 1-D dictionary having just the words and their frequency as elements. All words in the web-page were processed in the same manner before being added to the inverted index: the word was converted to lower case; all trailing and leading whitespaces were removed and finally the words were stemmed. The Porter Stemmer from the Natural Language Tool Kit (NLTK) was used here.

No more description of these concepts is required here.

4 PageRank

Once again, the traditional random surfer model [5] is used in this experiment. The PageRank of a webpage is given by the following formula:

$$PR(p_i) = \frac{1-d}{N} + d(\sum_{p_j \text{ links to } p_i} \frac{PR(p_j)}{L(p_j)} + \sum_{p_j \text{ has no out-links}} \frac{PR(p_j)}{N})$$

The above formula handles cycles in the graph and deals with dangling nodes. Nodes which do not have outgoing links are called dangling node (or) sinks. These can cause the page rank to collapse, causing the page rank score to be concentrated at the sink. In the experiment, verification was done by adding up the page rank of all 6000 crawled and making sure this summed up to a value of approximately 1, ensuring the score is split in the correct way.

Table 1: shows the top 10 pages according to the PageRank algorithm used in the experiment.

S. No	URL	Page Rank score
1	uic.edu	0.03627
2	maps.uic.edu	0.01350
3	disabilityresources.uic.edu	0.01228
4	library.uic.edu	0.01150
5	uic.edu/apps/departmentsaz/search	0.01131
6	emergency.uic.edu	0.01059
7	catalog.uic.edu/ucat/acade mic-calendar	0.00911
8	today.uic.edu	0.00805
9	dos.uic.edu/studentveteran affairs.shtml	0.0077
10	uic.edu/about/jobopportunities	0.00727

It is clear from the table, that the page rank algorithm has done well. The URL https://uic.edu has been ranked #1 which is no surprise considering the number of pages linking to it i.e., it has a high in-link count. Similarly, we have other important pages from the UIC subdomain in the list. The PageRank algorithm was run for 20 iterations which is more than enough usually for convergence of the score.

5 Extracted Features

Table 2 shows the list of features extracted from each web page and if that feature is dependent or independent of the search query given by user. This feature can be from the title (T), body(B) or URL(U) of the webpage. These features were chosen based on Microsoft's Learning To Rank dataset [3] which hosts 136 features used to find relevance of a page with respect to others. Since several of Microsoft's features were privately held [4] i.e., not released to the public, those features could not be tested on our experiment. These privately held features include user dwell time, quality scores of pages based on Microsoft's private algorithm, URL click count, Query-URL click count, etc. Hence, for our experiment we use only the 30 features listed out in Table 2. Most of the features listed out can be intuitively understood. Covered query terms is the number of terms from query present in the web page. The ratio would mean the covered query terms divided by the query length. Stream length is the number of characters in the title and body of the page. IDF is the inverse document frequency i.e., sum of IDF of each query term present in the web page. Similarly, TF is the term frequency of each query term in the web page. We also calculate the minimum, maximum and mean of term frequency. Like TF, we calculate the sum, maximum, minimum and mean of TF-IDF score for each page. We also have the length of URL and number of slashes in the URL. Finally, we have important features which are not query dependent namely, in-link count (number of pages pointing to that page), out-link count (number of pages that the current page points to) and the page rank of that page.

Table 2: List of features extracted from each web page

Feature number	Feature name	Portion of text (T/B/U)	Independent /Dependent on query? (I/D)
1, 2	Number of covered query terms	T, B	D
2, 3	Covered query term ratio	T, B	D
5, 6	Stream length	T, B	I
7, 8	IDF	T, B	D
9, 10, 11	Sum of TF	T, B, U	D
12, 13	Minimum of TF	T, B	D
14, 15	Maximum of TF	T, B	D
16, 17	Mean of TF	T, B	D
18, 19	Sum of TF-IDF	T, B	D
20, 21	Min. of TF-IDF	T, B	D
22, 23	Max. of TF-IDF	T, B	D
24, 25	Mean of TF-IDF	T, B	D
26	Number of slashes	U	I
27	Length of URL	U	I
28	Inlink count	U	I
29	Outlink count	U	I
30	PageRank	U	I

6 MAIN CHALLENGES

The main challenges that I have experienced during the development of this project are:

- Initially it was difficult to choose what kind of smart component to design, since I had no experience in this type

of application whatsoever and thinking about improvements without having a demo to test on it is just so difficult.

