UNT Search Engine using Vector Space Model Build using Java

Sai Madhu Bhargav Pallem
Department of Computer Science
University of North Texas
Tel: +1 (940) 208 8229
sp0723@unt.edu

ABSTRACT

This project shows the vector space model [1] retrieval of the pages belonging to the UNT Domain. These pages are downloaded using the JSOUP java library, which contains various functions for opening a html connection and downloading it to a Document object. These documents are then saved into a local directory in the form of a XML Document. These documents are then given as input to the vector retrieval function which will build the document index. After the index is built, the search query is sent to the same function, which will calculate the cos similarity between all the documents and the given query. Then a new XML Document is created which will contain the results of the given search query in the descending of the cos similarities. This XML document is then used to display in a JEditorFrame in the GUI developed in Java (on NetBeans IDE).

CCS Concepts

- **Information Systems → Web Crawler**
- **Information Systems → Vector Retrieval**

Keywords

Search Engine, Web Crawler, Vector Space Model, jFrames, BFS.

1. INTRODUCTION

This project is developed to index the websites in UNT domain and develop a search engine for the same. This crawler is developed using the Breadth First Search Traversal starting from the http://www.unt.edu as the root node. Then these documents that we get by performing this BFS [3] are saved into a local directory and these documents are used for indexing. The steps used in processing these files are discussed in the following sections.

1.1 Web Crawler

Web crawler is a program that will take a page as root and starting from this page. This root page will be parsed in order to get all the links available in this page and save all the links in a stack or a queue based on the type of crawler. If the crawler is using BFS [3], then a queue is used and if it is a DFS crawler, it uses a stack. The Web crawler is implemented using the Jsoup Libraries [7], which will open a connection to the required url. After a connection is established the link's html is downloaded into a document object. This document object is later parsed to get the links that are present in the href attribute of the anchor tag. Among these links, we have to filter out the links that are only valid. Sometime, we need to check if the link is dead or not also weather it is working or not. All such exceptions must be handled properly. We also should take care that, all the links are within the UNT domain and must not let the crawler to go outside the UNT domain. Then the valid links that are obtained from one link are then added to a queue for processing next. Before adding a link to the queue, it is checked with the list that contains the list of processes links, if a particular link is present in this list, then it is not passed into the queue. At the same time, when the links are being processed, the text form the document is been extracted and saved into a file in a Local directory, along with the document Title, URL, text and Last modified date, in the form of SGML. Each file is saved as an SGML because, SGML files can be easily processed and maintained by a Java code. This part of the code is only done once and later on these files are used to build the index for vector retrieval model.

1.2 Stop word removal and Porter Stemmer implementation

Each document is parsed and checked for stop words in it. All the stop words are removed from the document. Each word in the document is then provided as an input to the porter [5] stemmer function which returns the word after performing the porting and stemming on the word. This is done in order to reduce the number of distinct words in the document, which in turn reduces the number of dimensions in the vector space model [1].

Consider an example, where he words tree and trees. If the word, tree is present in a document for 100 times and the word trees, is repeated for 50 times, then after performing the porting stemming on these two words, we get the word tree for both the words. Hence, resulting in the word tree repeating for 150 times in the document. Also in a document, the words like the, an etc. are often repeated and contribute majorly to document length. So, we need to remove them.

1.3 Vector Space Model

The vector space model [1] is a vector representation of the document, where each distinct word is a different dimension and the weight of these words is the magnitude of the

corresponding dimension. In this model, the words are represented as vectors only after performing the stop word removal and performing the porter stemmer [5] on these words, so as to reduce the number of dimensions. Each document is represented as a vector with different number of dimensions in this model. Along with the documents, all the search queries will also be represented in the similar manner. By doing so, we can find the dot product of the query and each document, which will provide us, the angle present between the document and query. This process is called as calculating the cos similarity between the document and query. The search results are returned based on these cos similarity values.

