CS494: A Web Search Engine

Sharath S Bhargav

1 Introduction

This work is a search engine for the pages on https://uic.edu domain. It uses techniques like TF-IDF and Word2Vec word embedding and page rank to order the results for a query. A simple front-end is built in python Flask to allow users to enter the query they want to search and view the results .

2 Description

This project involves multiple components. The important ones are described in the following sections.

2.1 Crawler

This component is further divided into several components. First the core crawler. The python packages requests, BeautifulSoup are at the core of this sub component. The crawler is built using a thread pool in python that allows us to scale vertically. A thread pool executor with defined number of threads is created and jobs are submitted to it. There is a global queue that stores the URLs that have to crawled, there is a global list of URLs already crawled and their corresponding web document.
The individual jobs take in a URL from the front of the queue as input and fetch the document at that URL if it is not already fetched. Then the document is parsed using BeautifulSoup library to clean any unnecessary SGML tags and also to obtain the other URLs in the document. These URLs are extracted and added to the list of URLs to be crawled if they are not already crawled. This is a de-duplication mechanism that reduces unnecessary crawling. Further the document text is passed through a pre-processing pipeline.

2.2 Preprocessing

The pre-processing pipeline consists of multiple stages. The pre-processing pipeline mainly uses

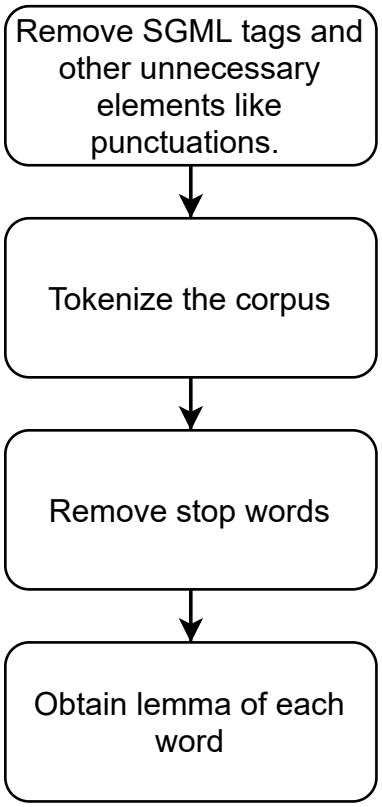

Figure 1: Preprocessing of text

NLTK (Natural language tool kit) library of python. The web document is first tokenized into words using NLTK's Punkt Tokenizer. Then the punctuation is removed since they don't contribute significantly to indexing and searching. NLTK library has a standard set of words which are considered stop words. These are words like "is", "are", "and" etc which don't add value to a text. Such words are filtered from our documents. Subsequently each word in our corpus is converted to its lemmatized form. Lemmatization is a technique where in a word's root form or dictionary representation is obtained. This step reduces the total vocabulary size of our corpus. I am using NLTK's WordNetLemmatizer. Once the document text is pre-processed it is stored along with all the links that the page

contains to other pages. This information gives us the outgoing links that will be crucial for page rank calculation in later stages.

2.3 Similarity Measurement Engine

This component is described in detail in the later section.

2.4 Importance ranking engine

While it is important to fetch pages related to a query text, it is equally important to show credible pages to user for every query. Google has approximately indexed about 50 billion pages as of December 2021. A simple query string can fetch hundreds of sites if not thousands or millions by text similarity relevance. Many of these websites will be spam or setup with malicious intentions. It is important to return websites that are credible and useful to user. This can be achieved in multiple ways. One of the ways is the HITS algorithm which recognizes websites as either hubs of information or a authority of information. An importance of a authority website increases when important hub websites point to it. In the same way, the importance of a hub website increases when it has outgoing links to important authority websites.

$$h(v) \leftarrow \sum_{v \mapsto y} a(y)$$

$$a(v) \leftarrow \sum_{y \mapsto v} h(y)$$

Here $h(v)$ is the hub score of a page v which points to a page y and $a(v)$ is the authority score of a page v which is pointed to by page y. This technique depends on query string and has to be done in real time.

