Retrieval Engine for The University of Memphis

Anjana Tiha
Department of Computer Science
University of Memphis, Memphis, TN
Email: atiha@memphis.edu

Supervised by: Professor Visali Rus

November 28, 2017

Introduction

Web search engine is a system that allows user to search world wide web using using a interface. Search or retrieval engine allows the user to retrieve information from worldwide web based on provided query. Retrieved document can be in the form of web page(HTML, PHP, ASP) or other document format including text, doc, docx, pdf. There are other retrieval option for different file format like image, audio videos. Closely coupled term information retrieval is another technique to obtain relevant information from a collection of information resources according to the information need. Information retrieval is very important and powerful subfield especially in current era. The world is filled with overwhelming amount of information. People use world wide web, electronic media heavily. Source of information can be books, periodicals, WWW, memos, ads, published/refeered Film, Photos, other Images, Broadcast TV, Radio,Telephone Conversations, Databases. This platforms are filled with information that can easily cause information overload. Also, in our daily life we are constantly taking in information from around us both actively and passively. All these accumulated information can lead to overwhelming stress. They can also cause decision paralysis. It has been seen that in presence of too much information people often fail to make decision and their productivity reduces significantly. Therefore building better information retrieval system and improving retrieval techniques is crucial. To understand

Lets take a look at part of information source.

The annual production of printed media:

- 968,735 Books (8 Terabytes (compressed image)
- 22643 Newspapers(25 Terabytes)
- 40000 Journals(2 Terabytes)
- 80000 Magazines(10 Terabytes)of 357 terabytes
- 12x109 pages Office Documents (312 Terabytes)

There is a total of 357 Terabytes of printed information. Library of congress has 3 petabytes of information including books, audio, maps and images. The books in print we can buy today is 26 terabytes. Annual production of photographs is 410 petabytes and film is 16 terabytes. Including information in print, media, optic media and magnetic media, there are 3,277,440 terabytes of new information stored per year.
This vast amount of information makes it difficult to find the useful information.

Information retrieval is can be used for retrieving useful information from web with better performance and precision. Information retrieval can be confused with data retrieval. Whereas data retrieval has well defined semantics and sensitive to error and checks for key words, information retrieval has flexible semantics and more tolerant to errors and is focused on topic, subject or category.

There are couple of information retrieval models.

- Boolean models (set theoretic)
  - Extended Boolean
- Vector space (VS) models (statistical/algebraic)
  - Generalized
  - Latent Semantic Indexing
- Probabilistic models

Web document indexing and searching can be difficult due to rapid change that occurs in web every seconds. So, popular search engines often crawl though news and rapid changing media websites more often than other webpages.

For improvement of information retrieval other techniques like natural language processing and machine learning can be used. Using history of user or trend in web machine learning can help retrieve more relevant information. Also, natural language processing can be used to improve information retrieval by using semantic information.

In the project the "Retrieval Engine for The University of Memphis" I have built a search engine using Python programming language. Interface was developed using Python web framework Django.

Approach

For this project, documents were collected from world wide web of domain "memphis.edu". Raw documents including webpages(including php, asp along with other types), text document(.txt extension), PDF files were collected. The custom web crawler crawled in breadth search manner. All the documents in their raw format was saved in local directory after markup and script removal. While saving each document, PDF documents were converted to text removing the PDF markups and

saved in local directory. After convertion of a document to text, each was pre-processed by removing everything other than Alphabetical letters, URL strings, stop words and performed stemming. For information retrieval task, vector space model was used. TF-IDF was generated from documents. For query assimilation, cosine similarity metrics was chosen. For similarity only documents with terms of query was chosen for performance. Returned the the documents with highest similarity score in descending order and urls was shown in the web page developed using Django framework.