Example for vector space model:

Table 1: Table showing TF - IDF values of a document

Word	Tf - Idf
Titanic	9.2
Ship	5.6
Sink	5.2
Atlantic Ocean	3.1
Drown	2.8

This table shows the tf-idf values of 5 words present in the document Titanic. Now, the vector representation of this document will be as follows:

*9.2*Titanic + 5.6*Ship + 5.2*Sink + 3.1*Atlantic Ocean + 2.8*Drown.*

The formula for calculating the TF – IDF value for a word present in a document is as follows:

$$tf_{ij} = \frac{f_{ij}}{\max(f_{ij})}$$

Tf is the term frequency, f is the frequency of the word i in document j, max(f) represents the frequency of the word that is having maximum frequency in document j

$$idf_i = log_2\left(\frac{N}{df_i}\right)$$

Here idf is the inverse document frequency of the word i, N is the total number of documents in the collection, df is the document frequency of the word i in entire collection. Finally, the weight of each word in the document is calculated as follows:

$$w_{ij} = tf_{ij} idf_i$$

$$w_{ij} = \frac{f_{ij}}{\max(f_{ij})} * log_2\left(\frac{N}{df_i}\right)$$

Where w represents the weight of the word i in document j. Instead of taking the entire frequency of the word in a document, we consider the weights because, this will reduce the complexity of multiplication and when a query is given, it has to multiply the frequency, which will result in greater processor speeds if larger numbers need to be multiplied. Hence we use the weights of each document instead of using the frequencies of these words. The entire collection is only parsed single time and at that time only, all these values are being calculated. Which in turn will reduce the computation costs. After this step, we need to perform the cos similarity between the query and every document present in the collection.

1.4 Cos Similarity

The cos similarity is the method used to find the angle that is present in between any two vectors, as our entire collection and the query is represented in the form of vectors, we can use this cos similarity method in order to find out the similarity between the documents and the query. The document having the highest cos similarity value will be much similar to the given query. For this, we will first perform the dot product between the query vector and the document vector. Then divide this dot product with the magnitude of the query and magnitude of the document. The resultant value is the applied with the inverse of cos, which will give the angle present in between the document and the query. The document having the last value for this angle, is considered as the document that is most similar to the given search query.

The formula for finding out the cos similarity of the document and the give query is as follows:

$$CosSim(d_j, q) = \frac{\langle d_j, q \rangle}{||d_j|| \, ||q||}$$

Here CosSim is the cos similarity, d represents the document and q represents the query, on Right Hand Side of the equation, we can see dot product of the document j and the query on the numerator and in the denominator, we can see the magnitude of both document and the query. The same equation can also be represented as follows, using the weights of document and query.

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

In this formula, w_{ij}represents the weight of the word in document j and w_{iq}represents the weight of the word in query.

1.5 SGML and XML

The entire document collection is represented as the form of a SGML file, which contains, the title of the document in the TITLE tag, text in TEXT tag and url in the URL tag. This is done in this way because, there are many SGML parsers present in java and we can use them to retrieve back the data form them hence we are using the SGML document types.

Once the query is given to the program, it calculates the cos similarity of query and the document, which is used to get the relevant documents. After getting the cos similarity values of the document and the query, these results are sorted according to the Cos Similarity values and the information of these documents are saved in an XML file that is present in the local directory. After the XML file is saved into the local directory, this file is then parsed and converted into a HTML document, that is given as input to the JEditorFrame which is set to type HTML. In this JEditorFrame, only 4 results are displayed at a time and all the results are present in the form of pages, user

need to press the next and previous buttons in order to navigate through these pages.

2. INTERFACE

The entire interface of the project is done using Java Swings and Abstract Window Toolkit. The swings are used to develop a text box input which will take the search query as input. Two buttons are also developed using the swings, where one button is used to invoke the query processor method, which will return the relevant results to the given query. Another button is used to update the databases present with us. These will again download all the files from the web again but will take a lot of time based on the Network speed. Two other button are used, which will be used for navigating through the search results (Next and Previous buttons). All these functions are done using threads. Each function will create a new thread that will be running parallel to the other threads, resulting in less time required in processing the query.

A JEditorFrame is also used in the interface, which will be used to show the results of the query in it. This JEditorFrame is set to type html, in order to display the html content in it. After XML file is generated to the given search query, the results are displayed in this using another function, that will develop the html equivalent to the generated search query. Only four results are displayed in the Frame at a time. When the user presses, the next or previous button, the next set of results will be rendered on to the JEditorFrame window. The search results are displayed in Google style, where the title of the document is present, and it is a link to the equivalent page. Below the title, a small description about the page is also shown. Once the user clicks on a link, it will open in an external web browser that is present in the user's computer.

As this uses Abstract Window Toolkit, the project is only confined to Windows operating system and cannot run on any other operating systems.

The output screens and their description is discussed in the following section.

3. IMPLEMENTATION

3.1 Requirements

The requirements to run the project are as follows, these requirements are mandatory in order to run the project:

Operating System: Any Windows Operating System.

Platform: A Latest Java Virtual Machine is required to run the project.

Space Required: 60 MB of HDD space is required to maintain the files that are downloaded from the web.