Another algorithm that can be used to rank webpages is Page rank algorithm. PageRank is a system for ranking web pages that Google's founders Larry Page and Sergey Brin developed at Stanford University. This technique assigns every node in a web graph a numerical score between 0 and 1. Working of the page rank algorithm is quite simple yet powerful. Consider a random surfer who begins at a web page (a node of the webgraph) and executes a random walk on the Web as follows. At each time step, the surfer proceeds from his current page A to a randomly chosen web page that A hyperlinks to. Figure 2 shows a toy web graph. If the random surfer starts at node A, the probabilities of him visiting each of the outgoing nodes B, C, D are equal to 1/3. With each time step, the surfer visits a node in a similar model and intuitively it can be seen that, the more number of incoming links a page has, the more frequently the surfer visits that page. This is the idea behind PageRank algorithm: the more frequently a page is visited, the more important it is.

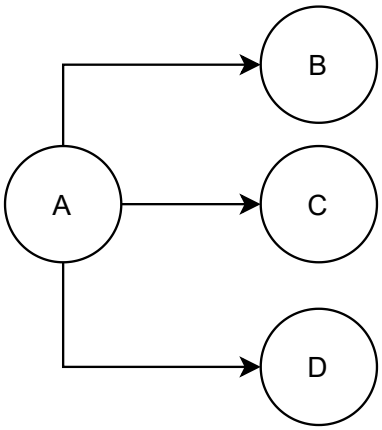

Figure 2: Preprocessing of text

Another aspect that has to be considered is, what if there are no outgoing links from a page. In this case the surfer will be stuck in that page. To avoid this, PageRank adds a small probability α of visiting any random webpage on the graph from any other graph. This action of visiting any page is called teleport operation. It is predefined and equal to some small probability (say $\alpha = 0.15$) divided by number of pages in the graph. The random surfer uses teleport operation in two ways: 1) if there are no outgoing links from a page, the teleport operation is invoked, 2) if there are outgoing links, the teleport operation is invoked with probability of α and the random walk (follow an out-link chosen uniformly at random) with probability of $1 - \alpha$. Thus PageRank of a page is defined as the sum of the PageRanks of all the pages that have outgoing links to this page.

And what it important to understand is that PageRank is all about links. The higher the PageRank of a link, the more authoritative it is. We can simplify the PageRank algorithm to describe it as a way for the importance of a webpage to be measured by analyzing the quantity and quality of the links that point to it.

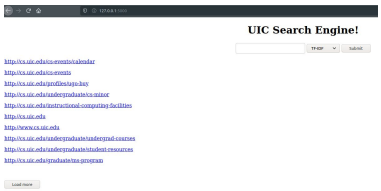

Figure 3: User interface of search engine

2.5 Front-end

The user-interface is a simple web application built using Python Flask framework. The user can enter a search string and choose the type of scoring of web pages: TF-IDF similarity, Word2Vec similarity, Wrod2Vec similarity combined with PageRank. Accordingly the first 10 pages are shown to user. The user has the option of loading 10 more pages by clicking a button.

3 Challenges

One of the main challenges of the building the search engine was writing multi threaded crawler. The crawler had to wait for half a second before crawling next page. The threading library of python was used to write the crawler and run with 4 threads. While crawling filters had to be applied so that both the normal and secured webpages was not stored. In the current implementation only unsecured webpages are crawled. The other major challenge was writing the PageRank algorithm.

4 Word embedding and Similarity measures

Two different type of word embedding has been used: Tf-idf and Word2Vec. The following sections describe each in detail.

4.1 Tf-idf

Term Frequency and Inverse Document Frequency are set of weights assigned to words in a collection of documents. Term frequency depends on the number of occurrences of the term in the document. The simplest way to assign a weight to a term t in document d is assign it the weight equal to number of occurrences of term t. It is denoted as $tf_{i,d}$. This technique suffers from one crucial problem: all terms are considered equal. But usually some terms will not be contributing in determining the relevance of a document but occur in majority of the documents. To counter this effect, we could reduce the weight of a term that frequently appears in a document. This leads us to Inverse Document Frequency (idf).

$$idf_i = log\frac{N}{df_i}$$

where N is the number of documents in the collection and df_i is the number of occurrences of the term i in the document collection. Thus, the idf of a rare term is high while the idf of a frequent term is low.

Tf-idf weightage scheme combines the tf and idf of a term by multiplying and assigns it as the weight of the term.

$$tf - idf_{i,d} = tf_{i,d} * idf_t$$

Thus for the collection of web pages, tf-idf score for each term is calculated.