For TF-IDF following concepts were adopted:

- More frequent terms in a document are more important, i.e. more indicative of the topic
  f_{ij} = frequency of term i in document j
- normalize term frequency (tf) within document:
  $tf_{ij} = f_{ij}/max_l(flj)$
- Terms that appear in many different documents are less indicative of overall topic.
  - df_i = document frequency of term i
    = number of documents containing term i
  - idf_i = inverse document frequency of term i,
    $= log_2(N/df_i)$
    $(N : totalnumberofdocuments)$
- A typical combined term importance indicator is tf-idf weighting:
  $w_{ij} = tf_{ij}idf_i = tf_{ij}log_2(N/df_i)$
- Cosine similarity:

$$\cos sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2}\sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

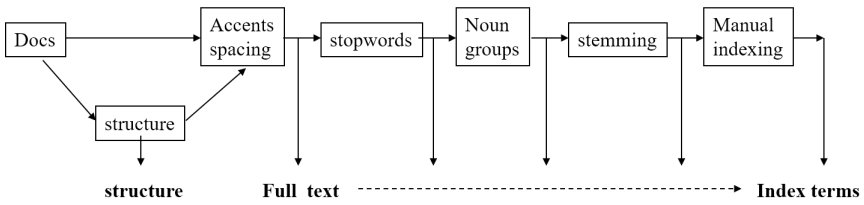


Figure 1: Information retrieval Preprocessing

Design

The program first crawled the memphis.edu web page. For crawling, I have built a crawler that crawls iteraively using a queue. Each time of visit, the crawler dequeues a link from queue and visits the page. After visiting, puts all the link present in current page at the end of queue. Crawler crawls in iterative manner and dequeues a link queue from queue for visiting. This way most important pages that is at the top of page hierarchy gets visited first. While visiting the pages, web page text and pdf are fetched using standard library function. Web pages are cleaned using beautifulsoup. Also, all the scripts are mark ups are removed. PDF files are saved in local directory. Then converted to text using texttopdf tool. text files are saved directly. Each document is converted, fetched, preprocessed at the time of fetching. After processing, urls of documents are saved in pickle file. Text files are tokenized after removing accents and stop words. Words are then stemmed and all the word in a document is saved in another text file in processed directory if number of word is greater than 50. Also, inverted document index is built during document processing. In the TF-IDF vector generation stage, all the documents are used to built the TF-IDF vector. Query preprocessing is similar to document preprocessing. After tokenization, TF-IDF is generated for query. Then document cosine similarity is calculated for all the documents where query keywords are present. Then return a list of document with descending order of similarity score. Then, the corresponding urls for documents are shown from most relevant to least relevant. For web interface, Django web framework was used. For interacting with the core program with Django framework intermediate scripts were generated.

Implementation

Program collected 10, 000 documents from web domain (memphis.edu). The collected documents included raw webpages, text(.txt) files, PDF files.

Webcrawling:

For web crawling iterative breadth search approach was followed. Each web page address was first queued in a global queue. When visiting the page, all the links present in the page was added to the end of the queue. After completing web document fetching from page, the program dequeued a link from the beginning of the queue and visited the web page. For the following pages, any link present in the current page was similarly added to the end of queue. The program ran iteratively until the queue was emptied or number of proper documents was greater than 10,000.

Document Preprocessing

Each web document was completely preprocessed before fetching another. For each page raw html was fetched using python library urlib. The mark up and scripts was removed using python Beautifulsoup library. Then text was saved in local directory. Text documents were saved directory and preprocessed similarly as web pages. For PDF files, they were first fetched from internet and written as PDF file in local directory. Then they were converted using "texttopdf" tool to convert to text file and next they were preprocessed.
When collecting the documents, following text operations were performed to preprocess each document.

1. Convert PDF to text files.
2. Removal of meta information, HTML tags, Scripts or other structure information.
3. Removal of the followings from text after initial preprocessing:
    - digits
    - punctuation
    - stop words
    - urls and other html-like strings
    - uppercases
    - morphological variations
4. Tokenized the text
5. If more than 50 token was available in a document, then saved the tokens of the document in processed file directory.

Vector Space Modeling of Document

Generated inverted document index for documents containing mapping from word to document. Also during document saving document name to url mapping was also saved using python pickle file for later use. Generated TF-IDF vector for the document collection using hash map. TF-IDF was developed iteratively for each document using global hashmap although normalization was done at the end of processing. Two hash maps (term, document, term_count_in_document) and (document, term, term_count_in_document) was generated and saved using pickle.

Query Processing

Input query was taken from web interface. Query was preprocessed similar to other documents. Then generated TF-IDF vector for the query. Documents containing the query tokens was then selected for cosine similarity calculation. After calculating cosine similarity, documents were ranked based similarity score and arranged in descending order in a list. Preprocessed the query similar to document preprocessing. Showed the corresponding urls for matching ranked documents from most similar to least.

For preprocessing the following was removed for query.

- digits
- punctuation
- stop words
- uppercases
- morphological variations

Web Interface

Developed web interface using Django open-source web framework. To use search/retrieval engine using the web interface, Django web framework is required.

Following are the steps for installing Django Framework:

1. Install pip for python if not installed already.
2. Move to python directory or scripts directory in Anaconda.
3. Enter -"pip install Django" for installing Django.

To open project in web interface:

1. To runserver for current project go to project folder "search_engine_website" where manage.py file is located.
2. Open command prompt in the directory where manage.py is located and type manage.py preceded by absolute location of python.exe and python in the following manner:

3. `C:\Users\Anjana\Anaconda3\pythonmanage.pyrunserverserver`
4. `(format->locationofpython.exe\pythonmanage.py)`
5. Finally to use the web search engine interface, go to http://127.0.0.1:8000/

Figure 2: Web Search/Retrieval Engine Interface

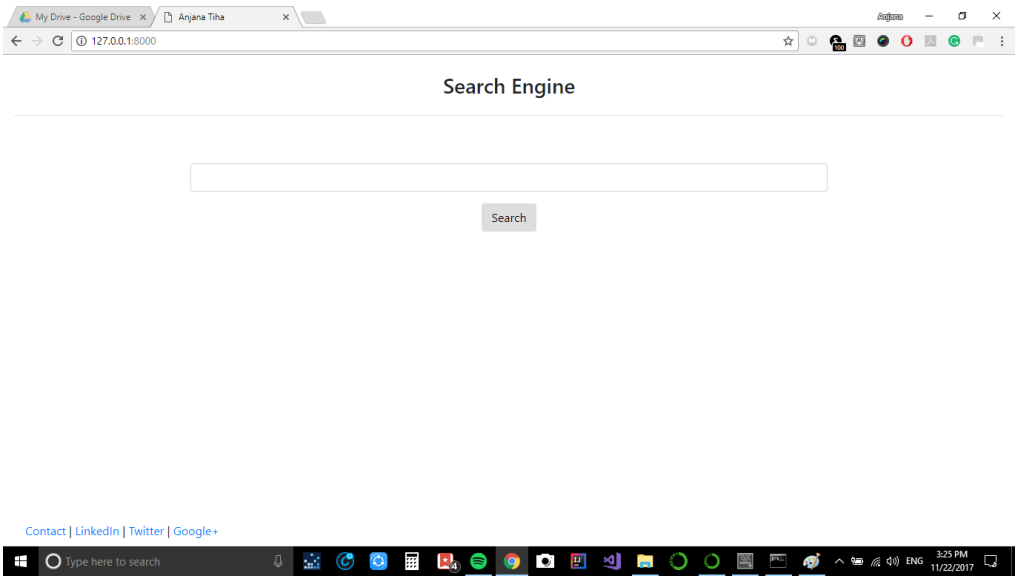
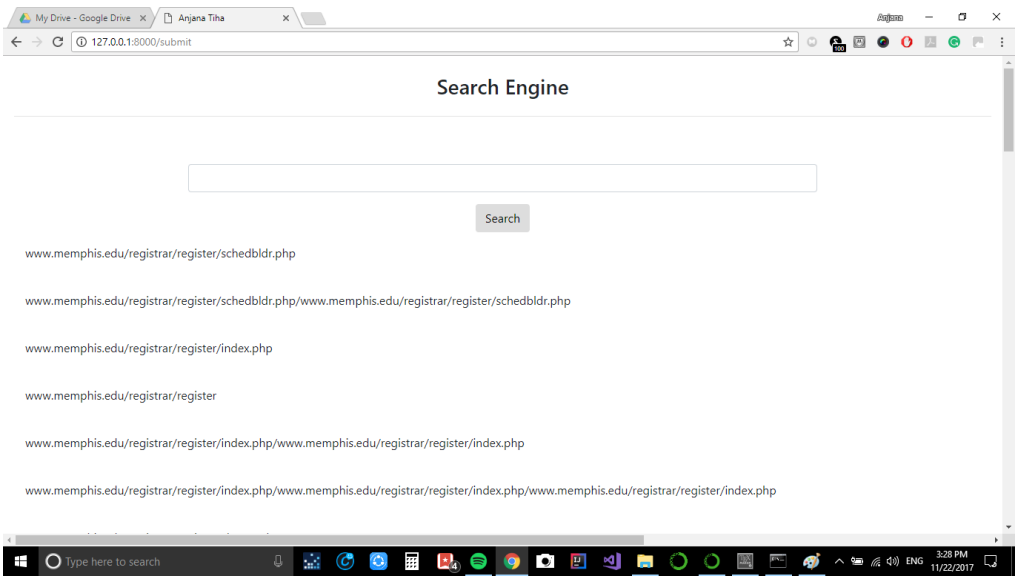