RAM: 4 GB of RAM is required, but 8 GB is recommended for the faster performance results.

Network: A working Internet connection is required to implement the update Databases module. High speed Internet Connection is recommended.

3.2 Specifications

The following shows the Specifications of the PC that I've used in order to evaluate and test the results on. All the results discussed in this paper are completely based on the results that are noted from the system with following specifications.

Operating System: Windows 10 Home Operating System.

RAM: 8 GB

Processor: Intel Core i7 – 5500U CPU @ 2.40 GHz

Network: High Speed Internet connection at 60 Mbps

Data: 10, 000 files belonging to the UNT Domain are downloaded and are used as the data source for this project.

3.3 Main Components

The main components of the project are discussed in this part of the document. Various classes used in this project are as follows (In the order of execution):

initLoadingPage.java: This file contains the code for building the indexes of all the documents from the data source. It also contains the code for GUI, which displays the progress bar along with the number of files, that are currently been processed and the elapsed time is also shown at the bottom left corner of the UI.

Time: it takes about 55 – 60 Secs on an average for processing the 10 000 files present in the Local Directory.

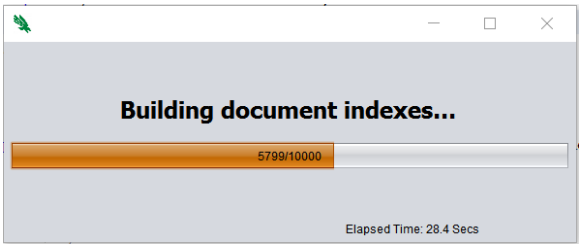

Figure 1:UI shown to the user while building the document indexes

VectRetrieval.java: This java class is used to build the document indexes, this will take all the document present in the local directory, read them and construct a vector space model for all the files. This will calculate the tf, idf and tf-idf values for all the words in the document and develop a vector space model [1] for all the documents. The same java file contains the code for performing the cos similarity measure and using it to find the relevant results and save them in a XML file which is later used to display in the GUI.

MainWindow.java: This file contains the code for the main UI of the project, it contains the text filed which takes in the query string as input and sends the query string to the query Analyzer function in VectRetrieval.java file, which will calculate the cos similarity of the query with all the documents in the collection. Later these results are shown in the HTML file in the main UI, where a JEditorFrame is preset and it renders the HTML content into it. Below the

JEditorFrame, there are two buttons, one for previous page and one for the next page. This will change the results in the JEditorFrame, also by showing the page number.

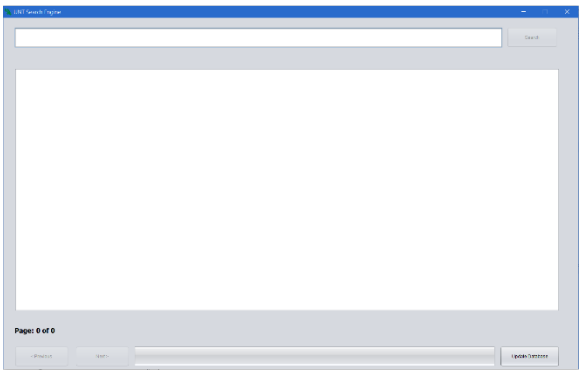

Figure 2: Basic UI for the Search Engine

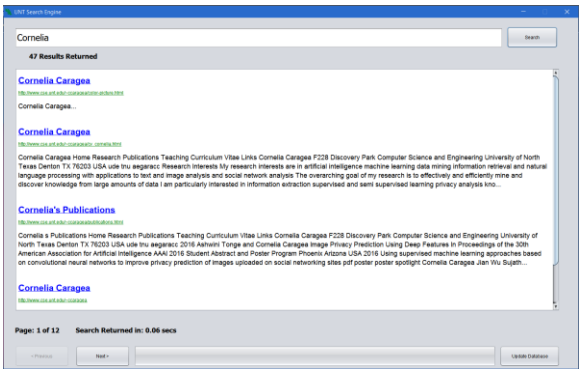

Figure 3: Showing results for the search query "Cornelia Caragea"