- During the implementation part, I spent a lot of time in learning about Python libraries and constructs that I did not know yet, like web crawling, how to implement or do web scraping to get the links out of an HTML page. Another thing that I had to learn from scratch was how to create a GUI in Python.
- I had to crawl 6000 pages more than once, because I found in the pages, there are various wrong formats. In the end the entire list of formats that I had to blacklist had a size of 18 and it included ".docx", ".doc", ".avi", ".mp4", ".jpg", ".jpeg", ".png", ".gif", ".pdf", ".gz", ".rar", ".tar", ".tgz", ".zip", ".exe", ".js", ".css", ".ppt".
- A very challenging thing was the hyperparameters tuning and the integration of page rank to rank documents. The parameter "***e***" to decide how much importance to give to the tokens of the extended query is the most difficult to tune because you cannot know the average performances of your search engine if you don't have labeled data (relevant documents for each query). Also, query relevance is subjective, you never know what a person wants to retrieve, and you can only guess based on the query. I decided the *e* should be at most 0.5 and at the end I set it around 0.1-0.2, you don't want to bias the query to much by giving a lot of importance to words that are not typed by the user.

7 Intelligent Component

An attempt was made to train a neural network using linear regression [6] on "Microsoft's Learning To Rank" dataset. The 136 features in the original dataset was cut down to 30 features which are the ones listed in Table 1. However, we faced several problems training the network. One was the high amount of training data coupled with number of features which posed a problem for training on a standard CPU. Hence, GPU power was required for training this for several epochs. Another problem is that Microsoft's dataset has output labels within the range 0-4; 0 stood for no relevance of the page with respect to the query and 4 stood for high relevance with respect to the query. Several training attempts were made on their data. Initially, the model would overfit at a value of 2.5 (outputting value of 2.5 for any input page). Given output labels within 0-4, 2.5 is a good output all the time as the mean squared error would always be low. This indicated that the model overfits given such a small output range. What we tried next was to multiply output labels by 100, thereby dataset having output labels (0, 100, 200, 300 and 400) was formed. However, the model never converged on this dataset outputting high error values and very low validation accuracy. Hence, as a last attempt, we assign the weights manually for each feature and sum them up to form relevance score as explained in the next section.

7.1 FEATURE WEIGHT ASSIGNMENT

Firstly, the importance of each feature was determined. Then a trial and error method of plugging in different weight values and checking the output results was done to ensure that acceptable weight values were assigned.

It is no doubt that the number of covered query terms is an important feature in determining whether a page is relevant or not. The covered query term for title and the body is set to a high weight value of 40 and 100 respectively. Then, the weight for in-link count of any page was calculated as follows:

$$Contribution(inlink) = \frac{inlinkcount(currentdoc)}{max(inlinkcount(docs))} * 110$$

The denominator is the page having highest in-link value. So, this page would be having a value of 110 which is the highest value possible by in-link feature.

The amount contributed by page rank was found in the following way:

$$Contribution(PageRank) = \frac{5000 - PageRank(doc)}{5000} * 110$$

What this formula does is assign page with rank 1 a value of 110. The number 5000 is used as that is the number of pages crawled. Lower PageRank give lower scores.

The next important features are the length of the URL and number of slashes. Both features are penalty incurring i.e., they decrease relevance score of the page. Higher URL lengths leads to more penalty, similarly with number of slashes. This is crucial in identifying authority pages. The effect of this feature is explained in the next section.

All these weight values were assigned after several trial and error tests with a lot of queries. Other features mentioned in Table 1 have lower weight value as their contribution to the overall score is not clear. Hence, the other features in the list were arbitrarily assigned lower weights of 0.001. These features include the different term frequency and TF-IDF features.

Therefore, the final relevance of a page is given by the following formula:

$$Relevance(page) = \sum_{featurelist} weight(feature) * feature$$

8 EXPERIMENTS AND RESULTS

We list out the precision for the 5 queries given in Table 3 for both the intelligent and unintelligent method. The unintelligent method outputs the list of pages based on TFIDF of each page according to the query. The intelligent method performs better; at least in most of the queries, as it takes both relevance of page and importance of page.