Figure 3: Web Search/Retrieval Engine Interface(Search Result)



Results

Figure 4: Precision recall and F1 score for websearch using first 200 results.

Query No	Query	Precision	Recall	F-1 Score
1	"computer science"	0.96	0.3254	0.4781
2	"Information Retrieval"	0.615	0.9248	0.7387
3	"TENTATIVE SCHEDULE"	0.775	0.9873	0.8683
4	" Liberty Bowl"	0.3	0.9523	0.4562
5	" Leading Memphis.Innovation. "	0.485	0.3579	0.4119
6	"State of Tech in Memphis"	0.95	0.3668	0.5292
7	"Greater Memphis"	0.935	0.4240	0.5834
8	"FedEx Institute of Technology"	1.0	0.0311	0.0604
9	"machine learning"	0.735	0.7616	0.7481
10	"quantum computing"	0.04	0.25	0.0689

The above was result was found by running 10 queries on the retrieval engine. For evaluation number of query result was set to 200 for optimal result. For query 1-"computer science", precision was very high. It may be due to the term being very common in memphis.edu domain. Relatively less common query, "quantum computing " resulted in very low precision and recall. "FedEx Institute of Technology" also resulted in low recall, as it is very common in memphis.edu domain. Therefore increasing number of documents may increase in better result. Overall, the queries result has to attain optimal result for most cases. Therefore 200 or such seems reasonable.

Future Work

In future, more web domain can be added for evaluating the performance of retrieval engine. Also, other retrieval model could be incorporated to improve result, including machine learning techniques for user based retrieval system. Also, natural language processing techniques can help build for semantically robust retrieval engine.

Conclusion

The web search or retrieval engine was implemented sequentially. I have another version partially developed in Perl. This project helped has gain essential knowledge of information retrieval system. In future, the interface can be developed more. Also, other retrieval techniques can be added to improve retrieval.

Bibliography

[1] Boyan, Justin, Dayne Freitag, and Thorsten Joachims. "A machine learning architecture for optimizing web search engines." In AAAI Workshop on Internet Based Information Systems, pp. 1-8. 1996.

[2] Lee, Dik L., Huei Chuang, and Kent Seamons. "Document ranking and the vector-space model." IEEE software 14, no. 2 (1997): 67-75.

[3] Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013). `https://en.wikipedia.org/wiki/Matrix_multiplication`