When a query is issued to the system, all the documents that contain the words in the query string are found. Then for each term in query and for each document in filtered document set, cosine similarity score is calculated using the formula

$$score(q,d) = \frac{\vec{V}(q).\vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}$$

where $\vec{V}(q)$ is the vector representation of query and $\vec{V}(d)$ is the vector representation of document.

4.2 Word2Vec word embedding

Word2Vec is one of the most popular technique to learn word embeddings using shallow neural network. It was developed by Tomas Mikolov in 2013 at Google. Word embedding is one of the most popular representation of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc. The objective is to have words with similar context occupy close spatial positions in a vector space. Mathematically, the cosine of the angle between such vectors should be close to 1, i.e. angle close to 0.

Word2Vec is a method to construct such an embedding. It can be obtained using two methods (both involving Neural Networks): Skip Gram and Common Bag Of Words (CBOW).

CBOW Model: This method takes the context of each word as the input and tries to predict the word corresponding to the context.

Skip-Gram model: This method uses the target word (whose representation we want to generate) to

Figure 4: Query 1

Figure 5: Query 2

Figure 6: Query 3

predict the context and in the process, we produce the representations.

Both of above models have their own advantages and disadvantages. According to Mikolov, Skip Gram works well with small amount of data and is found to represent rare words well. On the other hand, CBOW is faster and has better representations for more frequent words.

In this implementation a pre-trained Word2Vec model trained on news articles from Google News is used. This model is open-sourced by Google and has 3 million terms as vocabulary, each term represented as a 300 dimension vector. For a given webpage the word vectors of all words were taken and averaged to obtain the vector representation of document. Same process was followed for query text. Finding the cosine similarity between a query vector and a webpage vector resulted in number between 0 and 1; 0 indicating query and webpage have nothing in common and 1 indicating that query is very close in terms of context to the webpage.

5 Manual evaluation of results for five queries

The manual evaluation was performed on result set obtained by using TF-idf similarity score. Five diverse queries were run out of which first three were run with Tf-idf and Page rank combination while last two were run with just Tf-idf.

Query 1: The query term was "nlp" and got a precision of 1 for 10 pages.

Query 2: The query term was "undergraduate" and got a precision of 0.6 for 10 pages.

Query 3: The query term was "cta" and got a precision of 0.7 for 10 pages.

Query 4: The query term was "computer science" and got a precision of 1 for 10 pages.

Query 5: The query term was "covid" and got a precision of 0.9 for 10 pages.

6 Results

As can be seen from manual evaluations we are getting good results for some queries. This is partly due to techniques used and partly due to number of pages in the corpus. 3500 pages is quite less for a search engine. The weightage technique that worked was Tf-idf. Although its a simple technique to represent a text corpus in vector space, it has proven to work effectively. The search engine also implements Word2Vec embedding and cosine similarity on vectors obtained using this embedding. It was expected that these word embeddings would work well compared to Tf-idf since Word2Vec takes into consideration term context as well. Surprisingly the results obtained using Word2Vec embeddings was quite poor. By multiplying the Tf-idf weights of each term with the word vectors obtained for the corresponding term yields a weighted Word2Vec representation. This model also performed poorly due to the bad representation of Word2Vec was amplified further.

Page rank of each page was used while ordering the pages. A harmonic mean between the page rank score and Tf-idf similarity scores gave good results as shown in manual evaluation.

Figure 7: Query 4

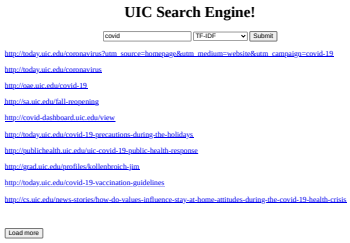

Figure 8: Query 5

7 Related work

There are plenty of search engines out there, Couple of blogs on web crawling and text similarity was researched to implement this project.

8 Future work

In future work the following things can be tried: Increase the number of webpages crawled, experiment with other weighting schemes and ranking algorithms like HITS, investigate why Word2Vec leads to poor results and experiment with it.

References

[1] https://www.worldwidewebsize.com/

[2] *The PageRank Citation Ranking: Bringing Order to the Web. Technical Report. Stanford InfoLab.* Page, Lawrence and Brin, Sergey and Motwani, Rajeev and Winograd, Terry (1999)

[3] *An introduction to Information Retrieval* Manning, Chistopher

[4] https://www.semrush.com/blog/pagerank/

[5] https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa