Web Crawling on UNT Domain

Raghavender Muppavaram
University of North Texas, Denton, Texas
raghavender-raomuppavaram@my.unt.edu

ABSTRACT:

In this paper, the implementation of web based search engine for the University of North Texas (UNT) is discussed. The search engine implements crawler to crawl the UNT domain pages, implements vector space model to index and retrieve the pages for any query given to the engine. Crawler follows BFS approach for crawling the documents.

Keywords

Search Engine, Information Retrieval, Web Crawling, Information retrieval

1. INTRODUCTION

The aim of this project is the implementation of search engine for UNT domain, using vector space model. This report describes the working of the search engine implemented and presents the results of manual evaluation of the system.

This report is used to explain the basic concepts of web search engine for UNT. The designed search engine will crawl around 3000 pages from the UNT domain and the search engine will search and retrieve documents for a given input query.

2. MAIN COMPONENTS

In this section, we will discuss the various components that are used for implementing the search engine. Crawler implementation for this project obeys Breadth first search order for crawling web pages. The components that are used in the implementation are.

- Web Crawler.
- Text processing and Indexing
- Retrieving and Ranking.

The components shown in the architecture are discussed in detail in the following sections. The project represents the development and integration of the components that are shown in figure 1.

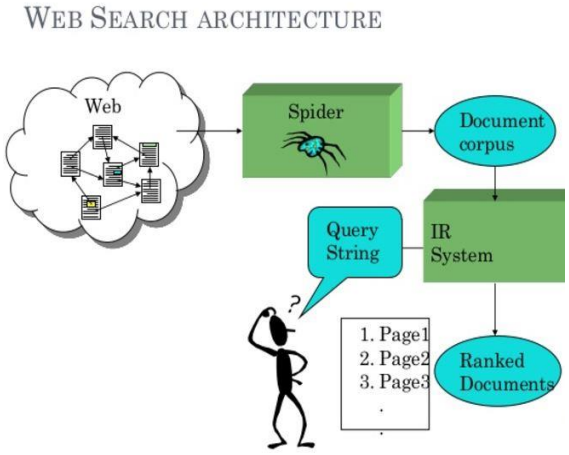

Fig:1

2.1 Web Spider

The implemented crawler will crawl the web pages of UNT domain (http://www.unt.edu). Breadth first strategy is used to keep track of previous visited link. Beautifulsoup package is used to parse the html pages. The parsed URL's are then stored into a SQLite database. The html pages are converted in to a plain text, the tags are all removed and the content is saved as a file. The file is saved as the ID of the URL. The URLS are handled and are changed to lower case.

2.2.1 Text Processing and Indexing

In this process, the downloaded html pages are preprocessed and indexed for efficient retrieval of documents. Vector space model implemented for the project deals with processing the content of the crawled URLs to build an inverted index for this crawled document collection. Elimination of irrelevant tags and processing of tokens is also handled in this module. The documents and user query are treated as vectors and document vectors closer to the query vector are more like the query and are displayed to the user. Tokenization, stemming and preprocessing of text are included in this process.

The character sequence from each document is tokenized by splitting into tokens with each token having a semantic meaning. Splitting is done on whitespace by throwing away the punctuation. If there are any stop-words they are removed by stemming.

The next step is indexing, the tokens that are obtained from the preprocessing module forms vocabulary. Each entry for each token in the vocabulary keeps a list of all the documents where it appears together with the corresponding frequency which forms Term Frequency (TF) and also keeps the total number of documents in which the corresponding word appears which forms Document Frequency (IDF). A dictionary (map) is maintained for each entry as a key and values is a posting list containing first element as IDF and second element again a dictionary whose key is document name and value is TF of that entry. The term frequency is normalized by formula

$$TFnorm = tf / max\_tf.$$

The IDF is calculated for each document as follows

$$IDF = \log_2(N/df).$$

A typical tf-idf weighting for each token i and each document j is given as follows,

$$wij = tfij * idfi$$

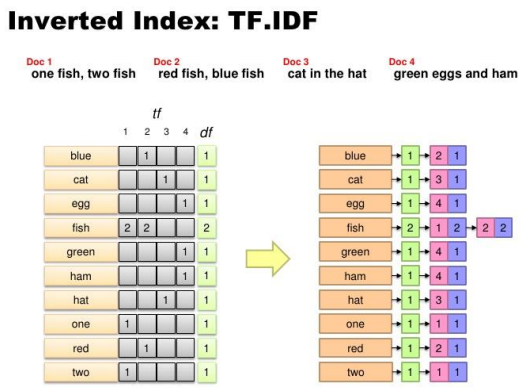

Fig:2

2.2.2 Retrieval and Ranking

The documents are retrieved by comparing the similarity between each document with the query. Similarity measure computes the degree of similarity between two vectors which are represented as query vector and document vector. We use cosine similarity measure to measure the similarity. Cosine similarity computes the cosine of the angle between the two vectors. Inner product of the vectors is normalized by vector lengths.

$$CosSim(d_j, q) = \frac{\langle d_j, q \rangle}{\|d_j\| \cdot \|q\|} = \frac{\sum_{i=1}^{t} w_{ij} w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$

Similarity between vectors for the document d_j and query q can be computed as above. Where w_{ij} is the weight of term I in document j and w_{iq} is weight of term I in query. For each document cosine similarity is calculated with respect to given query and are stored in a dictionary where key is document and value is similarity measure value. The dictionary is then sorted with similarity measure in descending order.

3. IMPLEMENTATION

This section provides description of the python implementation of the major components of the search engine. A brief explanation to the execution of the software implementation has also been presented here.

3.1 Tools Used

The following tools have been used for implementation:

Programming: Python

Libraries: Beautifulsoup

Editor: SublimeText

3.2 Modules

Following is a description of the modules implemented in the search engine.

Crawling.py: This python code is the implementation of the breadth first search crawler that crawls within the UNT domain and returns a list of URLs crawled. The implementation also keeps track of previously visited links to avoid duplication. And also it makes use of the BFS crawler to get the crawled URLs and download the content using Beautifulsoup library.

stemming.py: This file is used for stemming, which is downloaded https://tartarus.org/martin/PorterStemmer/.

Main.py: This is the main file which we run to get the output. This file reads an input string and returns the top search results based on input.

Tokeniser.py: Extract terms from the documents in a directory and calculates terms frequency.

3.3 Executing the Search Engine

The implementation can be executed with the following options:

Using Existing Index: A total of 3000 UNT webpages have been crawled and the document content is present in the folder. This document collection has been indexed with the Inverted Index.

Main.py file is responsible for executing the application. We should input the query when its asked.

4. CHALLENGES

In this section, the challenges faced while implementation are discussed.

Time complexity, precisely speaking about the time taken by the web crawler to parse the web sources is around 60 minutes. Some other major challenges are handling the duplicate pages and URL's.
.

5. RESULTS AND EVALUATION

The implemented search engine search is limited to only UNT domain. The experiment results show that the implemented method is efficient in retrieving the pages. To evaluate we will take the help of a metric called precision. Here Precision is nothing but the ratio of relevant documents retrieved to total number of documents retrieved. And results are presented in the table 1.

Table 1. Table captions should be placed above the table

Query Given	Precision
Cornelia Caragea	1
Computer engineering	1
Rec	0.7

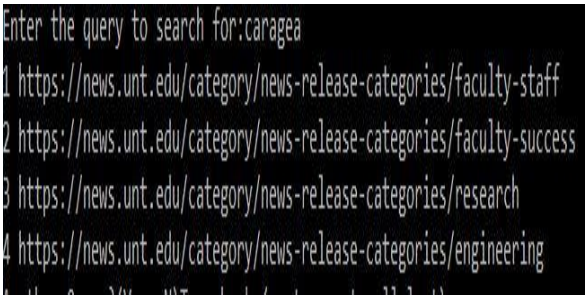

Fig3: Output for query 1

Fig4. Output for query 2

Fig5. Output for query 3

7. CONCLUSION

Search engines have become part of everyday life. The information required by the people is growing so search engines are alternatives for gathering quick information on a given topic. This project aims at implementing a simple vector space retrieval model based search engine that searches for user input query in the UNT domain. This project is an implementation of search engine based on a specific domain. It gives the basic idea on how information retrieval plays a key role in building a search engine. Implementing this system can give us a good understanding of how a search engine works and also addresses some challenges that we face when crawling the data from the web.

7. FUTURE WORK

There is always scope for improvement as new things evolve and problems get addressed. As part of future work, we need to develop a hybrid model by integrating probability based models and vector space model. Which possibly may increase the precision of the documents. Also crawling more number of documents so the scope for search also increases. Implementation is console based. I would like to extend the implementation as a web application.

References:

[1]. The Porter Stemming Algorithm

http://tartarus.org/~martin/PorterStemmer/

[2]. Cornelia Caragea. CSCE 5200 Information

Retrieval and Web Search Lectures, 2016

[3]. Session

https://en.wikipedia.org/wiki/Session

[4]. Serialization

https://en.wikipedia.org/wiki/Serialization.

[5]. C. D. Manning, P. Raghavan, and H. Schutze. Introduction to information retrieval, volume 1. Cambridge university press Cambridge, 2008.