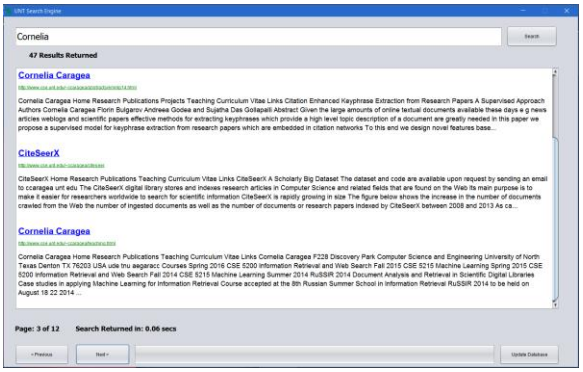

Figure 4: Showing page 4 of the previous Search Query

These images show the search results for the query, "Cornelia Caragea", these also show, that the search has returned in 0.30 secs and also that the search has returned 40 results, i.e. among the 10000 documents in our data set, 40 documents are having cos similarity value greater than zero (0). Hence only those results having cos similarity non zero are shown as the relevant results. The blue ones are the title of the pages, which are present inside the anchor tag, so when you click the title, it will open the link in an external web browser. This is shown in the following image:

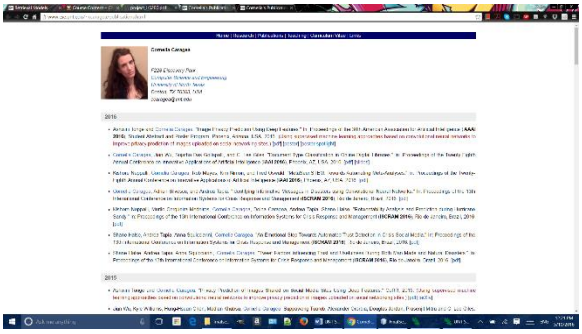

Figure 5: Link opened in my default web browser (Google Chrome)

Similarly, when a non-relevant query is entered, the cos similarity of the query with all the documents will be resulting in a zero, hence no results will be returned. When no results are returned, all the UI displays the message saying "No results are returned".

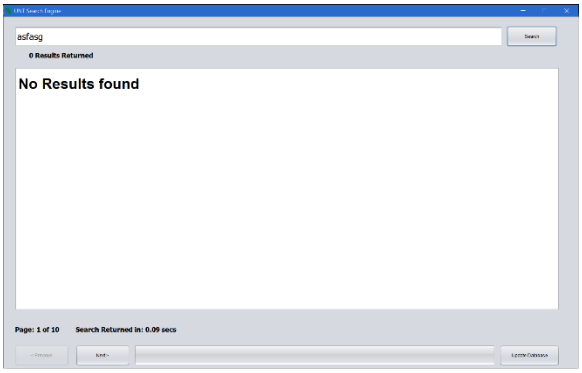

Figure 6: No results returned for a non-relevant query

A UI is also developed for displaying the progress of updating the databases. There is a progress bar available in the UI, that will display the current percentage of the documents that are downloaded.

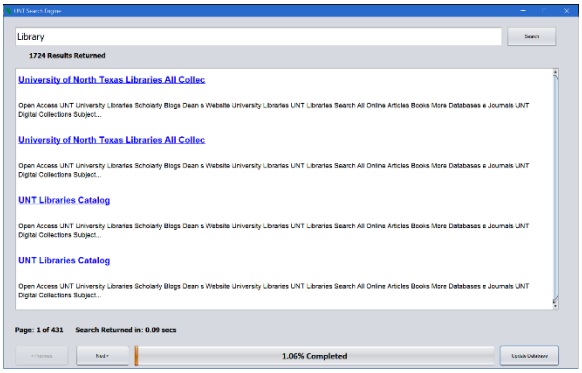

Figure 7: Progress bar showing the percent of files that are currently being updated

4. CHALLENGES

The main challenges that I have encountered during the implementation phase of the project are:

Filtering the URL: For this, I have developed a function which will filter the unwanted URLs, that are not in the UNT domain, that are not html files, pages that contains links to content within the page. All such URLs are filtered out by my code.

Checking for loops and duplication: Sometimes, different URLs redirect to the same page, this will result in duplication of data. So, for this I considered the pages that are ending with 'index.htm' or some similar extensions and filtered out them. Such links will also give scope to form cycles in the graph. Hence, by handling such links, I've got rid of loop and duplication of the URLs.

Page Redirecting: Some pages like *http://www.cse.unt.edu* will have a meta tag that have an attribute 'HTTP-EQUIV' which have a value 'REFRESH'. All such links will also have a 'CONTENT' attribute, which contains the value in the form of zero followed by the link to redirected page, we need to extract this link and append it to the base URL in order to get the actual html of the page. For the CSE webpage, the meta tag information is as follows:

```
<META HTTP-EQUIV="REFRESH" CONTENT="0;
URL=/site/index.php">
```

From this tag, we need to extract the text present after the URL = and append it at the back of http://www.cse.unt.edu, which will result in http://www.cse.unt.edu/site/index.php.