Some queries are dependent on the keywords and not on importance of page: say we are searching for a person. The results of such a query are shown in Figure 1

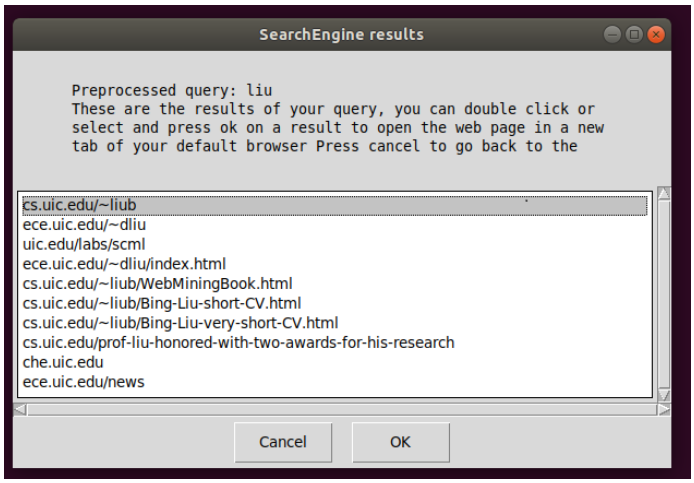

Figure 1. Intelligent Component output for "Liu"

This query is highly keyword dependent as it is the name of a person. Both the intelligent and unintelligent engines perform well for this query. When we search for a keyword, we get the following results:

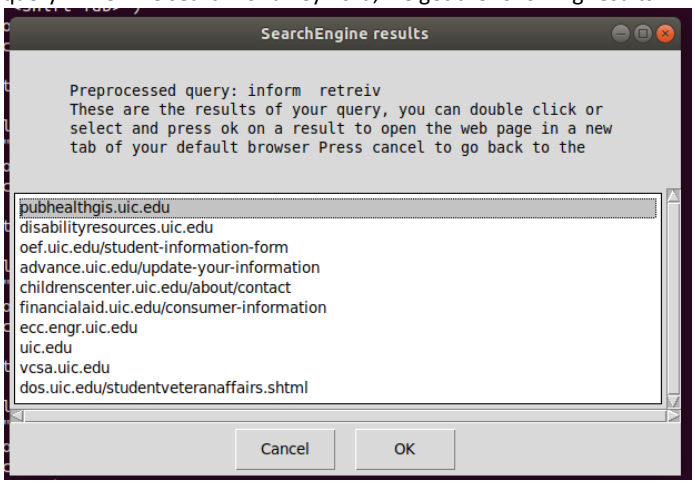

Figure 2. Intelligent engine output for "Information Retreival"

This gives a satisfiable result with most results relating to web page listing the details of the PhD programs of several schools. The unintelligent search engine output results as shown in Figure 3.

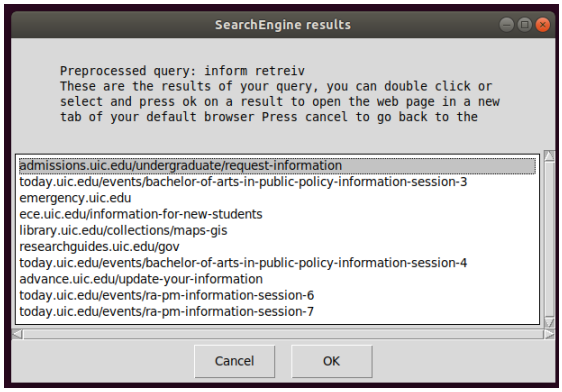

Figure 3. Unintelligent engine output for "Information Retreival"

There are some acceptable results, however a lot of them are irrelevant or are subpages of graduate programs which have the word "Information" or "Retreival" repeated several times. The reason why the intelligent search engine is performing well is because it incurs a penalty on the relevance score of the webpage depending on the URL length. Also, the covered query terms along with page rank helps in identifying "root" pages. Similarly, query "CAREER" returns results as in Figure with intelligent component. This highlights the quality of the intelligent engine even better. The unintelligent search engine is stuck on pages which have high frequency of the term "CAREER", however this is not a desirable thing. The intelligent one on the other hand shows a good mix of pages, showing root pages of several parts of the UIC subdomain.

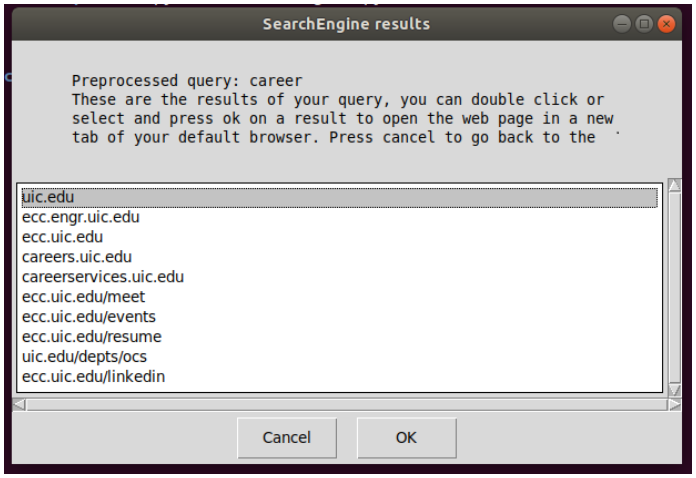

Figure 4. intelligent engine output for "CAREER"

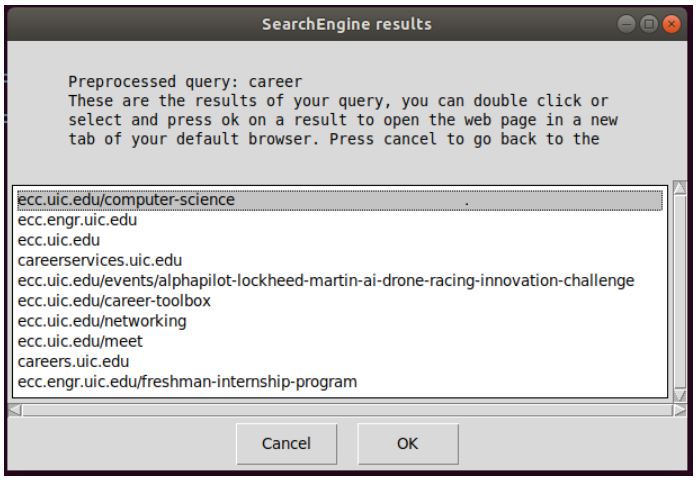

Figure 5. Unintelligent engine output for "CAREER"

Table 3 shows the precision of 5 different queries for the top 10 results of both intelligent and unintelligent search engines.

Table 3. Precision values for top 10 query results

#	Query	Precision @10	
		Intelligent	Unintelligent
1	Liu	0.7	0.6

2	Information Retreival	0.7	0.3
3	CAREER	0.4	0.1
4	caragea	0.8	0.2
5	CAREER services	0.7	0.3

It is clear from the table that the intelligent search engine performs better than the unintelligent one.

9 PROBLEMS FACED

The use of a properly trained neural network would have yielded better weights and would identify more complex connections between the features. However, since the basic regression model which was tried did not work, we had to resort to using manual weights as explained in Section 6. This required a lot of trial and error as a lot of plugging in of weights was done and a "right" mix of weights was to be found between the tf-idf and page rank features.

8. CONCLUSION AND FUTURE WORKS:

The overall performance of the proposed search engine is decent enough. More research on neural networks and learning to rank methods could be done in order to extend the scope of this project and achieve better results. Microsoft uses an algorithm called SVM-MAP [7] instead of regression which is something I would possibly consider trying out in the vacation.

9. REFERENCES:

[1] J. M. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," [Online]. Available: https://www.cs.cornell.edu/home/kleinber/auth.pdf.

[2] S. Brin and L. Page, "The PageRank Citation Ranking: Bringing Order to the Web," 29 January 1998. [Online]. Available: http://ilpubs.stanford.edu:8090/422/1/199966.pdf.

[3] T. Qin, T.-Y. Liu, J. Xu and H. Li, "LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval," *Information Retrieval,* vol. 13, no. 4, pp. 346 - 374, August 2010.

[4] S. Lei and X. Han, "Feature Selection and Model Comparison on Microsoft Learning-to-Rank Data Sets," University of California, Santa Barbara, December 2017. [Online]. Available: https://arxiv.org/pdf/1803.05127.pdf.

[5] K. Dai, "PageRank Lecture Note," 22 June 2009. [Online]. Available: http://www.ccs.neu.edu/home/daikeshi/notes/PageRank.pdf .

[6] C. Burges, T. Shaked and E. Renshaw, "Learning to Rank using Gradient Descent," *International Conference on Machine Learning,* 2015.

[7] Y. Yue and T. Finley, "SVM Map," Microsoft, 31 October 2011. [Online]. Available: http://projects.yisongyue.com/svmmap/.

- relevant to the query and possibly what the user is searching for, I noticed this for the query *"information retrieval"* as explained in paragraph 5.3.