These are the challenges that I've encountered while I implementing the crawler.

5. POSSIBLE ALTERNATIVES

Probability Distribution [4]: The vector space model will look for the number of time the word is present in the document and return the cos similarity of the query with all other documents. If a particular query term is not present in the document, then its weight is given by zero. So, instead of using this Vector Space Model, if we use Probability method, we can get much better results.

Bi Gram or n-Gram models: In this project all the terms in the documents and the query terms are all considered as unigrams. Instead if we use any bigram or n-gram model in building the document index, we can get much better results.

Authorities and Hubs: All the pages are given the same ranking in the project I've implemented. Instead if we use Authority and hub scores to rank the pages based on the number of in links and out links, we can expect a better ranking of the results.

All the results work good when the query is short. If a long query is given, it returns the pages that contain the query terms and more number of pages are returned.

6. EXPERIMENTAL RESULTS

The evaluation of the following three queries for the top 5 results is shown as follows:

Query: "Information Retrieval"

URL Returned	Relevant or not
http://www.cse.unt.edu/~ccaragea/cse5200.html	Yes
http://courses.unt.edu/acleveland/homepage/introduc.htm	Yes
http://www.cse.unt.edu/~ccaragea/teaching.html	No
http://www.cse.unt.edu/~ccaragea/russir14.html	No
http://courses.unt.edu/YDu/slis5600.htm	Yes

Table 2: Top 5 results for Query "Information Retrieval"

Precession: 3/5 = 0.6

Query: "HiLT Lab"

URL Returned	Relevant or not
http://hilt.cse.unt.edu	Yes
http://hilt.cse.unt.edu/about.html	Yes
http://hilt.cse.unt.edu/Undergrads.html	Yes
http://hilt.cse.unt.edu/news-events.html	Yes
http://hilt.cse.unt.edu/academics.html	Yes

Table 3: Top 5 results for Query "HiLT Lab"

Precession: 5/5 = 1.0

Query: "Bruce Hall"

URL Returned	Relevant or not
http://reslife.unt.edu/residence_halls/bruce_hall	Yes
http://housing.unt.edu/residence_halls/bruce_hall	Yes
http://www.dining.unt.edu/brucehall	Yes
http://housing.unt.edu/staff_directory/building/Bruce%20Hall	Yes
http://calendar.unt.edu/celebrate-may-4th-be-you-star-wars-day-bruce-cafeteria	No

Table 4: Top 5 results for Query "Bruce Hall"

Precession: 4/5 = 0.8

The precession for the given three queries are 0.6, 1.0, 0.4. So, the Mean Average Precession [2] of the given three queries is given by:

$$MAP = \frac{0.6 + 1.0 + 0.4}{3} = \frac{2.0}{3} = 0.667$$

$$\therefore MAP = 0.667$$

This shows the precession for the top 5 results is above 0.5 on an average. But if we go beyond top 5, we can get reduced precession values.

7. CONCLUSIONS AND FUTURE WORK

The web crawler is developed with the vector space model, implementation and the crawler is developed with BFS crawling. This can further be improved by implementing the probability distribution [6] model or applying the Authority and Hub scores algorithm. The crawler will download 10000 files to the local directory and these files are used to build the index. When a search query is sent, it calculates the cos similarity with the built indexes and returns the documents in the descending order of the cos similarities.

8. REFERENCES

[1] Vector Space Model. 2015. *http://www.cse.unt.edu/~ccaragea/cse5200/lectures/lecture5-invidx.pdf*

[2] Mean Average Precession. 2015. *http://www.cse.unt.edu/~ccaragea/cse5200/lectures/lecture5-invidx.pdf*

[3] Breadth First Search. 2015. *http://www.cse.unt.edu/~ccaragea/cse5200/lectures/lecture10-crawling.pdf*

[4] Probabilistic Models for Information Retrieval. 2015. *http://www.cse.unt.edu/~ccaragea/cse5200/lectures/lecture12-probIR.pdf*

[5] Text Processing and Text Properties for Porter Stemmer. 2015. *http://www.cse.unt.edu/~ccaragea/cse5200/lectures/lecture3-text.pdf*

[6] Hema Raghavan, Rukmini Iyer. Evaluating Vector-Space and Probabilistic Models for Query to Ad Matching. *ACM Library*. Published Online. 2007.

[7] JSOUP Documentation for Functions and classes in JSOUP. https://jsoup.org/apidocs/org/jsoup/nodes/Document.html